Synthesis Lectures on
Computer Architecture

SYNTHESIS
COLLECTION OF TECHNOLOGY

Luiz André Barroso · Urs Hölzle ·
Parthasarathy Ranganathan

The Data Center as a Computer

Designing Warehouse-Scale Machines

Fourth Edition

OPEN ACCESS

Springer

Synthesis Lectures on Computer Architecture

Series Editor

Natalie Enright Jerger, University of Toronto, Toronto, Canada

This series covers topics pertaining to the science and art of designing, analyzing, selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. The scope will largely follow the purview of premier computer architecture conferences, such as ISCA, HPCA, MICRO, and ASPLOS.

Luiz André Barroso · Urs Hölzle ·
Parthasarathy Ranganathan

The Data Center as a Computer

Designing Warehouse-Scale Machines

Fourth Edition

Springer

Luiz André Barroso
Google LLC (United States)
Mountain View, CA, USA

Urs Hölzle
Google LLC (United States)
Mountain View, CA, USA

Parthasarathy Ranganathan
Google LLC (United States)
Mountain View, CA, USA

ISSN 1935-3235 ISSN 1935-3243 (electronic)
Synthesis Lectures on Computer Architecture
ISBN 978-3-031-99488-3 ISBN 978-3-031-99489-0 (eBook)
https://doi.org/10.1007/978-3-031-99489-0

1st–3rd editions: © Springer Nature Switzerland AG 2022
Reprint of original edition © Morgan & Claypool 2019
4th edition: © The Editor(s) (if applicable) and The Author(s) 2026. This book is an open access publication.

Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

This fourth edition serves as an enduring memory of our dear friend and colleague, Luiz Barroso (1964–2023).[1] *A pioneer of Google's work on large-scale computing, Luiz was the first to see the benefits of sharing the ideas behind this new field with a broader audience, and it was he who coined the term Warehouse-Scale Computing. We will miss him dearly.*

[1] Cade Metz, New York Times. "Luiz Barroso, Who Supercharged Google's Reach, Dies at 59", https://www.nytimes.com/2023/12/13/technology/luiz-barroso-dead.html.

Foreword

When we wrote the first edition of this book in 2008-2009, warehouse-scale computers (WSCs) were exotic and rare, and primarily targeted at online services like web search. Mobile phones had barely started to enter the smartphone era, enterprise cloud computing was a niche, and artificial intelligence (AI) wasn't a thing. But today, WSCs are not just more widespread but also increasingly well-matched to these new applications as well. No longer exotic or niche, today even the physical data centers themselves have gained public awareness.

Thus, today it's best to view WSCs as the lower-level infrastructure that powers cloud computing of all kinds. When you're using GCP (Google Cloud), you're using the same WSC that all Google services are built on, and other public clouds like AWS or Azure have built similar WSCs for their own clouds. Via public clouds, WSCs have become societal infrastructure: the applications running in these clouds power everyday life, from Google's consumer applications to thousands of businesses, large and small, from banks to supermarkets to hospitals.

Last but not least, this latest edition serves as an enduring memory of our dear friend and colleague, Luiz Barroso (1964 - 2023)[2]. A pioneer of Google's work on large-scale computing, Luiz was the first to see the benefits of sharing the ideas behind this new field with a broader audience, and it was he who coined the term, Warehouse-Scale Computing. We miss him dearly.

[2] Cade Metz, New York Times. "Luiz Barroso, Who Supercharged Google's Reach, Dies at 59", https://www.nytimes.com/2023/12/13/technology/luiz-barroso-dead.html.

Preface

Warehouse-scale computers (WSCs) power cloud computing and all the great web services we use daily, including recent advancements in Artificial Intelligence/Machine Learning (AI/ML). This book examines how WSCs treat the data center itself as one massive computer designed at warehouse scale, with hardware and software working in concert. The book details the architecture of WSCs and covers the main factors influencing their design, operation, and cost structure, as well as the characteristics of their software base. Each chapter contains real-world examples, including detailed case studies and previously unpublished details of the infrastructure used to power Google's online services. Though targeted at the architects and programmers of today's WSCs, this book will also be broadly interesting to those who want to understand the infrastructure powering cloud computing and hyperscale and AI services.

The fourth edition reflects seven years of advancements since the previous edition and includes significantly revamped and enhanced content including four new chapters, new case studies, and new illustrations optimized for online reading. New topics include exascale AI processing, software-defined infrastructure, advanced networking, modular data center design, chip packaging and interconnects, silent data corruption, trusted computing, sustainability, and using AI to design better systems. Also new to this edition, the book includes a retrospective discussing key innovations and learnings from 25 years of warehouse-scale computing, and an annotated bibliography of our selection of top 25 papers from this period. Updated discussions of emerging trends and opportunities ensure that this revised edition remains an essential resource for educators and professionals working on the next generation of WSCs.

When we wrote the first edition of this book in 2008-2009, warehouse-scale computers (WSCs) were exotic and rare, and primarily targeted at online services like web search. Mobile phones had barely started to enter the smartphone era, enterprise cloud computing was a niche, and artificial intelligence (AI) wasn't a thing. But today, WSCs are not just more widespread but also increasingly well-matched to these new applications as well.

No longer exotic or niche, today even the physical data centers themselves have gained public awareness.

Thus, today it's best to view WSCs as the lower-level infrastructure that powers cloud computing of all kinds. When you're using GCP (Google Cloud), you're using the same WSC that all Google services are built on, and other public clouds like AWS or Azure have built similar WSCs for their own clouds. Via public clouds, WSCs have become societal infrastructure: the applications running in these clouds power everyday life, from Google's consumer applications to thousands of businesses, large and small, from banks to supermarkets to hospitals.

While we draw from our direct involvement in Google's infrastructure design and operation over the past several years, most of what we have learned and now report here is the result of the hard work, the insights, and the creativity of our colleagues at Google. The topics covered here are supported by the work of multiple teams at Google, and therefore, we are particularly grateful to them for allowing us to benefit from their experience.

Our thanks to numerous people who contributed and reviewed the content in the fourth edition: Dan Ardelean, Gary Borella, Dan Brougher, Mark Caine, Prashant Chandra, Liqun Cheng, David Culler, Mike Dahlin, Sundar Dev, Gaurav Dhiman, Amer Diwan, Nandita Dukkipati, Phillipa Gill, Md Ehtesam Haque, Marisabel Hechtman, Chris Heiser, Dave Hiniker-Roosa, Peter Hochschild, Tim Hockin, Kristine Jassmann, Saurabh Kadekodi, Svilen Kanev, Emily Kang, Sagar Karandikar, Leah Karasek, Natalie Kee Johnson, Darragh Kelly, Josh Kihm, David Lo, Maire Mahony, Jon McCune, Richard McDougall, Kathryn McKinley, Tipp Moseley, Sam McVeety, Ashish Naik, Pandu Nayak, Avinash Panga, Pavlos Papageorge, Christina Peabody, Pete Pellerzi, Lucas Pereira, Gang Ren, Varun Sakalkar, Kai Shen, Bob Silva, Arjun Singh, Joel Stephens, Suvinay Subramanian, Michael Terrell, Gautham Thambidorai, Amin Vahdat, and Matt Zelesko. Special thanks to Sagar Karandikar who helped with an end-to-end review of the book as well as helped with automating the conversion to the epub format. Thanks also to Emily Kang for her eagle-eyed review of everything, and to Dmytro Zaporozhtsev for the figures and graphs (as well as Andriy Achyn for earlier versions of some figures). We also appreciate and acknowledge the help from several people at Springer, notably Charles Glaser, Editorial director, as well as Ambrose Berkumans, production administrator. Thanks also to Michael Morgan and Natalie Enright Jerger for their support and encouragement of the overall series of Synthesis Lectures on Computer Architecture.

Our acknowledgements also go to the numerous people who helped with previous editions of this book.

The third edition benefited from contributions and feedback from: Jichuan Chang, Liqun Cheng, Mike Dahlin, Tim Hockin, Dave Landhuis, David Lo, Beckett Madden-Woods, Jon McCune, Tipp Moseley, Nishant Patil, Alex Ramirez, John Stanley, Varun Sakalkar, and Amin Vahdat. Ana Lopez-Reynolds, along with Marcos Armenta, Gary

Borella, and Loreene Garcia helped with all the illustrations, creating awesome new figures, and making sure all the illustrations had a consistent look. Thanks to our reviewers: Christos Kozyrakis and Thomas Wenisch for their valuable comments, and to Bob Silva for his proofreading.

The second edition greatly benefited from input and feedback from David Andersen, Parthasarathy (Partha) Ranganathan and Christos Kozyrakis, as well as contributions and corrections by Tor Aamodt, Dilip Agrawal, Remzi Arpaci-Dusseau, Mike Bennett, Liqun Cheng, Xiaobo Fan, David Guild, Matthew Harris, Mark Hennecke, Mark Hill, Thomas Olavson, Jack Palevich, Pete Pellerzi, John Reese, Ankit Somani and Amin Vahdat. We are also grateful for the feedback and corrections to the first edition submitted by Vijay Rao, Robert Hundt, Mike Marty, David Konerding, Jeremy Dion, Juan Vargas, Artur Klauser, Pedro Reviriego Vasallo, Amund Tveit, Xiau Yu, Bartosz Prybylski, Laurie Doyle, Marcus Fontoura, Steve Jenkin, Evan Jones, Chuck Newman, Taro Tokuhiro, Jordi Torres, Christian Belady and Catherine Warner. Thanks to Robert Stroud, Andrew Morgan, and others who helped with data and figures. Special thanks to Jimmy Clidaras who wrote the original content on cooling and power distribution for Chapter 5 of the second edition.

Thanks to the work of Kristin Berdan and Michael Morgan at Morgan & Claypool, we were able to make this lecture available electronically without charge. We are very grateful to Mark Hill and Michael Morgan for inviting us to this project, and to Michael and Margaret Martonosi for the encouragement to make this book possible. Ricardo Bianchini, Fred Chong, Jeff Dean, and Mark Hill provided extremely useful feedback on the proposal and early drafts of the first edition of this book.

And finally, we appreciate all the support for this book, from you, our readers, and all the feedback on the prior three editions of this book. We will continue to appreciate any thoughts, suggestions, or corrections you might have on this edition as well.

Sunnyvale, California — Urs Hölzle
April 2025 — Parthasarathy Ranganathan

Competing Interests The authors have no competing interests to declare that are relevant to the content of this manuscript.

Abstract

Warehouse-scale computers (WSCs) power cloud computing and all the great web services we use daily, including recent advancements in Artificial Intelligence/Machine Learning (AI/ML). This book examines how WSCs treat the data center itself as one massive computer designed at warehouse scale, with hardware and software working in concert. The book details the architecture of WSCs and covers the main factors influencing their design, operation, and cost structure, as well as the characteristics of their software base. Each chapter contains real-world examples, including detailed case studies and previously unpublished details of the infrastructure used to power Google's online services. Though targeted at the architects and programmers of today's WSCs, this book will also be broadly interesting to those who want to understand the infrastructure powering cloud computing and hyperscale and AI services.

The fourth edition reflects six years of advancements since the previous edition and includes significantly revamped and enhanced content including four new chapters, new case studies, and new illustrations optimized for online reading. New topics include exascale AI processing, software-defined infrastructure, advanced networking, modular data center design, chip packaging and interconnects, silent data corruption, trusted computing, sustainability, and using AI to design better systems. Also new to this edition, the book includes a retrospective discussing key innovations and learnings from 25 years of warehouse-scale computing, and an annotated bibliography of our selection of top 25 papers from this period. Updated discussions of emerging trends and opportunities ensure that this revised edition remains an essential resource for educators and professionals working on the next generation of WSCs.

Contents

1 Introduction 1
- 1.1 Setting the context 1
- 1.2 Warehouse-Scale Computers (WSCs) 2
- 1.3 Cost efficiency at scale 4
- 1.4 Why WSCs might matter to you 5
- 1.5 Overview of the book 6
- 1.6 What is new in the fourth edition 6

2 Overview of WSC Architecture 9
- 2.1 Regions and zones 9
- 2.2 Buildings and Infrastructure 11
- 2.3 Servers 13
- 2.4 Storage 14
- 2.5 Networking Fabric 16
- 2.6 Energy Usage 17
- 2.7 Handling Failures and Repairs 17
- References 18

3 Overview of WSC Workloads 19
- 3.1 WSC workloads 19
  - 3.1.1 WSC systems stack 19
  - 3.1.2 Compute consumption models 21
  - 3.1.3 Public cloud vs hyperscale 22
- 3.2 Hyperscale applications 23
  - 3.2.1 Broad overview 23
  - 3.2.2 Case study: Search 24
  - 3.2.3 Case study: YouTube 27
  - 3.2.4 Case study: Gmail 28

3.3 AI workloads . . . 30
3.3.1 Training . . . 32
3.3.2 Inference . . . 32
3.3.3 Large Language Models . . . 32
3.3.4 AI computation characteristics . . . 33
3.4 Case study: Google Compute Engine . . . 35
3.4.1 Verified boot . . . 35
3.4.2 Tiered offloads . . . 36
3.4.3 Provisioning workflows . . . 37
3.4.4 Live migration . . . 38
3.5 Case study: Data processing infrastructure . . . 39
3.5.1 BigQuery . . . 39
3.5.2 Cloud Dataflow . . . 41
3.6 WSC workload tradeoffs and implications for infrastructure . . . 42
3.6.1 Replication and sharding . . . 43
3.6.2 Error codes and integrity checking . . . 43
3.6.3 Brownouts . . . 44
3.6.4 Health monitoring and canaries . . . 45
3.6.5 Load balancing . . . 45
3.6.6 Consensus . . . 46
3.6.7 Compression and encryption . . . 47
3.6.8 Tail tolerance . . . 47
3.7 Some recurring themes . . . 48
3.7.1 Ample parallelism . . . 48
3.7.2 Workload evolution . . . 49
3.7.3 Platform homogeneity and abstraction . . . 49
3.7.4 Fault management . . . 50
References . . . 51

4 WSC Software: Distributed Systems Infrastructure . . . 55
4.1 Overview of the software stack . . . 55
4.2 Distributed systems . . . 55
4.2.1 Cluster-level infrastructure . . . 55
4.2.2 Unstructured storage systems . . . 58
4.2.3 Structured WSC storage . . . 61
4.2.4 WAN and edge networking . . . 62
4.3 Infrastructure management . . . 66
4.3.1 Resource provisioning . . . 66
4.3.2 Configuration and observability . . . 68
4.3.3 Practices: DevOps and SRE . . . 71

4.4 Server-level software ... 72
4.4.1 OS and firmware ... 72
4.4.2 Containerization and virtualization ... 73
References ... 74

5 WSC Hardware: Data Center Infrastructure ... 79
5.1 Data center infrastructure basics ... 79
5.1.1 Planet to campuses to buildings ... 79
5.1.2 Basic structure of a data center ... 79
5.1.3 Data center taxonomy: Tiers ... 81
5.1.4 Cloud and AI implications for data center design ... 83
5.2 Data center power systems ... 84
5.2.1 Overall electrical power delivery architecture ... 84
5.2.2 Comparing AC and DC power distribution architectures ... 88
5.2.3 Uninterrupted power supplies (UPS) ... 89
5.2.4 Power distribution units (PDUs) ... 90
5.2.5 Power microgrids and other future opportunities ... 91
5.3 Data center cooling systems ... 93
5.3.1 Overall cooling/mechanical architecture ... 94
5.3.2 Computer room air conditioners ... 96
5.3.3 Chillers ... 97
5.3.4 Free cooling ... 98
5.3.5 Cooling towers ... 99
5.3.6 Airflow considerations ... 101
5.3.7 In-rack, in-row, and liquid cooling ... 102
5.4 Other data center considerations ... 106
5.4.1 Modular data center design for velocity and efficiency ... 106
5.4.2 Economic models: The five envelopes ... 107
5.4.3 Physical automation in the data center ... 108
5.5 Historical retrospective ... 111
References ... 113

6 WSC Hardware: Computing Building Blocks ... 117
6.1 WSC building blocks and design considerations ... 117
6.1.1 Overall architecture ... 117
6.1.2 Building scale-out systems ... 117
6.2 Computing: servers and racks ... 123
6.2.1 Server hardware ... 123
6.2.2 Hardware racks ... 126
6.2.3 Individual servers as distributed systems ... 129

6.3 Accelerators and custom silicon . . . 133
6.3.1 Specialization and the end of Moore's Law . . . 133
6.3.2 ML accelerators and TPU systems . . . 133
6.3.3 Video accelerators and VCU systems . . . 139
6.4 Networking . . . 143
6.4.1 Overall networking from tray to planet . . . 143
6.4.2 Host networking and smartNICs . . . 143
6.4.3 Cluster networking . . . 146
6.4.4 Spine-less networking . . . 149
6.4.5 Optics in warehouse-scale data centers . . . 151
6.4.6 Special-purpose networks . . . 152
6.4.7 WANs . . . 152
6.5 Storage . . . 153
6.5.1 Storage media . . . 153
6.5.2 Diskless servers . . . 155
6.5.3 Disk appliances . . . 155
6.5.4 Flash appliances . . . 156
6.5.5 Smart storage . . . 158
6.6 Cross-cutting issues . . . 158
6.7 Historical retrospective . . . 159
6.7.1 Servers and accelerators . . . 159
6.7.2 Networking . . . 160
6.7.3 Storage . . . 161
References . . . 162

7 Software-Defined Infrastructure . . . 167
7.1 Software-defined servers . . . 167
7.1.1 Case study: Compiler optimizations . . . 170
7.1.2 Case study: Library optimizations . . . 172
7.1.3 Case study: Platform-aware scheduling . . . 173
7.1.4 Case study: Software-defined memory . . . 174
7.2 Software-defined accelerators . . . 176
7.3 Software-defined networks . . . 177
7.3.1 Jupiter topology and traffic engineering . . . 179
7.3.2 Network aware scheduling . . . 180
7.3.3 Bandwidth enforcer . . . 180
7.3.4 B4 traffic engineering . . . 180
7.3.5 DDoS attack mitigation . . . 181
7.3.6 What's next for SDN . . . 182
7.4 Software-defined storage . . . 182
7.4.1 Storage workload diversity . . . 183
7.4.2 Transparent SSD caching . . . 184

7.4.3 Trading data center space between compute and storage ... 185
7.4.4 Other software-managed hardware optimizations 185
7.5 Software-defined data center 185
7.5.1 Software-defined power 185
7.5.2 Software-defined fleet 187
7.6 Self-driving systems 188
References 189

8 Performance and Costs 193
8.1 Performance 193
8.1.1 Broad system performance characterizations 193
8.1.2 Fleetwide workload characterization 196
8.1.3 Performance modeling 197
8.2 Costs 200
8.2.1 Total cost of ownership model 200
8.2.2 TCO breakdowns and case studies 204
8.2.3 Real world costs: utilization matters! 205
8.2.4 Cost advantages of public clouds 207
References 211

9 Energy and Sustainability 213
9.1 The importance of energy in WSCs 213
9.2 Energy efficiency of data center infrastructure 214
9.2.1 PUE 215
9.2.2 Issues with the PUE metric 216
9.2.3 Breakdown of energy inefficiencies 218
9.2.4 Improving data center efficiency 219
9.2.5 Beyond PUE to SPUE 220
9.3 Energy efficiency of computing 222
9.3.1 Measuring energy efficiency 222
9.3.2 Improving energy efficiency: Energy proportionality 223
9.4 Improving energy efficiency via specialization 237
9.5 Data center power provisioning 239
9.5.1 Power stranding 239
9.5.2 Power oversubscription: The concept 240
9.5.3 Oversubscription variants 242
9.5.4 Case study: Power oversubscription at Google 243
9.5.5 Intermittent energy supply 245
9.5.6 Other energy optimizations 246
9.6 Environmental sustainability for data centers 247
9.6.1 Emissions: Scope 1, 2, 3 247
9.6.2 Carbon-aware scheduling 250

9.6.3 Water ... 250
9.6.4 Other sustainability considerations ... 251
9.7 Energy usage of AI ... 252
9.8 Energy usage of the internet ... 255
9.8.1 Data center energy usage ... 255
9.8.2 Cryptocurrency energy usage ... 256
9.8.3 End-to-end internet energy usage ... 257
References ... 260

10 Trusted Computing: Reliability, Availability, Security, Privacy ... 269
10.1 Fault tolerance at scale ... 269
10.1.1 Introduction to fault tolerance at scale ... 269
10.1.2 Fault tolerance in hardware vs software ... 271
10.1.3 Categorizing faults ... 273
10.1.4 Machine-level failures ... 277
10.1.5 Storage errors ... 282
10.1.6 Predicting and localizing faults ... 284
10.1.7 Repairing faults ... 285
10.1.8 Tolerating faults ... 286
10.1.9 Zonal and regional failures ... 287
10.1.10 Silent data corruptions and corrupt execution errors ... 289
10.1.11 Operational practices ... 290
10.2 Security at warehouse scale ... 292
10.2.1 Data center security: Building and systems ... 292
10.2.2 Security of servers ... 294
10.2.3 Root of trust ... 295
10.2.4 Encryption ... 297
10.2.5 Confidential computing ... 299
10.2.6 CPU vulnerabilities ... 300
10.2.7 Secure service deployment ... 301
10.2.8 Additional reading ... 303
References ... 303

11 A Twenty Five Year WSC Journey ... 309
11.1 Historical retrospective of twenty-five years of WSC at Google ... 309
11.1.1 Epoch 1: Scale-out computing, scrappy specialization ... 311
11.1.2 Epoch 2: Foundational WSC innovations ... 312
11.1.3 Epoch 3: Getting networking and security right ... 313
11.1.4 Epoch 4: Accelerators, scaling Moore's Law ... 314
11.1.5 Epoch 5: Cloud & AI, building sustainable societal infrastructure ... 315

11.2 Key lessons from 25 years of WSCs ... 318
11.2.1 Scale as a first-class design consideration ... 318
11.2.2 Risk management ... 318
11.2.3 Managing security deeply ... 319
11.2.4 Vertically-integrated systems design ... 319
11.2.5 Hardware-software co-design and specialization ... 320
11.2.6 Technology islands and industry ecosystems ... 320
11.2.7 A culture of landings versus launches ... 321
11.2.8 A culture of revisiting successful decisions ... 321
11.2.9 Creating moonshots through roofshots ... 321
11.3 Looking ahead ... 322
11.3.1 Efficiency innovation to extend Moore's Law ... 322
11.3.2 Optimizing velocity ... 323
11.3.3 Trusted computing and sovereignty ... 323
11.3.4 Sustainability for a societal infrastructure ... 323
11.3.5 AI for WSC design ... 324
11.4 Closing remarks ... 324
References ... 325

12 A Selection of 25 Top Papers for 25 Years of WSC ... 329
12.1 The anatomy of search: Google's early architecture ... 330
12.2 Web Search for a Planet: The Google Cluster Architecture ... 330
12.3 The Google File System ... 330
12.4 MapReduce: Simplified Data Processing on Large Clusters ... 331
12.5 Bigtable: A Distributed Storage System for Structured Data ... 331
12.6 The Chubby Lock Service for Loosely-coupled Distributed Systems ... 332
12.7 The Case for Energy-Proportional Computing ... 332
12.8 Dremel: Interactive Analysis of Web-Scale Datasets ... 333
12.9 Dapper, a Large-Scale Distributed Systems Tracing Infrastructure ... 333
12.10 Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers ... 334
12.11 Spanner: Google's Globally-Distributed Database ... 334
12.12 B4: Experience with a Globally Deployed Software Defined WAN ... 335
12.13 The Tail at Scale ... 335
12.14 Profiling a Warehouse-scale Computer ... 335
12.15 Large-scale cluster management at Google with Borg ... 336
12.16 Site Reliability Engineering: How Google Runs Production Systems ... 336
12.17 In-Data center Performance Analysis of a Tensor Processing Unit ... 337

12.18 Attack of the killer microseconds 337
12.19 Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization 337
12.20 Snap: a Microkernel Approach to Host Networking 338
12.21 Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale 338
12.22 Swift: Delay is Simple and Effective for Congestion Control in the Data center 339
12.23 Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild 339
12.24 Jupiter Evolving: Transforming Google's Data center Network via Optical Circuit Switches and Software-Defined Networking 340
12.25 Pathways: Asynchronous Distributed Dataflow for ML 340
References 341

Glossary 343

List of Figures

Fig. 2.1 Google Cloud Platform regions as of March 2025 10
Fig. 2.2 Aerial view of Google data center, Iowa, USA 11
Fig. 2.3 Power and network distribution 12
Fig. 2.4 Data center cooling hall 12
Fig. 2.5 Cooling towers and water storage tanks 13
Fig. 2.6 Building blocks for WSCs: server, TPU accelerator, disk tray 14
Fig. 2.7 Top: traditional compute servers; bottom: a TPU accelerator pod 15
Fig. 2.8 Power usage in a 2024 Google server 17
Fig. 3.1 Google software stack for warehouse-scale computers 20
Fig. 3.2 Diversity of workloads in WSCs in 2014 and 2024 23
Fig. 3.3 Logical view of a web index 25
Fig. 3.4 Daily traffic for a search service in one data center 26
Fig. 3.5 The YouTube video processing pipeline 28
Fig. 3.6 A Deep Neural Network (DNN) 31
Fig. 3.7 Development and deployment phases of a DNN model 33
Fig. 3.8 Compute demand of ML workloads over the past decade 34
Fig. 3.9 GCE node with tiered offloads 35
Fig. 3.10 Verified boot sequence 36
Fig. 3.11 GCE provisioning flow 38
Fig. 3.12 BigQuery architecture 41
Fig. 3.13 Cloud Dataflow architecture 42
Fig. 3.14 Service latency variation with tail events 48
Fig. 4.1 Google software stack for warehouse-scale computers 56
Fig. 4.2 Simplified architecture of the Colossus file system 59
Fig. 4.3 Media Content Distribution Network (CDN) locations 64
Fig. 4.4 Structure of Google's global network 65
Fig. 4.5 Google's global submarine network 66
Fig. 5.1 Different Google data centers 80

Fig. 5.2 Aerial view of a Google data center campus 81
Fig. 5.3 A Google data center building 81
Fig. 5.4 The main components of a typical data center 82
Fig. 5.5 Power density distributions with and without accelerators 84
Fig. 5.6 Radial power architecture 85
Fig. 5.7 Concept for medium voltage power plane architecture 86
Fig. 5.8 Concept for distributed redundant (DR) architecture (HAC = hot aisle containment) 87
Fig. 5.9 Common AC distribution architecture 88
Fig. 5.10 DC distribution architecture 89
Fig. 5.11 Microgrids overview 92
Fig. 5.12 Fluctuating power draw of a ML training run 93
Fig. 5.13 Raised floor data center with hot-cold aisle setup 94
Fig. 5.14 Three-loop data center cooling system 95
Fig. 5.15 Water-cooled centrifugal chiller 97
Fig. 5.16 Airflow schematic of an air-economized data center 98
Fig. 5.17 Data center cooling towers 100
Fig. 5.18 How a cooling tower works 100
Fig. 5.19 CFD model of a rack with under-provisioned airflow 101
Fig. 5.20 In-row air conditioner 103
Fig. 5.21 Cross-section of a cold aisle and associated hot air plenums 103
Fig. 5.22 Copper cold plates provide liquid cooling for TPUv3 104
Fig. 5.23 Cooling distribution unit (CDU) and a row of TPU racks 105
Fig. 5.24 Hierarchical composition in modular data centers 106
Fig. 5.25 Five envelopes of data center design 108
Fig. 5.26 Rack Mover for moving Google racks 109
Fig. 5.27 Disk shredder with media verification device 110
Fig. 5.28 Early colo with Google servers and “external cooling aid” 111
Fig. 5.29 Google’s 2005 container design 112
Fig. 5.30 A prototype container being installed on the Google campus 113
Fig. 6.1 Performance in hypothetical SMP clusters 119
Fig. 6.2 Deteriorating performance advantage of large SMP servers 120
Fig. 6.3 Annotated breakdown of a server tray 124
Fig. 6.4 Example server trays: Intel, AMD, and ARM 125
Fig. 6.5 Block diagram of a two-socket server 126
Fig. 6.6 Block diagram of a bifurcated 2x 1-socket system 127
Fig. 6.7 Empty and populated server racks 128
Fig. 6.8 Rack design for accelerator rack 129
Fig. 6.9 Evolution of servers into multi-node machines 129
Fig. 6.10 Single-node vs. multi-node “multi-brained” systems 130
Fig. 6.11 Multi-arena architecture for servers 132

Fig. 6.12 TPU generations over the years 134
Fig. 6.13 TPU chip block diagram 135
Fig. 6.14 OCS interconnect topology 136
Fig. 6.15 Illustrative TPU pod (4096-chip TPU-v4 pod) 137
Fig. 6.16 GPUs connected to servers 138
Fig. 6.17 GPUs racks with 2U server trays connected to 8U GPU trays 139
Fig. 6.18 Overview of VCU system architecture at all scales 140
Fig. 6.19 VCU chip floorplan 141
Fig. 6.20 Two VCU chips on a board 141
Fig. 6.21 VCU transcode throughput in VCU versus CPU and GPU systems 142
Fig. 6.22 Overview of Google's IPU integration into the host 144
Fig. 6.23 Google's Titanium IPU: PCIe card and key architectural blocks 145
Fig. 6.24 Sample three-stage Clos topology 146
Fig. 6.25 Jupiter network topology and photo of Jupiter rack 148
Fig. 6.26 Operating principle of an Optical Circuit Switch 150
Fig. 6.27 OCS connecting aggregation blocks 151
Fig. 6.28 4 MB IBM 350 hard disk in 1956, 4 TB 3.5" disk in 2013 154
Fig. 6.29 Rack of disk appliances; block diagram of a disk tray 156
Fig. 6.30 An SSD appliance with 16 slots 157
Fig. 7.1 Software-defined servers: Abstraction and efficiency layers 170
Fig. 7.2 AutoFDO end-to-end system diagram 171
Fig. 7.3 Flow of requests by size into components in TCMalloc 172
Fig. 7.4 An application's relative performance on different platforms 174
Fig. 7.5 Tiered memory architecture 175
Fig. 7.6 Simulated vs actual miss rates in first tier of memory 176
Fig. 7.7 Theoretical disk sizes (bytes versus IOPS) 184
Fig. 7.8 Power capping architecture 186
Fig. 7.9 Power throttling response over time 187
Fig. 8.1 Storage hierarchy of a WSC 194
Fig. 8.2 Latency and bandwidth of WSC storage hierarchy levels 195
Fig. 8.3 Latency numbers that every WSC engineer should know 196
Fig. 8.4 Fleetwide cycle spent on "data center tax" 197
Fig. 8.5 Tiered performance simulation approach 198
Fig. 8.6 Bitcoin mining data hut in Alberta, Canada 202
Fig. 8.7 Cost breakdown for case study A (left) and B (right) 205
Fig. 8.8 TCO case study C (50% filled facility) 207
Fig. 8.9 On-premises vs. cloud comparison 210
Fig. 9.1 2012 survey of PUE for 1100+ data centers 216
Fig. 9.2 Uptime Institute survey tracking PUE over time 216
Fig. 9.3 Average global PUE for Google data centers over time 217

Fig. 9.4 Energy losses in an average low-efficiency data center 219
Fig. 9.5 SPUE overhead in a 2024 Google server 221
Fig. 9.6 CPU utilization of Google clusters in 2013 and 2024 223
Fig. 9.7 CPU utilization of Google fleet sample 224
Fig. 9.8 CPU utilization across Google Cloud zones 225
Fig. 9.9 Example benchmark result for SPECpower_ssj2008 226
Fig. 9.10 Power and efficiency of three hypothetical systems 227
Fig. 9.11 Power usage in an x86 server optimized for performance 229
Fig. 9.12 Normalized system power of servers from 2007 to 2022 230
Fig. 9.13 Efficiency impact of Turbo on Demand 233
Fig. 9.14 Power fluctuations in a synchronized ML workload 236
Fig. 9.15 Unoptimized and optimized power and temperature variation 237
Fig. 9.16 Relative performance of GPUs, TPUs, and CPUs 238
Fig. 9.17 Cumulative distribution of time spent below a given power level 241
Fig. 9.18 Block diagram for the power capping service 244
Fig. 9.19 Nonproduction power capping while on generator 245
Fig. 9.20 Estimates of global data center energy use 253
Fig. 9.21 Global IT energy usage across categories 257
Fig. 9.22 Differing analyses of ICT's carbon footprint, excluding TVs 259
Fig. 9.23 Estimates for global ICT's carbon footprint in 2020 260
Fig. 10.1 Machine restarts at Google over six months in 2018 277
Fig. 10.2 Machine downtime observed over six months 278
Fig. 10.3 Distributions of machine crashes at Google over six months in 2024 280
Fig. 10.4 Google's system health monitoring and analysis infrastructure 286
Fig. 10.5 Highly available cluster architecture 287
Fig. 10.6 Structure of a cloud service with zones and regions 289
Fig. 10.7 Circular single-person door to prevent tailgating 293
Fig. 10.8 Data encryption at rest in Google storage 298
Fig. 11.1 Twenty five years of warehouse-scale computing at Google 310
Fig. 11.2 First Google colocation order 311
Fig. 11.3 Generations of TPUs 314
Fig. 11.4 Multiple generations of servers 317
Fig. 11.5 Housing servers in warehouse-scale computers: An evolution 317
Fig. 11.6 Timeline for key WSC innovations, both internal and open source 317

Introduction

1

1.1 Setting the context

The modern internet was born in 1983, yet the technologies that emerged from this remarkable milestone continue to transform industries and cultures today, with no signs of slowing down. Mobile devices combined with cheap, high-speed connectivity have accelerated a trend toward server-side or "cloud" computing that extends the limited compute and storage capacity of these devices. While this movement started in consumer services more than 25 years ago, it's now also embraced by mainline enterprise workloads. In 2024, the combined cloud computing revenue of AWS, Azure, and GCP exceeded $200B.

Computation and storage have thus moved away from the phone, laptop, and desktop computers and the corporate server room to warehouse-scale cloud data centers. The end-user client doesn't need to store all the photos, emails, documents, CAD drawings locally; it merely caches the most recent ones locally and leaves actual storage to the cloud.

Cloud-based computing offers many advantages. Web-based applications and app stores with auto-update functionality greatly increase software development velocity; nearly every client is on the same version of the application, and new releases can happen frequently. Similarly, server-side computing allows for faster hardware innovations, such as accelerators that can be encapsulated with well-defined interfaces and APIs. Combining these two advantages allows new functionality to be introduced more quickly. For example, a new speech recognition algorithm may initially be too expensive to run locally on a phone but can run in a cloud data center, so users don't need to wait for better phone hardware to get better speech recognition. As newer phones become more capable, some or all of the new functionality may migrate back to the phone for better latency or economics.

Moreover, data center economics allow many application services to run at a substantially lower cost per user: servers shared among thousands of active users (and many more inactive

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_1

ones) enable better utilization. Similarly, the computation itself may become cheaper in a shared service; for example, an email attachment received by multiple users can be stored once rather than many times. Finally, servers and storage in a data center are easier to manage with automated tools, thereby also lowering costs.

Cloud computing can be made much more secure than traditional on-premise IT systems. In part, that's simply a consequence of much better updateability: you can't secure what you can't update. But more fundamentally, a key advantage of cloud computing is its simplified governance: all hardware, software, and operational practices can be designed and implemented together, resulting in a more coherent architecture that is easier to secure. In comparison, every on-premise IT data center is a snowflake, unlike any other IT data center, because it arranges different versions of the same systems in different ways, making (and having made) different architectural and operational choices than everyone else. Securing a million moving targets is literally impossible, and thus the common integration point of a well-curated cloud stack fundamentally changes what is possible in information security.

These trends toward server-side computing combined with the exploding popularity of internet services created a new class of computing systems that, in the first edition of this book, we named warehouse-scale computers, or WSCs. By calling them WSCs, we wanted to call attention to the most distinguishing feature of these machines: the massive scale of their hardware and software infrastructure. This perspective is a departure from a view of the computing problem that implicitly assumes a model where one program runs on a single machine. In warehouse-scale computing, the program is a large-scale service, consisting of potentially hundreds of individual services that interact to implement complex services such as public clouds, search, or YouTube. These programs might be implemented and maintained by different teams of engineers, perhaps even across organizational, geographic, and company boundaries.

The computing platform required to run such large-scale services bears little resemblance to a pizza-box server or the refrigerator-sized, high-end multiprocessors popular in prior decades. The hardware for such a platform consists of thousands of individual computing nodes with their corresponding networking and storage subsystems, power distribution and conditioning equipment, and extensive cooling systems. The enclosure for these systems is in fact a building, which is often indistinguishable from a large warehouse. And, as hinted above, the fundamentally different architecture of WSCs allows much better approaches for developing, operating, and securing systems.

1.2 Warehouse-Scale Computers (WSCs)

Had scale been the only distinguishing feature of these systems, we might simply refer to them as data centers. Data centers are buildings where multiple servers and communication gear are co-located for ease of operation. In that sense, a WSC is just a large data center. Traditional data centers, however, typically host a large number of relatively small-

or medium-sized companies, each running on dedicated hardware that is decoupled and protected from other systems in the same facility. Different computing systems within such a data center often have little in common in terms of hardware, software, or maintenance infrastructure, and tend not to communicate with each other at all.

In recent years, the term "hyperscale" has become popular to denote large-scale infrastructure. While the two terms are related, they're not the same. For example, a pure data center provider or colo provider may build a large data center that's of a scale similar to WSCs, but they provide only the building, excluding the software stack that could turn all the servers in that building into a warehouse-scale computer. The term "hyperscaler" is commonly used to describe companies who build both WSCs and a WSC stack, either purely for their own hyperscale applications (Meta), to provide cloud computing (AWS, Microsoft), or both (Google).

Such WSCs differ significantly from traditional data centers: they belong to a single organization, use a relatively homogeneous hardware and system software platform, share a common systems management layer, and are one or even two orders of magnitude larger than corporate data centers. Often, much of the application, middleware, and system software is built in-house compared to the predominance of third-party software running in conventional data centers. Most importantly, WSCs can run very large applications (or internet services) that individually might use tens of thousands of servers each, and the common resource management infrastructure allows significant deployment flexibility. The requirements of homogeneity, single-organization control, and enhanced focus on cost efficiency motivate designers to take new approaches in constructing and operating these systems.

Though initially designed for online data-intensive web workloads, WSCs now power public clouds such as those from Amazon, Google, and Microsoft. While such public clouds do run many small applications, as seen by the provider, all of these applications are identical VMs, and they access large, common services for block or database storage, load balancing, and so on, fitting very well with the WSC model. To Google's infrastructure, Google Compute Engine (GCE)—the GCP (Google Cloud Platform) service providing VMs—looks like a single large application. Google is unique in having both hyperscale consumer applications as well as a large public cloud, and in later chapters we'll see how the same infrastructure can support both large online internal services as well as large public cloud offerings.

Internet services must achieve high availability, typically aiming for at least 99.99% uptime ("four nines," or about an hour of cumulative downtime per year). Achieving fault-free operation on a large collection of hardware and system software is hard and is made more difficult by the large number of servers involved. Although it might be theoretically possible to prevent hardware failures in a collection of 10,000 servers, it would surely be expensive. Consequently, WSC workloads must be designed to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability.

The technical challenges of designing WSCs are as worthy of the expertise of computer systems architects than any other class of machines. First, they are a new class of large-scale machines driven by a new and rapidly evolving set of workloads. Their size alone makes them

difficult to experiment with or simulate efficiently; therefore, system designers must develop new techniques to guide design decisions. Second, fault behavior, security, power, and energy considerations have a more significant impact in the design of WSCs. Third, WSCs have an additional layer of complexity beyond systems consisting of individual servers or small groups of servers; WSCs introduce a significant new challenge to programmer productivity, a challenge even greater than programming the individual multicore systems that comprise the WSC. This additional complexity arises indirectly from virtualization and the larger scale of the application domain and manifests itself as a deeper and less homogeneous storage hierarchy, with higher fault tolerance, and possibly with higher performance variability.

In this book, we discuss WSCs as if they were one data center, even though many internet services use multiple data centers located far apart. Multiple data centers are sometimes used as complete replicas of the same service, with replication used mostly to reduce user latency and improve serving throughput. In those cases, a given user request is fully processed within one data center, and our machine definition is appropriate. In other cases, services are "sharded" across multiple WSCs, so that some users are served from one location and others from another. Finally, some services (especially for storage, database, and analytics workloads) transparently replicate across multiple WSCs for higher availability and durability.

1.3 Cost efficiency at scale

Computers are expensive to buy, and they need to be replaced every 4-6 years. Additionally, they're expensive to maintain and manage: Do all of them have the latest security updates installed? Which ones need to be repaired or upgraded? Will this cluster run out of SSD storage next month? Which machines run at low utilization and why? Can we take a cooling tower offline for cleaning next week, or will the weather be too warm? Where on the data center floor should this new rack go?

In a smaller enterprise data center, almost all of these questions require people to answer them, and more people to take action on the answers. One of the biggest benefits of WSCs is that such questions become easier because they are harder. That is, they require automation because manual processes would be too expensive and error prone, and the scale justifies the investment in automation. As a result, they become easier, because they're either fully or partially automated.

Take hardware failures as an example. Let's assume servers have an annualized failure rate (AFR) of 5% (one failure every 20 years), and that a typical enterprise data center has 100 servers, implying an average of one failure every two months. Clearly, that's not worth automating, so when a server fails it will likely cause several hours of work to coordinate and execute the repair.

In a WSC with 100,000 servers, on average one server will fail every two hours, around the clock. Even without sophisticated automation, the cost per repair will decrease by a

significant factor: a repair tech will be on site every day, and someone will have at least partially automated the OS reinstall and the inventory system update, and so on. In a large cloud provider with millions of servers world-wide, that automation is even more mature and thus the cost per repair will be even lower.

A similar effect improves other operational costs: data centers get cheaper and more reliable because providers have built dozens of them already, working out the bugs and standardizing the designs. Energy efficiency improves, because it's worth having a specialized energy efficiency team when your power bill can be over a billion dollars a year.

In summary, the scale of WSC computing capacity makes cost efficiency a primary metric of interest in the design and operation of WSCs. Cost efficiency must be defined broadly to account for all the significant components of cost, including hosting-facility capital and operational expenses (which include power provisioning and energy costs), hardware, software, management personnel, and repairs. Chapter 8 discusses these issues in more detail.

1.4 Why WSCs might matter to you

In the first edition of the book (2009), we discussed how WSCs might be considered a niche area, because their sheer size and cost render them unaffordable by all but a few large internet companies. We argued then that this wasn't true, and that the problems large internet services face would soon be relevant to a much larger constituency because many organizations would be able to afford similarly sized computers at a much lower cost.

That intuition has since come true. Today, the attractive economics of cloud computing platforms puts clusters of thousands of nodes within the reach of a relatively broad range of corporations and research institutions. Combined with the trends around ever-increasing numbers of processor cores on a single die, a single rack of servers today has more hardware threads than many corporate data centers had a decade ago. For example, a rack with 40 servers, each with two 100-core CPUs, contains 8,000 cores! Such systems exhibit the same scale, architectural organization, and fault behavior of WSCs from the last decade.

The objectives of this book are to introduce this design space, describe some of the requirements and characteristics of WSCs, highlight some of the important challenges unique to this space, and share some of our experience designing, programming, and operating them within Google. We are fortunate to be not only designers of WSCs but also customers and programmers of the platform. This has provided us an unusual opportunity to evaluate design decisions throughout the lifetime of a product. We hope that we succeed in relaying our enthusiasm for this area as an exciting target worthy of the attention of the research and technical communities.

1.5 Overview of the book

The first part of the book provides a discussion of the full WSC stack: hardware, software, and services. We first discuss the architecture of WSCs at a high level (Chapter 2). Chapter 3 provides a deeper overview of the key use cases for WSCs, the top two being public clouds and AI training and serving. Additionally, a few individual hyperscale services are so large that they need to be architected specifically for WSCs; for example, Google search runs on many millions of CPU cores. For services this large, even the best public cloud infrastructure can't relieve the developers from understanding some WSC attributes like physical constraints and failure modes. Chapter 4 reviews the software systems needed to run WSCs, from distributed software systems to node management.

The second part delves deeper into WSC infrastructure. Chapter 5 discusses the physical layer of WSCs: buildings, power, and cooling. Chapter 6 then covers the equipment hosted in these facilities, including servers, accelerators, networking, and storage. Chapter 7 introduces the concept of software-defined infrastructure, an approach making fixed hardware blocks more flexible and manageable via software so that users can maximize their benefits without having to understand every detail of the underlying hardware.

In the third part, we discuss how WSCs optimize for important metrics that the designer (and end-user) care about. Chapter 8 discusses performance and costs and builds a TCO (total cost of ownership) model for WSCs. Chapter 9 then discusses energy and sustainability aspects of warehouse-scale computing, including efficiency methods to reduce the data center footprint. We also discuss how WSCs have reduced the energy intensity of computing, and how data center energy consumption compares with end-user consumption. Chapter 10 illustrates the reliability and security improvements made possible by WSC design, and how future advances can further increase the protection of all data stored in the cloud and allow privacy-protected restricted processing of such data.

The last part brings all the pieces together. In Chapter 11, we conclude our discussions of WSC systems design with a retrospective of the past 25 years, using key epochal changes to tease out common foundational design principles that we believe WSC practitioners should know. Chapter 12 highlights 25 top papers from the past 25 years, providing an annotated bibliography for further exploration.

1.6 What is new in the fourth edition

The fourth edition reflects the advances in the field since the prior (third) edition published in June 2018. The content has been significantly revamped and enhanced, with twelve chapters compared to the eight in the prior edition. We added supporting pictures and illustrations and improved existing ones. Numerous other changes make the book more friendly to online reading.

WSC infrastructure designs are significantly influenced by WSC workloads, and thus we have expanded the detailed case studies of websearch, YouTube, Gmail, Google Cloud, and AI workloads. as well as new sections on data processing workloads. Similarly, we have added a new chapter on WSC distributed systems infrastructure, including covering the infrastructure management stack.

We refreshed all the content on WSC hardware, both data center infrastructure as well as server, accelerators, storage, and networking, with case studies of latest generation designs. The data center chapter now includes a discussion of modularity and automation, while the hardware discussion includes new content around multi-node servers, new accelerators beyond ML (e.g., video, networking), new networking designs (spineless, optics), as well as retrospective lessons across past designs. We have also added a new chapter about software-defined infrastructure. We have also deepened the discussion of key metrics—performance, cost, power, energy, and security, including new topics like power oversubscription and the energy usage of AI. The chapter on trusted computing has also been rewritten to discuss both fault tolerance and security and new areas like silent data corruption and new security threat models.

Recognizing twenty five years of warehouse-scale computing, a new chapter provides a retrospective on the journey so far, discussing key WSC epochs and teasing out important lessons – both technical and non-technical – and the opportunities ahead. Finally, a special bonus chapter contains an annotated bibliography of our pick of the top twenty five papers from the past twenty five years (likely the hardest choices we had to make on content for the book!).

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Overview of WSC Architecture

2

2.1 Regions and zones

At the highest level, WSCs are placed in geographic regions with a diameter of about 50 km. Regions consist of multiple campuses, arranged so that no two campuses are more than 50 km apart. This maximum distance isn't chosen arbitrarily: it's large enough to make it unlikely that a single disaster event (such as a flood, fire, or earthquake) or an outage caused by a malfunction or operator error impacts more than one campus, and it's small enough to allow direct optical paths between campuses without needing repeaters along the way, and the resulting links have a round-trip delay low enough to allow synchronous storage replication between campuses.

Regions used by hyperscalers (Google, Meta) for their own applications can have a variety of shapes ranging from a single campus to multiple campuses. Hyperscale services like YouTube and Facebook are global services spread across multiple regions, and thus depend less on intra-region replication for availability or durability. Instead, their reliability model assumes that any region can fail (due to a metro-wide network outage, for example), and consequently data is replicated across regions.

Regions built for public clouds typically have two or three separate campuses. Most enterprise customers prefer single-region architectures because they're simpler, cheaper, and better fit existing application models where a storage device or database uses synchronous replication for availability or durability (i.e., writes must be committed on two separate campuses before the write operation completes). Google and AWS build three zones per region so that distributed consensus algorithms can work cleanly: if there's a network partition or single-campus problem, the system receiving two "healthy" votes becomes or remains the primary system of record, and the remaining copy is prevented from making further mutations. Currently, Azure regions typically have only two campuses, mirroring the common use case of active-active or active-standby pairs for storage. This solution is simpler and may

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_2

provide a faster path to market in new locations, but it complicates automated failovers. It also makes it more expensive for customers to build regionally fault tolerant applications: to tolerate a single-campus failure with $N = 2$ requires $2N$ capacity, whereas $N = 3$ requires only 1.5x capacity (each campus has 0.5N).

Public clouds additionally abstract individual WSC buildings or campuses via the concept of zones. A region has three zones (or two in Azure), and the zone abstracts the exact physical placement of a resource. Exact details differ between providers, but if two VMs are in the same zone, you will get the fastest and cheapest bandwidth between them, but you may see correlated failures: a single downtime event may cause both VMs to become unavailable at the same time. A single zone may have some physical diversity (consist of multiple buildings or even multiple campuses) but those details are hidden, and all of them share a single control plane and thus may experience software-induced failures at the same time (Fig. 2.1).

Even though regions provide a high degree of physical availability (since it's unlikely that three different campuses 50 km apart will be affected by a single disaster), they can't necessarily provide that degree of availability at the API layer because the region's control plane is shared among all zones. For example, if the regional load balancer has a bug, applications in all zones are affected. For that reason, many APIs are available in both zonal and regional variants (the only difference being their fault domains) so that customers can choose the appropriate one. Choosing a regional version is easier but adds a dependency on the regional cloud API, and thus requires more trust in the cloud provider's SLO performance. Choosing the zonal version provides additional in-house control over the regional reliability properties of the application but requires additional in-house effort to replicate regional functionality.

Fig. 2.1 Google Cloud Platform regions as of March 2025

2.2 Buildings and Infrastructure

A campus consists of one or more data center buildings. Sometimes, buildings consist of multiple WSCs that each have largely independent infrastructure, i.e., separate power, cooling, and networking infrastructure, and are separated by multi-hour firewalls that would allow neighboring segments to continue operating even if one segment caught fire. For example, the top building in the aerial view of the Council Bluffs campus in Figure 2.2 consists of three separate WSCs. (They don't look identical because of design changes introduced between the originally-designed WSC on the left, and the subsequent two WSCs.) Figure 2.3 shows the inside of a server hall, and Fig. 2.4 shows part of a cooling hall.

WSC buildings house the computing, network, and storage infrastructure that power warehouse-scale computing, and the design of the building can impact key WSC properties. For example, in Chapter 5, we will discuss how building design decisions (and different tiers of data center construction) can dramatically influence the availability and uptime of the WSC.

WSCs have elaborate power delivery designs. At the scale that they operate, WSCs can often consume more power than mid-size cities. Therefore, they use a holistic power delivery design that feeds electricity from the utility, to the substation, to the power distribution units, to the bus ducts, all the way to the individual power rails and voltage regulators on the server board. The design also provides corresponding backup and redundancy such as uninterruptible power supplies (UPSs), generators, and backup batteries at different levels of the topology. For example, the Council Bluffs campus has two large high-voltage substations on the right, each connected to the region's highest-voltage grid connections and containing multiple high-capacity transformers. Section 5.2 discusses power delivery in more detail.

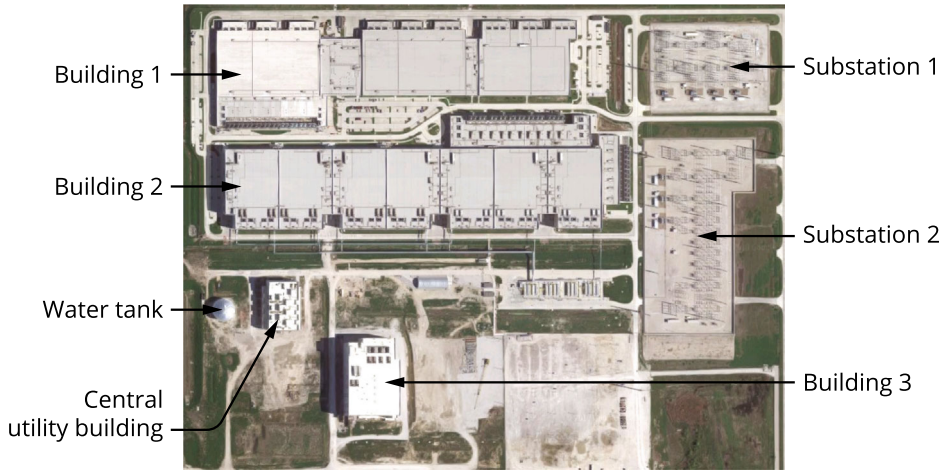


Fig. 2.2 Aerial view of Google data center, Iowa, USA

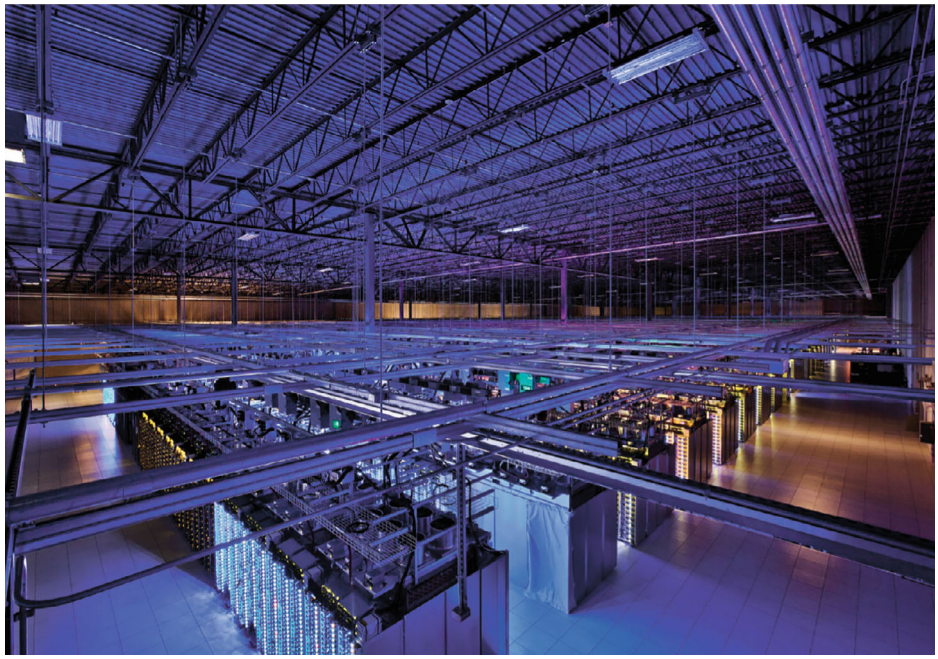


Fig. 2.3 Power and network distribution

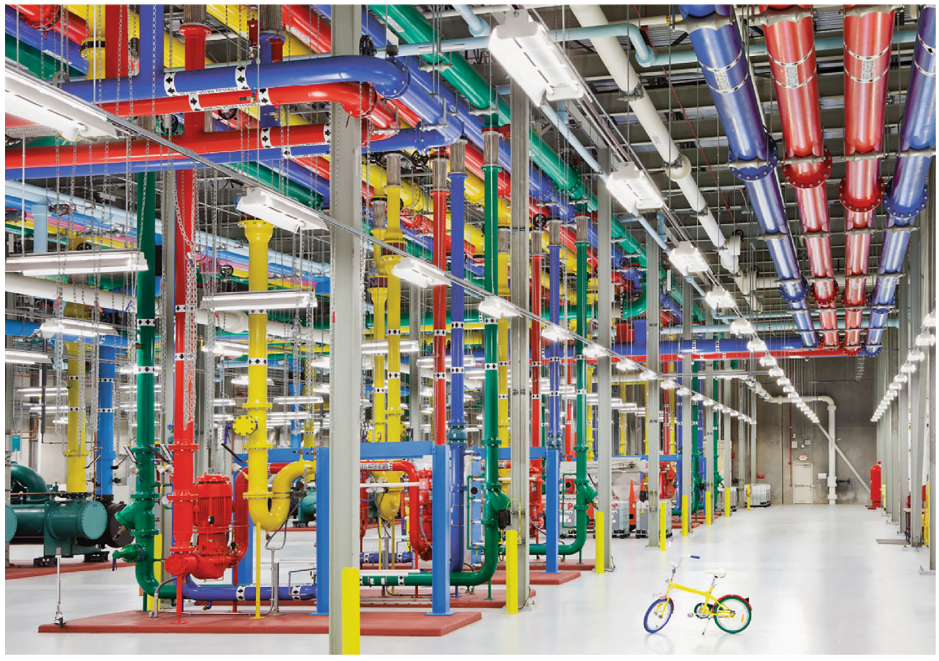


Fig. 2.4 Data center cooling hall

All that power ultimately turns into heat, so WSCs employ elaborate end-to-end cooling solutions with a hierarchy of heat-exchange loops, from circulated air (cooled by fan coils on the data center floor), to heat exchangers and chiller units, all the way to outside cooling units that reject heat into the air via evaporation or radiation (Fig. 2.5). Section 5.3 discusses cooling in more detail.

Building design and server design can't be fully decoupled. For example, the building's power density limits how many servers can be placed within the radius of a single short-reach optical cable, thus influencing the maximum size of a densely connected cluster of servers or accelerators, or the cost of building a larger one that uses more expensive optics. Similarly, how cooling is delivered to the rack can make it easier or harder to implement liquid cooling for accelerators. Thus, it's hard to determine the "best" data center design without understanding servers, and vice versa; sometimes, a little bit of additional cost at the data center level can save more money at the rack or server level. Last but not least, data centers last 20+ years and thus should be "best" not just for the current server generation but also for the servers introduced a decade from now.

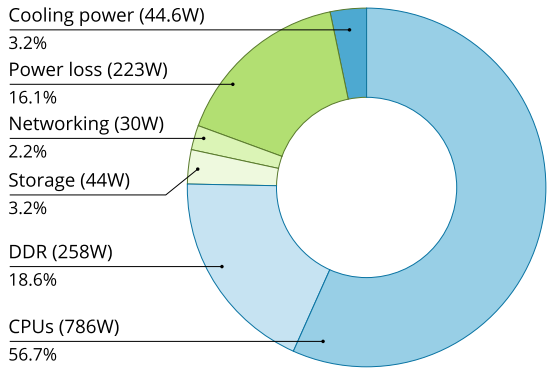


Fig. 2.5 Cooling towers and water storage tanks

2.3 Servers

Most of the energy used in a data center typically goes to servers, the hardware building blocks for WSCs. These commodity servers are mounted into a rack and interconnected using a local "rack top" Ethernet switch. These rack-level switches, which use 100-200Gbps links (as of 2025), in turn have a number of uplink connections to one or more cluster-level (or data center-level) Ethernet switches.

Being satisfied with neither the metric nor the US system, rack designers use "rack units" to measure the height of servers. 1U is 1.75 inches or 44.45 mm, and a typical rack is 42U

high. These standards were established 1922 as internal standards for equipment at AT&T (back then, American Telephone & Telegraph) and have survived until today.

Sometimes a tray or enclosure contains multiple servers that share some components. For example, several low-powered servers might share an expensive high-speed NIC. Thus, often they contain an additional first level of network aggregation within the enclosure where multiple processing nodes connect to a small number of networking nodes through an I/O bus such as PCIe.

Figure 2.6 illustrates Google server building blocks, starting with a regular server tray on the left. WSCs feature additional compute hardware building blocks, including GPUs and custom accelerators; the middle picture shows a TPU board. Similar to servers, these are connected through custom or industry-standard interconnects at the rack (or multi-rack *pod*) levels, leading up to the data center network. On the right is a storage tray with dozens of disks. Figure 2.7 shows how these building blocks are assembled into rows of racks of servers for both general-purpose servers and accelerators.

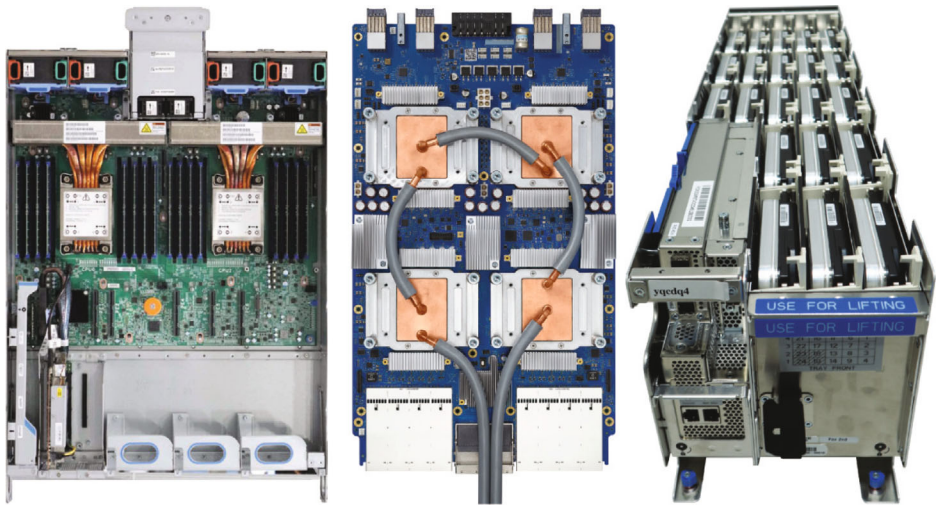


Fig. 2.6 Building blocks for WSCs: server, TPU accelerator, disk tray

2.4 Storage

Disks and Flash (NAND) SSDs are the building blocks of today's WSC storage system. Whereas servers typically use the majority of power in a WSC, storage typically occupies a majority of the space because racks full of disks use a lot less power, and thus a MW of disks occupies much more floor space than a MW of servers. Despite disks being too slow for

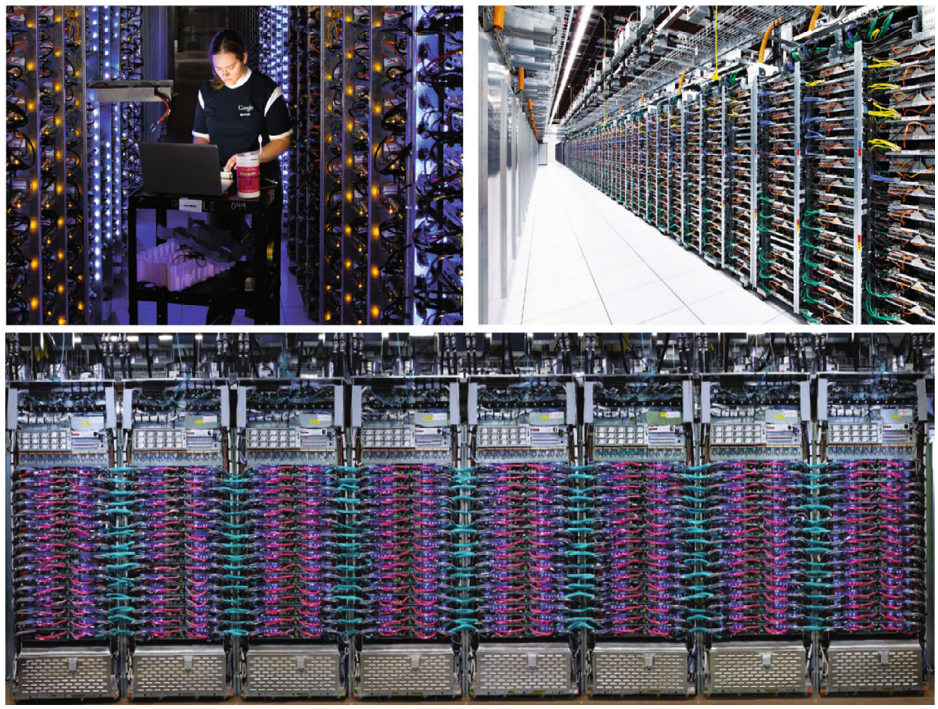


Fig. 2.7 Top: traditional compute servers; bottom: a TPU accelerator pod

many use cases, they still represent the majority of storage devices due to their unmatched cost per byte for cold data.

Storage devices are connected to the data center network and managed by distributed systems. WSC system designers need to make several trade-offs based on their requirements. For example, should disks and SSDs be connected directly to compute servers (Directly Attached Storage, or DAS) or disaggregated as part of Network Attached Storage (NAS)? While DAS can reduce raw hardware costs and improve network utilization (the network port is dynamically shared between compute and storage tasks), the NAS approach tends to simplify deployment and provide higher QoS to avoid performance interference from compute jobs. Additionally, it can improve utilization (that is, actually used space as a fraction of provisioned space) because NAS storage is shared among many systems, and aggregated demands are easier to forecast. Thus, storage devices in WSCs need to focus not just on raw hardware and costs but aim at an aggregated view of the global optima across several key metrics, including bandwidth, IOPS, capacity, tail latency, and TCO.

Distributed storage software not only manages storage devices, but also provides unstructured and structured APIs for application developers. The Google File System (GFS), and later Colossus and its Cloud cousin GCS [1], are examples of unstructured WSC storage

systems that use space-efficient Reed-Solomon codes and fast reconstruction for high availability. Google's BigTable [2], and Amazon's Dynamo [3] are examples of structured WSC storage that provides database-like functionality but with weaker consistency models. To simplify developers' tasks, newer generations of structured storage systems such as Spanner [4] provide a SQL-like interface and strong consistency models.

Over the past two decades, network speeds have improved much faster than disk speeds, and thus it's easy to support network-attached disk storage, dramatically simplifying management without needing to consider data locality. On the other hand, low-latency high-bandwidth storage such as Flash SSDs and emerging Non-Volatile Memories (NVMs) pose a new challenge for WSC networking: just like a single disk could consume an appreciable fraction of a server's network link two decades ago, today a single NVMe SSD can consume 25–50% of a 100 Gbps Ethernet link. Chapter 6 discusses system balance in more detail.

2.5 Networking Fabric

Choosing a networking fabric for WSCs involves a trade-off between speed, scale, and cost. As of 2025, switches offer a total bandwidth of 25-50 Tbps, supporting 64 servers at 400 Gbps per server within a single rack. As a result, bandwidth within a rack of servers tends to be homogeneous and plentiful. However, it would be very expensive to connect all other servers in a WSC at this same speed. No single switch can do that, and thus WSCs use a level of aggregation switches to tie together multiple racks into a connected block of racks, and then another layer to tie together these blocks into larger blocks, and so on. The further away a server is, the slower the average bandwidth to it if all servers communicate at the same time (bisection bandwidth). So while servers on the same rack may be able to communicate at 400 Gbps, the bandwidth to neighboring racks may be only 100 Gbps, and to a server all the way across the WSC it may be only 50 Gbps. We'll discuss such networks in Section 6.4.

Alternatively, one can improve cluster-level networking bottlenecks by spending more money on the interconnect fabric. For example, Infiniband interconnects can scale to a few thousand ports with high bandwidth and low latency, but cost significantly more than Ethernet on a per port basis. How much to spend on networking versus additional servers or storage is an application-specific question that has no single correct answer. However, as a rule of thumb, we will assume that intra-rack connectivity is cheaper than inter-rack connectivity.

2.6 Energy Usage

Energy use is an important concern in the design of WSCs because energy-related costs represent a significant component of the total cost of ownership. Figure 2.8 provides some insight into how energy is used in modern IT equipment by breaking down the peak power usage of a 2024-generation server by component groups. The server is a two-socket x86 server with 768 GB of DRAM and local SSD storage, running at a utilization of 80%.

Although this breakdown will vary significantly depending on how systems are configured for a given workload, CPUs are the dominant energy consumer in WSCs, followed by memory. This ratio has stayed relatively constant over the past two decades. Cooling overheads are relatively small, reflecting generations of improvements in this area. However, power losses have become more significant over time, roughly doubling in the last decade because modern-day CPU currents are very high: a CPU consumes 500 W or more, supplied at a core voltage or about 1V. Power losses include all Voltage Regulator Module (VRM) conversion losses, which increase with the square of the current. (The VRM steps down the 48V tray power feed to the core voltage.) Losses also include metal (or power delivery network, PDN) losses, which also increase with the square of current. In Chapter 9 we discuss WSC energy efficiency in detail.

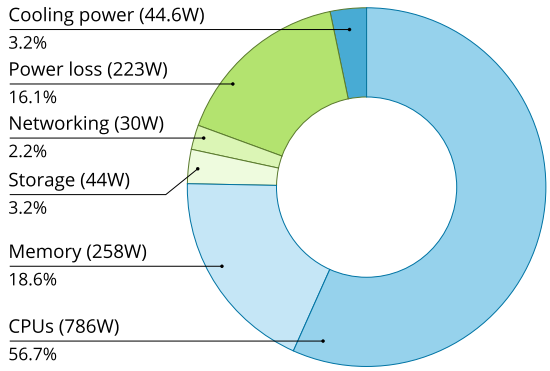


Fig. 2.8 Power usage in a 2024 Google server

2.7 Handling Failures and Repairs

The sheer scale of WSCs requires that internet services software tolerates relatively high component fault rates. Disk drives, for example, can exhibit annualized failure rates as high as 4% [5], [6]. Different deployments have reported between 1.2 and 16 average server-level restarts per year. With such high component failure rates, an application running across

thousands of machines may need to react to failure conditions on an hourly basis. We expand on this topic in Chapter 3, which describes the application domain, and Chapter 10, which deals with fault statistics.

References

1. D. Serenyi, "Cluster-level storage @Google," in *Keynote at the 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Intensive Computing Systems*, 2017.
2. F. Chang, J. Dean, S. Ghemawat, et al., "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 205–218.
3. G. DeCandia, D. Hastorun, M. Jampani, et al., "Dynamo: Amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2007, pp. 205–220.
4. J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Google's globally-distributed database," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12, USA: USENIX Association, Oct. 8, 2012, pp. 251–264.
5. E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, ser. FAST '07, San Jose, CA: USENIX Association, 2007.
6. B. Schroeder and G. A. Gibson, "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?," 3, vol. 3, New York, NY, USA: Association for Computing Machinery, Oct. 1, 2007, 8–es.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Overview of WSC Workloads

3

3.1 WSC workloads

3.1.1 WSC systems stack

A typical WSC deployment includes a multi-layer software stack from high-level workloads and application frameworks, to cluster-level and monitoring software, to development and platforms software. The applications and software that run on warehouse-scale computers (WSCs) dominate many system design trade-off decisions. In this chapter and next, we outline some of the distinguishing characteristics of software that runs in large internet services and the system software and tools needed for a complete computing platform. The different software layers in a typical WSC deployment are:

- **Application-level software**: Software that implements a specific service. It is often useful to further divide application-level software into online services and offline computations, since they tend to have different requirements. Examples of online services are Google Search, Gmail, and Google Maps. Offline computations are typically used in large-scale data analysis or as part of the pipeline that generates the data used in online services, for example, building an index of the web or processing satellite images to create map tiles for the online service.
- **Application frameworks**: Building blocks that make it easier to implement applications: serving infrastructure for front ends or image serving, user session analysis tools, UI widgets for common application elements, etc.
- **Cluster infrastructure**: The collection of distributed systems software that manages resources and provides services at the cluster level. Ultimately, we consider these services as an operating system for a data center. Examples are distributed file systems, schedulers and remote procedure call (RPC) libraries, as well as programming models that simplify

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_3

the usage of resources at the scale of data centers, such as MapReduce [1], Hadoop [2], Sawzall [3], BigTable [4], Dynamo [5], Dremel [6], Spanner [7], and Chubby [8].

- **Resource management**: Infrastructure for managing and allocating resources, including scheduling compute tasks onto specific machines and enforcing storage quotas (for example, Google's Borg scheduler [9], [10]).
- **Platform software**: The common firmware, kernel, operating system distribution, and libraries expected to be present in all individual server platforms to provide a basic machine abstraction layer.
- **Hardware**: servers, accelerators (machine learning, video), storage appliances, network switches, data center infrastructure, etc.
- **Tools**: Software that monitors application performance, identifies system bottlenecks, and measures the hardware health of clusters.

Figure 3.1 summarizes these layers. In this chapter, we will be focusing on the two top layers of the picture. (Chapter 4 will be discussing the other components in this picture.)

Because this book is taking the "inside view" of WSCs, our definition of layers above differs from another commonly-used industry taxonomy (discussed below) which differentiates between infrastructure, platform, and applications [11]:

- **Infrastructure as a Service (IaaS)** provides processing, storage, and networking to allow the consumer to run arbitrary software. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and

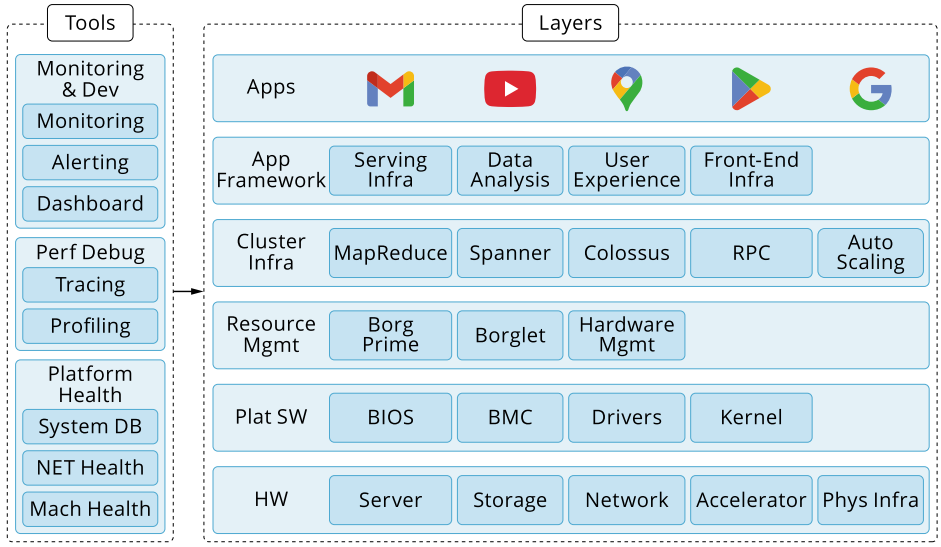


Fig. 3.1 Google software stack for warehouse-scale computers

deployed applications; and possibly limited control of select networking components (e.g., host firewalls). For example, GCP's IaaS product is GCE [12] which provides a variety of virtual machines and block storage options.

- **Platform as a service (PaaS)** provides tools to provision, instantiate, run, and manage applications, relieving developers and operators from having to build their own toolchains. For example, GKE [13] and Cloud Run [14] automate Kubernetes cluster management, Cloud SQL [15] provides fully managed databases, and tools like Terraform [16] allow standardized configuration.
- **Software as a Service (SaaS)** delivers fully managed applications, relieving the customer from most or all operational tasks related to that application (e.g., installing, provisioning, upgrading, etc.). Internet consumer services have always been SaaS services, so the term is primarily used to describe enterprise software. For example, Google Workspace [17] is a SaaS application.

3.1.2 Compute consumption models

Workloads in WSCs can be run in shared (multi-tenant) or dedicated hardware configurations. They can either be specific applications (Gmail, YouTube, search, etc) or broader application frameworks or platforms (GCE, Spanner, BigTable, VertexAI, etc). Additionally, they can be internal, traditional, hyperscale workloads or be external serving cloud customers. Within Cloud, customer workloads can run in infrastructure-as-a-service (IaaS) platforms like Google Compute Engine (GCE) or Google Kubernetes Engine (GKE), or serve higher-level managed services (e.g., VertexAI, DataFlow, DataProc, MemoryStore, CloudSQL, etc). In addition, accelerator platforms like TPU pods (discussed in Chapter 6) can run specific machine learning workloads and frameworks, and disk and flash appliances can run more specialized storage workloads.

Additionally, workloads are broadly classified into various tiers based on their service-level objectives (SLOs corresponding to service availability and latency sensitivity. For example, at Google we currently use these workload tiers:

1. **HALO**: high-availability latency optimized. This tier is intended for workloads that cannot take an outage and do not tolerate even small interruptions (e.g., from shared cores) and, thus, provides the strongest availability SLO. Typical examples are GCE VMs, databases, or application front ends.
2. **HALS**: high-availability latency sensitive. This tier provides the same availability as HALO but can tolerate small (millisecond) hiccups in performance. Examples include data warehouses and streaming data pipelines.

3. **HALT**: high-availability latency tolerant. This tier provides the same availability as HALO but for applications that can tolerate delays of a few minutes. Examples include important batch jobs or a capacity reserve that's rarely needed.
4. **MALS**: mostly-available latency-sensitive and **MALT**: mostly-available latency tolerant. These workloads are cost-optimized variants of the HA tiers whose capacity may be curtailed occasionally to allow oversubscription (see Chapter 9). A typical SLO could be "1% of the time you may lose up to 10% of your resources".
5. **WALS**: weakly-available latency-sensitive and **WALT**: weakly-available latency-tolerant. This tier provides an even more relaxed SLO intended for long-running batch jobs that only need a throughput guarantee ("You will receive at least 50% of requested task hours over a day").
6. **BE**: best-effort. This tier is for work that can be deferred indefinitely, and no resources are provisioned for demand in this tier. If free resources are available, jobs will run, but they may be preempted and resources may not be available for days or weeks so there is no guarantee that a given job will ever run.

Public clouds currently don't offer such tiers, except that most offer BE for VMs as preemptible VMs or Spot VMs. Managing capacity using tiers can provide substantial savings but adds significant complexity and thus is most suitable for cloud providers, not cloud consumers.

3.1.3 Public cloud vs hyperscale

Though initially designed for online data-intensive web workloads like Google Search, Facebook, or YouTube, WSCs also now power public clouds such as those from Amazon, Google, and Microsoft. Today, public cloud has become the largest workload running in WSCs, and the total data center capacity dedicated to public clouds greatly outnumbers the total capacity used by private hyperscaler clouds. In fact, except for Google's and Meta's services, almost all of the most popular consumer applications run on public clouds today.

The term *hyperscale* has become popular to denote large-scale infrastructure ("hyperscale data center"). While the two terms are related, they're not identical. For example, a pure data center provider may build a large data center that's of a scale similar to WSCs, but they only provide the building, excluding the software stack that could turn all the servers in that building into a warehouse-scale computer. However, the term *hyperscaler* is commonly used to describe companies who build WSCs either purely for their own hyperscale applications (Meta) or to provide cloud computing (AWS, Microsoft) or both (Google). Thus, a hyperscaler is a company that operates both large-scale data centers and large-scale services built on top of these.

At first glance, public cloud computing seems very different from hyperscale applications: whereas for the latter, a single application may run on tens of thousands of servers, most

enterprise applications running on public clouds use just a few servers. However, from the perspective of much of the WSC stack, the public cloud looks like one large application that consumes tens of thousands of servers. For example, to Google's infrastructure, GCE (the GCP service providing VMs) looks like a single large application. Google is unique in having both hyperscale consumer applications as well as a large public cloud, and in this chapter we'll illustrate how the same infrastructure can support both large online internal services as well as large public cloud offerings.

3.2 Hyperscale applications

3.2.1 Broad overview

Web search was one of the first large-scale internet services to gain widespread popularity. The amount of web content exploded in the mid-nineties, and organizing this massive amount of information required new infrastructure that didn't exist at the time. Search became the canonical large-scale processing challenge: how do you build a service that scales with the growth of the web (since all web pages need to be indexed) and the growth of internet users?

Fairly quickly, though, many other hyperscale applications emerged: email, social networks, streaming videos, games, and so on. Additionally, individual hyperscale applications may themselves consist of multiple parts; for example, today's web search also has image search, AI summaries, maps, and so on. Consequently, a hyperscale application won't be a single workload but consist of multiple hyperscale subservices. Figure 3.2 presents the cumulative distribution of cycles across workloads in Google data centers. The top 50 binaries account for only about 60% of the total WSC cycles, with a long tail accounting for the

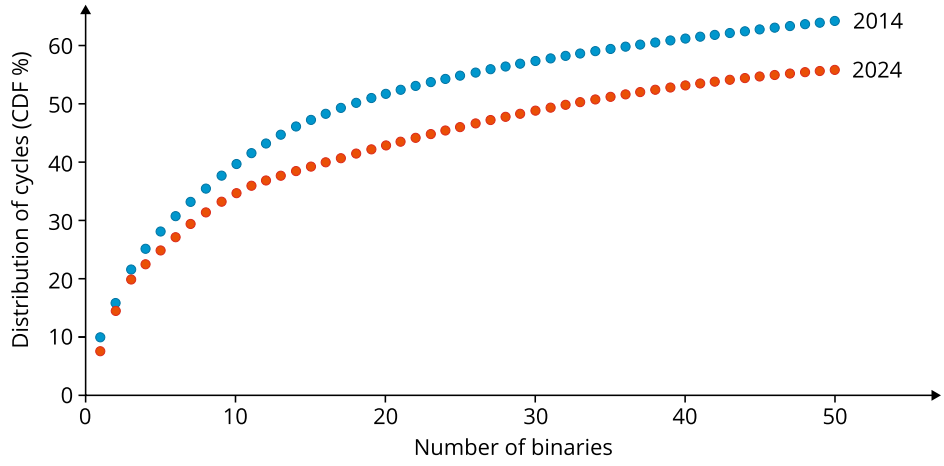


Fig. 3.2 Diversity of workloads in WSCs in 2014 and 2024

rest of the cycles [18]. The two trend lines, separated by nearly a decade, also show how the diversity of services has been steadily increasing.

Given the diverse requirements of multiple services, the data center itself must be a general-purpose computing system. Although specialized hardware solutions might be a good fit for individual parts of services, the breadth of requirements makes it important to focus on general-purpose system design. Another factor against too much hardware specialization is the speed of workload churn; product requirements evolve rapidly, and smart programmers will learn from experience and rewrite the baseline algorithms and data structures much more rapidly than hardware itself can evolve. Therefore, there is substantial risk that by the time a specialized solution is implemented, it is no longer a good fit even for the problem area for which it was designed, unless these areas are ones where there is a significant focus on hardware-software codesign. Having said that, there are cases where specialization has yielded significant gains, and we will discuss these further later in Chapter 6.

In the following sections, we describe workloads used for web search, video serving, machine learning, Gmail, data processing, and Cloud. We do not describe the implementation of these internet service workloads in detail, especially because the dynamic nature of this market will make those obsolete by publishing time. However, it is still useful to describe at a high level a few workloads to illustrate key characteristics of WSC workloads and highlight the distinctions between key categories of online services and batch (offline) processing systems. This discussion will set the stage for future architectural design tradeoffs later.

3.2.2 Case study: Search

Search is the quintessential "needle in a haystack" problem. Although it is hard to accurately determine the size of the web at any point in time, it is safe to say that it consists of trillions of individual documents and that it continues to grow. If we assume the web contains 100 billion documents, with an average document text size of 10 KB (after compression), the haystack is about 1000 TB. (In reality, the index is much bigger; Google publicly states that it covers hundreds of billions of webpages and is well over 100,000 TB in size [19].) The database for web search is an index built from that repository by inverting that set of documents to create a repository in the logical format shown in Figure 3.3.

A lexicon structure associates an ID to every term in the repository. The termID identifies a list of documents in which the term occurs, called a posting list, and some contextual information about it, such as position and various other attributes (for example, whether the term is in the document title).

The size of the resulting inverted index depends on the specific implementation, but it tends to be on the same order of magnitude as the original repository. The typical search query consists of a sequence of terms, and the system's task is to find the documents that

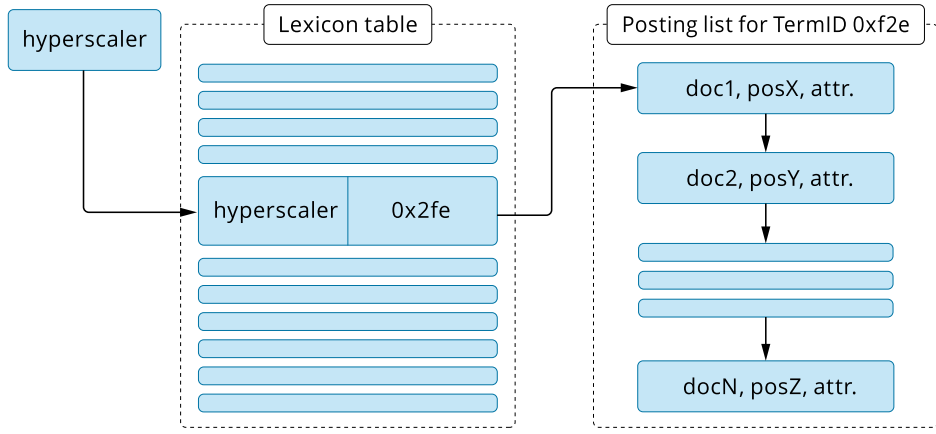


Fig. 3.3 Logical view of a web index

contain all of the terms (an AND query) and decide which of those documents are most likely to satisfy the user. Queries can optionally contain special operators to indicate alternation (OR operators) or to restrict the search to occurrences of the terms in a particular sequence (phrase operators). For brevity, we focus on the more common AND query in the example below.

Consider a query like [palo alto library]. The search algorithm must traverse the posting lists for each term (palo, alto, library) until it finds all documents contained in all three posting lists. Additionally, a language model interprets the query and would discover that, in this case, "palo alto" refers to a place name [20]. The algorithm then ranks the documents found using a variety of parameters, such as the overall importance of the document (in Google's case, that would be the PageRank score [21] as well as other properties such as number of occurrences of the terms in the document, positions, and so on) and returns the highest ranked documents to the user.

Given the massive size of the index, this search algorithm may need to run across a few thousand machines. That is accomplished by splitting (or "sharding") the index into load-balanced subfiles and distributing them across all of the machines. Index partitioning can be done by document or term. The user query is received by a front-end web server and distributed to all machines in the index cluster. As necessary for throughput or fault tolerance, multiple copies of index subfiles can be placed in different machines, in which case only a subset of the machines is involved in a given query. Index-serving machines compute local results, pre-rank them, and send their best results to the front-end system (or some intermediate server), which selects the best results from across the whole cluster. At this point, only the list of doc_IDs corresponding to the resulting web page hits is known. A second phase is needed to compute the actual title, URLs, and a query-specific document snippet that gives the user some context around the search terms. This phase is implemented

by sending the list of doc_IDs to a set of machines containing copies of the documents themselves. Once again, a repository this size needs to be partitioned (sharded) and placed in a large number of servers.

The total user-perceived latency for the operations described above needs to be a fraction of a second; therefore, this architecture places heavy emphasis on latency reduction. However, high throughput is also a key performance metric because a popular service may need to support many thousands of queries per second. The index is updated frequently, but at the granularity of handling a single query, it can be considered a read-only structure. Also, because there is no need for index lookups in different machines to communicate with each other except for the final merge step, the computation is very efficiently parallelized. Finally, further parallelism is available by exploiting the fact that there are no logical interactions across different web search queries.

If the index is sharded by doc_ID, this workload has relatively small networking requirements in terms of average bandwidth. The amount of data exchanged between machines is typically not much larger than the size of the queries themselves (a few hundred bytes or so) but does exhibit some bursty behavior. Basically, the servers at the front-end act as traffic amplifiers as they distribute a single query to a very large number of servers. This creates a burst of traffic not only in the request path but possibly also on the response path as well. Therefore, even if overall network utilization is low, careful management of network flows is needed to minimize congestion.

Finally, because web search is an online service, it suffers from normal traffic variations because users are more active on the web at different times of the day. Figure 3.4 illustrates this effect, showing that traffic at peak usage hours can be five times as high as during off-peak periods. Such variability presents a challenge to system operators because the service must be sized for peak traffic, not average traffic.

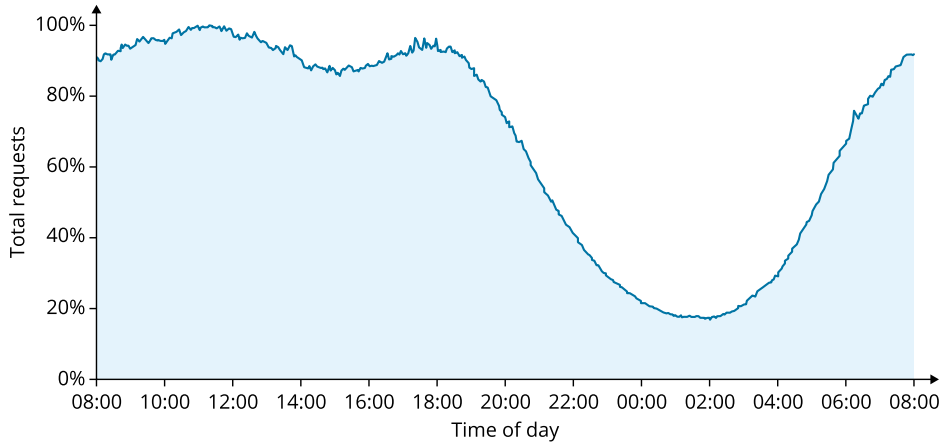


Fig. 3.4 Daily traffic for a search service in one data center

Global load balancing can help with provisioning. Figure 3.4 shows interactive search traffic at a typical cluster in the US. This cluster receives traffic proportional to user demand in North America. The small dip at 14:00 PST (5pm EDT) is fewer searches during the commute hours. The large dip at 2am is most (but not all) of North America going to sleep. During peak times, traffic levels oscillate since the load balancer fine-tunes traffic distribution across different data centers based on latency.

3.2.3 Case study: YouTube

IP video traffic constitutes 75–85% of the global internet traffic. On YouTube, users upload hundreds of hours of video per minute and watch billions of hours of YouTube video every day. Video transcoding (decoding video in one format and encoding it into a different format) is a crucial part of any video sharing infrastructure: video is uploaded in a plethora of combinations of format, codec, resolution, frame rate, color space, and so on. These videos need to be converted to the subset of codecs, resolutions, and formats that client devices can play, and adapted to the available network bandwidth to optimize the user experience.

Video serving has three major cost components: the compute costs due to video transcoding, the storage costs for the video catalog (both originals and transcoded versions), and the network egress costs for sending transcoded videos to end users. Improvements in video compression codecs improve the storage and egress costs at the expense of higher compute costs. The YouTube video processing pipeline balances the three factors based on the popularity profile of videos, only investing additional effort on compressing highly popular videos. Below, we describe how video on demand works; other video serving use cases, like on-the-fly transcoding or live video streaming, are similar at a high-level but differ in the objective functions being optimized, leading to different architectures.

Every video uploaded to YouTube is first transcoded [22] from its original upload format to a temporary high-quality common intermediate format, to enable uniform processing in the rest of the pipeline (Figure 3.5). Then, the video is chunked into segments and transcoded into multiple output resolutions and codecs, so that when a user requests the video, the available network bandwidth and the capabilities of the player device are matched with the best version of the video chunk to be streamed. Video chunks are distributed across multiple machines, parallelizing transcoding of each video segment into multiple formats, and optimizing for both throughput and latency. Finally, if a video is identified as highly popular, it will undergo a second video transcoding pass, where additional compute effort is invested to generate a smaller video at the same perceptual quality. This enables users to get a higher resolution version of the video at the same network bandwidth, and the higher compute costs are amortized by egress savings on many playbacks.

Once video chunks are transcoded to their playback ready formats in the data center, they are distributed through the Edge network, which caches the most recently watched videos

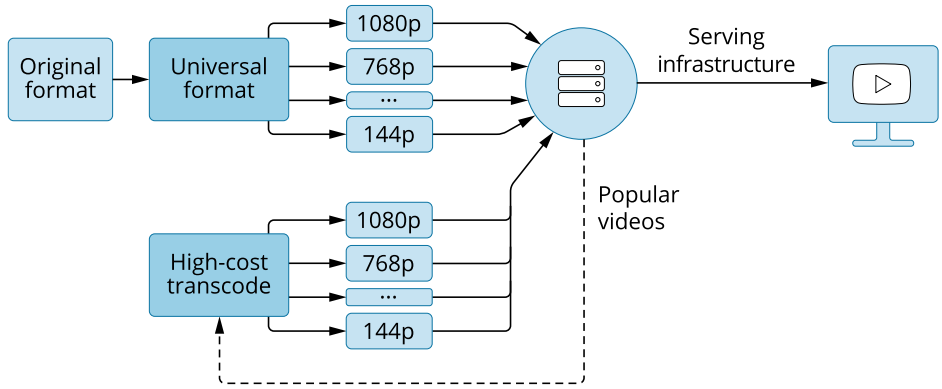


Fig. 3.5 The YouTube video processing pipeline

to minimize latency and amplify egress bandwidth. When a user requests a video that is not readily available in the Edge caches, the request is forwarded to the nearest data center or a peer Edge location, and the requested video is uploaded from the catalog.

3.2.4 Case study: Gmail

Gmail is a large email service: with billions of active accounts, the number of emails in Gmail is in the same order of magnitude as the number of web pages. Gmail's features include the obvious (sending and receiving emails) but also many other features such as protection from spam, phishing, and malware, automatically triaging and categorizing email, fast search over emails, and AI tools helping users write and respond to emails. Gmail cannot serve data at this scale with latency and reliability from a single data center; consequently, Gmail is a globally distributed system.

To enable parallel and independent development of features, Gmail has many components, each owned by a separate team. Collectively, these components implement Gmail's many features. These components broadly fall in two categories: (i) the application components that implement business logic and have no durable state; and (ii) the storage components that are responsible for durably and consistently storing the data. Gmail uses two primary storage systems: Spanner stores emails, indices, and metadata; and Blobstore (an API similar to GCS or S3) stores attachments (images, videos, etc.). Gmail needs multiple storage systems because of varying needs for different kinds of data: Spanner stores data that is mutable on a fine granularity (e.g., changing labels on emails) while Blobstore stores "bags of bytes" which Gmail reads and writes as a unit. To ensure that a bad Spanner event (e.g., corruption of a root table) does not cause widespread unavailability, Gmail partitions its users across 100+ Spanner databases and no critical operation can modify more than one database. All data, regardless of storage system, is encrypted at rest and in flight.

Most Gmail requests involve multiple roundtrips between the application and storage components. For example, when a user opens an email, Gmail first reads signals to determine if the user's account is compromised, then it reads the contents of the email including attachments, then it updates the storage to mark that email as having been read, and finally updates the indices so users can readily search for messages that they have already read (or not read). Consequently, every user request transitively results in hundreds of RPCs (remote procedure calls) and thus every request has to contend with long-tail behavior: with 100s of RPCs per user request, more than half of the user requests would be affected by a 1 in a 100 failure event if we assume uniform failure probability. Multiplied with 1.5B+ accounts, even very low-probability problems become commonplace. Thus, handling events such that they do not cause high latency or unavailability to users is central to Gmail's design.

To minimize the latency overhead of the RPCs that Gmail issues to satisfy user requests, Gmail colocates the application and storage systems: a component in a data center primarily communicates with other components in the same data center. Besides benefiting latency (e.g., RPCs are all within the same data center and we can use in-data center caches), this also benefits reliability: for example, a bad push of a component in a cell only affects traffic in that cell and thus bounds the blast radius of a problem.

To ensure high availability, Gmail uses Spanner to consistently replicate each user's data in three data centers. Gmail users expect consistent reads: for example, after applying a label to a message, a user expects that any subsequent search for messages with that label must include that message. Spanner supports consistent reads but they can be expensive unless they go to the Paxos leader [23] which always has the most up-to-date state for the user. To avoid this latency, Gmail and Spanner coordinate to pick the Spanner leader for a user's account: all of a user's requests route to the data center serving this leader and thus this data center is "primary" for the user. If this data center becomes unavailable, Gmail initiates a protocol that picks one of the other data centers with a replica of the user's data as the primary for the user.

Gmail also uses Blobstore to replicate a user's attachments in multiple locations. Since attachments are read-only and do not support partial mutations, attachment replication uses a simpler (and cheaper) background replication mechanism in contrast to Spanner's Paxos implementation.

To enable high availability for users even when their ISP is unreliable, Gmail clients synchronize recent emails and log unapplied mutations (e.g., sending an email) on the user's device. When the client connects to the server, Gmail transactionally applies these mutations. This also reduces client perceived latency since it moves some transactions off the critical path.

To minimize latency for Gmail users, a naive strategy would be to place a user's data in data centers that are physically close to the user. Unfortunately, this strategy turns out to be a poor choice for cost (to Google), provisioning complexity (to Google), and latency. It is a poor choice for cost for two reasons. First, during working hours in one time zone, the local data centers will be busy while data centers that are a few time zones away will be

idle but cannot be used; thus it leads to poor resource utilization. Second, each geographical region will need enough local capacity to handle data center outages in that region, further lowering utilization. It is a poor choice for complexity because every data center will have a different utilization, provisioning, and latency profile depending on the user population near the data center; consequently, Gmail will need a possibly different provisioning strategy for each data center. It is a poor choice for latency because Gmail will serve users from busy local data centers which will have poor long-tail latency.

Thus, rather than employing the naive strategy, Gmail uses consistent hashing to spread the replicas of a user's data across data centers globally, or across particular regions in case of data location requirements. This strategy ensures uniform utilization across data centers and better long-tail latency since user requests are not limited to busy local data centers. Gmail's placement strategy simplifies provisioning by ensuring that all data centers have similar utilization and user mix. Specifically, Gmail can use a simple mapping from accounts to resources: each account needs (on average) a certain amount of CPU, memory, IOPS, and storage regardless of the location of the account or the user. Thus Gmail can use the same provisioning strategy in all data centers. Additionally, the clean separation of storage and processing allows both pools to be scaled independently, further simplifying provisioning and increasing utilization.

In summary, Gmail is a distributed system optimized for availability (via replication in geographically diverse locations), latency (e.g., by avoiding hot-spots in data centers or by using different storage systems optimized for each kind of data), and cost (e.g., via its use of consistent hashing).

3.3 AI workloads

So far, we have discussed specific services like search, Gmail, etc. Next, we will discuss the category of AI/ML (artificial intelligence/machine learning) workloads.

Deep neural networks (DNNs) and AI workloads have become a cornerstone of modern data center applications, driving advancements in various fields such as natural language processing ("AI bots"), speech recognition and synthesis, computer vision, protein folding, or weather prediction. These workloads are characterized by their intensive computational and data demands, requiring specialized infrastructure to handle training and inference at scale.

DNNs are a class of machine learning algorithms inspired by the structure and function of the human brain. They consist of multiple layers of interconnected neurons, where each neuron processes input data and passes the result to the next layer (Figure 3.6). The depth of a neural network refers to the number of layers it contains, with "deep" typically indicating a network with several hidden layers – large data sets in the cloud enable more accurate models, by using larger and extra layers to capture more complex patterns and higher-levels of abstractions.

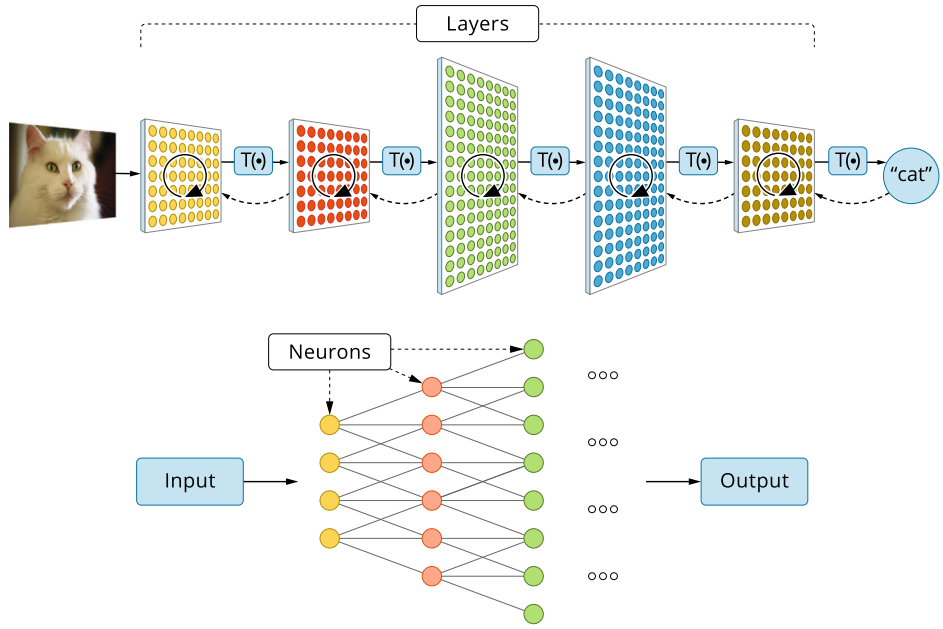


Fig. 3.6 A Deep Neural Network (DNN)

DNNs encompass a variety of architectures tailored to specific data types and tasks. Multi-layer Perceptrons (MLPs) are early models used for classification and regression. Convolutional Neural Networks (CNNs) excel at image processing, while Recurrent Neural Networks (RNNs) handle sequential data like text. Transformers [24] have revolutionized natural language processing with their attention mechanism, powering all Large Language Models (LLMs), which are scaled-up Transformer architectures trained on massive datasets. They excel at understanding and generating human-like text, and through integration with diffusion models, their capabilities extend to other modalities like image, video, and audio generation. Recommendation models leverage user-item interactions to predict preferences, crucial for personalized advertising and ranking. Each of these classes of DNNs, entails unique computational characteristics, necessitating careful design of the compute infrastructure.

Developing and deploying DNNs involves distinct phases, each with unique computational demands. (Pre-)Training, involves learning model weights from (labeled) data. This is followed (optionally) by additional fine-tuning and distillation on task-specific data. Finally, inference, or serving the model, makes predictions on new data. (More recently, with thinking models, we also have additional inference-time computing as well.)

3.3.1 Training

Training, the foundation of DNN development, involves iteratively adjusting model parameters using labeled data, often at massive scale with thousands of specialized accelerators (TPUs, GPUs) over periods of several weeks. This computationally intensive process leverages distributed learning techniques, where multiple learners process data subsets and reconcile parameters through centralized servers or collective reduction. DNN training algorithms typically operate in a highly synchronous manner, necessitating frequent communication between worker nodes to ensure consistent parameter updates and model convergence. This synchronous nature requires careful optimization to avoid communication bottlenecks, as even a single laggard can stall the entire training process. Additionally, it demands robust fault tolerance mechanisms at scale, including periodic model checkpointing, rapid recovery without performance degradation, and programming abstractions to seamlessly handle failures. The synchronicity of such large workloads also needs new design optimizations to deal with power and thermal spikes. Finally, efficient data processing pipelines are crucial for handling large-scale datasets, pre-processing them in a performant manner, and feeding data to the specialized accelerators effectively.

3.3.2 Inference

Inference (or serving), the application of the trained model, makes predictions on new data, often with strict latency requirements for user-facing applications. Serving at scale requires deployability across diverse hardware and software environments, ensuring scalability to handle millions of queries per second, and maintaining high availability and fault tolerance. Efficient resource utilization and load balancing are critical to meet performance and cost-effectiveness targets. Figure 3.7 shows the major workloads during model development. Pretraining ingests vast amount of training data to determine the base model weights. Supervised fine tuning (SFT) and reinforcement learning with human feedback (RLHF) tune the model for a specific use case (like a chatbot), and distillation reduces its size to optimize performance.

3.3.3 Large Language Models

Large Language Models (LLMs) introduce additional stages to the workflow: pre-training on large, general-purpose datasets for extended periods demands vast computational resources. Fine-tuning for task-specific adaptation uses smaller datasets and less computation. Optional distillation creates smaller, resource-efficient models for serving. LLM pre-training requires tens to hundreds of thousands of chips, sometimes spanning multiple data centers. LLM fine-tuning similarly requires significant computational power, though it can be conducted over

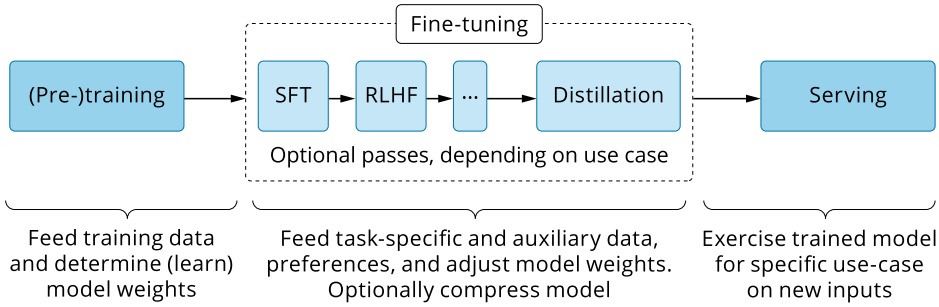


Fig. 3.7 Development and deployment phases of a DNN model

a more localized cluster of accelerators. LLM serving presents its own unique challenges: the large memory footprint of an LLM must be distributed onto multiple host CPUs and accelerators (TPUs, GPUs), and inference involves distinct phases that range from highly compute-intensive to highly memory-bandwidth intensive. Additionally, serving LLMs at scale demands efficient context and state management and low-latency responses, while also being cost effective. Fortunately, inference efficiency has progressed extremely well over the past few years, resulting in improvements (at the same or better quality) of more than two orders of magnitude.

3.3.4 AI computation characteristics

Large-scale linear algebra operations, particularly matrix multiplications, dominate DNN workloads. Figure 3.8 plots the FLOPS needed to train state-of-the-art models over time, starting from early DNNs (ResNet) to recent LLMs (Google Gemini [25], OpenAI GPT-4 [26], [27], Meta Llama [28], Anthropic Claude [29]). This fundamental attribute has driven the development of specialized hardware accelerators such as TPUs and GPUs, optimized to efficiently execute these operations. Unlike traditional and HPC workloads that utilize high bit-width numerics (fp64, fp32), DNN models can employ lower-precision numerics (bf16, fp8, fp4, ocp-mx). These lower-precision formats reduce memory footprint and enable more efficient provisioning of FLOPS, enhancing both area and energy efficiency on accelerators. Scalability is critical as models continue to increase in size and train on more data, necessitating systems with high-bandwidth interconnects to scale out effectively. Additionally, power and carbon footprint are increasingly important. Successive accelerator generations (TPUs, GPUs) focus on operational energy efficiency, complemented by techniques like intelligent scheduling to utilize green energy in data centers and optimizing power delivery (e.g., oversubscription) to capitalize on the power characteristics of large-scale DNN workloads.

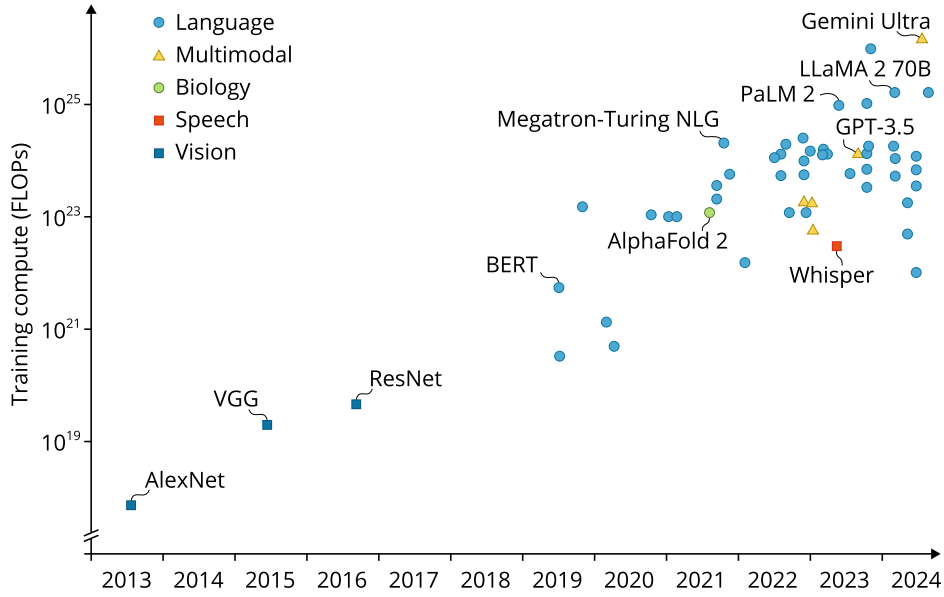


Fig. 3.8 Compute demand of ML workloads over the past decade

Complementing these hardware advances, dedicated software abstractions empower researchers to innovate at a rapid pace. High-performance compilers and runtimes translate algorithms into efficient code for domain-specific hardware. Fusion techniques ensure effective utilization of memory bandwidth by combining multiple operations into a single kernel, reducing memory access overhead. Efficient prefetching and overlapping compute with communication operations help to minimize idle times and maximize throughput. As models grow in size, they need to be sharded across multiple chips, requiring various parallelization strategies: data parallelism (distributing different data batches to different chips), model parallelism (splitting model weights across multiple chips), and pipeline parallelism (dividing model layers into stages across different chips).

Software frameworks like TensorFlow, PyTorch, and JAX provide high-level abstractions, enabling researchers to express complex models with ease, while incorporating features like auto-differentiation. Cross-stack optimizations, such as exploiting reduced-precision numerics (e.g., bf16, fp8, int8, int4) and employing auto-ML techniques, enable AI workloads to be tailored to specific hardware platforms and application requirements, maximizing efficiency and performance.

In summary, AI workloads, particularly DNNs and LLMs, are engineered for massive computational scale (e.g., through distributed training on specialized accelerators like TPUs/GPUs) and efficient, low-latency inference (e.g., via model/software optimizations and diverse hardware deployment). However, AI workloads are evolving rapidly, and this discussion should be viewed as a snapshot at the time of writing of a rapidly advancing field.

3.4 Case study: Google Compute Engine

Google Compute Engine (GCE) lets customers securely and efficiently run their workloads on Google's infrastructure. GCE's infrastructure supports three kinds of compute instances: performance-optimized virtual machines (VMs) where virtualized resources map directly to the underlying hardware, cost-optimized VMs where customers can use custom machine shapes that are dynamically mapped to the underlying hardware based on their resource needs, and bare metal instances where customers get direct access to the underlying compute hardware.

Figure 3.9 shows an overview of the GCE architecture. The host's hardware includes specialized hardware accelerators for security (Titan) and networking and storage virtualization (Titanium), so that even a bare metal instance does not have direct hardware access to them. The on-host I/O accelerators don't perform all I/O functions themselves but offload some functionality to network "hoverboards" (specialized high-performance packet processors) and storage agents. Unlike individual services like mail or search, GCE represents a platform to run other workloads. Below we discuss three key aspects of this platform relevant to WSC design.

3.4.1 Verified boot

Google's custom-designed Titan [30] security microcontroller provides a hardware root of trust to cryptographically validate all firmware and software [31] on the host. The hardware root of trust cryptographically validates the host's boot firmware. Then the boot firmware validates the system software including the kernel and KVM-based hypervisor that GCE uses to support virtual machine instances. Finally, the system software uses binary authorization

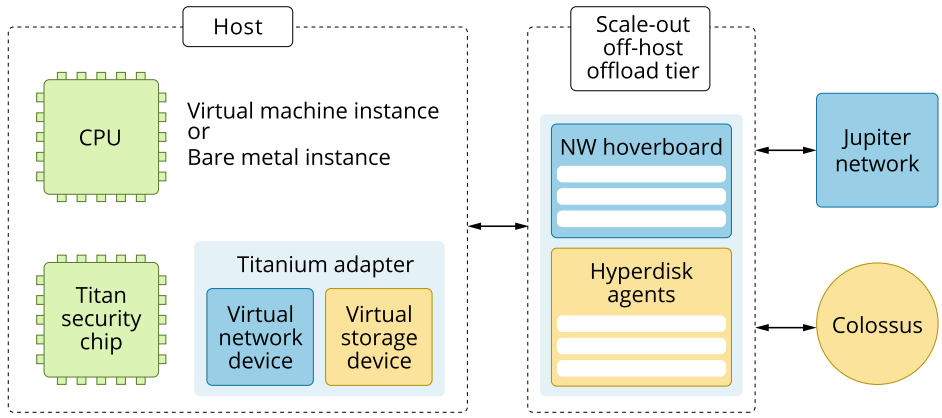


Fig. 3.9 GCE node with tiered offloads

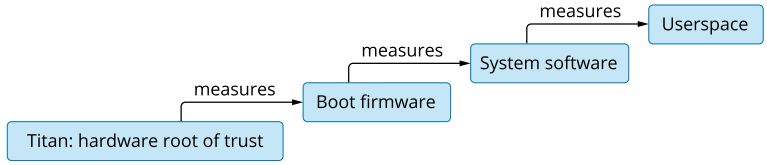


Fig. 3.10 Verified boot sequence

to validate userspace software like system daemons. If a host's firmware or software deviates from the intended state, it cannot access the credentials that it needs to operate within the fleet. Such machines cannot participate in running customer workloads or other Google services until machine management infrastructure triggers automated repair actions.

A verified boot architecture provides two important benefits. First, it serves as the foundation for a layered security architecture where higher layers can rely on the proper state (code and configuration) of lower layers (Figure 3.10). Without that, even a strongly secure operating system could be defeated by tampering with the firmware it depends on. Second, it provides a way to rapidly recover from a security breach: if we suspect that, somehow, an attacker is present on the host or guest, then a reboot is guaranteed to remove that attacker and restore the default state. The same functionality can be used to clean a machine before assigning it to a new customer, even if the previous customer had bare metal access.

3.4.2 Tiered offloads

Historically, virtual machines relied on host software to convert a physical network interface card (NIC) with a physical IP address on a physical network into a virtual NIC with a virtual IP on a virtual private LAN by encapsulating outgoing packets and decapsulating incoming packets. (For historical reasons, such a virtual LAN is often called VPC, for Virtual Private Cloud.) Similarly, virtualizing physical I/O devices with PCIe or NVMe interfaces required host support.

Today, GCE supports customer workloads running in virtual machine or bare metal instances through Titanium [32], [33], an offload system that complements on-host offloads with scale-out offloads that run outside the host. This architecture improves security, performance, and flexibility over software-based approaches. Other clouds use similar architectures but without tiers; for example, Amazon's Nitro accelerator for Elastic Block Store (EBS) contains all of the block storage functionality in a single card. This architecture has the benefit of simplicity but makes it harder to provide flexible I/O capacity where a host or VM's I/O capacity can be changed without a hardware upgrade.

GCE's on-host Titanium Adaptor is an Infrastructure Processing Unit (IPU) (we will discuss the IPU in detail in Chapter 6) to minimize the work done by the host CPUs. It provides hardware acceleration for virtualizing I/O. For networking, it accelerates encap-

sulating/decapsulating network packets and encrypting network traffic with the scale- and offload-optimized PSP security protocol [34]. For storage, it maps virtual NVMe block devices to high-performance, flexible, scalable, and durable Hyperdisk volumes.

Beyond on-host offloads, other (scale-out, off-host) offloads let GCE scale VM services beyond what is practical to deliver on each host. Instead of being limited to the storage virtualization performance of the accelerator card that fits on a host, Hyperdisk block storage [35] performs that work in two places. Basic processing happens on the local Titanium Adaptor, but actual persistent storage processing happens in a scale-out offload tier on a separate host which distributes I/O across Google's cluster-level filesystem, Colossus. With traditional offload architectures, higher block storage IOPS requires VMs with more vCPUs (or bigger accelerator cards) whether the workload needs them or not. This tight coupling results in wasted resources and higher costs. Hyperdisk decouples compute instance size from storage performance since GCE can independently size the instance itself and the storage offload task. Thus, GCE can support high storage performance even with relatively small, general-purpose VMs.

As another example of scale-out offloads, GCE uses "hoverboards" [36] to scale network virtualization. Hoverboards are standalone software switches that act as default routers for some flows. With Titanium, Google's Andromeda virtual networking stack on the Titanium Adapter (again, more about Andromeda in Chapters 6 and 7) sends all packets for which it does not have a route to Hoverboard gateways, which have forwarding information for all virtual networks.

3.4.3 Provisioning workflows

GCE's control plane lets users create and configure virtual clusters and lets Google run them reliably and efficiently (Fig. 3.11). For example, a customer can use the GCE API to store a representation of the configuration they want for a virtual machine, specifying the number of CPUs, amount of memory, number and size of Hyperdisk volumes, network firewall rules, and so on.

After storing updates to a VM's intended state in Spanner, the GCE API server communicates with several specialized control planes to carry out the user's intent. First, GCE calls Borg (discussed more in Chapter 4) to efficiently bin-pack the VM instance onto a host while meeting other constraints like spreading instances from the same project across hosts and avoiding hosts with pending maintenance events. Borg then starts a VM Instance Controller on the target host.

Once the VM Instance Controller is running on the right host, GCE sends it a VM specification that includes both basic information like the VM size and information to configure virtual devices including Hyperdisk volumes. The Hyperdisk client then configures a set of scale-out Hyperdisk agents to serve the VM's volumes, and they open the relevant Colossus files.

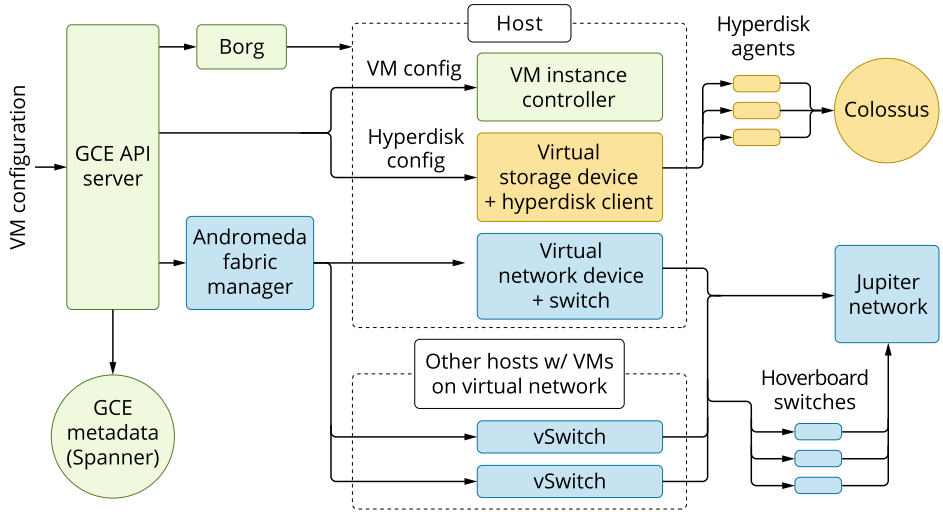


Fig. 3.11 GCE provisioning flow

GCE then calls the Andromeda [36] Fabric Manager, which configures the virtual switch on the new VM's host's Titanium Controller to set up routes to communicate with other virtual machines on the virtual network. The Fabric Manager may also proactively program other virtual switches on other hosts so that other VMs on the virtual network can send packets directly to the new VM. Alternatively, it may omit this step, in which case packets sent to the new VM will be routed via the Hoverboard switches.

3.4.4 Live migration

Live migration (migrating a VM from one physical host to another without shutting down the VM [37]) illustrates how many of these pieces fit together. During a planned maintenance event on a virtual machine (VM) instance's underlying hardware, Compute Engine might use live migration to move the running VM to another host. After selecting a VM for migration, GCE notifies the guest that a migration is happening soon. After a delay, the migration process begins with GCE using Borg to reserve capacity on a destination host.

GCE then begins the pre-copy phase of migration, copying memory pages from the source host to the destination host while tracking any pages that are written by the still-running VM. GCE also flushes dirty Hyperdisk blocks to storage while continuing to buffer and write any additional blocks that are updated during pre-copy.

The next phase of migration is a short blackout phase when the VM on the source host is paused, its VCPU state is transferred to the destination host, any remaining dirty Hyperdisk blocks are flushed, a list of any remaining dirty memory pages is sent to the destination

host, and Andromeda networking begins programming Hoverboards and other VMs' virtual switches to direct the VM's packets to the destination host. For most VMs, this blackout period can be kept to a fraction of a second.

Finally, in the post-copy phase, GCE resumes the VM. Once the VM is resumed, GCE asynchronously copies any remaining dirty memory pages from the source host to the destination host; if the VM accesses a dirty page before it is copied to the destination, the access causes a page fault and the destination host fetches the page from the source host to allow the VM to continue with minimal interruption. Additionally, while the distributed Andromeda system is updating virtual switches to reference the VM's new location, the source host forwards any of the VM's packets it receives to the destination host.

In summary, Google Compute Engine (GCE) is a platform optimized for secure workload execution (via its verified boot process with Titan hardware) and flexible, high-performance I/O (through tiered offloads including Titanium IPUs and specialized scale-out systems for storage and networking). Efficient VM provisioning workflows and non-disruptive live migration for customer workloads optimize efficiency.

3.5 Case study: Data processing infrastructure

The final set of WSC workloads we will discuss is around data processing, including database and data analytics workloads. Globally, data analytics is a $100B+ industry, with many exabytes of structured and unstructured data being generated and analyzed daily. Traditionally, data analytics has focused on structured data, but AI is increasingly expanding its focus towards a hybrid world where structured and unstructured data can be seamlessly integrated [38]. For these hybrid use cases, disaggregated systems are tremendously useful for decoupling rapidly changing requirements for storage (from tabular to multi-modal) and compute (from traditional CPUs to accelerators). Two products, BigQuery and Dataflow, illustrate the benefits of disaggregation provided by WSC.

3.5.1 BigQuery

BigQuery is a data warehouse that can operate at data center scale, dynamically provisioning compute capacity for incoming queries from a shared pool of workers. BigQuery's precursor Dremel [6] initially served as a logs processing tool, providing a simplified and streamlined query interface across log files stored in a disaggregated storage system (Colossus and its predecessor, GFS). Many of these use cases required iterative, ad hoc querying of data. Thus, a crucial design goal for Dremel was to separate compute and storage to avoid imbalances inherent in pre-planned compute/storage ratios (e.g. bottlenecking large queries on limited compute and/or requiring data movement to access infrequently queried data).

More than a decade later, this separation of compute and storage is indispensable. When a data warehouse tightly couples compute and storage, the resulting system has many drawbacks. Inevitably, optimizing the shape of a given warehouse requires careful tuning to avoid wasteful overprovisioning of compute or storage. Even if such tuning was possible, it assumes that the warehouse has a relatively uniform steady-state load, or otherwise requires complex autoscaling logic to avoid idling resources. Autoscaling is inherently more complex for systems that couple compute and storage. For example, simply adding more compute to the system may not immediately translate to additional query bandwidth unless the corresponding storage is also dynamically replicated.

BigQuery's core system design is elegant in its simplicity (Fig. 3.12). Workers read from disaggregated storage (Colossus [39]), using a specialized columnar file format (Capacitor [40]) for efficiency and performance. Workers can also read from common OSS file formats. Past this initial read stage, the system performs aggregations and joins across workers using a distributed shuffle layer that can tier storage across memory, SSD, and HDD. Asynchronously, the system performs updates to the storage layer (compactions, key rotations, repartitioning, even cross-data center moves) using a separate pool of compute resources, transparently optimizing storage performance without impacting the live query path.

Disaggregation is crucial at every one of the above stages: by dynamically assigning workers, BigQuery can optimize capacity usage and minimize idle resources. This allows large and/or high-criticality workloads to temporarily burst usage [41], without requiring steady-state overprovisioning. When reading data, the disaggregation of compute and storage allows the system to maximize parallelism according to the needs of a given job, rather than being bottlenecked on a predetermined number of available data copies.

Disaggregation is even more crucial during shuffle operations. During a shuffle, worker nodes redistribute data such that all data belonging to one key is located on the same worker node. In traditional shuffles, skewed data distributions can create performance bottlenecks, but BigQuery's in-memory shuffle tier allows for dynamic repartitioning of data [42], eliminating such issues. Further, this shuffle tier decouples the producer stages of the processing graph from the consumer stages, thereby providing fault tolerant data processing.

As data volumes continue to increase, disaggregation is important even for the metadata level. Once a data warehouse reaches a certain size, its metadata no longer fits on a single shard. Some systems deal with this by splitting metadata into multiple shards, but if these shards themselves couple compute and storage (i.e. to query a given piece of metadata, you must use the compute that is resident on a given storage shard), they create the same drawbacks described above. Instead, BigQuery approaches metadata the same way it approaches data. By using similar mechanisms [43] for metadata that separate compute and storage, BigQuery accommodates very large metadata sizes without the corresponding drawbacks of sharding schemes (Fig. 3.12).

BigQuery began a decade ago as a data warehouse, primarily designed to store structured data in SQL tables. Data is cleaned and processed into a predefined schema before being stored. In recent years, it has evolved into a data "lakehouse" (data lake + data warehouse)

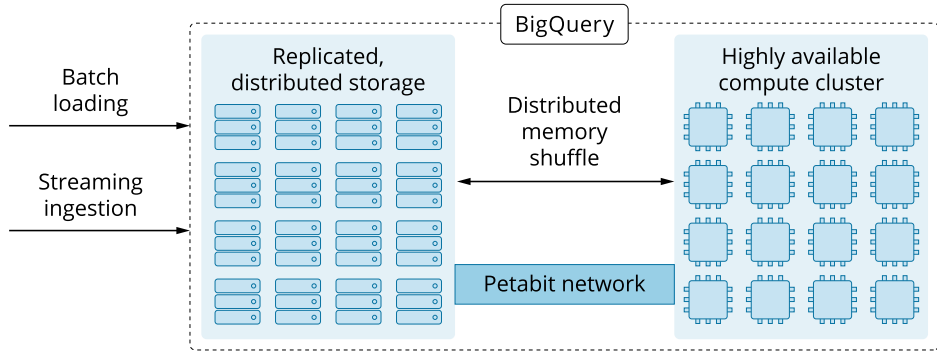


Fig. 3.12 BigQuery architecture

which stores all types of data, including structured, semi-structured (like JSON or XML), and unstructured data like images, audio, and video files [44]. Additionally, extensions to the query language now allow users to process data with a combination of SQL, AI model evaluation, and external systems like Flink or Spark, and to process data stored on multiple clouds. While this evolution significantly impacts many aspects of the system, the core processing architecture has remained.

3.5.2 Cloud Dataflow

Dataflow [45] is a data processing engine that can perform both streaming and batch processing over heterogeneous inputs at petabyte scale (Fig. 3.13). Rather than use a dedicated file format, Dataflow can be customized to read arbitrary sources of data, residing in any cloud-addressable location. Similarly to BigQuery, disaggregated data sources are crucial to realizing parallelism in Dataflow. Once data enters the system, batch data can be shuffled across stages using a similar mechanism to BigQuery (tiered across memory and disk).

When designing Dataflow and its internal precursor (Flume [46]), a core goal was to decouple the physical and logical data processing plan (conceptually, another form of disaggregation). Prior to these systems, users would chain together complex topologies of MapReduces, explicitly defining a physical plan (e.g. run job A after job B), while implicitly defining a logical plan (e.g. join datasets A and B and then perform an aggregation). Instead of inputting a physical plan, users provide a logical plan to Dataflow, which then performs various optimizations (e.g. skipping unnecessary dataset materializations) that compile to the physical plan that the system actually executes.

The above optimization logic can apply to both batch and streaming workloads. For streaming workloads specifically, Dataflow automatically partitions incoming streams according to dynamically determined key ranges. Like batch shuffle operations, these ranges may need to be rebalanced [47] to avoid bottlenecks (and in the case of streaming, stalling

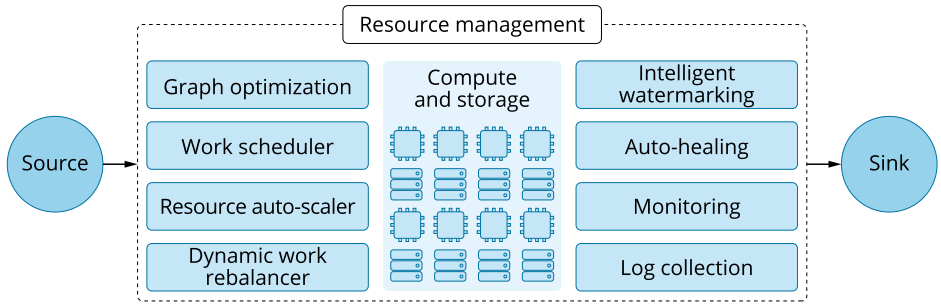


Fig. 3.13 Cloud Dataflow architecture

latency-sensitive pipelines). Here, disaggregation is fundamental to the viability of the system: Dataflow stores streaming state and checkpoints in a system-managed store that is disaggregated from the compute layer, allowing it to split and merge key ranges independently.

Disaggregation of storage and compute also allows for dynamic placement of different physical stages of a workload. Often, colocation of compute and storage in the same geographic location is the best policy, but the costs and benefits are dynamically scored against other execution strategies (e.g. moving small datasets between locations) in order to optimize the physical plan. This allows a single logical plan to evolve over time (according to changes in physical constraints), which is particularly important of heterogeneous compute across different accelerator families. While a given stage is executing, compute resources are dynamically scaled according to resource availability and desired throughput (Fig. 3.13).

In summary, disaggregation of compute and storage is a vital mechanism for providing efficient, scalable services across Google's data processing portfolio. As demonstrated by BigQuery and Dataflow, this disaggregation operates at multiple layers of the respective system designs. Both systems amplify these benefits by expressing processing logic using higher level abstractions (SQL for BigQuery, logical plans for Flume).

3.6 WSC workload tradeoffs and implications for infrastructure

Many large-scale services face similar challenges in achieving high performance or high availability. This section describes some of the most prevalent concepts; many papers and books provide interesting further perspectives from different organizations [48], [49], [50] and [51].

3.6.1 Replication and sharding

Data replication can improve both throughput and availability. It is especially powerful when replicated data is not often modified, since replication makes updates more complex.

In contrast, sharding splits a data set into smaller fragments (shards) and distributes them across a number of machines. Operations on the data set are dispatched to some or all of the shards, and the caller then coalesces results. The sharding policy can vary depending on space constraints and performance considerations. Using very small shards (or microsharding) is particularly beneficial to load balancing and recovery.

Often, keeping multiple replicas up-to-date using the traditional guarantees offered by a database management system significantly increases complexity, hurts performance, and reduces availability of distributed applications [50]. Fortunately, large classes of applications have more relaxed requirements and can tolerate inconsistent views for limited periods, provided that the system eventually returns to a stable consistent state (eventual consistency). However, storage using eventual consistency tends to push complexity into the application since it now needs to correctly handle non-intuitive situations. For example, after updating a value, a subsequent read may go to a different replica and return the previous value. Such complications motivated the design of scalable yet still fully consistent storage systems like Spanner [7].

3.6.2 Error codes and integrity checking

Data replication provides both fault tolerance and throughput improvements: if every copy of data is replicated (N) times, we get (N) times greater read throughput and exponentially higher availability and durability. For example, if a disk has an annual failure rate (AFR) of 2%, the annual risk of data loss is reduced from 2% with (N)=1 to $(2\% * 2\% * 2\% = 0.0008\%)$ for (N)=3 even if we didn't replace failed disks, and much lower than that if we replaced a failed disk in 48 hours.

Replication increases storage costs, and if an application doesn't require additional throughput (e.g., for rarely accessed archival storage), error correcting codes can provide availability and durability in more efficient ways. Most distributed storage systems use erasure codes, so called because they protect against the erasure (loss) of a subset of bits. The most common form of erasure coding is Reed-Solomon (RS) coding, which enables regeneration of missing data from the remaining pieces of data. A (k, m) RS code encodes a set of k data "chunks" into $(k + m)$ chunks. The total set of chunks comprises a stripe. As long as at least (k) out of $(k + m)$ chunks are available, $RS(k, m)$ can recover the entire data.

WSC storage faces two additional challenges: data corruption and storage degradation. While disks, SSDs, networks, and DRAM all have error-correcting codes (ECC) to protect against data corruption, these protections aren't strong enough to detect all bit errors at scale. For example, a TCP checksum only detects one-bit errors.

However, it wouldn't necessarily be efficient to improve low-level ECC to be strong enough! For example, suppose an SSD has a one-in-a-million chance of encountering a multi-bit flip that's not detected by ECC. Clearly, in a storage system with a million SSDs, this level of protection is insufficient. At the same time, expanding each device's ECC bits to improve detection to an acceptable level is inefficient if all of these SSDs are part of a file system already using erasure coding and software integrity guarantees (which also guard against possible data corruption from SSD firmware bugs). Thus, high durability is provided by the filesystem, not the hardware. Later in Chapter 10, we will also discuss the recent increase in errors from silent data corruption.

A second challenge is data degradation: ECC bits are checked on accesses only and thus don't discover slow degradation in infrequently accessed data. To guard against such degradation, WSC storage systems implement background integrity checking that periodically sweeps all data, ensuring that each block's integrity is verified at least once every N weeks or months.

3.6.3 Brownouts

A surprising insight from WSC operational experience is that failures are relatively easy to handle, but near-failures are hard. Near-failures are when failing devices enter a degraded state where they don't fail outright but where their performance drops to unacceptable levels. For example, a disk with surface contamination may not actually fail reads because the device's ECC protection can still recover all data, but it needs to read the corresponding media multiple times before it gets a "clean" read, leading to much slower seek performance. Borrowing from electrical terminology, such situations are labeled brownouts, to distinguish from clean failures (blackouts).

To guard against failures, WSC systems employ multiple protections. At the request level, it is critical to quickly determine that a server is too slow or unreachable and steer new requests away from it. Remote procedure calls must set well-informed timeout values to abort long-running requests, and infrastructure-level software needs to continually check for failing servers or devices and take appropriate action. Doing this successfully requires a delicate balance: being too permissive impairs service SLOs (in a sea of a million devices, brownouts happen pretty much continuously) but being too aggressive risks losing throughput and can even induce outages. For example, if a service is running near its provisioned capacity, responses will be slower than normal. Some servers or devices may be slow enough to suggest they're suffering from a brownout. Removing those systems from service, or retrying requests by sending them to replicas, may push the service into complete overload and cause a full outage.

3.6.4 Health monitoring and canaries

Monitoring service health can be surprisingly difficult, as the above example illustrates. At a basic level, it includes monitoring each device and each task involved in a service. Monitoring can involve passive data collection (e.g., CPU utilization) as well as active health checks where a monitoring system sends periodic requests to verify that the service component is still healthy.

A rare but realistic catastrophic failure scenario consists of a "query of death" (QoD) which causes the respective server to crash because of a bug. With QoDs, higher-level mechanisms intended to improve availability will actually multiply the problem; for example, retrying the QoD will crash additional servers. Similarly, in high fan-out services where a single request is sent to all servers, QoDs will lead to simultaneous crashes and full outages since the client making the request will usually retry it after getting an error, so even fast restarts won't help.

To guard against QoDs, traffic management systems like load balancers or RPC retry logic need to keep track of requests that are correlated with crashes of the server receiving the request. Since servers can have thousands of pending requests at any time, it's hard to catch QoDs after the first crash. Once identified as a QoD, the request is added to a blocklist that prevents it from being sent again, and an alert is raised. High fan-out services may additionally use canarying, i.e., sending each request to a smaller subset of servers and only submitting it to the rest of the system upon successful completion of that (canary) request. At the cost of doubling latency, canaries are highly effective in preventing QoD-driven outages.

More commonly, canaries guard rollouts of new versions. To avoid outages, a new version is first rolled out to a small percentage of servers or users (the "canaries"), and this canary set is monitored separately from the larger fleet, so that deviations (e.g., performance degradations, errors, crashes) aren't obscured by the signals from the much larger fleet running the older version (reporting "everything looks good on average"). Only once the new release passes this canary stage does the progressive rollout continue until it eventually reaches 100% of servers or users.

3.6.5 Load balancing

In large-scale services, service-level performance often depends on the slowest responder out of many servers: if a search query is sent to a thousand shards, its response time is the maximum of the shard response times. Reducing response-time variance is therefore critical.

As a high level, load balancing should equalize the amount of work per server. For example, each server should ideally be responsible for an equal amount of storage in a distributed storage service. This goal becomes more difficult when servers have unequal or varying capacity; for example, they may have different hardware, performance may be impaired by other tasks running on the same server, or the server may be performing

additional work at the time of the request (e.g., compaction, or chunk reconstruction after a failure). Requests may have highly variable cost, so that balancing requests per second doesn't achieve balanced load. Most importantly, actual request loads may not be evenly distributed, leading to hotspotting where many clients simultaneously make requests that are serviced by a single shard.

To achieve good performance, load balancers often require cooperation from the service being load balanced. For example, servers may report their current load (so that expensive requests are correctly reflected), and support health checks that allow the load balancer to drop servers that aren't healthy. In the simplest case, health checks might be ordinary requests (e.g., to fetch the homepage of a web server), or they're special requests implemented by the server (e.g., checking the consistency of its configuration). Either way, health checks can cause unintentional global outages if they have bugs, and thus need to be carefully tested.

Load balancing can be simpler if it's possible to separate (disaggregate) compute and storage, which allows expanding compute capacity independently of storage capacity. We saw a practical example in the discussion of the BigQuery system: if a server handling a particular shard experiences persistent hotspotting, disaggregated storage allows that shard to be split dynamically, distributing request processing to multiple servers.

3.6.6 Consensus

We all agree that consensus is important! In distributed systems, "consensus" refers to an agreement on the system's state by a group of participants. For example, is a particular service up or down? Among these two servers, which one is the primary and which one is the backup (i.e., is the primary healthy, or has the backup taken over the role)?

In theory, a distributed system with a single primary server limits the resulting system availability to the availability of that server. In practice, centralized control is simpler to implement and generally yields more responsive control actions. At Google, we have tended towards centralized control models for much of our software infrastructure, unless it's impossible to do so.

For applications needing consensus (e.g., on configuration state), the Chubby lock service [8] provides a simple way to establish consensus. Chubby provides a POSIX-like file API with coarse-grained locking. For example, it's simple to elect a primary using Chubby by creating a file containing the identity of the current primary. All the entities wanting to become a primary try to open the file in write mode. Only one of those will get write mode access and then writes its identity to the file. All the others get a file modification event and now know about the current primary. "Keep-alive" calls by the primary maintain its ownership; if they expire, another writer's attempt will succeed and that server becomes the new primary. Several popular open-source systems implement similar APIs, including Apache ZooKeeper [52], and Kubernetes' etcd [53].

3.6.7 Compression and encryption

Storage comprises a significant cost in modern data centers, and compression can significantly reduce these costs. Compression trades increased compute costs (for compression and decompression) for reduced storage cost. Modern CPUs and NICs contain hardware support for compression, so that this tradeoff usually is beneficial. Although generic compression algorithms can do quite well, application-specific compression schemes can achieve significantly superior compression factors or better decompression speeds. For example, Google's search index is compressed with an application-specific technique.

Compressed data increases effective cache sizes, which can be an important side benefit for services with high throughput requirements. For most compression algorithms, decompression is much cheaper than compression, making compressed caches especially useful for applications where read traffic dominates.

All data should be encrypted at rest. Unfortunately, good encryption produces near-random data which cannot be compressed. Thus, storage systems must carefully coordinate the two to obtain both benefits. For example, compression can be performed at the client as part of a write() call, before the data is encrypted.

3.6.8 Tail tolerance

In very large-scale systems, the completion of a parallel task can be held up by the slower execution of a very small percentage of its subtasks. The larger the system, the more likely this situation can arise. Sometimes a small degree of redundant execution of subtasks can result in large speedup improvements.

Earlier in this chapter, we described a number of techniques commonly used in large scale software systems to achieve high performance and availability. As systems scale up to support more powerful online web services, we have found that such techniques are insufficient to deliver service-wide responsiveness with acceptable tail latency levels. (Tail latency refers to the latency of the slowest requests, that is, the tail of the latency distribution.)

At large enough scale, simply stamping out all possible sources of performance variability in individual system components is as impractical as making all components in a large fault-tolerant system fault-free [54].

Consider a hypothetical system where each server typically responds in 10 ms but with a 99th percentile latency of 1 second. In other words, if a user request is handled on just one such server, 1 user request in 100 will be slow (take 1 second). Figure 3.14 shows how service level latency in this hypothetical scenario is impacted by very modest fractions of latency outliers as cluster sizes increase. If a user request must collect responses from 100 such servers in parallel, then 63% of user requests will take more than 1 second . Even for

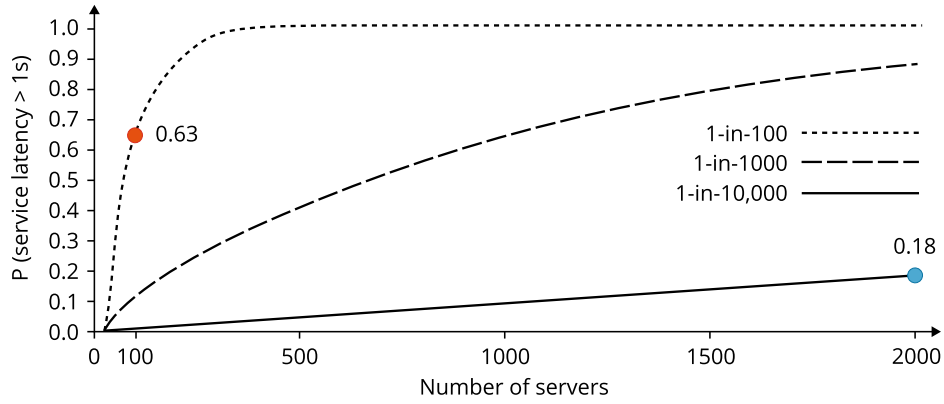


Fig. 3.14 Service latency variation with tail events

services with only 1 in 10,000 requests experiencing over 1 second latencies at the single server level, a service with 2,000 such servers will see almost 1 in 5 user requests taking over 1 second.

To tolerate these kinds of latency variability and still deliver low tail latency at the service level, special efforts are required [54]. Often, these techniques take advantage of resource replication that has already been provisioned for fault-tolerance, thereby introducing only small additional overheads to existing systems. For example, with hedged requests, a client first sends one request to the server believed to be the most appropriate, but then falls back on sending a secondary request after some brief delay (e.g., until the first request has been outstanding for more than the 95th-percentile expected latency for this class of requests).

Independently, as networks and storage have reduced their latencies into the microsecond range, it has become increasingly important to optimize systems for latency. It is relatively easy to squander all the benefits from microsecond devices if supporting software is not tuned for such small latencies [55].

3.7 Some recurring themes

Software development in WSC services differs from the traditional desktop/server model in many ways:

3.7.1 Ample parallelism

WSC services exhibit a large amount of parallelism stemming from both data- and request-level parallelism. Usually, the problem is not to find parallelism but to manage and efficiently harness the explicit parallelism inherent in the application. Data parallelism arises from the

large data sets of relatively independent records that need processing, such as billions of files stored by cloud customers, or billions of log lines. These large data sets often require significant computation for each parallel (sub) task, which in turn helps hide or tolerate communication and synchronization overheads. Similarly, request-level parallelism stems from the hundreds or thousands of requests per second that such services receive. These requests rarely involve read-write sharing of data or synchronization across requests. For example, search requests are essentially independent and deal with a mostly read-only database; therefore, the computation can be easily partitioned both within a request and across different requests. Similarly, requests from different GCP users are essentially independent from each other since each customer only accesses their own data, creating natural units of data partitioning and concurrency. As long as the update rate is low, even systems with highly interconnected data (such as social networking backends) can benefit from high request parallelism.

3.7.2 Workload evolution

Users of WSC services are isolated from the services' implementation details by relatively well-defined and stable high-level APIs, making it much easier to deploy new software quickly. For example, thousands of individual binaries are released every single day across Google Cloud's stack.

This environment favors rapid product innovation but makes it hard for a system designer to extract useful benchmarks even from established services. Moreover, new products and services may emerge, and their success with customers directly affects the resulting workload mix in the data center. For example, video services such as YouTube have flourished in relatively short periods and present a very different set of requirements from the existing large customers of computing cycles in the data center, potentially affecting the optimal design point of WSCs. A beneficial side effect of this aggressive software deployment environment is that hardware architects are not necessarily burdened with having to provide good performance for immutable pieces of code. Instead, architects can consider the possibility of significant software rewrites to leverage new hardware capabilities or devices.

3.7.3 Platform homogeneity and abstraction

The data center is generally a more homogeneous environment than on-premise servers or desktops and laptops as a target platform for software development. Large internet services operations typically deploy a relatively small number of hardware and system software configurations at any given time. In fact, in the early days of hyperscalers when Moore's Law was still going strong and a new hardware generation could be twice as fast as any

previous platform, it was common to only deploy a single "machine of the day" at any given time, switching to the next one when the next generation arrived.

That's not the case anymore. Server platforms improve much more slowly, so that specialized platforms (e.g., optimized for memory bandwidth per core, or maximum number of cores) can provide TCO improvements for certain workloads that are bigger than those provided by the next-generation generic platform. Additionally, now that server lifetimes have increased to six years or more, more generations of hardware are in service at the same time. Even if a provider only introduces a handful of platforms in a given year (with different CPU types, motherboards etc, vs just different configurations of the same platform), many dozens of platforms will be live on the data center floor at any given time.

The best way to handle this hardware heterogeneity is to hide it from most of the system stack by moving management functions elsewhere. For example, the offload cards discussed above move virtualization into a separate card, reducing or eliminating server-specific code. Similarly, Chapter 6 describes how server management and monitoring (e.g., fan speed control) has moved to a separate card. Chapter 7 will discuss how software-defined infrastructure further abstracts hardware complexity through software wrappers and management.

WSC providers still see a much more uniform environment than on-premise providers. Software for on-premise systems can make few assumptions about the hardware or software platform they are deployed on, and their complexity and performance characteristics may suffer from the need to support thousands or even millions of hardware and system software configurations.

3.7.4 Fault management

Because internet service applications run on hundreds of thousands of machines, some type of fault is expected every few hours or less (more details are provided in Chapter 10). As a result, although it may be reasonable for desktop-class software to assume a fault-free hardware operation for months or years, this is not true for WSC services: such services need to work in an environment where faults are part of daily life. Ideally, the cluster-level system software should provide a layer that hides most of that complexity from application-level software, although that goal may be difficult to accomplish for all types of applications.

Although the plentiful thread-level parallelism and a more homogeneous computing platform help reduce software development complexity in internet services compared to desktop systems, the scale, the need to operate under hardware failures, and the speed of workload churn have the opposite effect.

References

1. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," 1, vol. 51, New York, NY, USA: Association for Computing Machinery, Jan. 1, 2008, pp. 107–113.
2. Apache Software Foundation, *Apache Hadoop*, https://hadoop.apache.org/.
3. R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall," *Scientific Programming*, vol. 13, no. 4, pp. 277–298, 2005.
4. F. Chang, J. Dean, S. Ghemawat, et al., "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 205–218.
5. G. DeCandia, D. Hastorun, M. Jampani, et al., "Dynamo: Amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2007, pp. 205–220.
6. S. Melnik, A. Gubarev, J. J. Long, et al., "Dremel: Interactive analysis of web-scale datasets," 1-2, vol. 3, VLDB Endowment, Sep. 1, 2010, pp. 330–339.
7. J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Google's globally-distributed database," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12, USA: USENIX Association, Oct. 8, 2012, pp. 251–264.
8. M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 335–350.
9. A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2015, pp. 1–17.
10. M. Tirmazi, A. Barker, N. Deng, et al., "Borg: The next generation," in *Proceedings of the 15th European Conference on Computer Systems*, ser. EuroSys '20, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2020, pp. 1–14.
11. P. Mell, *The NIST definition of cloud computing*, https://csrc.nist.gov/pubs/sp/800/145/final, 2011.
12. Google Cloud, *Compute Engine: Virtual Machines (VMs)*, https://cloud.google.com/products/compute.
13. Google Cloud, *Google Kubernetes Engine (GKE)*, https://cloud.google.com/kubernetes-engine.
14. Google Cloud, *Cloud Run - Serverless Container Platform*.
15. Google Cloud, *Cloud SQL: Relational Database Service*, https://cloud.google.com/sql/.
16. HashiCorp, *Terraform by HashiCorp*.
17. Google, *Google Workspace*, https://workspace.google.com/sql.
18. S. Kanev, J. P. Darago, K. Hazelwood, et al., "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 158–169.
19. Google, *Organizing information - How Google Search Works*, https://www.google.com/intl/en/search/howsearchworks/how-search-works/organizing-information/.
20. P. Nayak, *Understanding searches better than ever before*, 2019.
21. S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, 1998.
22. A. Lottarini, A. Ramirez, J. Coburn, et al., "Vbench: Benchmarking video transcoding in the cloud," in *Proceedings of the 23th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18, New York, NY, USA: Association for Computing Machinery, Mar. 19, 2018, pp. 797–809.

23. T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proceedings of the twenty-sixth annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '07, New York, NY, USA: Association for Computing Machinery, Aug. 12, 2007, pp. 398–407.
24. A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is all you need," vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, et al., Eds., 2017.
25. G. Team, R. Anil, S. Borgeaud, et al., "Gemini: A family of highly capable multimodal models," arXiv preprint arXiv:2312.11805, 2023.
26. T. B. Brown, B. Mann, N. Ryder, et al., *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].
27. OpenAI, J. Achiam, S. Adler, et al., "GPT-4 technical report," 2024. arXiv: 2303.08774 [cs.CL].
28. H. Touvron, T. Lavril, G. Izacard, et al., "Llama: Open and efficient foundation language models," 2023. arXiv: 2302.13971 [cs.CL].
29. Anthropic, *Introducing the next generation of Claude*, News: https://www.anthropic.com/news/claude-3-family, Mar. 2024.
30. U. Savagaonkar, N. Porter, N. Taha, B. Serebrin, and N. Mueller, "Titan in depth: Security in plaintext," *Google Cloud Identity and Security Blog*, 2017.
31. Google Cloud, *Shielded VM Overview*, Google Cloud Security Documentation: https://cloud.google.com/docs/security/boot-integrity.
32. Google Cloud, *Titanium underpins Google's workload-optimized infrastructure*, Google Cloud Blog: https://cloud.google.com/blog/products/compute/titanium-underpins-googles-workload-optimized-infrastructure.
33. Google Cloud, *Google Cloud Titanium*, https://cloud.google.com/titanium.
34. Google Cloud, *Announcing PSP security protocol is now open source*, Google Cloud Blog: https://cloud.google.com/blog/products/identity-security/announcing-psp-security-protocol-is-now-open-source.
35. Google Cloud, *About Hyperdisk block storage*, Compute Engine Documentation: https://cloud.google.com/compute/docs/disks/hyperdisks.
36. M. Dalton, D. Schultz, J. Adriaens, et al., "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 373–387.
37. Google Cloud, *VM live migration process*, Compute Engine Documentation: https://cloud.google.com/compute/docs/instances/live-migration-process.
38. Google Cloud, *Introduction to object tables*, BigQuery Documentation: https://cloud.google.com/bigquery/docs/object-table-introduction.
39. D. Hildebrand and D. Serenyi, *Colossus under the hood: A peek into Google's scalable storage system*, https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system?e=48754805, 2021.
40. Google Cloud, *Inside Capacitor, BigQuery's next-generation columnar storage format*, Google Cloud Blog: https://cloud.google.com/blog/products/bigquery/inside-capacitor-bigquerys-next-generation-columnar-storage-format.
41. Google Cloud, *Introduction to BigQuery autoscaling*, BigQuery Documentation: https://cloud.google.com/bigquery/docs/slots-autoscaling-intro.
42. Google Cloud, *In-memory query execution in Google BigQuery*, Google Cloud Blog: https://cloud.google.com/blog/products/bigquery/in-memory-query-execution-in-google-bigquery.
43. V. Edara, Z. Zhao, K. Agrawal, and I. Gupta, "Network-compute co-design for distributed deep learning," in *Proceedings of the VLDB Endowment (PVLDB)*, vol. 14, 2021, pp. 3083–3091.
44. J. Levandoski, G. Casto, M. Deng, et al., "Biglake: BigQuery's evolution toward a multi-cloud lakehouse," in *Companion of the 2024 International Conference on Management of Data*, ser.

SIGMOD/PODS '24, New York, NY, USA: Association for Computing Machinery, Jun. 9, 2024, pp. 334–346.
45. T. Akidau, R. Bradshaw, C. Chambers, et al., "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, Aug. 2015, ISSN: 2150-8097.
46. C. Chambers, A. Raniwala, F. Perry, et al., "Flumejava: Easy, efficient data-parallel pipelines," in *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '10, Toronto, Ontario, Canada: Association for Computing Machinery, 2010, pp. 363–375.
47. Google Cloud, *No shard left behind: Dynamic work rebalancing in Google Cloud Dataflow*, Google Cloud Blog: https://cloud.google.com/blog/products/gcp/no-shard-left-behind-dynamic-work-rebalancing-in-google-cloud-dataflow.
48. J. R. Hamilton et al., "On designing and deploying internet-scale services.," in *LISA*, vol. 18, 2007, pp. 1–18.
49. E. Brewer, "Lessons from giant-scale services," *IEEE Internet Computing*, vol. 5, no. 4, pp. 46–55, 2001.
50. W. Vogels, "Eventually consistent: Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability," 6, vol. 6, New York, NY, USA: Association for Computing Machinery, Oct. 1, 2008, pp. 14–19.
51. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google runs production systems.* " O'Reilly Media.", 2016.
52. P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.
53. *etcd - Distributed reliable key-value store*, https://etcd.io/.
54. J. Dean and L. A. Barroso, "The tail at scale," 2, vol. 56, New York, NY, USA: Association for Computing Machinery, Feb. 1, 2013, pp. 74–80.
55. L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Check for updates

WSC Software: Distributed Systems Infrastructure 4

4.1 Overview of the software stack

In the previous chapter, we focused on the applications that run on warehouse-scale computers (WSCs) and how they impact system trade-offs and design decisions. In this chapter, we will focus on the lower parts of the stack. As a quick recap, Figure 4.1 presents the different layers.

4.2 Distributed systems

In a single system, the operating system layer manages resources and provides basic services to that single computer. Much like that, a WSC system composed of thousands of computers, networking, and storage also requires software that provides analogous functionality at a larger scale. This section discusses the most important components of that software: cluster-level systems, storage, and WAN and edge networks.

4.2.1 Cluster-level infrastructure

Many WSCs are linked into a global network and provide global services. Clusters represent the smallest domain being managed as a unit. A cluster (sometimes also called a cell) consists of machines linked into a single high-bandwidth cluster network like Jupiter (Chapter 6). Typically, clusters contain tens of thousands of servers located in a single data center hall or building (i.e., 10-30 MW of power). All the servers in the cluster are connected to the same network with a single control plane, so there is a chance they can all become unavailable at the same time.

© The Author(s) 2026
L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_4

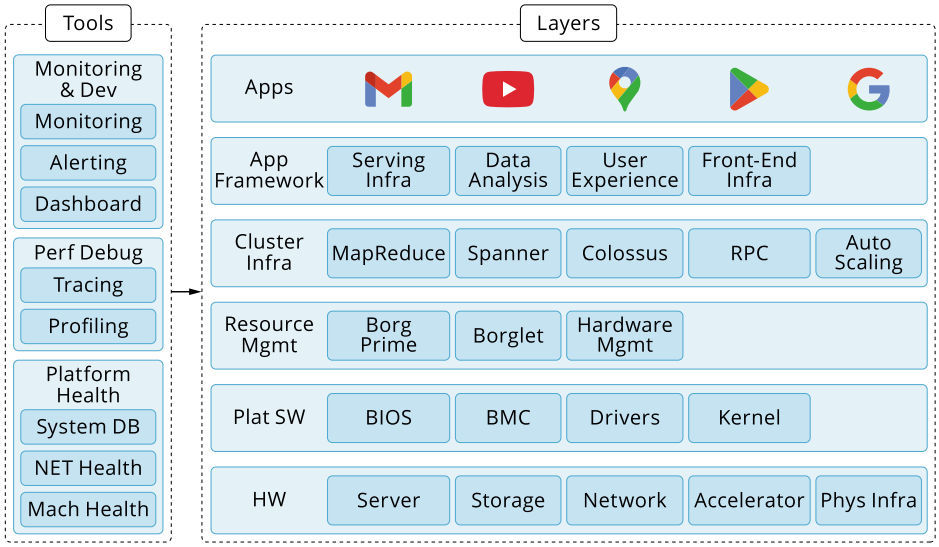


Fig. 4.1 Google software stack for warehouse-scale computers

Clusters are lower-level than zones (see Chapter 2) because a zone can contain multiple clusters spread over multiple buildings. For example, a new zone can start as a single cluster and add additional clusters over time.

4.2.1.1 Resource Management

Resource Management is perhaps the most indispensable component of the cluster-level infrastructure layer. It controls the mapping of users and jobs to hardware resources, enforces priorities and quotas, and provides basic task management services. Resource management has four distinct responsibilities:

- **Hardware management** allows hardware to be added, upgraded, repaired, and removed. Managing hardware resources effectively requires inventory management (e.g., what's on the floor, where, and what components are inside), state management (e.g., is the hardware powered on and healthy, etc), and workflow tools (e.g., to direct a technician to a server, identify a component to be replaced, and record the barcode of the old and new components).
- **Machine state management** deals with bits stored locally on each device, such as firmware images, hardware configuration parameters (e.g., clock speed settings), and device identities (e.g., certificates).
- **Job management** schedules workloads onto individual machines and reacts to changing supply (e.g., machines becoming unavailable) or demand (e.g., jobs wanting to scale up

to handle additional load). Depending on the level of abstraction provided to users, the scheduler may allocate raw machines (bare metal servers), virtual machines (typically occupying fractional machines), or containers (abstracting even the virtual machines).

- **Capacity management** extends resource management to the physical supply chain, balancing the cost of having too much (paying for idle resources) with the cost of having too little (impacting applications via stockouts of critical resources). In contrast to job management, which makes sub-second allocation decisions, capacity management operates on timeframes from days to months, depending on the lead time of hardware resources.

In its simplest form (still practiced in small-scale enterprises), resources are manually and statically allocated to a given user or job, typically leading to low utilization. In contrast, WSC resource management must be highly automated, driven by necessity (manual tracking of millions of resources is infeasible) and by opportunity (a 1% efficiency gain may represent millions of dollars of savings).

Google's Borg scheduler [1], [2] schedules jobs into a heterogeneous pool of servers. Each job represents a service (e.g., a search backend) consisting of multiple tasks. Each task represents a single binary (with accompanying configuration files) that runs on a single machine. Thus, the job of scheduling a search backend job may consist of scheduling thousands of tasks, each handling a shard of the search index. All machines are shared, turning task scheduling into a bin packing problem whose goal is to schedule all tasks (from all jobs) onto a pool of machines with the least amount of internal fragmentation, i.e., stranded resources. For example, if a task requires 32 cores and 40 GB of memory, it is undesirable to schedule it onto a server that has 32 remaining idle cores and 100GB of memory, since this would strand the remaining 60GB of memory.

Jobs may specify scheduling constraints. Affinity (or anti-affinity) constraints specify that particular tasks should (not) be co-scheduled on the same machine or switch or power domain; for example, a database job with two replicas would want its secondary replica to be on a different machine, and different switch, than the primary machine, so no single failure can affect both replicas. Other constraints may include machine types, to optimize efficiency and performance. Some of these constraints may be generated automatically based on profiling information. For example, the Borg ecosystem includes tools that recognize when a job runs particularly well (or poorly) on a particular hardware, and biases the scheduler to take this into account.

Kubernetes [3] is a popular open source container manager that is based on ideas and concepts from Borg (in fact, the initial Kubernetes team consisted primarily of engineers who previously worked on Borg). Kubernetes aims to be highly portable and extensible, running on many public and private clouds and on physical and virtual machines. Unlike Borg, it doesn't cover the lifecycle of the machines themselves, so it assumes an underlying infrastructure that provides VMs.

While many Kubernetes users treat it as a cloud API abstraction, some seek to abstract even more. Kubernetes includes many extension points, including within the API, which

allows organizations to create bespoke platforms for their internal users. These internal developer platforms allow companies to embed and enforce their own opinions about how services should be run. Kubernetes's extensibility has also spawned several open-source PaaS and Function as a Service (FaaS) projects, and has influenced other container-oriented products. For example, GCP's Cloud Run [4] provides a Kubernetes-like abstraction that hides not just VMs but most of the operational details, taking only the user's container as an input and running it as a service.

Today, resource managers also incorporate power, energy, and carbon optimizations. For example, they allow oversubscription of the power infrastructure (discussed in Chapter 9), capping low-priority workloads when needed [5], and cross-regional schedulers may move workloads to the region with the lowest-carbon energy [6].

4.2.1.2 Foundational infrastructure software

Nearly every large-scale distributed application needs a small set of basic functionalities. Examples are reliable distributed storage, remote procedure calls (RPCs), and cluster-level synchronization and locking. Implementing this type of functionality correctly with high performance and high availability is complex in large clusters. It is wise to avoid re-implementing such tricky code for each application and instead create modules or services that can be reused. Colossus (successor to GFS) [7], [8], Dynamo [9], and Chubby [10] are examples of reliable storage and lock services for large clusters.

Many tasks that are simple manual processes in a small deployment require a significant amount of infrastructure in large-scale systems: software image distribution and configuration management, monitoring, triaging alerts in emergency situations, and many more. Autopilot [11] offers an example design for some of this functionality for Windows Live data centers. Monitoring the overall health of the hardware fleet also requires careful monitoring, automated diagnostics, and automation of the repairs workflow. Google's System Health Infrastructure [12] is an example of the software infrastructure needed for efficient health management. Finally, performance debugging and optimization in systems of this scale need specialized solutions as well. Dapper [13] was one of the first real-word systems for collecting RPC traces to enable debugging of performance problems and inspired today's Open Telemetry ecosystem [14].

4.2.2 Unstructured storage systems

Google's GFS [7] was the first cluster-scale storage system with a simple file-like abstraction. Initially, GFS was designed to support a single use case, the web search indexing system (the system that turned crawled web pages into index files for use in web search). Thus, it focused on high throughput for thousands of concurrent readers/writers and robust performance under

high disk failures rates. GFS users typically manipulated large quantities of data, and thus GFS was further optimized for large reads and writes.

The system architecture consisted of a primary server which handled metadata operations, and thousands of chunkserver (secondary) processes running on every server with a disk drive, to manage the data chunks on those drives. In GFS, fault tolerance was provided by replication across machines instead of within them, as in conventional RAID systems. Cross-machine replication allowed the system to tolerate machine and network failures and enables fast recovery, since replicas for a given disk or machine were spread across thousands of other machines.

GFS made several choices to accelerate its deployment. Most prominently, the file system's metadata had to fit into memory on a single server. This limitation greatly reduced the implementation effort but limited both the scale of a storage cell and the achievable uptime when the central metadata server failed. As the demands on GFS broadened (e.g., from Gmail and YouTube use cases), GFS was replaced by its successor Colossus [15].

A Colossus cell consists of many folders; each folder is effectively its own filesystem. Each folder's metadata is stored as an independent hierarchy of BTrees that hold the file metadata. This top directory layer is called Metadata1 or MD1. The metadata of MD1 BTree files is stored in another level of BTree called MD0, and the metadata of that tree (the file descriptors of the MD0 files) is stored in a distributed log system implemented with Paxos [16]. (MD0 and MD1 are omitted from Figure 4.2 for simplicity.)

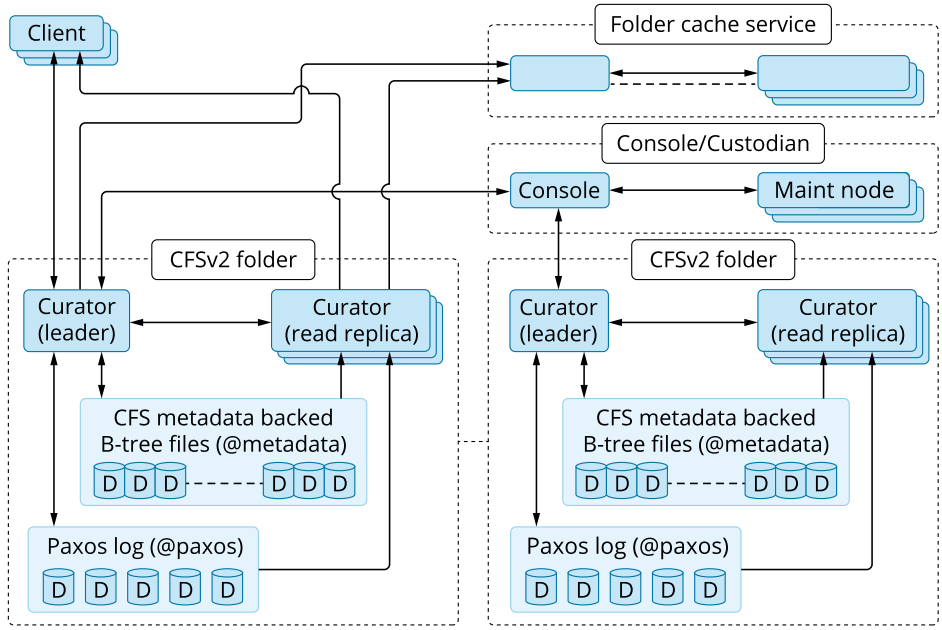


Fig. 4.2 Simplified architecture of the Colossus file system

Curators implement a highly available, horizontally scalable metadata server. They are responsible for managing the namespace, file system structure, and associated metadata (access control lists, timestamps, etc). Curators handle file operations (create, delete, rename, etc.) but aren't involved in reads and writes. Paxos determines which curator is the leader (the server able to mutate metadata) of this folder.

All data is stored in a lower-level system called D which implements a simple API that resembles that of network-attached disks, thus the name. D is not a distributed file system; each D server provides network access to files only on its local storage devices. D does not provide any serious reliability guarantees other than per-file checksums. Any kind of replication, error correction, or other backup systems must be implemented by the system that uses it (i.e., Colossus).

The Colossus client library implements significant portions of the overall functionality, including replication, load balancing, and handling unresponsive D servers. For example, if a file uses three-way replication (r=3), the client is responsible for sending the write to three replicas. Putting this functionality into the client provides effective scalability: the more readers and writers a file system has, the more CPU and network bandwidth is available for read and write operations, and the overall system performance is less likely to bottleneck on the actual Colossus servers since they are only involved in a subset of operations, like opening a file.

The initial version of GFS supported only simple replication, but Colossus added support for more space-efficient Reed-Solomon codes, which reduce the space overhead of replication by roughly a factor of two over simple replication for the same level of availability (see section 3.3.2). Additionally, Colossus distributes file chunks across the whole cluster such that correlated failures are extremely unlikely to lead to data loss (e.g., replicas are stored on different racks and in different network blocks).

Wide distribution of chunks across devices over a whole cluster also speeds up recovery. Since replicas of chunks stored on a given device are spread across many machines in a storage cluster, reconstruction of lost data chunks is performed in parallel at high speed. Quick recovery is important since long recovery time windows would leave under-replicated chunks vulnerable to data loss, in case additional faults hit the cluster. A comprehensive study of availability in distributed file systems at Google can be found in Ford et al. [17]. A good discussion of the evolution of file system design at Google can also be found in McKusick and Quinlan [15].

Putting a lot of functionality into the Colossus client comes with one notable disadvantage: to fix bugs, or provide new functionality, all Colossus clients need to be recompiled and then restarted! That is a highly undesirable property, since it's usually impossible to control the pace of upgrades of the applications using a file system. However, for purely internal usage within a single company (the cloud provider, i.e., Google) it is a practical choice: at Google, a standard policy requires all binaries to be rebuilt regularly for a variety of reasons, and this policy keeps Colossus clients up to date as well. However, Colossus' public cousin GCS [18] made the opposite choice because it is impossible to force cloud customers to update their

applications in a timely manner. Thus, the GCS client library is fairly simple and forwards requests to an API server containing the actual Colossus client.

4.2.3 Structured WSC storage

The simple file abstraction of Colossus and GCS may suffice for systems that manipulate large blobs of data, but application developers also need database-like functionality, where data sets can be structured and indexed for easy small updates or complex queries. Structured distributed storage systems, such as Google's Bigtable [19], and Amazon's DynamoDB [9], were designed to fulfill those needs. Compared to traditional database systems, Bigtable and DynamoDB sacrifice some features, such as the richness of schema representation and strong consistency, in favor of higher performance and availability at massive scales. Bigtable, for example, presents a simple multidimensional sorted map consisting of row keys (strings) associated with multiple values organized in columns, forming a distributed sparse table space. Column values are associated with timestamps to support versioning and time-series. Compared to even the highest-performing traditional SQL databases, Bigtable can reach throughputs many orders of magnitude higher (billions of queries per second).

However, providing just eventual consistency in Bigtable and DynamoDB shifts the burden of achieving full consistency to the applications using these systems. In particular, multi-step operations can become hard to write, and thus error-prone. For example, in an eventually consistent system, it's possible for an application to write new data but then, at a later time, see the old data when it is read again, if the second read goes to a different replica than the initial write. Second-generation structured storage systems such as Megastore [20] and especially Spanner [21] were designed to address such concerns. Both Megastore and Spanner provide richer schemas and SQL-like functionality while providing simpler, stronger consistency models. Megastore sacrificed write throughput to provide synchronous replication, limiting its usefulness and adoption.

Spanner uses a new time-based API called TrueTime to efficiently serialize globally-distributed transactions without giving up horizontal scalability, providing a simple consistency model to applications that need seamless wide-area replication for fault tolerance. TrueTime is a globally distributed clock based on GPS and atomic clocks that provides a tight bound on clock uncertainty. Instead of returning a single point in time, TrueTime gives each machine an interval [earliest, latest] where the true time lies. Spanner uses TrueTime to assign timestamps to transactions. By relying on these globally consistent timestamps, Spanner can determine the order of transactions even if they originate from different servers across the globe.

Spanner provides external consistency, which is the strictest consistency property for transaction-processing systems. All transactions in Spanner satisfy this consistency property, not just those within a partition. External consistency states that Spanner executes transactions in a manner that is indistinguishable from a system in which the transactions

are executed serially. In addition, if one transaction completes before another transaction starts to commit, the system guarantees that clients can never see a state that includes the effect of the second transaction but not the first [22].

Spanner offers a fully consistent, SQL-like database with horizontal scaling and very high availability (99.999% SLO). This appears to violate the limitations imposed by the CAP theorem [23] which states that a system can only achieve two out of the three properties: Consistency, Availability, and resilience to network Partitions. NoSQL systems must trade consistency to get the other two properties, so how can Spanner provide all three? In reality, Spanner only provides CP (dropping Availability) but in practice its availability is so high that users can consider it a CA system [24]. This combination is possible in practice if one has full control over the network, which is rare over a WAN but is achieved in Google's global network. Even then, it requires significant redundancy of network paths, architectural planning to manage correlated failures, and very careful operations, especially for upgrades. If outages do occur, Spanner chooses consistency over availability.

Because of its advantages in consistency and reliability, Spanner has become the transactional storage system of choice for almost all WSC users. For example, at Google, Gmail and Google Photos store all their data in Spanner, as do Google's clickstream and ads data management systems. For the latter, Google's F1 system [25] uses Spanner as data store, and manages all AdWords data with database features such as distributed SQL queries, transactionally consistent secondary indexes, asynchronous schema changes, optimistic transactions, and automatic change history recording and publishing.

At the other end of the structured storage spectrum are systems that aim almost exclusively at high performance. Such systems tend to lack support for transactions or geographic replication, use simple key-value data models, and may have loose durability guarantees. Memcached [26], developed as a distributed DRAM-based object caching layer, is widely used for simple caching. Fast key-value stores attracted significant interest in the academic community once their value became clearer. For example, the Stanford RAMCloud [27] system used a distributed DRAM-based data store but aimed at much higher per-node performance and durability in the presence of storage node failures. The FAWN-KV [28] system also presented a key-value high-performance storage system but instead used NAND flash as the storage medium, and has an additional emphasis on energy efficiency, a subject we cover more extensively in Chapter 9. Today, several open source key-value storage systems are popular, including Redis ([29]).

4.2.4 WAN and edge networking

While this book primarily discusses WSCs as large building blocks, most WSCs are part of a global infrastructure operated by a hyperscaler or public cloud provider. Here we sketch the infrastructure that's outside the WSCs themselves; while it is much smaller, it has an outsized impact on the value of WSCs themselves.

We'll illustrate this infrastructure using an example request to www.google.com from a user in Germany. As a first step, Google's DNS server needs to return an IP address for that endpoint, preferably located near the user. Today, this is accomplished via Anycast DNS [30]: the same IP address (say, 1.2.3.4) is announced from all regions that have servers for it, and the internet's standard Border Gateway Protocol (BGP) routing directs the user's request to the region that is the fewest network hops away (e.g., Hamburg). Google uses Anycast both for the DNS server itself and the frontend IP receiving the request.

An alternative approach is to use a smart DNS server to compute the best location for a request, using the request's source IP. This approach relies on having a global network database that maps IP ranges to locations; for example, it might know that the user's IP is in a subnet used by Deutsche Telekom in Hamburg, and thus that Hamburg is the right region to direct the request to. Google's DNS used such an approach before switching to Anycast. The primary drawbacks of IP-based name resolution lie in the complexity of building and maintaining the network map, and in the inherent limitation that too little is known at the time of the DNS request. In particular, the DNS request reaching Google's DNS often isn't directly sent by the user but by an intermediate (recursive) DNS resolver owned by the user's ISP or company, and these resolves can be far away from the user. Thus, the DNS request's source IP address can be a poor proxy for the user's actual location. (The HTTP/3 specification now includes an option for the server to specify a preferred IP address that's different from the current one [31], but as of 2024 this feature was not widely implemented yet.)

The user's request then reaches an edge location, so called because it is at the edge (outer boundary) of the network, vs near its core (e.g., an actual WSC). "Edge" is an imprecise term that is used differently depending on context. For example, sometimes it refers to the client device (i.e., edge computing happens on the user's phone or laptop), sometimes to a highly distributed network location like a neighborhood network cabinet or tower, or to a network POP (point of presence) in a larger city.

Google's network has two layers of edge serving. Google Global Cache (GGC) [32] uses edge caching, content compression, and intelligent routing to deliver content quickly and efficiently. Most GGC nodes consist of just a few servers and are located inside a partner ISP's network. In that sense, they are located beyond Google's network edge, since they are not directly connected to Google's network, nor are they hosted in Google's space.

GGC is an example of a Content Delivery Network (CDN), a network specifically engineered to serve static content (images, videos) or stream live content. GGC can greatly speed up content delivery (e.g., YouTube videos or Spotify songs) while reducing the ISP's infrastructure cost via caching. Globally there are tens of thousands of GGC locations; the map in Figure 4.3 shows just the larger caching nodes that are part of the Cloud Media CDN.

The second edge layer is at the real network edge, typically at an internet exchange or networking POP [33]; there are well over a hundred such locations worldwide. Continuing our example, at the time of this writing (2024), Google operates five edge locations in Germany, including one in Hamburg, so the user's request would likely be served from the

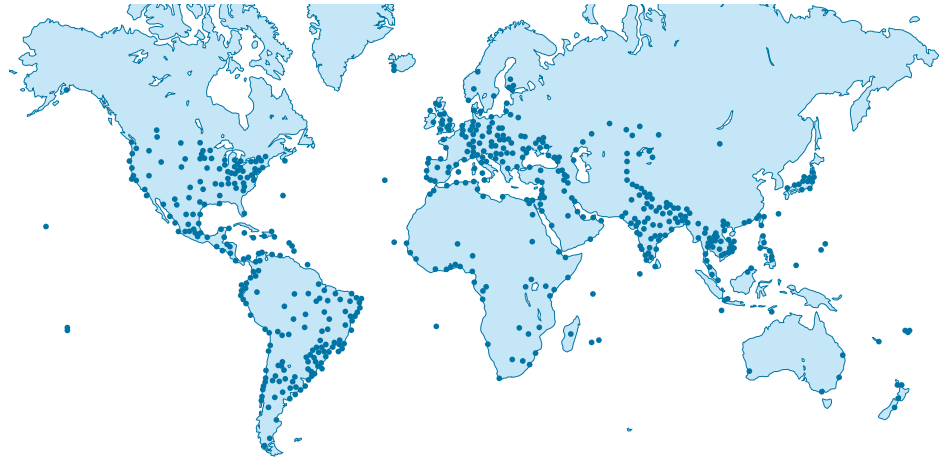


Fig. 4.3 Media Content Distribution Network (CDN) locations

Hamburg edge POP. Edge POPs come in a variety of sizes depending on the area they serve, and typically contain both frontend servers to handle or route requests, and caching servers to form a second level of caching for popular content. Requests that can't be handled locally are forwarded to the nearest data center with available servers for that service. E.g., a GCP request might be forwarded to the Frankfurt or Berlin GCP region, but a search request might be forwarded to a nearby data center serving search (e.g., in Denmark or The Netherlands).

In addition to handling requests, network POPs also house peering routers that connect Google's network with that of ISPs present in that location. "Peering" denotes arrangements where two networks agree to directly interconnect and exchange traffic between themselves, with neither side paying the other. Peering can improve network performance for both sides because it provides direct visibility into traffic levels; for example, if a port to ISP A in Hamburg becomes overloaded, Google can redirect traffic to a less congested port elsewhere.

Different networks have different routing and peering policies. "Hot potato" routing minimizes network cost by letting packets exit the network as soon as possible; for example, if a packet is destined for ISP B but there's a local ISP A that can route it to B, the packet is handed off immediately to A and then travels over the public internet, possibly through several ISPs, until it reaches B. "Cold potato" routing does the opposite, keeping the packet on the internal backbone until it is handed off to B at a location close to the user. With a cold potato routing policy, an open peering policy, and a presence in most peering locations, more than 98% of Google's packets only see a single hand-off from Google's network directly to the end user's ISP or the end user's corporate network.

Edge POPs are connected to each other, and to the main data centers, via a network backbone (Figure 4.4). This backbone network is organized hierarchically; for example, several POPs in a metropolitan area are linked together via a metro network, and that network

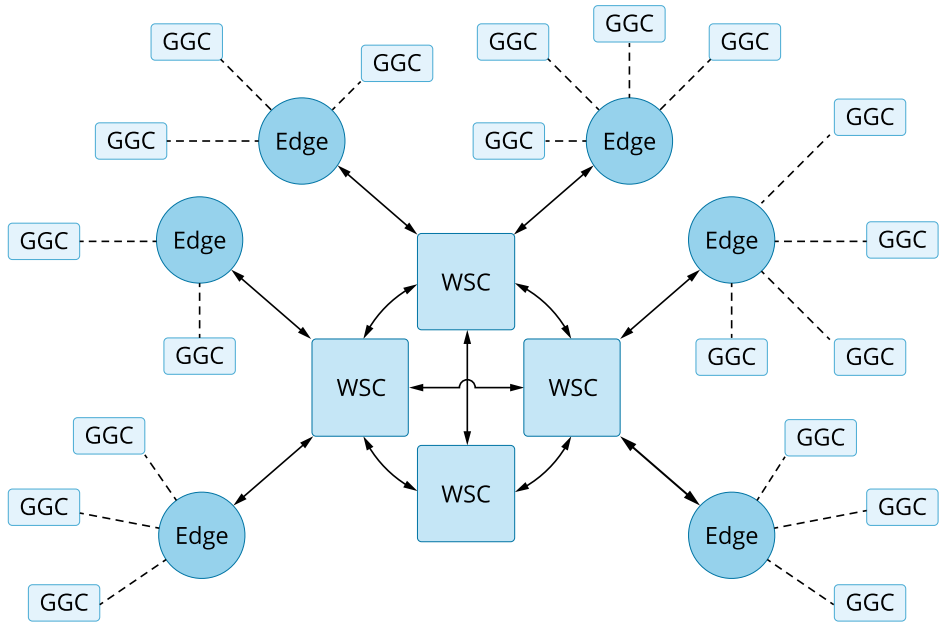


Fig. 4.4 Structure of Google's global network

connects to the regional backbone in one or more locations, and the regional backbone in turn connects to the global backbone (Figure 4.5). In many backbone networks, the majority of traffic is between data centers and not between the data centers and users, because of data replication between zones and regions.

As discussed in Chapter 2, regions have multiple zones located on multiple campuses, and thus the network backbone connects regions, not data centers directly. The regional backbone network connects all campuses in a region; typically, these are only a few tens of kilometers apart and thus the regional backbone can provide high capacity bandwidth at a low cost. Larger campuses that consist of multiple buildings additionally have a campus network that connects these buildings with even higher-speed connections at an even lower cost. Each of these networks layers (e.g., campus and region) are connected by gateways that route traffic between them. While similar in functionality, the gateways have vastly different scales: whereas a gateway consisting of a single peering router can connect multiple ISPs to the network, a gateway between a multi-petabit campus network and a multi-petabit regional network consists of dozens of chassis with thousands of ports, more closely resembling a cluster network than an internet POP.

To recap our example, the user's packet to Google will travel over their ISP's network to a Google edge location where a peering router connects the ISP and Google. If it can't be handled locally, it will then traverse Google's metro network to a backbone gateway and then to another backbone gateway near the data center serving the request. There it will

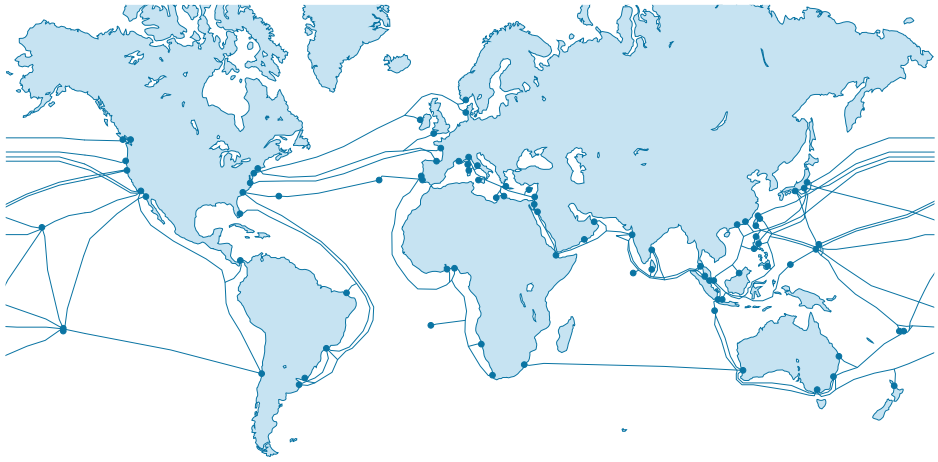


Fig. 4.5 Google's global submarine network

continue on the destination's metro network to a campus network gateway and from there to the building hosting the service, where another gateway routes it to the cluster network holding the actual servers.

4.3 Infrastructure management

A WSC cluster contains millions of physical parts, tens of millions of compute cores, and at least tens of thousands of binaries. Managing resources at this scale becomes an art in itself. Today, many practices that reduce the toil of managing even small deployments are derived from practices originally developed for WSCs.

4.3.1 Resource provisioning

Most developers using the WSCs behind public clouds never have to deal with the physical infrastructure underlying the VMs, containers, or storage they're using. Provisioning a new resource takes just an API call, and the resource is available seconds later. Behind the scenes, this illusion of always-available capacity meets the reality of the physical world. How do we ensure that free resources are available whenever a customer requests them?

First, we need a planning system to calculate the current demand (what we should order today) and also forecast future demand (what we'll need in a few months to a few years). Note that "current" demand doesn't refer to what we need right now, it refers to what we

need to order right now, i.e., demand that is at lead time for its supply. For example, if it takes a week to order, ship, and install a server, then we need to know today what our demand will be a week from now.

Thus, lead time is a central metric for provisioning: demand that materializes within lead time can't usually be fulfilled, so shorter lead times are better. For some resources like individual servers, realizing lead times of a few days is possible within limits (demand for one server is very different from demand for 1000 servers!). For others, supply chains can be much longer. For example, a data center takes on the order of a year to build, and some components like utility-scale transformers take more than a year from order to delivery. Even small components like chips can have long lead times: as of 2024, chips manufactured in TSMC's 5nm process take about 5 months to manufacture!

One way to shorten or hide lead times is to order ahead and keep a buffer of inventory somewhere. For example, by keeping a quarter's worth of 5 nm chips in a buffer, unanticipated demand can be fulfilled much more quickly than in 5 months. But the opposite is true as well: we now have to determine at least three months ahead of time that demand for this chip is ending, so the buffer can be drained. And even if that prediction is perfect, the buffer might be costly: every dollar spent buying inventory that's held in reserve is a dollar spent on something just sitting on a shelf, hardly an effective use of money. Thus, buffering, while necessary, isn't a complete solution.

The output of the planning process then becomes the input to fulfillment. In the supply chain world, fulfillment is defined as the process of receiving, preparing, and delivering orders. For a WSC supply chain, this may include ordering components (typically the "A" parts, i.e., the most expensive parts) and assigning them to manufacturers (who buy B and C parts), booking manufacturing and transportation capacity, planning deployment (are there enough technicians available in this location next Tuesday?), etc.

In WSCs, supply may not come from new purchases only. For example, maybe a big user just moved out of the same (or a nearby) data center and its machines can be used to fulfill new demand. Or maybe a service in the location A can be moved to B which has idle resources, freeing up resources for new demand in A. Or maybe demand can be reshaped: a job requesting 100 GPUs of type A could run equally well on 150 GPUs of type B, avoiding an additional purchase of type A GPUs. Generally speaking, the larger the pools across which capacity can be shared, and the more flexible the capacity demand is in terms of location and specific hardware, the lower the total cost of fulfillment will be. Since the cost difference between good and great fulfillment may be high, planning and fulfillment have evolved into highly complex, multidimensional optimization problems. Or, as someone once put it, "planning and fulfillment are easy, until you try to do them yourself".

4.3.2 Configuration and observability

The size and complexity of both the workloads and the hardware infrastructure make the management and observability framework a fundamental component of any WSC deployments, and thus we describe them here in more detail.

4.3.2.1 Configuration management

Historically, application configuration was more of an afterthought than a discipline. Binaries received their configuration via command-line flags, shell environment variables, and maybe some configuration files. Modern software development has changed that, leading to configuration management as an integral part of software engineering [34].

Configuration management helps engineering teams build robust and stable systems with tools that automatically manage configuration data. Configuration data may include low-level machine data (e.g., a specific machine type with N cores and M GB of RAM), application parameters, credentials like API keys, names or endpoints denoting services needed by the application, and so on.

Configuration management aims to store all of this data in a version-controlled system that allows for a review of changes, rollbacks of bad configurations, and historical documentation ("what settings did we use last year?"). Often, such systems are implemented on top of a source control system since they were built for a very similar purpose. That's why this approach often is called "configuration as code": treat your config files like source files.

"Infrastructure as code" (IAC) goes one step further to suggest that setting up or deleting a service instance should be fully automated. This approach requires configuration as code but additionally includes all the steps to create VM or container images, start the respective VMs or containers, set up their networking and storage, and so on. Examples of IAC tools include Ansible [35], Chef [36], Puppet [37], and Terraform [38].

4.3.2.2 Observability

Beyond monitoring (i.e., collecting and reporting on metrics that define system health), observability requires systems that make sense of monitoring data to find the root cause of issues. While monitoring collects data on individual components, observability looks at the distributed system as a whole.

System operators must keep track of how well an internet service is meeting its service level indicators (SLIs). An SLI measures a particular service aspect such as the percentage of time a particular API call C is executed successfully. A service level objective (SLO) states the target SLI the service should reach ("C will successfully execute 99.9% of the time"). Finally, a service level agreement (SLA) is a legal agreement specifying the SLO and any penalties payable if the SLO is not reached ("if we fail to meet this SLO you may cancel your contract without penalty").

Monitoring information must be very fresh so that an operator (or an automated system) can take corrective actions quickly and avoid significant disruption—ideally within seconds, not minutes. Sometimes, the most critical information needed is restricted to just a few signals that can be collected from the front-end servers, such as latency and throughput statistics for user requests. In its simplest form, such a monitoring system can simply be a script that polls all front-end servers every few seconds for the appropriate signals and displays them to operators in a dashboard. Google Cloud Operations tools [39] provides similar products to monitor, troubleshoot, and operate cloud services. Load balancers are often used to automatically address certain problems within seconds, either by diverting traffic to a healthy server pool or by adding additional servers to the pool.

Large-scale services often need more sophisticated and scalable monitoring support, as the number of front-ends can be quite large, and more signals are needed to characterize the health of the service. For example, it may be important to collect not only the signals themselves but also their derivatives over time. The system may also need to monitor other business-specific parameters in addition to latency and throughput ("are any purchase button clicks failing?"). Monitoring systems typically support an expression language that lets operators create derived parameters based on baseline signals being monitored. Finally, the system generates automatic alerts to on-call operators depending on monitored values and thresholds. Fine tuning a system of alerts (or alarms) can be tricky because alarms that trigger too often because of false positives will cause operators to ignore real ones, while alarms that trigger only in extreme cases might get the operator's attention too late to allow smooth resolution of the underlying issues. The SRE book [40] contains an in-depth discussion of monitoring and alerting.

WSC developers also need application and crash logs to determine the source of problems. In traditional (single-machine) applications, these are stored on the local file system, but in WSCs they need to be collected and aggregated in a central repository so that a developer can view application logs across all instances of the service. Typically, such files are written to local storage first, and a log collection process copies them to the central repository where they are kept for a certain period of time. Often, this process also compresses logs and "sanitizes" them, removing sensitive data such as encryption keys that may be present in crash dumps.

4.3.2.3 Tracing tools

Although service-level dashboards help operators quickly identify service-level problems, they typically lack the detailed information required to know why a service is slow or otherwise not meeting requirements. Both operators and the service designers need tools to help them understand the complex interactions between many programs, possibly running on hundreds of servers, so they can determine the root cause of performance anomalies and identify bottlenecks.

Distributed system tracing tools address this need, and form an important step in moving from monitoring to observability. These tools attempt to determine all the work done in a distributed system on behalf of a given initiator (such as a user request) and detail the causal or temporal relationships among the various components involved. For example, when a user clicks the "Buy" button, the application front-end may transitively invoke hundreds of microservices in the process of completing the purchase. So if it fails, why does it fail? Tracing tools derive their name from their purpose: they aim to trace a request through all its subrequests until it is finished, producing a full record of all the other services involved, and how well they performed.

Tracing tools fall into two broad categories: black-box monitoring systems and application/middleware instrumentation systems. WAP5 [41], and the Sherlock system [42] are examples of black-box monitoring tools. Their approach consists of observing networking traffic among system components and inferring causal relationships through statistical inference methods. Because they treat all system components (except the networking interfaces) as black boxes, these approaches have the advantage of working with no knowledge of, or assistance from, applications or software infrastructure components. However, this approach inherently sacrifices information accuracy because all relationships must be statistically inferred. Collecting and analyzing more messaging data can improve accuracy but at the expense of higher monitoring overheads.

Instrumentation-based tracing schemes pioneered in academic systems such as Pip [43], Magpie [44], and X-Trace [45], take advantage of the ability to explicitly modify applications or middleware libraries for passing tracing information across machines and across module boundaries within machines. The annotated modules typically also log tracing information to local disks for subsequent collection by an external performance analysis program. These systems can be very accurate as there is no need for inference, but they require all components of the distributed system to be instrumented to collect comprehensive data. The Dapper system [13] is an annotation-based tracing tool that remains effectively transparent to application-level software by instrumenting a few key modules that are commonly linked with all applications, such as messaging, control flow, and threading libraries. GCP's Cloud Trace [46] is a publicly available implementation of the Dapper system, accepting traces from any OpenTelemetry compatible source.

So far we've considered tracing at the level of RPCs, but the concept can be applied to function calls as well. For example, XRay, a feature of the LLVM compiler, uses compiler instrumentation to add trace points at every function entry and exit, enabling extremely fine latency detail when active and very low overhead when inactive. Other tools can provide even deeper introspection, for example, allowing users to dynamically add logging to their programs without recompiling or redeploying.

4.3.2.4 Performance tools

Tracing tools can be very helpful for performance optimization as well. Since traces are annotated with timing information, we can get a good understanding of the factors contributing to the latency of an important service, like a Buy button click. Moreover, when traces are collected all the time, they can be very helpful in understanding the performance of a service under various loads, or understanding why the performance of a service instance in region A differs from that of region B.

CPU profilers based on sampling of hardware performance counters have been incredibly successful in helping programmers understand microarchitecture performance phenomena. Since WSC applications run on many machines, such tools need to profile multiple machines. Google-Wide Profiling (GWP) [47]] selects a random subset of machines to collect short whole machine and per-process profile data, and combined with a repository of symbolic information for all Google binaries produces cluster-wide view of profile data. GWP answers questions such as: which is the most frequently executed procedure at Google, or which programs are the largest users of memory? The Google Cloud Operations tools product [39] is inspired by GWP.

Performance tools don't just observe, they can also actively optimize. For example, profile-guided optimization [48] uses profiling data from production uses to optimize future versions of the same binary. Optimized memory allocators reduce cache and TLB misses [49]. Progress in this space is particularly challenging since the true performance impacts of optimizations only become visible at scale, and might be very different from benefits in smaller, experimental setups [50]. For example, heavily inlining or specializing an allocator may provide significant speedups in smaller benchmarks but may actually slow down production binaries because the code size growth from millions of inlined allocators may induce too many instruction cache misses.

4.3.3 Practices: DevOps and SRE

WSCs, and later public clouds, required a new approach not just to the design and implementation of applications but also to the way they are run. The two most popular new approaches are DevOps (a portmanteau of Development and Operations) and SRE (Site Reliability Engineering).

DevOps unifies development and operations into a single team: developers no longer hand off a finished product to an operations team that then runs and maintains it, and there's no formal distinction anymore between software engineers and operations engineers. The goal of DevOps is to shorten the time from committing new code to that code running in production by making the same team responsible for both, allowing them to balance the needs of both sides. For example, if there aren't enough application-level metrics or logs, debugging will become difficult, and since the same engineers are dealing with problems or outages in production, they are more likely to build applications with high observability.

A possible argument against DevOps is that the two professions require skill sets that are too different: great software engineers may not be great at reliability, or operational rigor, or capacity planning. SRE thus aims to solve the same problem in a different way: there still are two teams, but the "operations" team consists of software engineers specializing in reliability and other operational attributes (like scalability, efficiency, security etc).

Chapter 1 of the SRE book [40] contains an excellent overview of the principles of SRE, and they can be summarized in this simple sentence: SRE is what happens when you ask a software engineer to design an operations team. SRE has a durable focus on engineering (e.g., automating solutions or solving root causes, vs addressing symptoms) and pursues maximum change velocity while achieving the service's SLO. In other words, the goal is not to achieve 100% uptime, because that could encourage SRE to push back on changes, which are the number one contributor to outages. Instead, the SRE philosophy asks to "spend your downtime budget well" (and stay within that budget, i.e., achieve the SLO). If there is an outage, it should be caused by something new, and not by a "simple mistake". For example, if the outage was caused by a corrupted configuration file, that's a sign that the service isn't validating its configuration properly.

Operating at the scale of WSCs also requires a strong culture of seeing failures as learning opportunities. At scale, in a system with millions of cores and thousands of software engineers, something *will* go wrong. Even with the best team, the best technology, the best practices. Thus, the goal is to create a system that can be highly reliable despite imperfect hardware, software, and people. One key mechanism for reaching high levels of reliability is the blameless postmortem [40].

> Blameless postmortems are a tenet of SRE culture. For a postmortem to be truly blameless, it must focus on identifying the contributing causes of the incident without indicting any individual or team for bad or inappropriate behavior. A blamelessly written postmortem assumes that everyone involved in an incident had good intentions and did the right thing with the information they had. If a culture of finger pointing and shaming individuals or teams for doing the "wrong" thing prevails, people will not bring issues to light for fear of punishment (see Chapter 15 of [40] for more details).

While this book primarily deals with hardware and software topics, creating and maintaining the right engineering *culture* often is even more difficult and also more important! We will talk about this more in Chapter 11 when we look back at twenty-five years of WSCs.

4.4 Server-level software

4.4.1 OS and firmware

The basic software system image running in WSC server nodes isn't much different than what one would expect on a regular enterprise server platform. However, firmware, device drivers, or operating system modules in WSC servers can be simplified to a larger degree

than in a general purpose enterprise server. Given the higher degree of homogeneity in the hardware configurations of WSC servers, we can streamline firmware and device driver development and testing since fewer combinations of devices will exist. In addition, a WSC server is deployed in a relatively well known environment, leading to possible optimizations for increased performance. For example, the majority of the networking connections from a WSC server will be to other machines within the same building, and incur lower packet losses than in long-distance internet connections. Thus we can tune transport or messaging parameters (timeouts, window sizes, and so on) for higher communication efficiency.

WSC servers place a high emphasis on security, including the integrity of the software running on them (we'll discuss security further in Chapter 10). To ensure that a server has not been tampered with, it uses a silicon root of trust (RoT), a small, specialized security chip that controls the boot process and any updates to firmware. The RoT is completely independent of the CPU, so that even an attacker having full control (root) on the server cannot bypass the security measures.

The RoT helps ensure that the hardware infrastructure and the software that runs on it remain in their intended, trustworthy state by verifying that a server or a device boots with the correct firmware and hasn't been infected by a low-level malware. It can also provide a cryptographically unique machine identity, so an operator can verify that a server or a device is legitimate. Additionally, it protects secrets like encryption keys in a tamper-resistant way even for people with physical access (e.g., while a server or a device is being shipped). Last but not least, it provides authoritative, tamper-evident audit records and other runtime security services.

Silicon RoTs are common in cloud providers; examples include Google's Titan chip [51], or Amazon's Nitro [52], and the open source efforts OpenTitan [53], and Caliptra [54].

4.4.2 Containerization and virtualization

Containerization and virtualization are two distinct yet complementary technologies for deploying and managing applications that are isolated from each other.

Containerization is a lightweight form of operating system virtualization that allows multiple isolated user-space instances to run on a shared operating system kernel. Each container encapsulates an application and its dependencies, providing a consistent and portable runtime environment. This isolation ensures that applications running within different containers do not interfere with each other, simplifying dependency management and enhancing security. Containers are efficient because they share the underlying OS kernel, resulting in fast startup times and reduced resource consumption compared to traditional virtual machines.

Virtualization, on the other hand, emulates an entire computer system inside a virtual machine (VM), including the hardware, operating system, and applications. This emulation is achieved through a hypervisor, a software layer that sits between the virtual machines and the physical hardware. Each virtual machine operates independently, with its own dedicated

resources and its own "guest" kernel. Virtualization enables greater flexibility by allowing multiple operating systems and applications to run on a single physical server.

Both technologies play important roles in WSC infrastructures, enabling efficient resource utilization by scheduling multiple workloads onto the same server, and enhanced application management by isolating these workloads from each other. While this isolation is fairly strong, it is not perfect because separate workloads can interfere with each other's performance (the "noisy neighbor problem"). For example, if one workload saturates DRAM bandwidth, or displaces many L3 cache lines because it has poor locality, other workloads will see worse performance. Newer CPU architectures may include hardware features specifically designed for performance isolation. For example, Intel's Cache Allocation Technology (CAT) allows fine-grained control over cache allocation on a per-core basis, while Resource Director Technology (RDT) allows monitoring and controlling shared resources like cache and memory bandwidth [55].

Additionally, noisy-neighbor effects can create security problems via side-channel attacks where one process is able to reconstruct ("read") information from another process even though it cannot directly access its memory or registers. The first high-impact vulnerabilities of this kind (Spectre and Meltdown [56]) were discovered by researchers at Google and the University of Graz in 2017 and many similar vulnerabilities have been found since then. We will discuss hardware security vulnerabilities in more detail in Chapter 10.

References

1. A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2015, pp. 1–17.
2. M. Tirmazi, A. Barker, N. Deng, et al., "Borg: The next generation," in *Proceedings of the 15th European Conference on Computer Systems*, ser. EuroSys '20, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2020, pp. 1–14.
3. Kubernetes Authors, *Kubernetes*, https://www.kubernetes.io.
4. Google Cloud, *Cloud Run - Serverless Container Platform*.
5. S. Li, X. Wang, F. Kalim, et al., "Thunderbolt: Throughput-optimized, quality-of-service-aware power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI20)*, 2020, pp. 1241–1255.
6. A. Radovanovic, *Our data centers now work harder when the sun shines and wind blows*, The Keyword, https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows/, 2020.
7. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the nineteenth ACM Symposium on Operating systems principles*, ser. SOSP '03, New York, NY, USA: Association for Computing Machinery, Oct. 19, 2003, pp. 29–43.
8. D. Serenyi, "Cluster-level storage @Google," in *Keynote at the 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Intensive Computing Systems*, 2017.
9. G. DeCandia, D. Hastorun, M. Jampani, et al., "Dynamo: Amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles*,

ser. SOSP '07, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2007, pp. 205–220.
10. M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 335–350.
11. M. Isard, "Autopilot: Automatic data center management," 2, vol. 41, New York, NY, USA: Association for Computing Machinery, Apr. 1, 2007, pp. 60–67.
12. E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, ser. FAST '07, San Jose, CA: USENIX Association, 2007.
13. B. H. Sigelman, L. A. Barroso, M. Burrows, et al., "Dapper, a large-scale distributed systems tracing infrastructure," 2010.
14. OpenTelemetry Authors, *OpenTelemetry*, https://opentelemetry.io/.
15. M. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-forward: A discussion between Kirk McKusick and Sean Quinlan about the origin and evolution of the Google File System," 7, vol. 7, New York, NY, USA: Association for Computing Machinery, Aug. 1, 2009, pp. 10–20.
16. L. Lamport, "The part-time parliament," 2, vol. 16, New York, NY, USA: Association for Computing Machinery, May 1, 1998, pp. 133–169.
17. D. Ford, F. Labelle, F. I. Popovici, et al., "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10, USA: USENIX Association, Oct. 4, 2010, pp. 61–74.
18. Google Cloud, *Cloud Storage - Object Storage*, https://cloud.google.com/storage.
19. F. Chang, J. Dean, S. Ghemawat, et al., "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 205–218.
20. J. Baker, C. Bond, J. C. Corbett, et al., "Megastore: Providing scalable, highly available storage for interactive services.," in *CIDR*, vol. 11, 2011, pp. 223–234.
21. J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Google's globally-distributed database," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12, USA: USENIX Association, Oct. 8, 2012, pp. 251–264.
22. Google Cloud, *TrueTime and external consistency*, Cloud Spanner Documentation: https://cloud.google.com/spanner/docs/true-time-external-consistency.
23. E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29,2012.
24. E. Brewer, Spanner, *TrueTime and the CAP theorem*, https://research.google/pubs/spanner-truetime-and-the-cap-theorem/, 2017.
25. J. Shute, R. Vingralek, B. Samwel, et al., "F1: A distributed SQL database that scales," 11, vol. 6, VLDB Endowment, Aug. 1, 2013, pp. 1068–1079.
26. memcached team, *memcached - a distributed memory object caching system*, https://memcached.org/about.
27. J. Ousterhout, P. Agrawal, D. Erickson, et al., "The case for RAMClouds: scalable high-performance storage entirely in DRAM," 4, vol. 43, New York, NY, USA: Association for Computing Machinery, Jan. 27, 2010, pp. 92–105.
28. D. G. Andersen, J. Franklin, M. Kaminsky, et al., "Fawn: A fast array of wimpy nodes," 7, vol. 54, New York, NY, USA: Association for Computing Machinery, Jul. 1, 2011, pp. 101–109.
29. Redis, *Redis*, https://redis.io.
30. Cloudflare, *What is anycast DNS?* Cloudflare Learning Center: https://www.cloudflare.com/learning/dns/what-is-anycast-dns/.
31. J. Iyengar, M. Thomson, et al., "QUIC: A UDP-based multiplexed and secure transport," in *RFC 9000*, Internet Engineering Task Force (IETF) Fremont, CA, USA, 2021.

32. Google, *Google Peering Infrastructure Map*, https://peering.google.com/#/infrastructure.
33. Google Cloud, *Cloud CDN Locations*, https://cloud.google.com/vpc/docs/edge-locations.
34. Atlassian, *Configuration management in microservices*, https://www.atlassian.com/microservices/microservices-architecture/configuration-management.
35. Red Hat, *Ansible*, Website: https://ansible.io.
36. *Chef*, Website: https://chef.io.
37. *Puppet*, Website: https://puppet.com.
38. HashiCorp, *Terraform by HashiCorp*.
39. Google Cloud, *Cloud Operations Suite (formerly Stackdriver)*, https://cloud.google.com/products/observability.
40. B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, *The Site Reliability workbook: practical ways to implement SRE.* " O'Reilly Media, Inc.", 2018.
41. P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, "WAP5: Black-box performance debugging for wide-area systems," in *Proceedings of the 15th International Conference on World Wide Web*, ser. WWW '06, New York, NY, USA: Association for Computing Machinery, May 23, 2006, pp. 347–356.
42. P. Bahl, R. Chandra, A. Greenberg, et al., "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2007, pp. 13–24.
43. P. Reynolds, C. E. Killian, J. L. Wiener, et al., "Pip: Detecting the unexpected in distributed systems.," in *NSDI*, vol. 6, 2006, pp. 9–9.
44. P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modelling and performance-aware systems," in *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.
45. R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-Trace: A pervasive network tracing framework," in *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, 2007.
46. Google Cloud, *Cloud Trace API Reference*, https://cloud.google.com/trace/docs/reference.
47. G. Ren, E. Tune, T. Moseley, et al., "Google-Wide Profiling: A continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, 2010.
48. H. Shen, K. Pszeniczny, R. Lavaee, et al., "Propeller: A profile guided, relinking optimizer for warehouse-scale applications," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023, New York, NY, USA: Association for Computing Machinery, Jan. 30, 2023, pp. 617–631.
49. Z. Zhou, V. Gogte, N. Vaish, et al., "Characterizing a memory allocator at warehouse scale," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24, vol. 3, New York, NY, USA: Association for Computing Machinery, Apr. 27, 2024, pp. 192–206.
50. A. H. Hunter, C. Kennelly, P. Turner, et al., "Beyond malloc efficiency to fleet efficiency: A hugepage-aware memory allocator," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 257–273.
51. U. Savagaonkar, N. Porter, N. Taha, B. Serebrin, and N. Mueller, "Titan in depth: Security in plaintext," *Google Cloud Identity and Security Blog*, 2017.
52. Amazon Web Services, "Security Design of the AWS Nitro System," Tech. Rep.
53. OpenTitan, *OpenTitan - Open Source Silicon Root of Trust*, https://opentitan.org/.
54. Chips Alliance, *Caliptra - Hardware Root of Trust*, GitHub Repository: https://github.com/chipsalliance/Caliptra.

55. Intel, *Intel Resource Director Technology (Intel RDT)*, https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.
56. *Spectre Attack*, https://spectreattack.com/, 2018.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

WSC Hardware: Data Center Infrastructure

5

5.1 Data center infrastructure basics

5.1.1 Planet to campuses to buildings

Internet and cloud services run on a planet-scale computer with workloads distributed across multiple data center buildings around the world. These data centers are designed to house computing, storage, and networking infrastructure. The main function of the buildings is to deliver the utilities needed by equipment and personnel there: power, cooling, shelter, and security. By classic physics definitions, there is little work produced at the data center. Other than some departing photons, all of the energy consumed is converted into heat. Delivering input energy and subsequently removing waste heat are at the heart of the data center's design and drive the vast majority of non-computing costs. Data center construction costs are roughly proportional to the amount of power delivered and typically run in the range of $5–20 per watt (see Chapter 8), varying considerably depending on size, location, and design.

5.1.2 Basic structure of a data center

Data center sizes vary widely and are commonly described in terms of critical power, the total power that can be continuously supplied to IT equipment. Traditional enterprise servers were housed in data centers smaller than 5,000 sq ft (450 square meters) and with less than 1 MW of critical power [1], [1], [2]. Commercial data centers built to host servers from multiple companies (often called colocation data centers, or "colos") are larger, and can support a critical load of tens of megawatts. Data centers of large cloud providers are similar, though often larger than colos. Sometimes, many data centers are grouped together in a single

© The Author(s) 2026
L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_5

Fig. 5.1 Different Google data centers

campus. Most data centers are single story, while a few are multi-story (Figure 5.1). The critical power of WSC data center campuses are often above 100 MW.

A data center building has multiple components. A mechanical yard (or, alternatively, a central utility building) hosts the cooling systems, such as cooling towers and chillers. An electrical yard houses electrical equipment, such as generators and power distribution centers. Within the data center, the main server hall hosts the compute, storage, and networking equipment organized into hot aisles and cold aisles (and in some data centers, hot air containment structures). The server floor can also host repair areas for operations engineers. Most data centers also have separate areas designated for networking, including inter-cluster, campus-level, and long-haul connectivity and a separate facility management network. Given the criticality of networking for data center availability, networking areas often have additional physical security and high-availability features to ensure increased reliability.

Data center building construction follows established codes around fire-resistive and non-combustible construction, safety, and so on [3], and the design usually incorporates elaborate security for access, including circle locks, metal detectors, guard personnel, and an extensive network of cameras. A nice video tour on YouTube [4] shows a real-world facility security design.

Figure 5.2 shows an aerial view of a Google data center campus in Council Bluffs, Iowa. Figure 5.3 zooms in on a single building to highlight some of the typical components in greater detail.

Figure 5.4 shows the components of a typical data center architecture. Beyond the IT equipment (discussed in Chapter 6), the two major systems in the data center provide power delivery (shown in red) and cooling (shown in green). We discuss these in detail in Section 5.2.

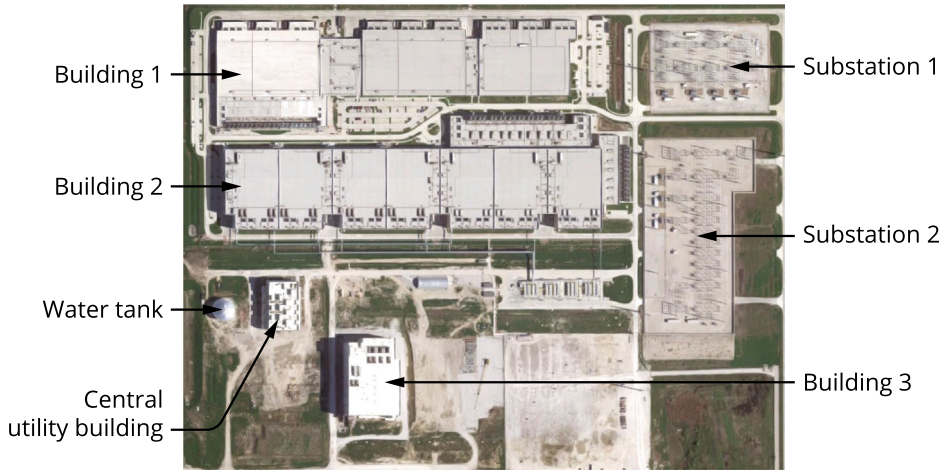


Fig. 5.2 Aerial view of a Google data center campus

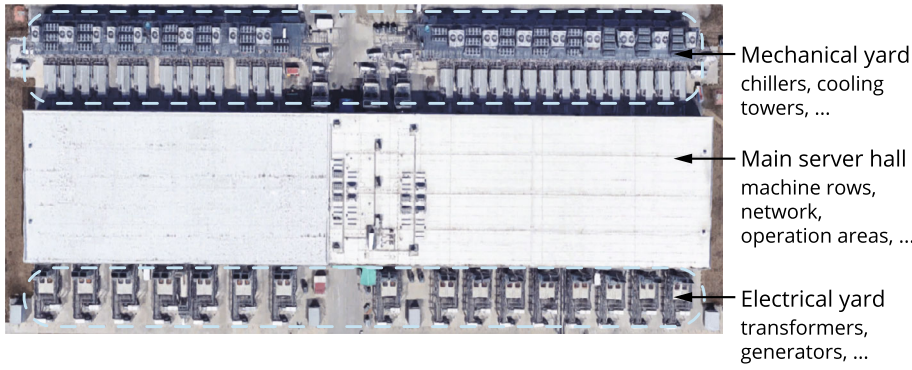


Fig. 5.3 A Google data center building

5.1.3 Data center taxonomy: Tiers

The design of a data center is often classified using a system of four tiers [5]. The Uptime Institute, a professional services organization specializing in data centers, advocates a 4-tier classification loosely based on the power distribution, uninterruptible power supply (UPS), cooling delivery, and redundancy of the data center [6].

- Tier I data centers have a single path for power distribution, uninterruptible power supplies (UPS), and cooling distribution, without redundant components.
- Tier II data centers add redundant components to this design (N + 1), improving availability.

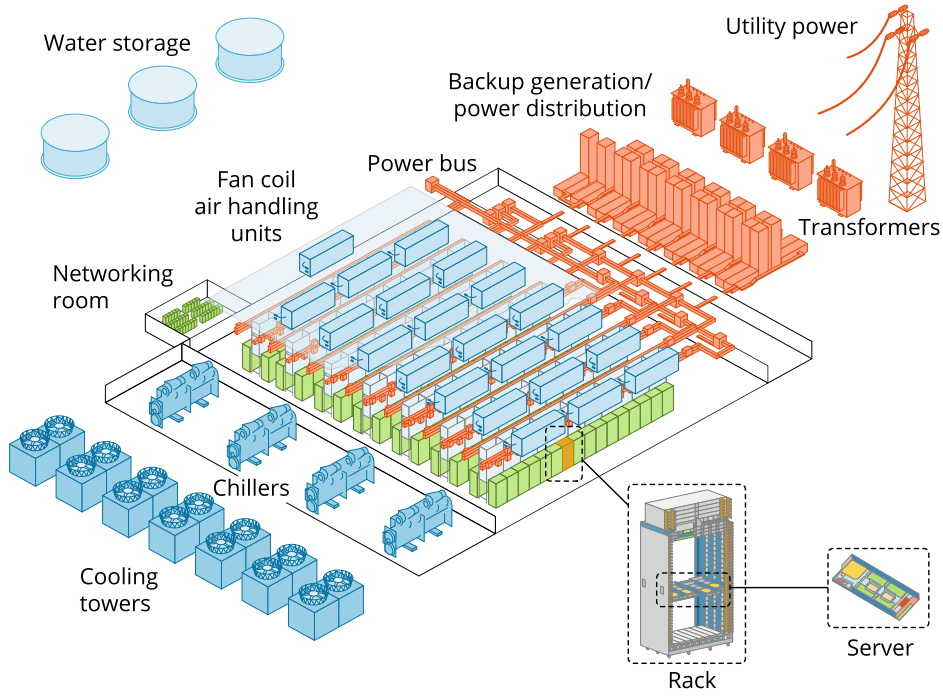


Fig. 5.4 The main components of a typical data center

- Tier III data centers have one active and one alternate distribution path for utilities. Each path has redundant components and is concurrently maintainable. Together they provide redundancy that allows planned maintenance without downtime.
- Tier IV data centers have two simultaneously active power and cooling distribution paths, redundant components in each path, and are supposed to tolerate any single equipment failure without impacting the load.

The Uptime Institute's specification focuses on data center performance at a high level. The specification implies topology rather than prescribing a specific list of components to meet the requirements. Notable exceptions are the amount of backup diesel fuel and water storage, and ASHRAE temperature design points [7]. Within the Uptime standards, there are many architectures that can achieve a given tier classification. In contrast, the TIA-942 standard is more prescriptive and specifies a variety of implementation details, such as building construction, ceiling height, voltage levels, types of racks, and patch cord labeling.

Formally seeking tier classification is labor intensive and requires a full review from one of the certifying bodies. For this reason most data centers are not formally rated. Most commercial data centers fall somewhere between tiers III and IV, choosing a balance between

construction cost and reliability. Generally, the lowest individual subsystem rating (cooling, power, and so on) determines the overall tier classification of the data center.

Real-world data center reliability is strongly influenced by the quality of the organization running the data center, not just the design. Theoretical availability estimates used in the industry range from 99.7% for tier II data centers to 99.98% and 99.995% for tiers III and IV, respectively [6]. However, real-world reliability often is dominated by factors not included in these calculations; for example, the Uptime Institute reports that over 70% of data center outages are the result of human error, including management decisions on staffing, maintenance, and training [6]. Furthermore, in an environment using continuous integration and delivery of software, software-induced outages dominate building outages. There are also application-specific considerations. For example, recall our discussions in Chapter 3 where we discussed how hyperscale workloads differ from emerging cloud and machine learning workloads in tradeoffs around reliability and uptime.

Section 5.4.2 also discusses another view of how we can think about data center buildings in the context of "envelopes," depending on the level of readiness of the infrastructure to house WSC equipment.

5.1.4 Cloud and AI implications for data center design

Traditionally, the key design philosophy underpinning WSC data center design embraced economies of scale, and targeted designs customized to hyperscale workloads and systems. But as discussed in Chapter 3, today's WSCs show a lot more diversity via Cloud and AI/ML workloads.

ML workloads in particular have posed interesting new challenges for data center design by greatly increasing power density. As an illustration, Figure 5.5 below shows two power density distribution curves. The curve on the left shows a mix of traditional compute and storage systems, while the curve on the right shows the impact of introducing machine learning racks. The right shift in the curves indicates the increased power densities of accelerator systems. Compared to 10-30kW/racks for traditional servers, accelerator systems can range from 50-200kW/rack, trending even higher in future generations. The bimodality of lower-density traditional servers and higher-power-density for emerging accelerators motivate new design tradeoffs to support this wider range.

ML workloads also have diverging availability and uptime requirements relative to cloud and hyperscale workloads. As discussed in Chapter 3, these workloads are bulk-synchronous and more throughput oriented and have mechanisms for checkpoint-restarting and customer SLOs that allow them to be more tolerant to sporadic infrastructure downtime. Consequently, compared to enterprise workloads on cloud that often need Tier-3 or higher data center requirements, some ML workloads (especially training workloads) can potentially use Tier-1-like data centers. As discussed earlier, the range of costs across these tiers is fairly significant, leading to new tradeoffs in data center design.

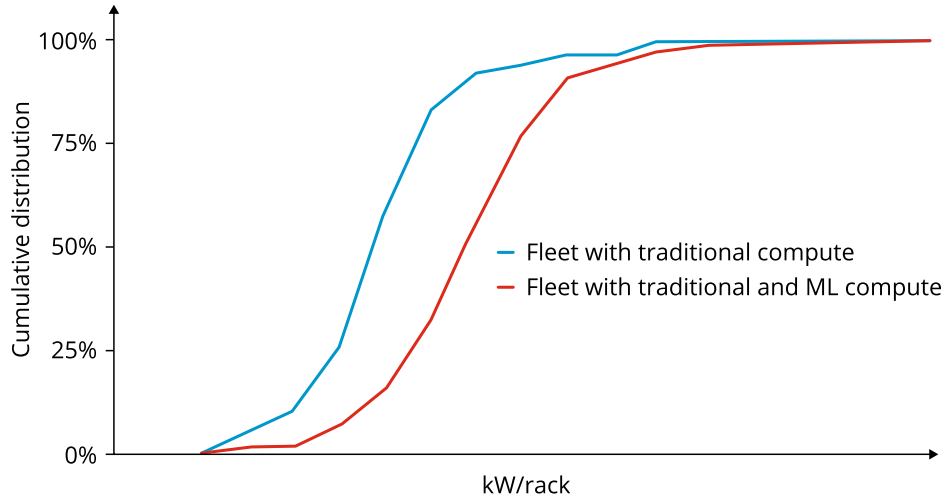


Fig. 5.5 Power density distributions with and without accelerators

Cloud data center deployments bring different considerations. They introduce more geo-diversity, matched to the wide diversity of customer requirements, and also impose more stringent requirements on regional and zonal isolation. Additionally, relative to the large-scale deployments that hyperscale and ML workloads prefer, cloud deployments prefer a "pay as you go" model especially for growth in new locations where data center capacity is more incrementally deployed in smaller chunks, and thus focus in on velocity of deployment over pure cost efficiency at scale.

We will discuss how these considerations change the design tradeoffs for WSC data centers below as we deep dive into the key elements of the data center – the power subsystems and the cooling subsystems.

5.2 Data center power systems

5.2.1 Overall electrical power delivery architecture

Power enters first at a utility substation which transforms high voltage (typically 110 kV and above) to medium voltage (typically less than 50 kV). Medium voltage is used for site-level distribution to the primary distribution centers (also known as unit substations), which include the primary switchgear and medium-to-low voltage transformers (typically below 1000 V). From here, the power enters the building with the low-voltage lines going to the uninterruptible power supply (UPS) systems. The UPS switchgear also takes a second feed at the same voltage from a set of diesel generators that cut in when utility power fails. The outputs of the UPS are routed to data center floor Power Distribution Units (PDUs). PDUs

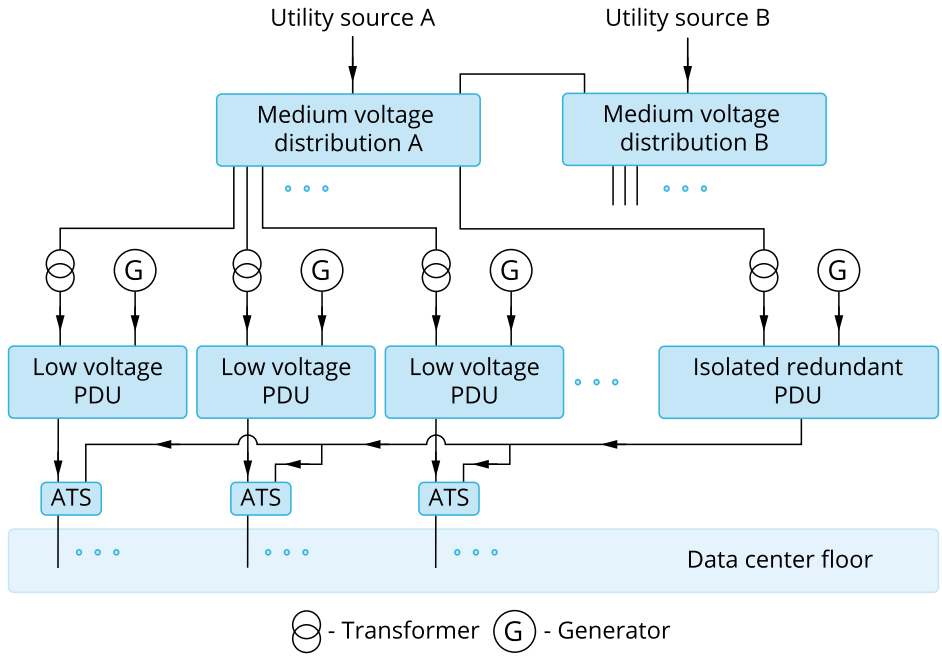


Fig. 5.6 Radial power architecture

are the last layer in the transformation and distribution architecture and route individual circuits to overhead busbars or to individual racks.

A conventional AC power distribution scheme for a large data center is shown in Figure 5.6. This topology is known as "radial" because power fans out to the entire data center floor from a pair of medium voltage buses that provide redundancy in case of loss of a utility feed. Low voltage (400-480 V) AC power is supplied to the data center floor by many PDUs, each fed by either a step-down transformer for utility power or a backup generator.

Each PDU has its own backup generator indicated by a "G." Power loss due to failures of low voltage equipment is greatly mitigated by the presence of an isolated redundant PDU, which can take the place of any other PDU if it fails or needs to be taken temporarily out of service. An automatic transfer switch (ATS) automatically switches power from the primary PDU to the backup source when the primary source fails. This $N + 1$ design is simple to implement with ATS switches; $N + M$ designs are more complicated, as we will see in the next example.

An alternative architecture for data center power distribution is Google's medium voltage power plane (Figure 5.7), which allows for sharing of power across domains as large as an entire data center [8]. High availability at the building level is provided by redundant utility AC inputs. Building-level transformers step the voltage down to a medium voltage of 11–15 kV for further distribution through the building's electrical rooms. For backup power,

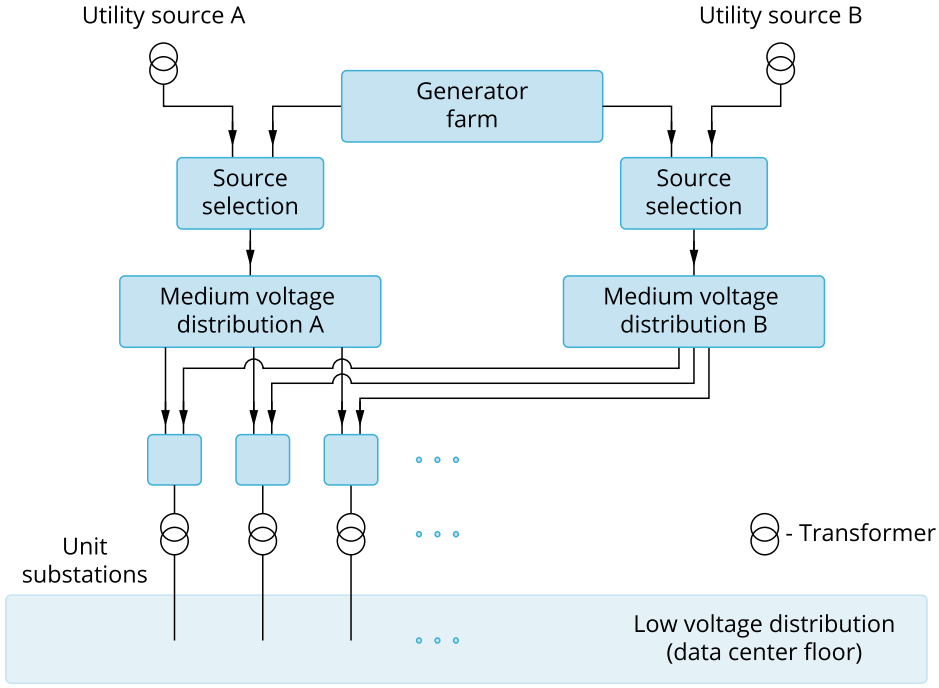


Fig. 5.7 Concept for medium voltage power plane architecture

$N + M$ medium voltage generators are paralleled to a bus, and automated systems consisting of breakers and switches select between the utility and generator sources. Redundant paths exist from both utility and generator sources to many unit substations. Each unit substation steps down the voltage to low-voltage AC for distribution to a row of racks on the data center floor.

The medium-voltage power plane architecture offers several advantages over traditional radial architectures. First, a large pool of diverse workloads can increase the opportunity for power oversubscription [9] (discussed in depth later in Chapter 9). Such power oversubscription with Google's power plane architecture significantly increases the quantity of IT equipment that can be deployed above and beyond a data center's critical power capacity. Second, the parallel generator farm offers resilience against generator failures with a minimum of redundant equipment, but at the cost of a more complex control system that can introduce new failure modes if not properly tested and maintained.

Finally, power is more fungible across the entire data center floor: with appropriate sizing of both medium and low voltage distribution components, a high dynamic range of deployment power density can be supported without stranding power. This flexibility is an important benefit given that rack power varies substantially depending on the type of IT equipment within the rack. For example, storage-intensive racks consume much less power

than compute-intensive racks. With traditional radial power architectures, a low power density in one region of the data center floor can result in permanently underused infrastructure. The medium voltage power plane enables power sharing across the floor: high power racks in one region can compensate for low power racks in another region, ensuring full utilization of the building's power capacity.

If the availability provided by the upstream power source is sufficient for an application (ML training, for example) then the row power can be used non-redundantly, effectively doubling the amount of power available. Thus, the amount of infrastructure used for backup can be as high as 50% if used in the 2N configuration, or as low as 0% or if used in the power optimized configuration.

Another popular configuration is the distributed redundant scheme, frequently used in colocation environments. It can be used with low or medium voltage infrastructure.

The unit substations and row level busways are arranged in an interleaved manner as shown in Figure 5.8. They provide the functionality of a (2N) system with an $(N + 1)$ approach. The example shown uses four units of infrastructure to create three units of capacity, commonly abbreviated as "4/3 DR". Each (double) row of racks is fed by all four power feeds, and each rack has two redundant power inlets. If one of the power feeds fails (either at the unit transformer or a busbar breaker below it), 25% of racks will switch to their alternate power source. Thus, as long as the power feeds are normally running at no more than 75% of their load, no load interruption occurs. At the cost of having dual fed racks or dual-fed servers, this scheme provides (2N) redundancy without requiring a (2N) facility.

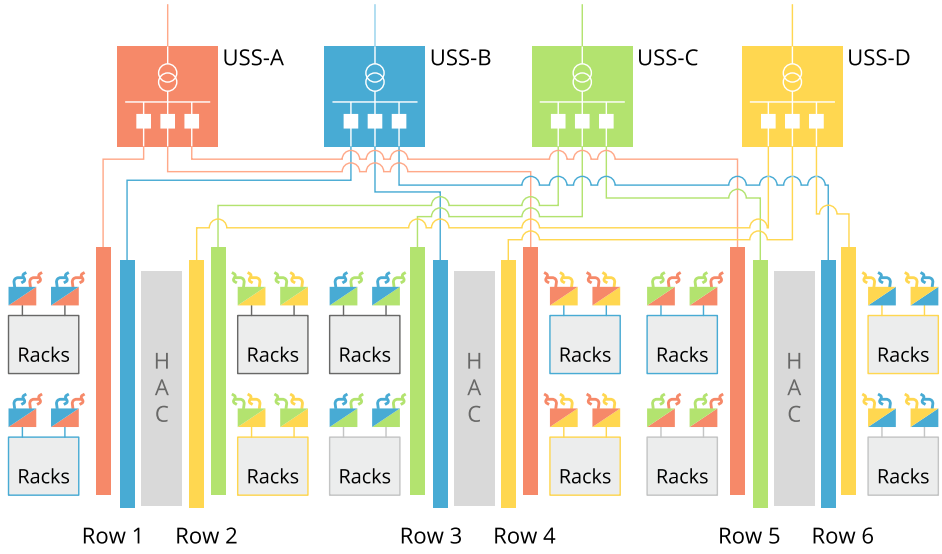


Fig. 5.8 Concept for distributed redundant (DR) architecture (HAC = hot aisle containment)

5.2.2 Comparing AC and DC power distribution architectures

High-voltage DC (HVDC) on the utility grid presents advantages for connecting incompatible power grids, providing resistance to cascading failures, controlling large scale power flows actively, and transmitting power efficiently across long-distances. In data centers, the case for DC distribution is centered around efficiency, increased reliability from reduced component counts, footprint reduction, and easier integration of distributed generators with native DC outputs. In comparison with the double-conversion UPS mentioned above, DC systems eliminate the final inversion step of the UPS. If the voltage is selected to match the DC primary stage of the server power supply unit (PSU), three additional steps are eliminated: PDU transformation, PSU rectification, and PSU power factor correction.

Figure 5.9 and Figure 5.10 compare AC and DC distribution architectures commonly used in the data center industry. State-of-the-art, commercially available efficiencies (based on [10]) are shown for each stage of the "power train." The overall powertrain efficiency using state-of-the-art components remains a few percent higher for DC distribution as compared to AC distribution; this difference was more pronounced in data centers with older components [11]. Note that the AC architecture shown corresponds to the voltage scheme commonly found in North America; in most other parts of the world the additional voltage transformation in the AC PDU can be avoided, leading to slightly higher PDU efficiency.

Commercial DC equipment for data centers is available, but costs remain higher than for comparable AC equipment. Similarly, building large data centers involves hundreds and sometimes thousands of skilled workers. While only a subset of these will be electricians, the limited availability of DC technicians may lead to increased construction, service, and operational costs. However, DC power distribution is more attractive when integrating distributed power generators such as solar photovoltaic, fuel cells, and wind turbines. These power sources typically produce native DC and are easily integrated into a DC power distribution architecture.

Emerging denser computing solutions (e.g., higher-power accelerators connected by copper interconnects) put more pressure on bus bars by increasing amperage and thermal chal-

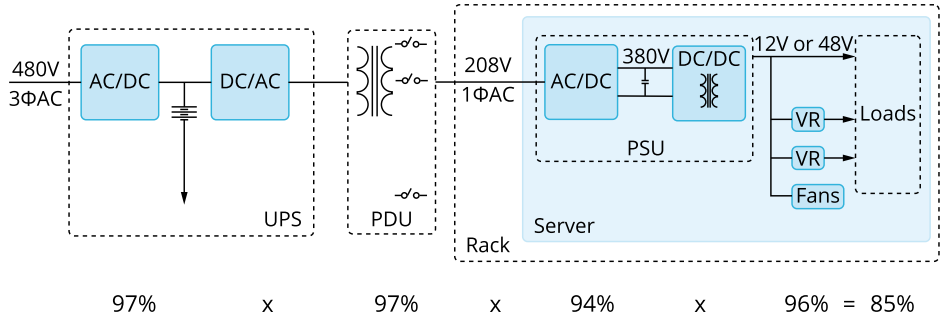


Fig. 5.9 Common AC distribution architecture

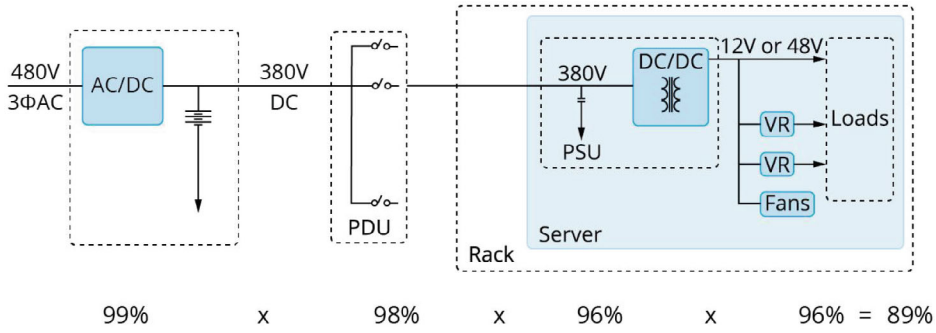


Fig. 5.10 DC distribution architecture

lenges. Since most new rack designs already consume DC power, future industry standards may employ DC bus bar architectures and higher voltages.

5.2.3 Uninterrupted power supplies (UPS)

The UPS typically combines three functions in one system.

- First, it contains a transfer switch that chooses the active power input (either utility power or generator power). After a power failure, the transfer switch senses when the generator has started and is ready to provide power; typically, a generator takes 10–15 seconds to start and assume the full rated load.
- Second, it contains some form of energy storage (electrical, chemical, or mechanical) to bridge the time between the utility failure and the availability of generator power.
- Third, it conditions the incoming power feed, removing voltage spikes or sags, or harmonic distortions in the AC feed. This conditioning can be accomplished via "double conversion."

A traditional UPS employs AC–DC–AC double conversion. Input AC is rectified to DC, which feeds a UPS-internal bus connected to strings of batteries. The output of the DC bus is then inverted back to AC to feed the data center PDUs. When utility power fails, input AC is lost but internal DC remains (from the batteries) so that AC output to the data center continues uninterrupted. Eventually, the generator starts and resupplies input AC power.

Traditional double-conversion architectures were robust but inefficient, wasting several percent of the power flowing through them as heat. Newer designs such as line-interactive, delta-conversion, or multi-mode systems operate at efficiencies in the range of 96-98% over a wide range of load cases. These newer designs vary in terms of speed and protection against fast voltage sags and swells, so they're not fully interchangeable.

Because UPS systems take up a sizable amount of space, they are usually housed in a room separate from the data center floor. Typical UPS capacities range from hundreds of kilowatts up to two megawatts or more, depending on the power needs of the equipment. Larger capacities are achieved by combining several smaller units.

Google pioneered distributed UPS designs that place UPS batteries into each server or server rack instead of a central UPS room. The original 12V "floating" battery architecture [12] placed a battery on the output (DC) side of the server's AC/DC power supply, thus requiring only a small trickle of charge and a simple switching circuit. If AC power fails, the 12V DC battery maintains the voltage of the 12V lines powering the server motherboard, providing protection in a very simple and natural way. (Telecom equipment has long used DC power supplies for reliability, and still uses it today.) The OpenCompute UPS [13], which distributes a rack of batteries for every four server racks, later adopted a similar strategy. These designs eliminate the upstream UPS and virtually eliminate UPS losses: ~0.1% compared to ~1%–3% for an in-line UPS with 480V AC input.

With today's increased power densities, racks operate on a 48V DC power bus [14]: rack-level rectifiers transform the incoming AC power to DC, and 48V battery modules connect directly to that same bus. Rectifier and battery modules can be scaled to match the power requirements of each rack's contents (a storage rack will need fewer modules than a server rack), and the natural paralleling on a common 48V bus allows for N+M redundancy if desired.

Until around 2015, data center UPSes used lead-acid batteries similar to 12 V car batteries. These batteries are relatively inexpensive and reliable but degrade over time, requiring replacement every five years. More recently, lithium ion batteries (Li-ion), including Lithium Iron Phosphate (LFP), have become more common since they offer higher discharge rates and longer battery life. However, they require careful fire prevention measures since damaged Lithium batteries can suffer from a thermal runaway reaction and catch fire. For example, in October 2020, a data center hosting the Korean company Kakao was destroyed by a Lithium battery fire. However, by carefully addressing safety, including rigorous testing and standardization, we can significantly benefit from Li-ion batteries for uninterrupted operation of WSCs. Recently, Google reached an important milestone: more than 100 million cells deployed in Li-ion battery packs across our global data center fleet [15].

5.2.4 Power distribution units (PDUs)

In our example data center, the UPS output is routed to PDUs on the data center floor. PDUs resemble breaker panels in residential houses but can also incorporate transformers for final voltage adjustments. They take a larger input feed and break it into smaller circuits that distribute power to the racks on the floor. Each circuit is protected by its own breaker. A traditional PDU handles 75–225 kW of load, and an individual circuit handles a maximum of approximately 6 kW (20 or 30 A at 110–230 V).

PDUs found in large data centers handle 2–3 MW, corresponding to the size of the backup generators, and typically feed a small number of busbars that run overhead along each server row. Tap-off units attached to the busbar drop individual circuits to the racks below and contain its circuit breakers, partially assuming the role of the PDU. Busbars are popular because they greatly reduce the amount of work for cabling: each rack now only needs a short cable from the overhead busbar instead of a long cable from a PDU at the end of the server row.

PDUs often provide additional redundancy by accepting two independent ("A" and "B") power sources and are able to switch between them with a small delay. The loss of one source does not interrupt power to the servers. In this scenario, the data center's UPS units are usually duplicated on A and B sides, so that even a UPS failure will not interrupt server power.

In North America, the input to the PDU is commonly 480 V 3-phase power. This requires the PDU to perform a final transformation step to deliver the desired 110 V output for the servers, thus introducing another source of inefficiency. In the EU, input to the PDU is typically 400 V 3-phase power. By taking power from any single phase to neutral combination, it is possible to deliver the desired 230 V without an extra transformer step. Using the same trick in North America requires computer equipment to accept 277 V (as derived from the 480 V input to the PDU), which unfortunately exceeds the upper range of standard power supplies.

Real-world data centers contain many variants of the simplified design described here. These include the "paralleling" of generators or UPS units, an arrangement where multiple devices feed a shared bus so the load of a failed device can be picked up by other devices, similar to handling disk failures in a RAID system. Common paralleling configurations include $N + 1$ (allowing one failure or maintenance operation at a time), $N + 2$ (allowing one failure even when one unit is offline for maintenance, or "concurrent maintainability"), and $2N$ (fully redundant pairs).

5.2.5 Power microgrids and other future opportunities

Like almost any building, data centers are grid connected, consuming power supplied by one or more utilities. A century ago, it was more common for industrial facilities to have their own private power plants, but the economic and reliability advantages of a shared grid supplanted virtually all of them. Only recently, the concept of microgrids has re-emerged, often in the context of geographies with abundant distributed energy resources but weak and unreliable grids. In some situations, microgrids can also support data centers.

Microgrids (Figure 5.11) can provide reliability and resiliency, cost savings and improved sustainability. Data centers already operate as microgrids during grid disturbances when generators provide backup power while the site is disconnected from the grid. Additionally, data centers employ key microgrid features such as power management (including power

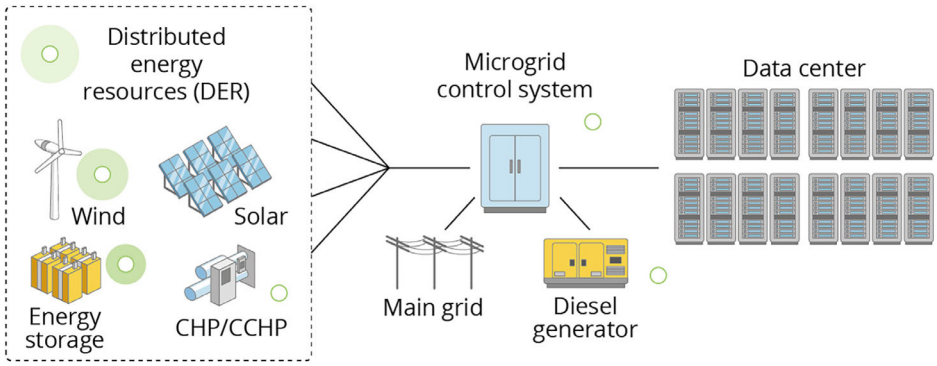


Fig. 5.11 Microgrids overview

monitoring and capping), and thus it is easier to integrate them into microgrids. For example, a data center could integrate carbon free energy (CFE) sources "behind the meter" to reduce energy costs and increase reliability by having an additional source of on-site backup power. Another use of data center microgrids could be using battery storage to avoid grid consumption at peak times.

Microgrid-based optimizations align with broader goals of transitioning to a 24/7 carbon-free energy model, including the sources of backup power. Treating data center loads as dispatchable loads allows cooperatively managing supply and demand with utilities. For example, shifting workloads in time or location can improve the carbon footprint of workloads [16]. This can include using standby power assets for capacity augmentation and power fluctuation control versus just backup [17], or introduce new carbon-free backup power sources like fuel cells.

Earlier, we discussed how ML workloads introduce new requirements to WSC data center designs. One specific example is shown in Figure 5.12, showing the power fluctuations of large-scale ML training workloads. At the start of a large job, thousands of TPUs or GPUs simultaneously move from idle to busy, causing a rapid change in power consumption from mostly idle to mostly full power (#1 in the figure). During a training run, worker tasks may briefly idle during a globally synchronous exchange of data at the end of each training step (#2). If some interruption occurs (e.g., loss of a rack), training may stall for a longer period (#4) while its tasks and data are redistributed to fit the new configuration (#3).

These start-stop cycles can lead to large power swings that can cause grid disruption if not managed appropriately. Currently these fluctuations are managed through software-managed power consumption (discussed more in Chapter 9). Microgrids and energy storage can help provide buffers to mask the grid from such fluctuations, but they can also exacerbate the problem since the fluctuations are large relative to the microgrid but small relative to the grid: a load oscillating between 25 and 50 MW will cause significant problems for a 50 MW microgrid but virtually no problems for a 10 GW regional grid.

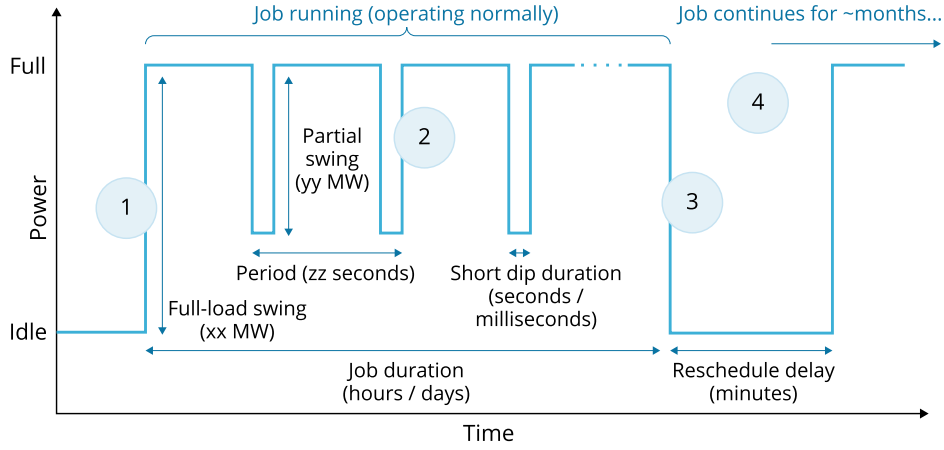


Fig. 5.12 Fluctuating power draw of a ML training run

Other future opportunities in data center electrical systems include higher voltage (600 V) systems for increased power delivery, onsite power generation technologies like fuel cells and hydrogen-based solutions, and the adoption of more advanced sensors and software-defined protection. Additionally, time synchronized disturbance recording systems promise improved observability and controllability.

5.3 Data center cooling systems

Data centers power the servers they contain and remove the heat generated. Now that we have seen how we deliver power in the previous section, this section examines the design of cooling systems.

Both power and cooling cause additional overhead. Historically, these overheads have been significant, nearly 100%, i.e., data centers have consumed twice as much energy as needed for the servers alone, due to these overheads. But, when best practices are employed, this overhead shrinks to 10–20%. When discussing power, we identified best-practice techniques like high-efficiency power distribution and UPS. Similarly, cooling subsystems include many optimizations including free cooling (further boosted by raising the target inlet temperature of servers) and well-managed air flow. Below, we will discuss these in more detail.

5.3.1 Overall cooling/mechanical architecture

Data center cooling systems remove heat generated by equipment. To remove heat, a cooling system must employ some hierarchy of loops, each circulating a cold medium that warms up via some form of heat exchange, to then be cooled again at a different place in the loop. An open loop replaces or supplements the outgoing warm medium with a cool supply from the outside, so that each cycle through the loop uses (some) new material. In contrast, a closed loop recirculates a separate medium, continuously transferring heat to either another loop via a heat exchanger or to the environment. All loops must eventually transfer heat to the outside environment, but inner loops do so indirectly by transferring their heat to outer loops.

Closed loop systems come in many forms, the most common being the air circuit on the data center floor. Its function is to isolate and remove heat from the servers and transport it to a heat exchanger. As shown in Figure 5.13**,** cold air flows to the servers, heats up, and eventually reaches a heat exchanger to cool it down again for the next cycle through the servers.

Many data centers employ raised floors, concrete tiles installed onto a steel grid resting on stanchions two to four feet above the slab floor. The underfloor area often contains power cables to racks, but its primary purpose is to distribute cool air to the server racks. The airflow through the underfloor plenum, the racks, and back to the CRAC (a 1960s term for computer room air conditioning) defines the primary air circuit. While convenient and flexible, raised floors add additional costs, and thus most hyperscale designs avoid them.

The simplest closed loop systems contain two loops. The first loop is the air circuit shown in Figure 5.13, and the second loop (the liquid supply inside the CRACs) leads directly from the CRAC to external heat exchangers or radiators (typically placed on the building roof) that discharge the heat to the environment.

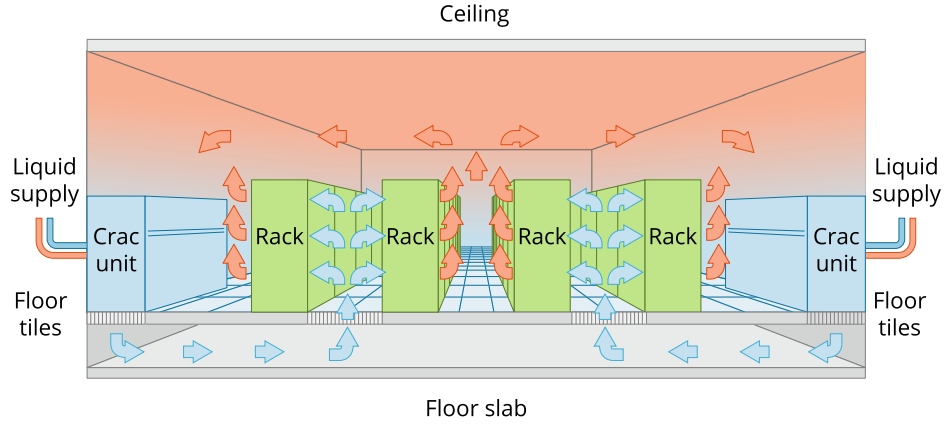


Fig. 5.13 Raised floor data center with hot-cold aisle setup

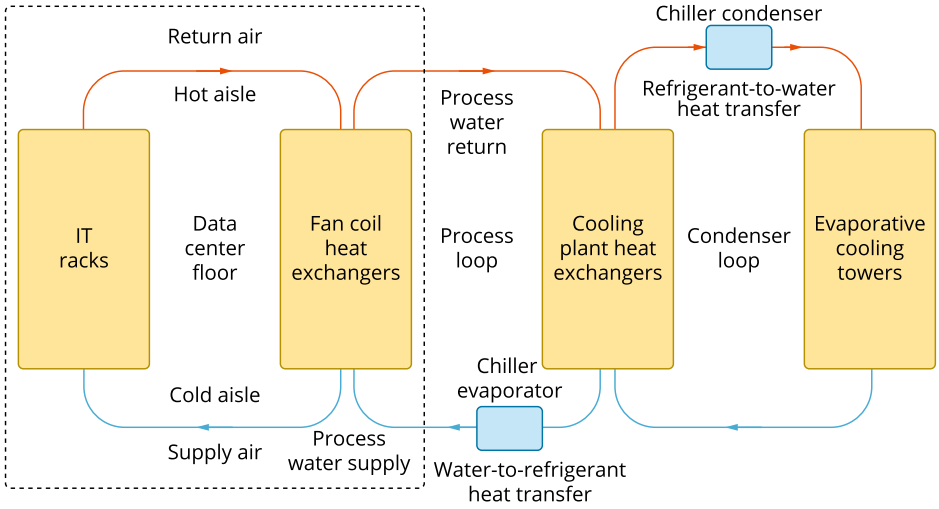


Fig. 5.14 Three-loop data center cooling system

A three-loop system commonly used in large scale data centers is shown in Figure 5.14. The first data center floor loop involves circulating air that is alternately cooled by fan coils and heated by IT equipment on the data center floor. This first air loop is kept very short: typically, racks exhaust their heat in the back into an enclosure (the "hot aisle") where fans with cool-water coils cool it and return it to the front of the racks. In the process water loop, warm water from the fan coils returns to the cooling plant which chills it and pumps it back to the fan coils. Finally, the condenser water loop removes heat received from the process water through a combination of mechanical refrigeration by chiller units and evaporation in cooling towers. The condenser loop is so named because it removes heat from the condenser side of the chiller. Heat exchangers perform much of the heat transfer between the loops while preventing process water from mixing with condenser water. In favorable weather conditions, the entire data center heat load can be removed by evaporative cooling of the condenser water; the chiller evaporator and chiller condenser heat transfer steps then become unnecessary.

Each topology presents tradeoffs in complexity, efficiency, and cost. For example, direct air cooling can be very efficient but does not work in all climates, requires filtering of airborne particulates, limits supported power density, and can introduce complex control problems. Two-loop systems are easy to implement, relatively inexpensive to construct, and offer isolation from external contamination, but typically have lower operational efficiency. A three-loop system is the most expensive to construct and has moderately complex controls, but offers contaminant protection and good efficiency when employing economizers.

Additionally, generators (and sometimes UPS units) must provide backup power for most mechanical cooling equipment because the data center may overheat in a matter of minutes without cooling. To see why, consider a large data center with a server hall measuring 100 * 50 * 5 meters (25,000 m^3) to support 25 MW load. The specific heat of air is 1 kJ/kg and the weight is 1.2 kg/m^3. A 25 MW load injects 25 MJ in one second, or 1kJ per m^3 of air in the server hall, thus heating it by 0.8°C *per second*. If the cooling plant fails, the air temperature will theoretically reach 100°C in two minutes!

In reality, the situation is less dire because other materials will absorb some of the heat from the air. For example, in water cooled buildings, the cooling pipes contain a fair amount of water that will warm up beyond its specified exit temperature and absorb a lot of heat, providing additional minutes of reserve. Similarly, racks, and the building itself, will start warming, thus delaying the ambient temperature rise. Nonetheless, there is little time to react to a cooling failure, since most equipment cannot tolerate an intake temperature above 40 °C.

Thus, if cooling fails, data center operators need to shut down IT equipment within 5–15 minutes of a cooling failure to prevent permanently damaging the equipment. Such an event is called Emergency Power Off (EPO) and typically indicates several hours of downtime even if cooling is restored quickly, since the uncontrolled shutdown of all equipment usually causes a lengthy cluster restart to validate all stored data and repair any lost storage servers.

In a typical data center, chillers and pumps can add 40% or more to the critical load supported by generators, significantly adding to the overall construction cost. CRACs, chillers, and cooling towers are among the most important building blocks in data center cooling systems. We take a closer look at each below.

5.3.2 Computer room air conditioners

Until the 1990s, data centers typically used computer room air conditioners (CRACs). CRACs resemble consumer air conditioners, containing a heat exchanger, air mover, and controls. The CRAC units typically sit on the periphery of the data center floor and blow cold air into the room or into the space under a raised floor. They mostly differ by the type of cooling they employ: direct expansion (DX), fluid solution, or water.

A DX unit is a split air conditioner with cooling (evaporator) coils inside the CRAC, and heat-rejecting (condenser) coils outside the data center. The fluid solution CRAC shares this basic architecture but circulates a mixture of water and glycol through its coils rather than a phase-change refrigerant. Finally, a water-cooled CRAC connects to a chilled water loop.

In a classical design, CRAC units pressurize the raised floor plenum by blowing cold air into the underfloor space, which then escapes through perforated tiles in front of the server racks. The air flows through the servers and is expelled into a "hot aisle." Racks are typically arranged in long rows that alternate between cold and hot aisles to reduce inefficiencies caused by mixing hot and cold air. In fact, many newer data centers physically

isolate the cold or hot aisles with walls [18].) As shown in Figure 5.13, the hot air produced by the servers recirculates back to the intakes of the CRACs, where it is cooled and exhausted into the raised floor plenum again.

5.3.3 Chillers

A water-cooled chiller as shown in Figure 5.15 is like a large water-cooled air conditioner.

Chillers submerge the evaporator and condenser coils in water in two large, separate compartments joined via a top-mounted refrigeration system consisting of a compressor, expansion valve, and piping. In the cold compartment, warm water from the data center is cooled by the evaporator coil before returning to the process chilled water supply (PCWS) loop. In the hot compartment, cool water from the condenser water loop is warmed by the condenser coil and carries the heat away to the cooling towers where it is rejected to the environment by evaporative cooling. The variable frequency drive (VFD) controls the speed of the chiller's motor. Because the chiller uses a compressor, a significant amount of energy is consumed to perform its work, so it is less efficient than just using cooling towers. However, because they work even on days where it's too warm for cooling towers to handle all cooling load, chillers are used to augment a cooling-tower based design and run only when needed.

Fig. 5.15 Water-cooled centrifugal chiller

5.3.4 Free cooling

Free cooling refers to the use of cold outside air to either help produce chilled water or directly cool servers. It is not completely free in the sense of zero cost, but it involves very low energy costs compared to chillers.

The simplest topology is fresh air cooling (or air economization) — essentially, opening the windows. Such a system is shown in Figure 5.16. This is a single, open loop system that is popular in geographies that don't experience high temperatures. Large wall fans direct outside air into cold aisles, and the warm air leaves the building at the top. Optionally, air economizers include water sprays in their intake path to lower cold aisle temperatures when the outside dry bulb temperature is high but the air is dry. Just like in cooling towers, evaporating water lowers the air temperature to be closer to the wet bulb temperature.

As mentioned above, air-economized data centers are open to the external environment and use low dry bulb temperatures for cooling. (The dry bulb temperature is the air temperature measured by a conventional thermometer). Large fans push outside air directly into the room or the raised floor plenum when outside temperatures are within limits (for an extreme experiment in this area, see [19]). Once the air flows through the servers, it is expelled outside the building. An air-economized system can be very efficient but requires effective filtering to control contamination, may require auxiliary cooling (when external conditions are not favorable), and may be difficult to control. Specifically, if there is a malfunction, temperatures will rise very quickly since air can store relatively little heat. By contrast, a water-based system can use a water storage tank to provide a significant thermal buffer.

Water-economized data centers take advantage of the wet bulb temperature. The wet bulb temperature is the lowest water temperature that can be reached by evaporation. The drier the air, the greater the difference between dry bulb and wet bulb temperatures. For example, air at 25° and 50% humidity has a wet bulb temperature of 18°, but at 90% humidity it's 23.7°. Thus, a water-economized data center can run without chillers for many more hours

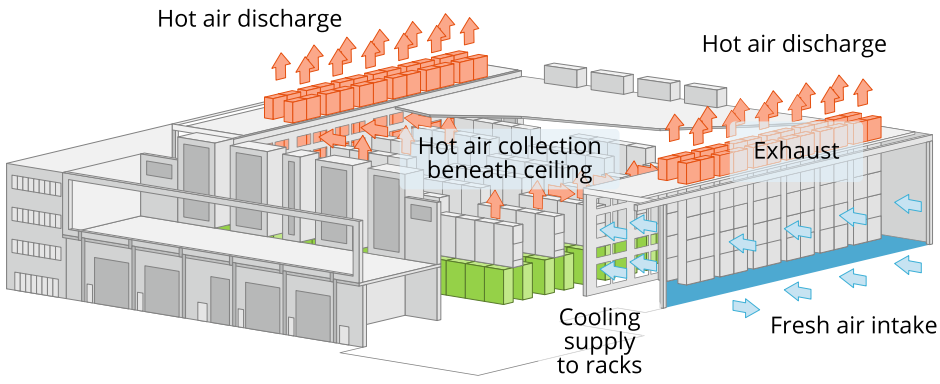


Fig. 5.16 Airflow schematic of an air-economized data center

per year. Some air-economized data centers employ a hybrid system where water is misted into the airstream (prior to entering the data center) to take advantage of evaporation cooling.

Typical water-economized data centers employ a parallel heat exchanger so that the chiller can be turned off when the wet bulb temperature is favorable. Depending on the capacity of the cooling tower (which increases as the wet bulb temperature decreases), a control system balances water flow between the chiller and the cooling tower.

Yet another approach uses a radiator instead of a cooling tower, pumping the condenser fluid or process water through a fan-cooled radiator. Similar to the glycol/water-based CRAC, such systems use a glycol-based loop to avoid freezing. Radiators work well in cold climates (say, a winter in Chicago) but less well at moderate or warm temperatures because the achievable cold temperature is limited by the external dry bulb temperature, and because convection is less efficient than evaporation.

5.3.5 Cooling towers

Cooling towers (Figure 5.17) cool a water stream by evaporating a portion of it into the atmosphere. The energy required to change the liquid into a gas is known as the latent heat of vaporization. Water that evaporates removes this latent heat and can thus cool the remaining water significantly given favorable weather conditions. The final temperature of the cooled water is close to the wet bulb temperature of the outside air; the difference between the achievable water temperature and the wet bulb temperature is called the cooling tower's "approach", typically 1-2°C. (To reach the actual wet bulb temperature, water would have to flow infinitely slowly.)

The water flowing through the tower comes directly from the chillers or from another heat exchanger connected to the PCWS loop. Figure 5.18 illustrates how it works.

1. Hot water from the data center flows from the top of the cooling tower onto "fill" material inside the tower. The fill creates additional surface area to improve evaporation performance.
2. As the water flows down the tower, some of it evaporates, drawing energy out of the remaining water and reducing its temperature.
3. A fan on top draws air through the tower to aid evaporation. Dry air enters the sides and humid air exits the top.
4. Cool water is collected at the base of the tower and returned to the data center.

Cooling towers work best in temperate climates with low humidity. Ironically, they do not work as well in very cold climates because they need additional mechanisms to prevent ice formation on the towers and in the pipes.

Fig. 5.17 Data center cooling towers

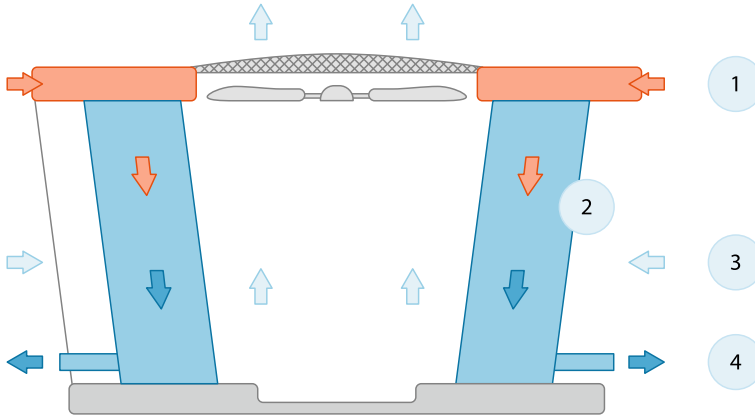


Fig. 5.18 How a cooling tower works

5.3.6 Airflow considerations

Conventional data centers use the raised floor setup discussed above. To change the amount of cooling delivered to a particular rack or row, we exchange perforated tiles with solid tiles or vice versa. For cooling to work well, the cold airflow coming through the tiles should match the horizontal airflow through the servers in the rack. For example, if a rack has 10 servers with an airflow of 100 cubic feet per minute (CFM) each, then the net flow out of the perforated tile should be 1,000 CFM (or higher if the air path to the servers is not tightly controlled). If it is lower, some of the servers will receive cold air while others will ingest recirculated warm air from above the rack or other leakage paths.

Figure 5.19 shows the results of a Computational Fluid Dynamics (CFD) analysis for a rack that is oversubscribing the data center's airflow. In this example, recirculation across the top of the rack causes the upper servers to ingest warm air. The servers on the bottom are also affected by a recirculation path under the rack. Blockages from cable management hardware cause a moderate warm zone about halfway up the rack. (In a design without raised floors but overhead cooling, the effects would be the same except that top and bottom are reversed.)

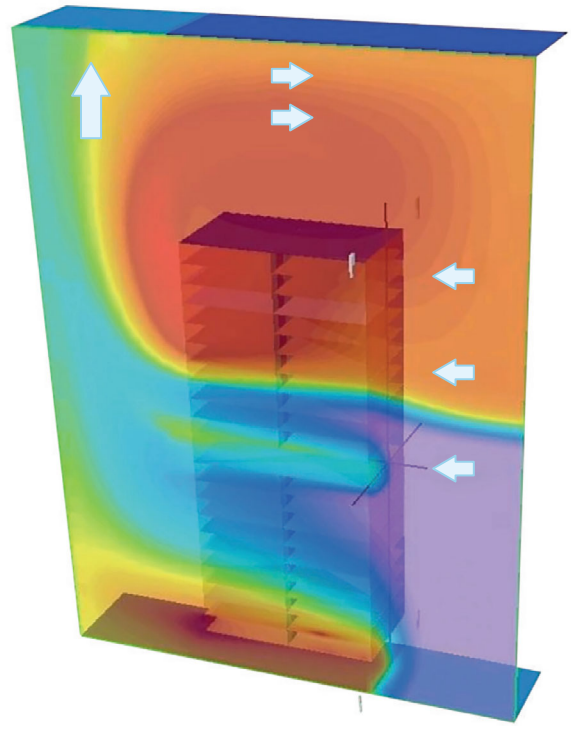


Fig. 5.19 CFD model of a rack with under-provisioned airflow

The facility manager's first response to such a situation might be to lower the temperature of the CRAC output. That works, but increases energy costs significantly, so it's better to fix the underlying problem instead and physically separate cold and warm air as much as possible, while optimizing the path back to the CRACs. In this setup the entire room is filled with cool air (because the warm exhaust is kept inside a separate plenum or duct system) and, thus, all servers in a rack will ingest air at the same temperature [18].

Even with hot aisle containment, airflow limits data center power density. For a fixed temperature differential across a server, a rack's airflow requirement increases with power consumption, and the airflow supplied to the front of the rack must increase linearly with power. That in turn increases the amount of static pressure needed in the underfloor plenum or the hot aisle. At low power densities, this is easy to accomplish, but at some point the laws of physics start to make it economically impractical to further increase pressure and airflow. Typically, these limitations make it hard to exceed power densities of more than 150–200 W/sq ft without substantially increased cost.

5.3.7 In-rack, in-row, and liquid cooling

In-rack cooling can increase power density and cooling efficiency beyond the conventional raised-floor limit. Typically, an in-rack cooler adds an air-to-water heat exchanger at the back of a rack so the hot air exiting the servers immediately flows over coils cooled by water, essentially short-circuiting the path between server exhaust and CRAC input. In-rack cooling might remove part or all of the heat, effectively replacing the CRACs. Obviously, chilled water needs to be brought to each rack, greatly increasing the cost of plumbing. Some operators may also worry about having water on the data center floor, since leaky coils or accidents might cause water to spill on the equipment.

In-row cooling works like in-rack cooling except the cooling coils aren't in the rack, but adjacent to the rack. A capture plenum directs the hot air to the coils and prevents leakage into the cold aisle. Figure 5.20 shows an in-row cooling product and how it is placed between racks.

Figure 5.21 illustrates the main features and air flow of a Google in-row cooling system (the innermost cooling loop). Hot exhaust from IT equipment rises in a vertical plenum space or hot aisle (1) and enters a separate plenum space above the drop ceiling (2). There, it is cooled by a heat exchanger (3) and the heat is removed by process water. Finally, a fan blows the cold air down into the cold aisle (4) where it will again enter the IT equipment.

Similar variations have become prevalent because hot-air containment is efficient and inexpensive. Tall, vertical hot aisle plenums duct the exhaust air from the rear of the IT racks to overhead fan coils. Fan coils receive chilled process water from an external cooling plant. This water flows through multiple tube passages attached to fins, absorbing heat from the incoming hot air. Blowers in the fan coil units force the cooled air downwards into the cold aisle, where it enters the intakes of servers and networking equipment. Together with the

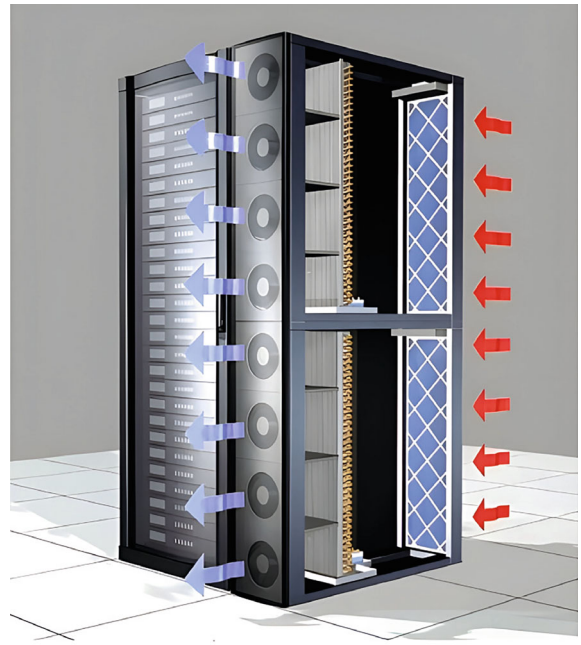


Fig. 5.20 In-row air conditioner

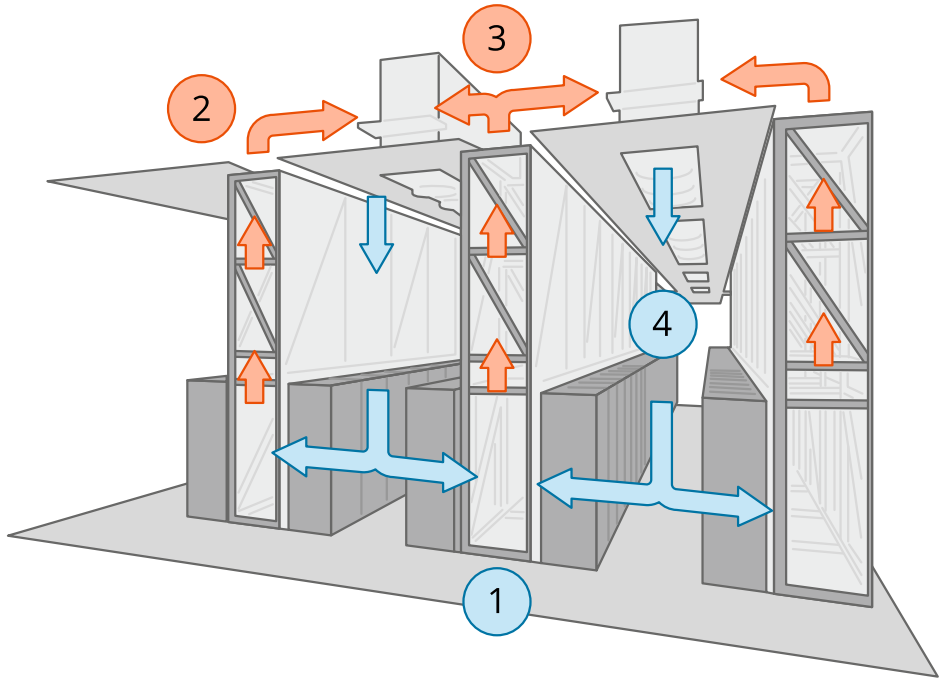


Fig. 5.21 Cross-section of a cold aisle and associated hot air plenums

cooling plant and process water loops, this air loop comprises a highly-efficient, end-to-end cooling system that adds <10% to the energy consumed by the IT equipment.

Finally, we can also directly cool server components using cold plates, that is, local, liquid-cooled heat sinks. It is usually impractical to cool all components with cold plates. Instead, components with the highest power dissipation (such as processor chips) are targeted for liquid cooling while other components are air-cooled. The liquid circulating through the heat sinks transports the heat to a liquid-to-air or liquid-to-liquid heat exchanger that can be placed close to the tray or rack, or be part of the data center building (such as a cooling tower).

In spite of the higher cost and mechanical design complexity, cold plates are becoming essential for cooling very high density workloads whose TDP per chip exceeds what is practical to cool with regular heatsinks (typically, 200–250 W per chip). An example is Google's third generation tensor processing unit (TPU): as shown in Figure 5.22, four TPUs on the same motherboard are cooled in series on a single water loop [20]. Cold water comes in from the rack (left) and then flows through all four cold plates before exiting via a tube. Each chip will heat the incoming water by about the same amount, so the outgoing water temperature is determined by the total power consumed by the four TPUs, divided by the water flow rate times the thermal mass of water. In principle, each chip could have its own cold water supply, but for TPU v3 this wasn't necessary.

To enable this water cooling, each TPU rack must be supplied with cold water. In principle, water is available already on the floor (it's flowing through the cooling infrastructure that's behind or above the racks, as discussed above) but on-tray cooling requires a much finer distribution of water to many more consumption points (trays). Figure 5.23 shows the cooling distribution unit (CDU) that's connected to the process water loop and breaks it up into flows

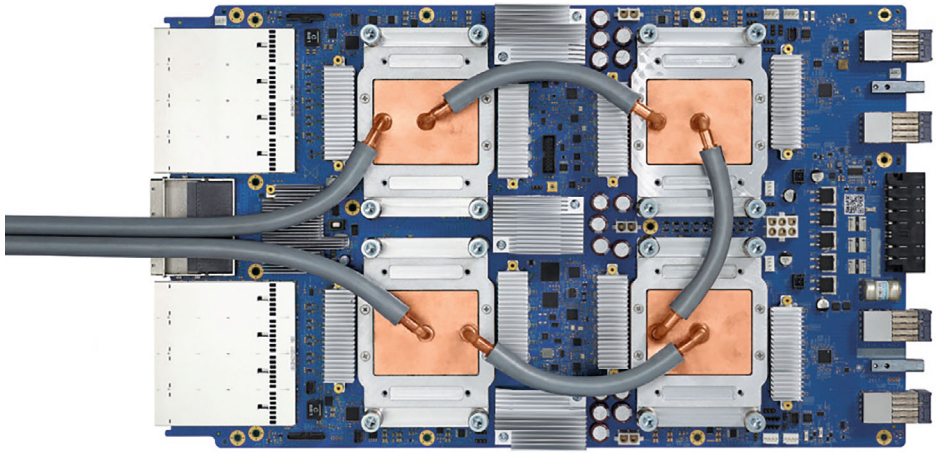


Fig. 5.22 Copper cold plates provide liquid cooling for TPUv3

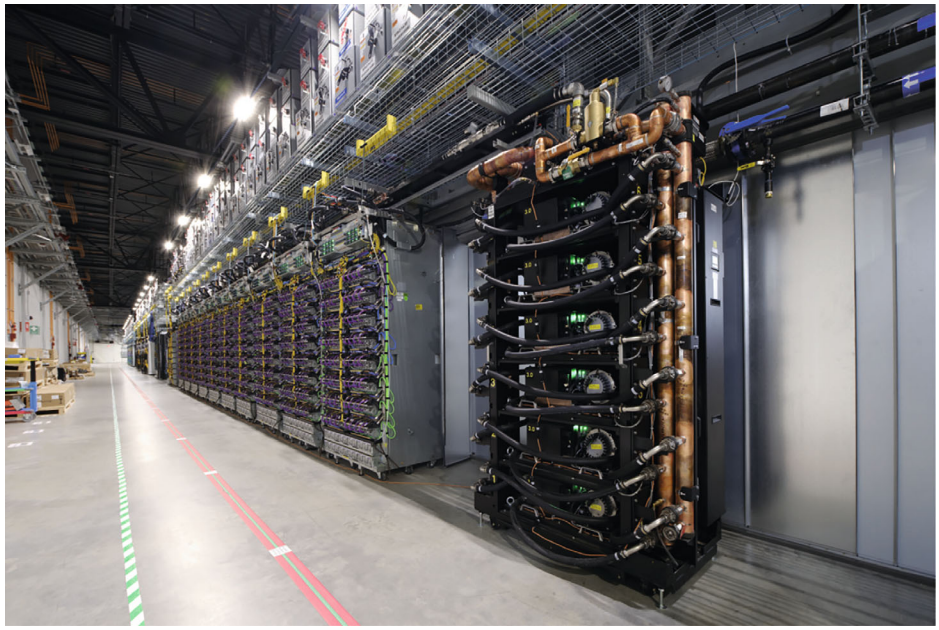


Fig. 5.23 Cooling distribution unit (CDU) and a row of TPU racks

for each rack. Water flows into (and out of) each rack via black hoses visible on top. Each tray taps into that supply via a no-drip quick-connect connector that allows individual trays to be serviced without spilling water.

As computing power (and thermal) density continues to increase, system designers are also exploring other new cooling solutions for WSCs such as liquid immersion cooling. As the name indicates, these designs immerse the entire server in non-conductive fluid that helps dissipate heat more effectively, using less power. The superior heat exchange capabilities of liquid cooling also eliminates the need for bulky heat sinks, fans, and air flow channels, potentially enabling new denser system designs (including novel 3D server designs [21]).

Two main kinds of immersion cooling solutions have been explored, single-phase (1P) and two-phase (2P). 1P immersion cooling [22] uses a pump to circulate a dielectric fluid such as mineral oil through a radiator cooling loop. 2P immersion cooling, on the other hand, cools server components by dissipating heat through phase change using liquids that have low boiling point (50-60 degrees Celsius) [23].

While immersion cooling can enable significantly lower PUEs, it has not yet been widely adopted. Server boards may have to be redesigned to be compatible with the cooling fluid and its impact on signal integrity, and optical connectors may also need redesigns. While these challenges can be solved, they move the design out of the mainstream (commodity) supply chain, thus creating additional costs. Additionally, immersed racks are significantly

harder to maintain, especially if overall power density needs to be high, thus requiring tall tanks.

5.4 Other data center considerations

5.4.1 Modular data center design for velocity and efficiency

WSC data centers are large and complex, and designing and deploying them is time-intensive, labor-intensive, and capital-intensive. Modular data center designs can improve these metrics.

Modular designs make it possible to create specific products (or variants targeting specific customer needs) out of individual building blocks. As shown in Figure 5.24, modular data centers follow a hierarchical composition approach, starting with individual components that combine to build modules, key data center subcomponents, and eventually full data centers. Modular data centers are characterized by two key elements:

- Flexible interfaces between components and subsystems allow building an overall system with multiple configurations. These interfaces are similar to software APIs and must be carefully designed and version-controlled.

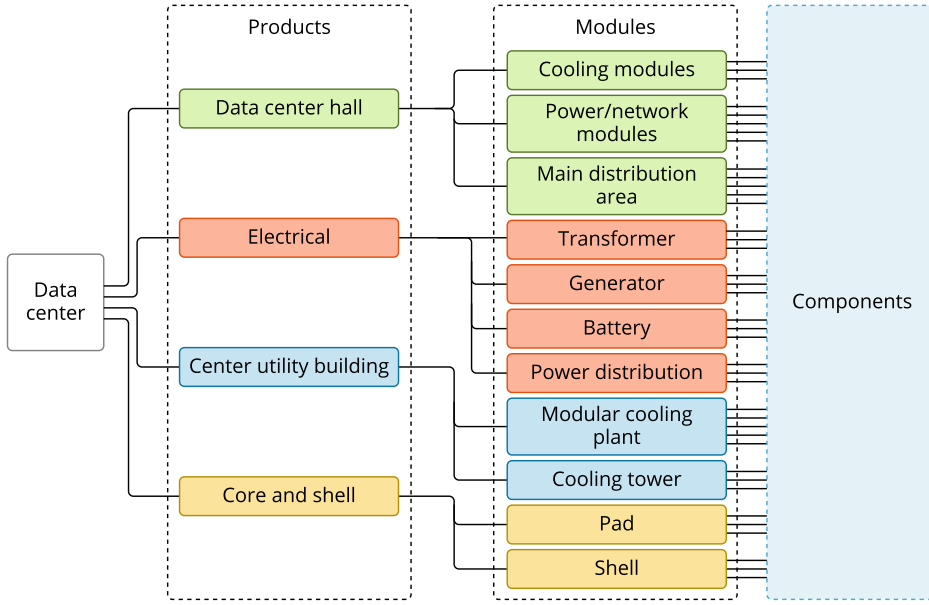


Fig. 5.24 Hierarchical composition in modular data centers

- "Kit-of-parts construction" uses a set of pre-designed, standardized components that can be assembled in various ways to create different components. Kit-of-parts construction is a subset of prefabrication that not only attempts to achieve flexibility in assembly and efficiency in manufacture, but also requires demountability, disassembly, and reuse.

Modular data centers enable late binding configuration and flexible reconfiguration to meet new or location-specific requirements or supply-chain constraints. For example, the hot aisle cooling structure can come in a regular variant and one that's earthquake rated. A new data center can be configured quickly from the standardized set of parts, and enable subsystems (network rooms, power, cooling, space) to evolve independently. Last but not least, they lend themselves to easier manufacturing, including pre-deployed modular construction and delivery (e.g., equipment skidding and pre-assembly).

5.4.2 Economic models: The five envelopes

To reduce lead times and cost, data center capacity can be held at different stages of completion. For example, it can take months or even years to find a suitable piece of land and obtain all the zoning approvals and utility connections required for a data center. Similarly, it might take a few months to "grade" a site, i.e., leveling the ground or creating slopes on a site to ensure proper drainage and a stable foundation for a building. Holding capacity as graded land is much cheaper than holding it as a partially completed facility, but it requires a longer lead time to build actual capacity for hosting servers. Thus, if demand is uncertain (which it always is!), the best supply strategy keeps additional buffers at several stages of the construction process, and keeps larger buffers at earlier stages because the carrying cost is much lower there.

The top row of Figure 5.25 shows the stages of data center construction. Entitled land is a parcel of land that we now own ("hold the title"). The next step is to connect it to utilities, primarily water, sewer, power, and maybe roads. Ideally, the land is already close to such utilities, at the required capacity. This stage can take months to years, depending on the responsiveness of utilities and their ability to obtain permits and execute the work. Pad and shell refers to site grading (creating a pad suitable for a building), and constructing the building shell without any fitout. Next in line typically is the electrical substation, which often has a long lead time (12-18 months are common) and is a significant investment. As a rule of thumb, a substation costs $0.10/W and sometimes comes with a minimum power consumption requirement since the utility won't build it to just stand there idle.

After that, the investments (in $/W) rapidly escalate since the next phases involve building the actual data center infrastructure. First, the facility part that's outside the floor, the cooling and electrical plants and their controls. Fitting out the data center hall is next, i.e., populating the inside of the data center with busbars, cable trays, cooling units, etc. Last, with the space now ready to receive equipment, cluster fit-out to provide infrastructure needed for the racks

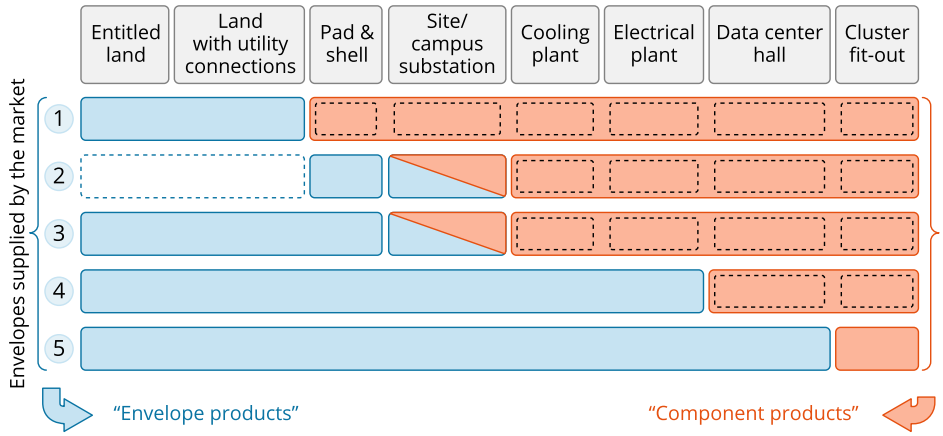


Fig. 5.25 Five envelopes of data center design

being deployed afterwards: tap boxes for rack power, network racks and fiber bundles for Clos networks, and so on.

Different entities may perform these construction phases, and thus they enable a variety of economic models. The five-envelope framework, illustrated in Figure 5.25, shows common divisions in responsibilities between vendors and WSC providers. Each envelope shifts the boundary between what is built by the final WSC operator and what is bought (or leased) from a provider.

Envelope 1 represents WSC data centers designed and built end-to-end internally by the hyperscaler; the only component bought from the market is land and utility connections. Envelope 2 additionally buys a building shell constructed by a third party instead of building it in-house. On the other end of the spectrum, Envelope 5 represents a colocation choice where everything but the actual equipment deployment is done by a third party. Envelope 4 is very similar but shifts the floor fit-out and possibly some of the operations to the WSC operator; it is more common when the WSC operator leases an entire facility.

WSC operators may use all of these envelopes, or just some, depending on local circumstances. For example, for a small network POP data center, Envelope 5 usually is the most cost effective, whereas for a large data center campus envelope 1 typically is cheapest.

5.4.3 Physical automation in the data center

Warehouse-scale computers resemble actual warehouses in more than just their external appearance and scale. WSCs also see significant movements of goods. Every day, trucks arrive to unload new equipment, move excess equipment to a different location, or remove

obsolete equipment. DIMMs and disks are delivered to or from trays on the floor, servers and racks are moved between rows or floors, and so on.

In actual warehouses, physical automation is common. Goods are moved via robots driving around on the warehouse floor, automated lifts servicing tall shelves, or conveyor systems moving them laterally. In WSCs, physical automation isn't as pervasive since material movement is less intensive and since many actions require precise work on expensive parts, vs moving cheap consumer goods to and from shelves.

However, as data center buildings and campuses become larger, it is increasingly important to enable safer and more efficient operations. One key advantage of automation is the in-built capability to scan and track material as it passes through the automation systems. This is particularly useful in improving the traceability of storage media with user data like hard drives and SSDs. Designing servers and racks to interface with automation also opens up new opportunities to improve server installation and repair.

Figure 5.26 shows a rack mover robot used to pick, move, and place fully-loaded racks in Google data centers. This method of moving racks is 5x faster and reduces the risk of injuries. Additionally, racks are also subjected to lower levels of shock and vibrations, resulting in faster rack bring up at new locations.

Fig. 5.26 Rack Mover for moving Google racks

One of the key objectives of data center operations teams is secure management of storage media. When technicians handle media containing user data, they need to take ownership of media and securely hand it off to dedicated secure areas for processing. At each step, devices are scanned so all movement is traceable and auditable. With growing numbers of storage media devices, automation has significantly helped this task.

Figure 5.27 shows a disk shredder with a verification hood that controls the access path of media to the shredder. The hood captures the media part number, verifies that the media is to be destroyed, and updates audit logs once the media is destroyed by the shredder. The logs capture the source data behind the certificate of destruction (COD), available for future audits. Media re-use after erasure and disassembly can support a circular economy [24].

Other physical automation can also help reduce the physical toil on data center technicians. For example, server tray lifts allow a single technician to insert or remove even the heaviest servers. Similarly, autonomous guided vehicles (AGVs) can ferry such trays to and from technicians, saving them the time that would be needed to walk to a collection point that may be 100 meters or more away. Since data center use cases have similar requirements than other industries, an emerging ecosystem of automation suppliers serves these use cases increasingly well.

Fig. 5.27 Disk shredder with media verification device

5.5 Historical retrospective

During the first twenty five years of WSCs at Google, WSC data centers have seen significant evolution. Google initially started off with cheap and scrappy rack solutions in third party colos. Figure 5.28 shows one such early deployment. Colos at that time charged by space and not by power, so the racks packed a lot more servers per square foot, sometimes getting to power densities higher than what the facility's cooling supported, requiring the external fan shown in the picture!

Soon, Google started building its own data centers, taking over large warehouses and populating them with computing infrastructure. This was the genesis for the term warehouse-scale computing. Many key innovations on WSC data centers were designed and deployed at this time: hot air capture, warm air cooling, local UPS. From day one, these data center designs emphasized minimizing the overheads associated with power and cooling – something that would later be institutionalized in the PUE (power usage efficiency) metric (discussed in Chapter 9). This was also the phase when Google experimented with the first container-based data centers (idea dating back to a Google patent application in 2003 [25]).

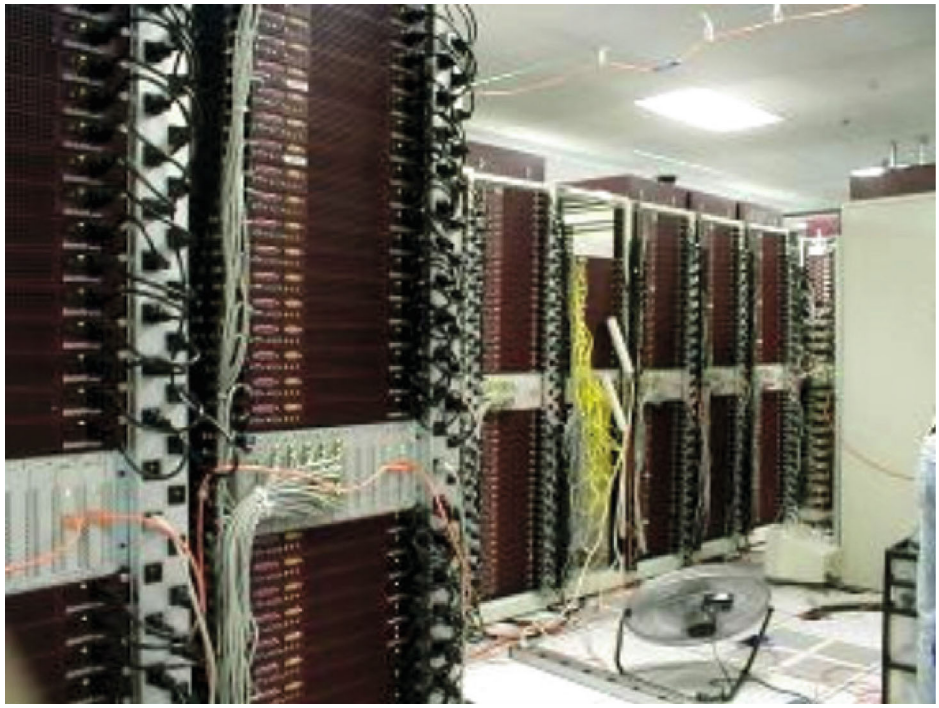


Fig. 5.28 Early colo with Google servers and "external cooling aid"

Container-based data centers (Figure 5.30) went one step beyond in-row cooling by placing the server racks inside a container (typically 20 or 40 feet long) and integrating heat exchange and power distribution into the container as well. Similar to in-row cooling, the container needed a supply of chilled water and used coils to remove all heat. Close-coupled air handling typically allowed higher power densities than regular raised-floor data centers. Thus, container-based data centers provided all the functions of a typical data center room (racks, CRACs, PDU, cabling, lighting) in one small package. Figure 5.29 shows an isometric cutaway of Google's container design. Like a regular data center room, containers must be accompanied by outside infrastructure such as chillers, generators, and UPS units to be fully functional.

However, subsequent generations of Google data centers moved away from containers, instead incorporating the same principles at a broader warehouse level. This next phase of data center design focused on scalability, right sizing, and uptime. Rack-level UPS were introduced, as also larger power and cooling shared domains. The next wave of innovations were headlined by a transition to an ambitious architecture that introduced medium voltage power delivery into the data center [26], coupled with software codesign for aggressive power oversubscription to increase data center utilization and cost efficiencies. Battery technologies also transitioned from lead acid to lithium ion during this time.

More recently, the growth of cloud computing workloads brought an increased focus on heterogeneous IT requirements and off-the-shelf components. Buildings were designed more modularly with pre-fabbed solutions (e.g., modular electrical buildings) to reduce

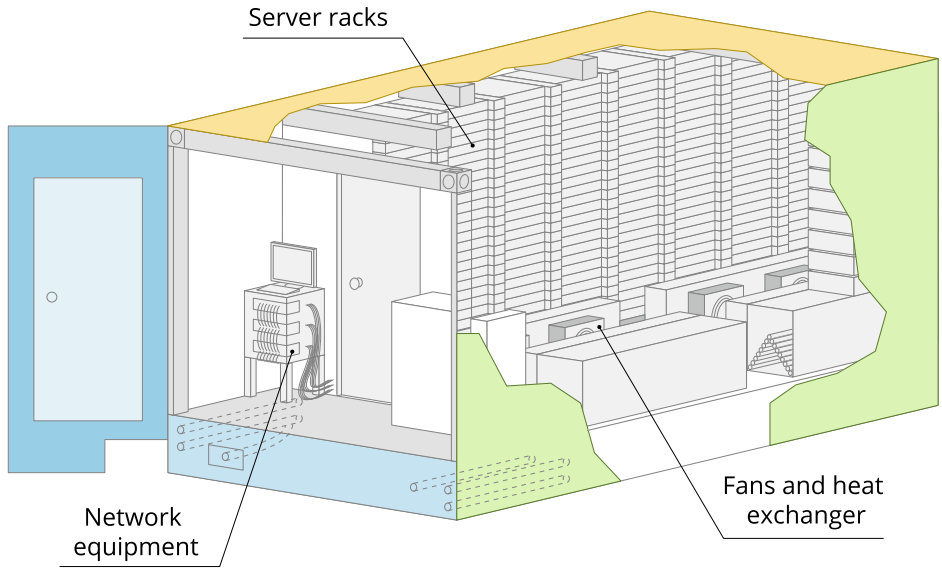


Fig. 5.29 Google's 2005 container design

Fig. 5.30 A prototype container being installed on the Google campus

onsite construction time. Machine learning deployments led to increased deployment of liquid cooling at scale to all rack spots. Physical automation improved the efficiency, safety, and security of material handling.

Today, WSC data center design is far from done. Optimizing for a complex set of tradeoffs between cost and energy efficiency, sustainability, time-to-design, modularity and incremental deployability is a gift that keeps giving.

References

1. US EPA, *Report to Congress on server and data center energy efficiency*, https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Report_Exec_Summary_Final.pdf, 2007.
2. J. Koomey et al., "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, no. 2011, p. 161, 2011.
3. International Code Council (ICC), *Chapter 6: Types of Construction - 2015 International Building Code(IBC)*, https://codes.iccsafe.org/public/document/IBC2015/chapter-6-types-of-construction, 2015.
4. dup.

5. W. P. Turner IV, J. PE, P. Seader, and K. Brill, "Tier classification defines site infrastructure performance," *Uptime Institute*, vol. 17, 2006.
6. Uptime Institute, *Tier Standard: Operational Sustainability*, https://uptimeinstitute.com/resources/asset/tier-standard-operational-sustainability.
7. Uptime Institute, *Tier Standard: Topology*, https://uptimeinstitute.com/publications/asset/tier-standard-topology.
8. V. Sakalkar, V. Kontorinis, D. Landhuis, et al., "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 497–511.
9. P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," in *33rd International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 66–77.
10. A. J. George and G. Ferrand, *Cost study of AC vs. DC data center power topologies based on system efficiency*, https://www.eltek.com/globalassets/media/downloads/white-papers_case-studies/cost-study-on-ac-vs-dc-data-center-based-on-system-efficiency-an-eltek-white-paper.pdf, 2017.
11. F. L. Rawson, J. Pfleuger, and T. Cader, "Comparing the energy consumption of servers using workload injection," Intel Corporation, Tech. Rep., 2008.
12. W. Whitted, M. Sykora, K. Krieger, et al., "Data center uninterruptible power distribution architecture," US Patent 7,560,831, Jul. 2009.
13. P. Sarti, *Battery cabinet hardware v1. 0*, https://telecombloger.ru/wp-content/uploads/2012/10/Open_Compute_Project_Open_Rack_v1.0.pdf, 2012.
14. S. McCauley and S. Jiang, *Google 48V update: Flatbed and STC*, https://www.opencompute.org/files/External-2018-OCP-Summit-Google-48V-Update-Flatbed-and-STC-20180321.pdf, 2018.
15. Google Cloud, *TPUs improved carbon efficiency of AI workloads by 3x*, Google Cloud Blog: https://cloud.google.com/blog/topics/systems/100-million-li-ion-cells-in-google-data-centers, 2025.
16. A. Radovanovic, *Our data centers now work harder when the sun shines and wind blows*, The Keyword, https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows/, 2020.
17. V. Kontorinis, L. E. Zhang, B. Aksanli, et al., "Managing distributed UPS energy for effective power capping in data centers," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 488–499.
18. M. K. Patterson and D. Fenwick, *The state of data center cooling*, https://www.ceclimited.com/sites/all/themes/creative/state-of-date-center-cooling.pdf, 2008.
19. D. Atwood and J. G. Miner, *Reducing data center cost with an air economizer*, https://www.intel.com/content/dam/doc/technology-brief/data-center-efficiency-xeon-reducing-data-center-cost-with-air-economizer-brief.pdf, 2008.
20. N. P. Jouppi, D. H. Yoon, G. Kurian, et al., "A domain-specific supercomputer for training deep neural networks," *Commun. ACM*, vol. 63, no. 7, pp. 67–78, Jun. 2020.
21. M. Kodnongbua, Z. Englhardt, R. Bianchini, et al., "Dense server design for immersion cooling," *ACM Trans. Graph.*, vol. 43, no. 6, Nov. 2024, ISSN: 0730-0301.
22. Y. Zhong, *A large scale deployment experience using immersion cooling in data centers*, Open Compute Project Summit, https://www.opencompute.org/events/past-events/2019-ocp-global-summit, 2019.
23. B. Ramakrishnan, H. Alissa, I. Manousakis, et al., "Cpu overclocking: A performance assessment of air, cold plates, and two-phase immersion cooling," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 10, pp. 1703–1715, 2021.

24. iNEMI, *iNEMI 2019 Value Recovery from End-of-Life Electronics Project Phase 2 Report*, https://thor.inemi.org/webdownload/2019/iNEMI-Value_Recovery2_Report.pdf, 2019.
25. W. H. Whitted and G. Aigner, *Modular data center*, US Patent 7,278,273, 2007.
26. V. Sakalkar, V. Kontorinis, D. Landhuis, et al., "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, New York, NY, USA: Association for Computing Machinery, Mar. 13, 2020, pp. 497–511.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

WSC Hardware: Computing Building Blocks

6

6.1 WSC building blocks and design considerations

6.1.1 Overall architecture

Building upon the previously discussed data center infrastructure, we now focus on the individual hardware building blocks that define the architecture of warehouse-scale computers (WSCs). Just as logic elements form a microprocessor or chipsets create a server, WSC systems are composed of diverse components across compute, storage, and networking—from individual chips to motherboards and distributed systems. This chapter aims to provide insights into the selection of these building blocks and the resulting hardware-software codesign decisions.

6.1.2 Building scale-out systems

When building large-scale distributed hardware systems, we have two choices: scale-up (large shared-memory systems), or scale-out (large clusters). WSCs have historically chosen scale-out to build clusters of low-end or mid-range servers [1], for a variety of reasons. The primary motivation has been the underlying cost-efficiency of such servers relative to high-end shared-memory systems. Continuing trends on increasing core count also mean that most VM or task instances can comfortably fit into a two-socket (or even a one-socket) server. In fact, a typical physical server often hosts tens or hundreds of VMs today.

In prior editions of this book, we have gone deep into comparisons of the TPC-C benchmark [2] on high-end versus low-end servers. For example, the first edition compared the HP Integrity Superdome-Itanium2 [2] and the HP ProLiant ML350 G5 [2] and showed how the cost-efficiency of the low-end servers was four times better than the high-end

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_6

shared-memory system. However, over subsequent years and editions, the high-end server market has increasingly become niche, with hyperscale servers now being much more common and higher-volume. So, rather than focus on any specific individual system comparisons, below, we present a simple mathematical model to reason about scale-up versus scale-out tradeoffs and discuss the choices around the CPU building blocks, including the "brawny versus wimpy cores" debates [3], and how those have evolved over the years. We also discuss how the scale-up versus scale-out consideration has been different for emerging accelerator systems.

6.1.2.1 A model to reason about scale-up versus scale-out

Simple processor-centric cost-efficiency analyses show that a single server of capacity $N * M$ is much more expensive than N servers of capacity M. However, they fail to consider that large Shared-Memory Multiprocessors (SMPs) benefit from drastically superior intercommunication performance than clusters of smaller servers connected by commodity fabrics. Nodes in a large SMP can communicate at latencies on the order of 100 ns, whereas LAN-based networks, usually deployed in clusters of servers, can experience latencies on the order of 100 µs. For parallel applications that fit within a single large SMP (for example, SAP HANA), the efficient communication can translate into dramatic performance gains. However, as discussed earlier in the discussion on WSC applications (Chapter 3), most large workloads are unlikely to fit within an SMP. Therefore, it is important to understand the relative performance of clusters of large SMPs with respect to clusters of low-end servers each with smaller number of CPU sockets or cores (albeit the same server-class CPU cores as used in large SMPs). The following simple model can help make these comparisons.

Assume that a given parallel task execution time can be modeled as a fixed local computation time plus the latency penalty of accesses to global data structures. If the computation fits into a single large shared memory system, those global data accesses will be performed at roughly DRAM speeds (~100 ns). If the computation fits in only a multiple of such nodes, some global accesses will be much slower, for the remote distributed systems access, on the order of typical LAN speeds (~100 µs). If the fixed local computation time is of the order of 1 ms—a reasonable value for high-throughput internet services—the equation that determines the program execution time is as follows:

$$time = 1ms \ + \ f \cdot (100ns \cdot LocalAccessesFraction + 100\mu s \cdot RemoteAccessesFraction)$$

where the variable f is the number of global accesses per work unit (1 ms).

Assume further that accesses to the global store are uniformly distributed among all nodes. Then, the fraction of global accesses that map to the local node is inversely proportional to the number of nodes in the system (N). The equation thus becomes:

$$time = 1ms \ + \ f \ \cdot \ (\frac{1}{N} \ 100ns \ + \ \frac{N-1}{N} \ 100\mu s)$$

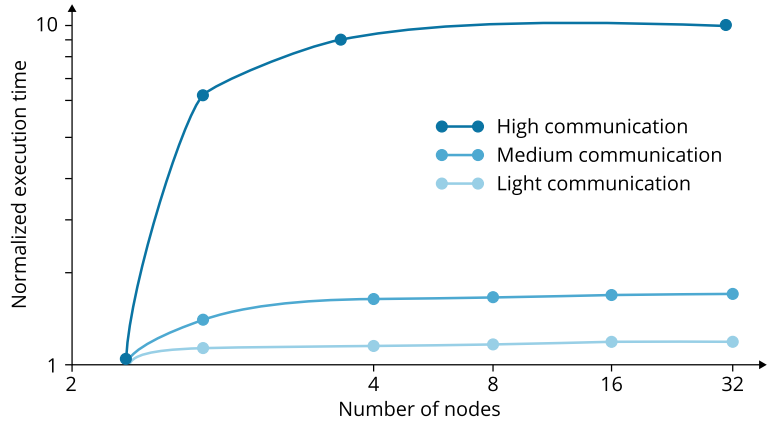


Fig. 6.1 Performance in hypothetical SMP clusters

In Figure 6.1, we plot the execution time of this parallel workload (Y-axis) as the number of nodes involved in the computation increases (X-axis). Three curves are shown for different values of *f,* representing workloads with light communication ($f = 1$), medium communication ($f = 10$), and high communication ($f = 100$) patterns. Note that in our model, the larger the number of nodes, the higher the fraction of remote global accesses. Both axes are logarithmic.

As expected, under light communication performance degrades only slightly when using clusters of multiple nodes. For medium- and high-communication patterns the penalties can be quite severe, but they are most dramatic when moving from a single node to two, with rapidly decreasing additional penalties for increasing the cluster size. Using this model, the performance advantage of a single 128-processor SMP over a cluster of thirty-two 4-processor SMPs could be more than a factor of 10×.

By definition, WSC systems consist of thousands of CPUs. Therefore, we would like to use this model to compare the performance of a cluster built with large SMP servers with one built with low-end ones. Here we assume that the per-core performance is the same for both systems and that servers are interconnected using an Ethernet-class fabric. Although our model is exceedingly simple (for example, it does not account for contention effects), it clearly captures the effects we are interested in.

In Figure 6.2, we apply our model to clusters varying between 512 and 4,192 cores. We show the performance advantage of an implementation using large SMP servers (128 cores in a single shared memory domain) versus one using low-end servers (four-core SMPs). The performance edge of the cluster based on high-end servers deteriorates quickly as the cluster size increases. If the application requires more than 2,000 cores, a cluster of 512 low-end servers performs within approximately 5% of one built with 16 high-end servers, even under a heavy communication pattern. With a performance gap this low, the price premium of the high-end server (4–20 times higher) renders it an unattractive option.

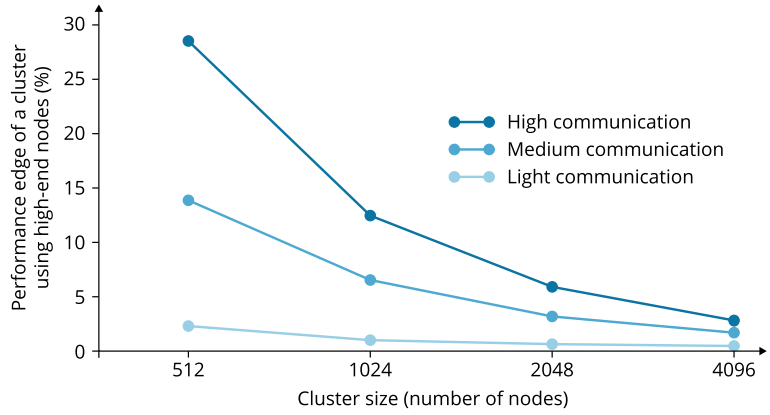


Fig. 6.2 Deteriorating performance advantage of large SMP servers

The point of this analysis is qualitative in nature: it illustrates how we need to reason differently about our baseline platform choice when architecting systems for applications that are too large for any single high-end server. The broad point is that the performance effects that matter most are those that benefit the system at the warehouse scale. Performance enhancements that have the greatest impact on computation, that are local to a single node (such as fast SMP-style communication in our example), are still very important. But if they carry a heavy additional cost, their cost-efficiency may not be as competitive for WSCs as it is for small-scale computers.

6.1.2.2 Granularity of scale-out: Brawny versus wimpy cores

Clearly one could use the argument laid out above to go further and use CPU cores that are even smaller (or "wimpier" [4]) than today's server-class CPU cores. Indeed, several prior studies have explored this direction: The Piranha chip multiprocessor [5] was one of the earliest systems to advocate the use of lower-end cores in enterprise-class server systems. Lim et al. [6] made the case for low-end power-efficient processor cores in server platforms. This work led to HP's Moonshot Servers [7] that had 45 ARM-based or mobile-x86-based blade servers in one 4.3U enclosure. Hamilton [8] made a similar argument for PC-class components, setting the stage for Amazon's Graviton-based line of ARM servers [9]. The FAWN (Fast Array of Wimpy Nodes) project [4] explored the utility of wimpy cores as the basis for building an energy efficient key-value storage system using flash memory, focused on more I/O- and memory-bound computation in storage servers.

The advantages of using even smaller, slower CPUs are very similar to the arguments for using mid-range commodity servers instead of high-end SMPs:

- Multicore CPUs in mid-range servers typically carry a price-performance premium over lower-end processors so that the same amount of throughput can be bought two to five times cheaper with multiple smaller CPUs.
- Many applications are memory- or I/O-bound so that faster CPUs do not scale well for large applications, further enhancing the price advantage of simpler CPUs.
- Slower CPUs tend to be more power efficient. Typically, CPU power decreases by $O(k^2)$ when CPU frequency decreases by a factor of k.

At the same time, there are perils in going down this path too indiscriminately [3].

First, although many internet services benefit from seemingly unbounded request- and data-level parallelism, such systems are not immune from Amdahl's law. As the number of offered parallel threads increases, it can become increasingly difficult to reduce serialization and communication overheads, limiting either speedup or scaleup [10], [6]. In the limit, the amount of inherently serial work performed on behalf of a user request by extremely slow single-threaded hardware will dominate overall execution time.

Also, the more threads handle a parallelized request, the larger the variability in response times from all these parallel tasks will be, exacerbating the tail latency problem discussed earlier in Chapter 3. As a result, although hardware costs may diminish, software development costs may increase because more applications must be explicitly parallelized or further optimized. For example, suppose that a web service currently runs with a latency of one second per user request, half of it caused by CPU time. If we switch to a cluster with lower-end servers whose single-thread performance is three times slower, the service's response time will double to two seconds and application developers may have to spend a substantial amount of effort to optimize the code to get back to the one-second-latency level.

Smaller servers may also lead to lower utilization. Consider the task of allocating a set of applications across a pool of servers as a bin packing problem—each of the servers is a bin, with as many applications as possible packed inside each bin. Clearly, that task is harder when the bins are small because many applications may not completely fill a server and yet use too much of its CPU or RAM to allow a second application to coexist on the same server.

Finally, even embarrassingly parallel algorithms are sometimes intrinsically less efficient when computation and data are partitioned into smaller pieces. That happens, for example, when the stop criterion for a parallel computation is based on global information. To avoid expensive global communication and global lock contention, local tasks may use heuristics based on local progress only, and such heuristics are naturally more conservative. As a result, local subtasks may execute longer than they might if there were better hints about global progress. Naturally, when these computations are partitioned into smaller pieces, this overhead tends to increase.

Also, from the perspective of cloud applications, most workloads emphasize single-VM performance, preferring higher-performance cores with a focus on single-threaded performance per VM. For example, a database serving many applications often wants to run on the

fastest server available since database performance limits application performance. Additionally, a bigger system is more amenable to being deployed and sold as smaller VM shapes, but the converse is not true.

On the other hand, throughput-oriented workloads like data analytics prefer systems that minimize the cost of computation, leading to servers with somewhat slower cores but higher throughput, and less memory per core to reduce system cost.

Recently, CPU vendors have started to provide CPU variants optimized for each of the two scenarios. For example, ARM offers the "V" series for performance, the "N" series for scale-out, and the "E" series for efficiency. Intel offers two parallel lines of Xeon processors, again focused on performance and efficiency. Similarly, AMD offers multiple processor lines in their EPYC systems, optimized for different performance and efficiency design points.

The success of ARM processors offered by cloud providers (for example, AWS's Graviton and GCP's Axion) illustrates the demand for "brawny" cores. Current ARM VMs provide better price/performance for two reasons. First, the respective CPUs do not use hyperthreading, and thus 1 vCPU = 1 physical core. On x86 VMs using two-way hyperthreading, 2 vCPUs = 1 physical core. Thus, when comparing performance per vCPU, ARM VMs can be substantially faster. Second, they are also cheaper because they have half the physical DRAM per core. Cloud VMs are provisioned with a standard amount of DRAM per vCPU (not per physical core), and thus the ARM server contains only half the memory, saving 20-25% in cost per server.

Our discussion above on the pros and cons of finer granularity provide broad guidelines for design decisions. Designs maximizing single-core speed, and designs improving per-socket throughput by reducing single-core speed, both have their place in server fleets. But the further apart their speeds are, the less likely any single-socket price-per-throughput advantage will play out in actual deployments, due to the overheads discussed above.

6.1.2.3 Scale-up vs scale-out for accelerators

While we have thus far focused on traditional compute servers, we face similar trade-offs when building large-scale emerging WSC accelerator systems. Later in this chapter, we will discuss the design of different accelerators — GPUs and TPUs for machine learning workloads, and VCUs for video workloads — and how we scale these systems for WSC scale. Since accelerator systems are custom-designed and target vertically-integrated solutions for specific workloads, often with lower volumes than for traditional servers, their design tradeoffs can be different.

Returning to the model discussed at the beginning of this chapter, a custom design opens up the opportunity to also customize the network, so that the latency variable for remote node access can differ from the 100 µs we assumed in the prior model. TPU systems, for example, have an innovative ICI-based network (ICI stands for Google's Inter-Chip Interconnect protocol) that allow larger TPU supercomputers (or "hypercomputers" [11]) with shared-memory abstractions. For ML accelerators, TPU designers chose to go fairly

deep on scaling up, with pod sizes of 9,216 chips. NVIDIA offers NVLink interconnects allowing up to 256 GPUs. A new industry consortium is working on an "ultra accelerator link (UAL)" [12] to enable similar faster networking as an industry standard. For large applications, multiple such highly-interconnected pods are connected via Ethernet-based scale-out network fabrics.

VCU systems (Section 6.3.3), on the other hand, fully embrace traditional WSC scale-out because video coding tasks don't communicate with each other. Thus, their mass deployment uses Ethernet-based networking and close integration with scale-out task scheduling.

Overall, while this area is still relatively nascent and evolving, we expect the same principles outlined earlier in our discussion to hold for the design of these systems. Scale-up will offer significant performance and related performance-per-cost advantages for smaller-sized workloads, but cost efficiency and the ability to leverage industry hardware and the large software investment in distributed systems management will always make scale-out important for much larger workloads.

6.2 Computing: servers and racks

6.2.1 Server hardware

Servers hosted in individual racks are the basic building blocks of WSCs. They are interconnected by hierarchies of networks, and supported by the shared power and cooling infrastructure.

Each server generation needs to optimize performance and cost for a wide range of WSC workloads and the flexibility to support their resource requirements. Servers are usually built in a tray format, housing the motherboard, chipset, and additional plug-in components. The motherboard provides sockets and plug-in slots to install CPUs, memory modules (DIMMs), a management module, local storage (such as Flash, SSDs, or HDDs), network interface cards (NICs), as well as accelerator cards (such as GPUs, TPUs, or VCUs) to satisfy the range of resource requirements. The Platform Controller Hub (PCH) manages slower, non-critical I/O operations like USB, SATA, and Ethernet, while the CPU handles faster connections like PCIe and DRAM. Figure 6.3 shows the components of a typical server.

Driven by workload performance, total cost of ownership (TCO), and flexibility, several key design considerations determine the server's form factor and functionalities.

- CPU: CPU power, often quantified by the thermal design power, or TDP; CPU packaging, number of CPU sockets and NUMA (non-uniform memory access) topology; CPU selection (for example, core count, core and uncore frequency, cache sizes, and number of inter-socket coherency links).

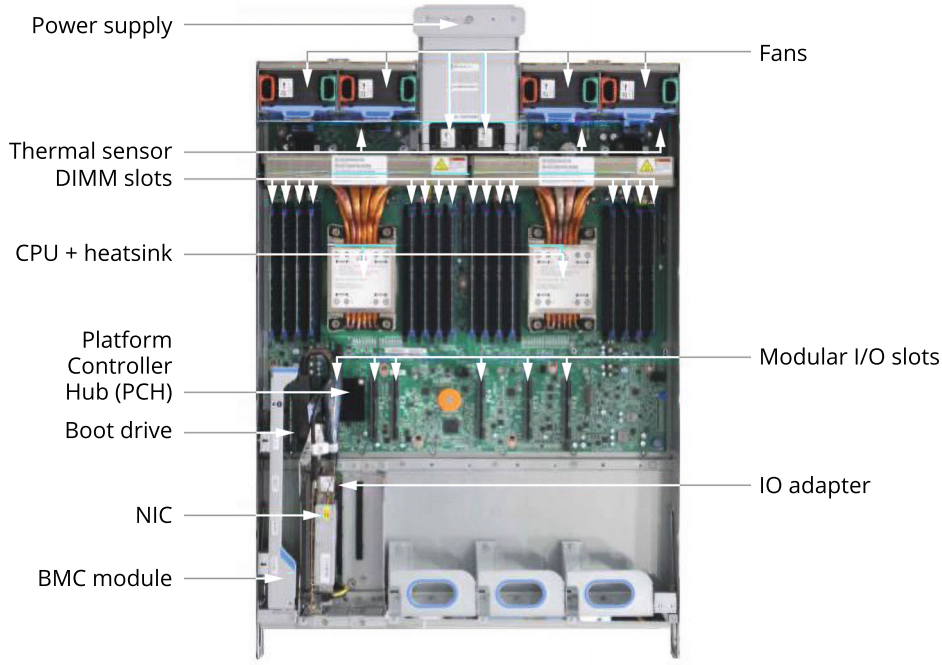


Fig. 6.3 Annotated breakdown of a server tray

- Memory: Number of memory channels, number of DIMMs per channel, DIMM types supported (such as RDIMM, LRDIMM, and so on), and DDR standard supported (DDR4, DDR5, LPDDR, etc.).
- Plug-in I/O cards: Number of PCIe cards needed for SSD, NIC, and accelerators; form factors; PCIe bandwidth and power; potential sharing of resources (e.g., NIC quadfurcation); new I/O devices like Compute Express Link (CXL), etc.
- Tray-level power and cooling: Voltage regulators, battery backup, cooling options (liquid versus air-cooled) and packaging, fans, support for power management, etc.
- Device management and security options: board management controller (BMC), root-of-trust security, etc.
- Mechanical design: how servers are designed and assembled requires careful consideration. Form factors (width, height, depth) can impact power density, and front or rear component access can affect serviceability.

Figure 6.4 shows photographs of different server trays. The leftmost server supports two Intel Sapphire Rapids CPU sockets. Each CPU can support up to 350W TDP. Each CPU has four memory controllers, each of which can control two DDR5 memory channels with two DIMM slots per channel (2DPC), for a total of 16 DIMMs per CPU and 32 DIMMs per

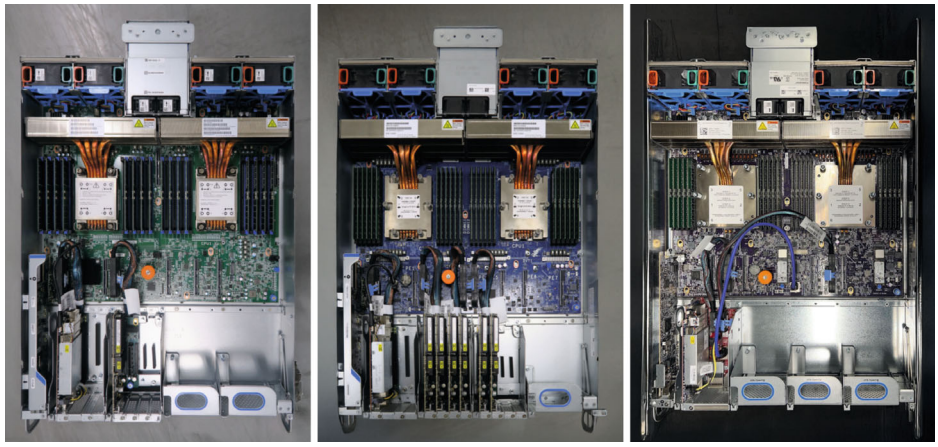


Fig. 6.4 Example server trays: Intel, AMD, and ARM

tray. With integrated Voltage Regulators, the platform allows per-core DVFS. The system supports 8 PCIe Gen5 connections that allow it to host PCIe cards for SSD, 200 Gbps NIC, and an array of special purpose accelerators. A Data Center Secure Control Module (DC-SCM) based on the OCP DC-SCM specification connects to the motherboard and enables separation of Root-of-Trust (RoT), Board Management Controller (BMC), BMC EEPROM, and BIOS EEPROM from the motherboard.

The second server is similar, except it contains AMD Genoa CPUs with higher core counts and TDP, support for Compute Express Link (CXL) interconnect, and twelve memory channels with one DIMM slot per channel (1DPC). The additional memory channels provide increased memory bandwidth to support the higher number of cores. It's difficult to provide enough memory bandwidth for all workloads, though, so some may still be limited by memory bandwidth. With each improvement in transistor size it becomes easier to add additional cores, but scaling memory bandwidth proportionally becomes harder since the chip area available for I/O connections (the "shoreline") is not increasing, and the bandwidth per memory channel also doesn't scale with transistor size and performance.

The third server uses Google's custom Axion CPU with ARM Neoverse V2 cores. Each CPU has 10 DDR5 memory controllers and supports up to 350W TDP. The tray contains two compute nodes, each with their own socket. This design balances compute capacity, in terms of core count, frequency, and cache sizes, with the available memory system bandwidth. The design also supports NIC quad-furcation, where a single NIC is shared by 4 ARM servers (two trays) to improve Perf/TCO.

Figures 6.5 and 6.6 show the functional architecture for a two-socket server and for a bifurcated 2x1-socket system design.

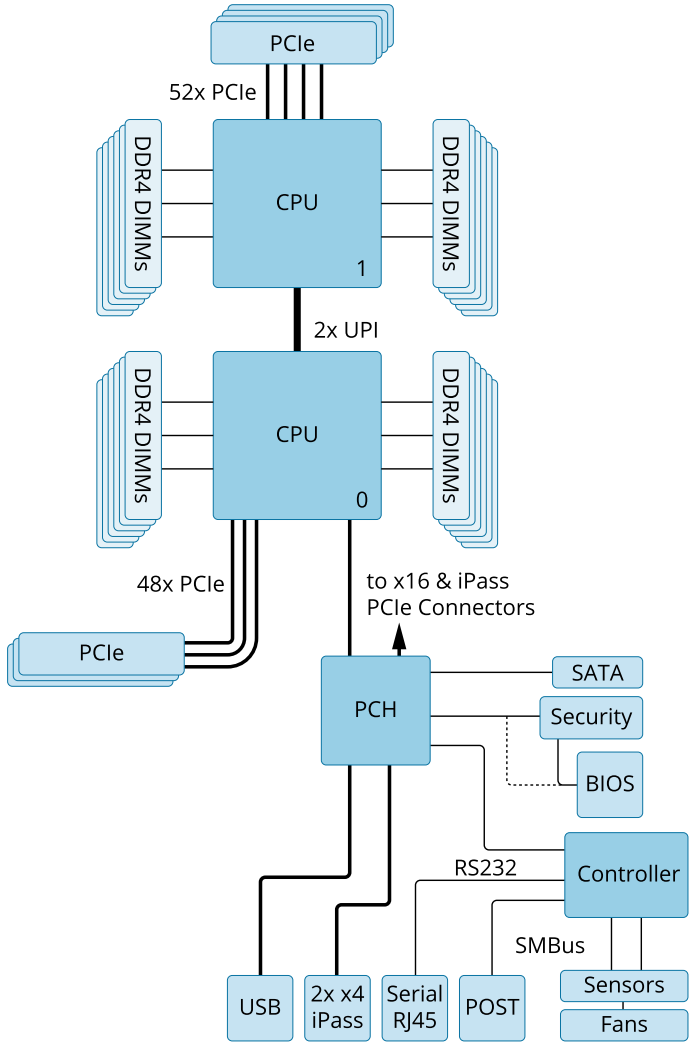


Fig. 6.5 Block diagram of a two-socket server

6.2.2 Hardware racks

The rack is the physical structure that holds tens of servers together. Racks not only provide the physical support structures, but they also handle shared power infrastructure, including power delivery, battery backup, and power conversion (such as AC to 48V DC). The width and depth of racks vary across WSCs: some are classic 19-in wide, 48-in deep racks, while others can be wider or shallower. Rack design can also vary depending on whether they are targeted at server trays, networking, or accelerators, and whether they are used in the primary

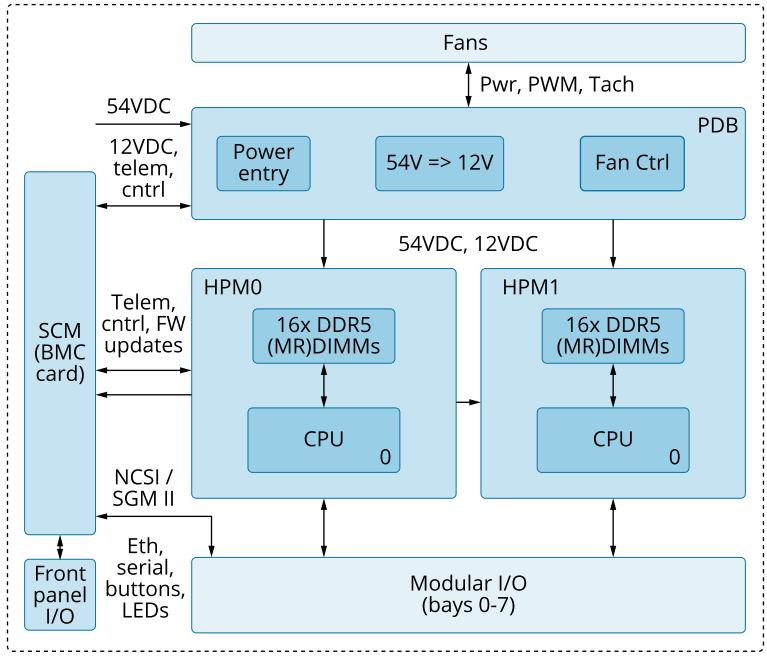


Fig. 6.6 Block diagram of a bifurcated 2x 1-socket system

campus or in edge or colo campuses. Rack design considerations can also change when used in "pods" or "cliques" [13] and with different cooling considerations (air vs liquid). We will discuss accelerator rack design in pods more in the next section.

The width and depth of a rack constrains tray form factors. It is often convenient to connect the network cables at the top of the rack; such a rack-level switch is appropriately called a Top of Rack (ToR) switch. We will describe the networking in more detail shortly in Section 6.4.

Figure 6.7 shows an example rack in a Google data center. It supports a configurable physical structure for server trays with different widths and heights; for example, it can host four narrow server trays per row. It also provides configurable power conversion and battery backup support, with redundancy, to match the IT power load. The ToR switch and rack management unit are located at the top of the rack and are further connected to a data center fabric to connect many racks.

Figure 6.8 shows high-density racks containing TPU accelerators and their associated networking. Compared to regular racks, these racks have much higher bandwidth and power per rack.

The Open Compute Project [14] provides detailed specifications of many WSC hardware components.

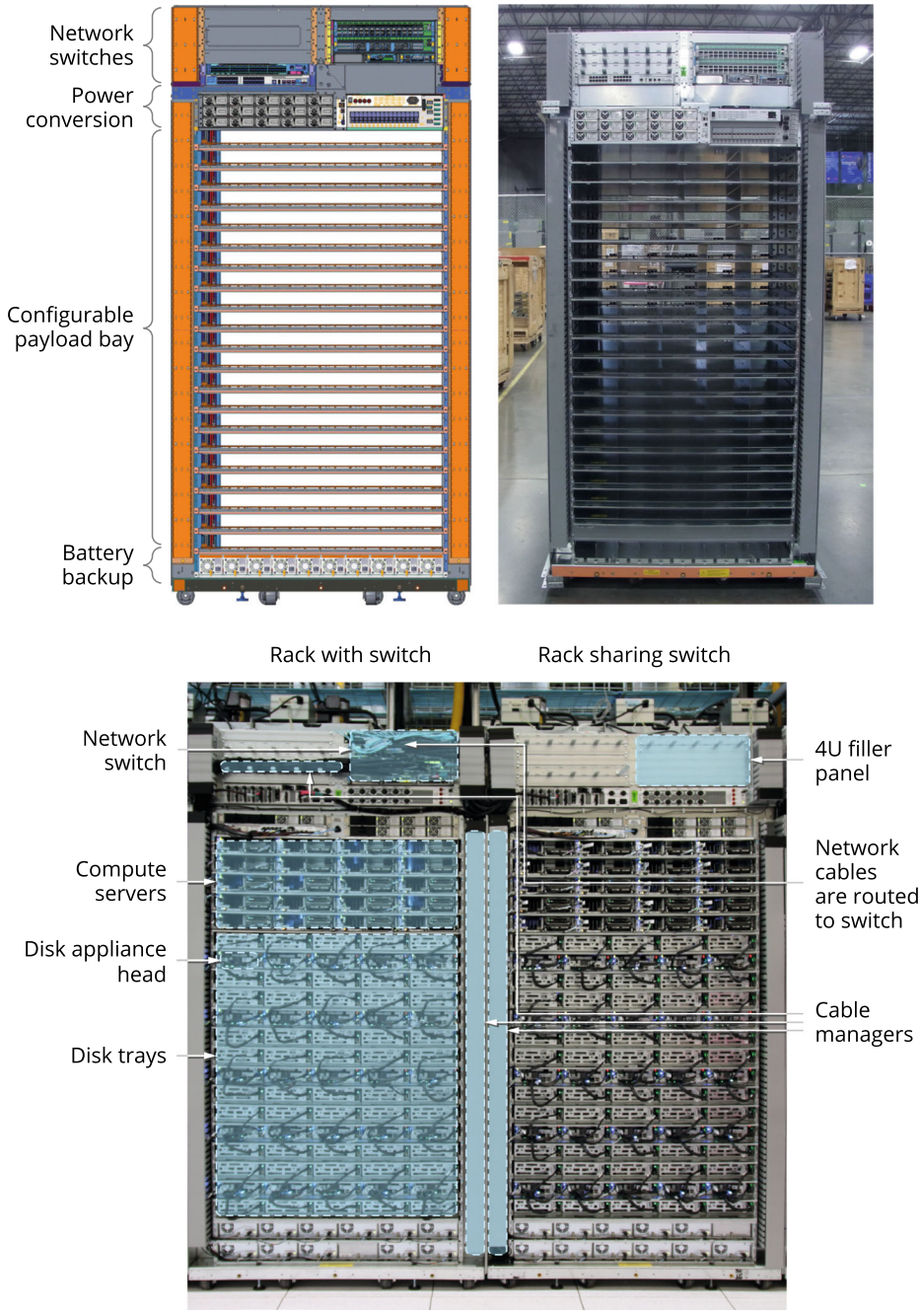


Fig. 6.7 Empty and populated server racks

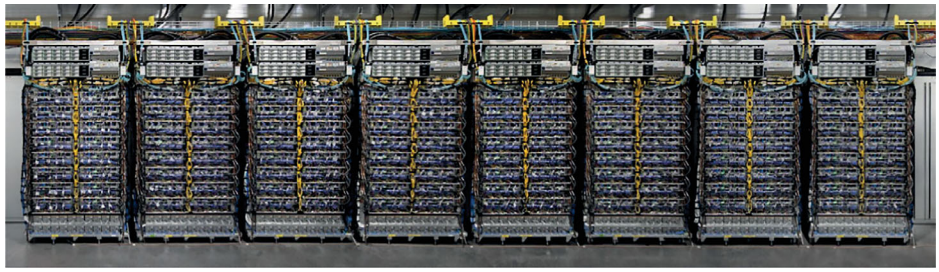


Fig. 6.8 Rack design for accelerator rack

6.2.3 Individual servers as distributed systems

In the past, server designs were "monoliths", single node machines akin to Figure 6.9 (a), with one main compute complex and managed by a single operating system. However, as shown in Figure 6.9 (b), systems today include multiple compute elements outside the primary CPU: management controllers, smart NICs, accelerators – each running their own operating systems. This approach improves security and manageability, and also enables "bare-metal" servers where cloud customers have non-virtualized access to the CPU. Disaggregated designs further redefine the server as a virtual construct where system resources are configured as dynamically re-allocatable pools with resource slices multiplexed over a network of individual hardware components. For example, instead of physically containing a local SSD or GPU, the server may connect to a rack-level pool of such devices via a high-

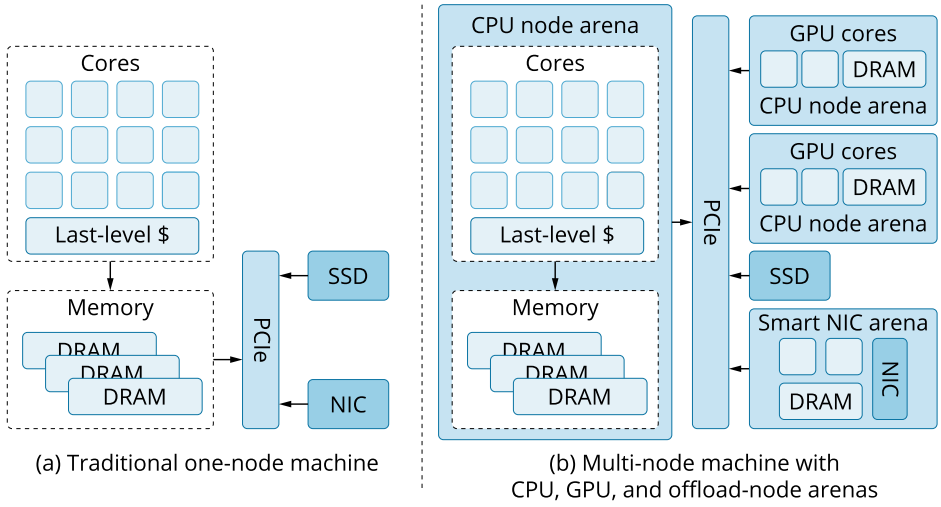


Fig. 6.9 Evolution of servers into multi-node machines

speed bus like PCIe or CXL, or even via Ethernet. Figure 6.9 demonstrates how a server has become a complex distributed system running multiple firmware and systems stacks.

Thus, a server no longer is a set of hardware components contained in sheet metal, but a logical composition of mutually-distrusting abstract components, called arenas. A machine is the composition of one or more arenas. Such servers provide a clear separation of security and trust domains (arenas), with each arena running its own operating system. Thus, the OS that runs the computing workload cannot access or modify the OS that manages the system.

The best way to handle this hardware heterogeneity is to hide it from most of the system stack by moving management functions elsewhere (Figure 6.10). For example, the offload arenas discussed above move virtualization into a separate card, reducing or eliminating

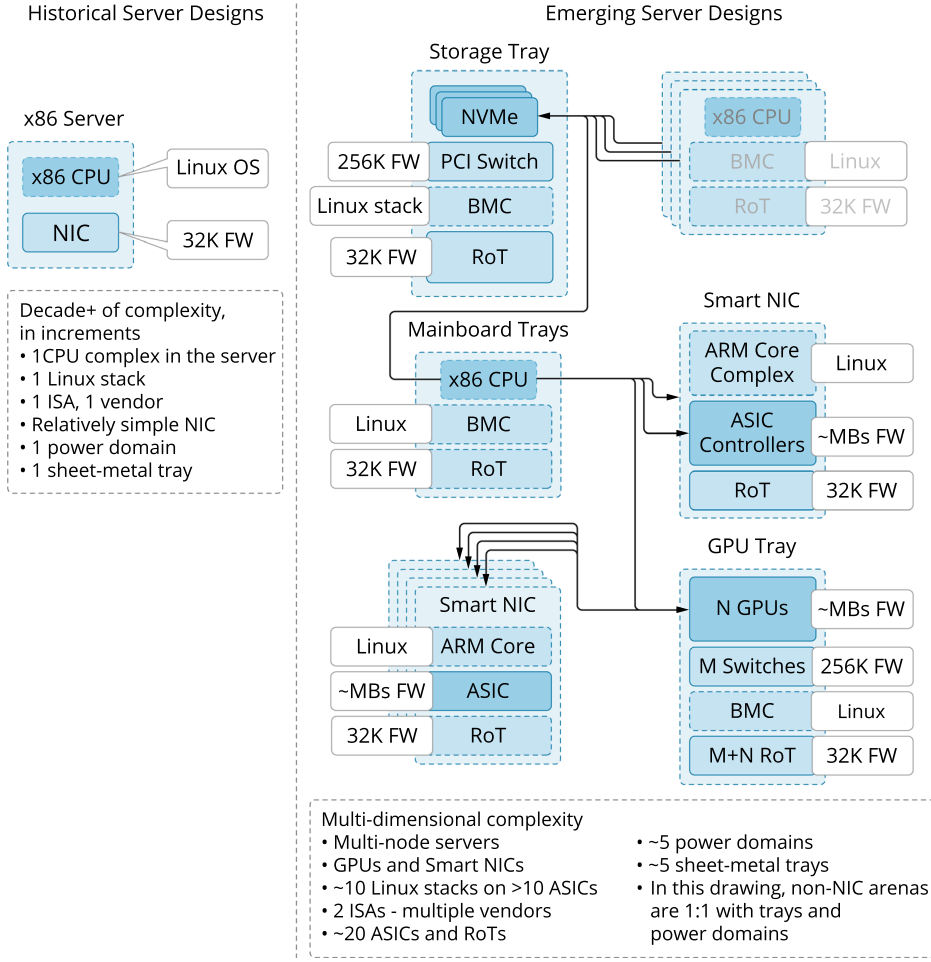


Fig. 6.10 Single-node vs. multi-node "multi-brained" systems

server-specific code. Similarly, server management and monitoring (e.g., fan speed control) has moved to a separate card (called Baseboard Management Controller, or BMC). Each of these arenas contains its own firmware stack, protected by a hardware root of trust. Chapter 7 will discuss how software-defined infrastructure further abstracts hardware heterogeneity through software wrappers and management.

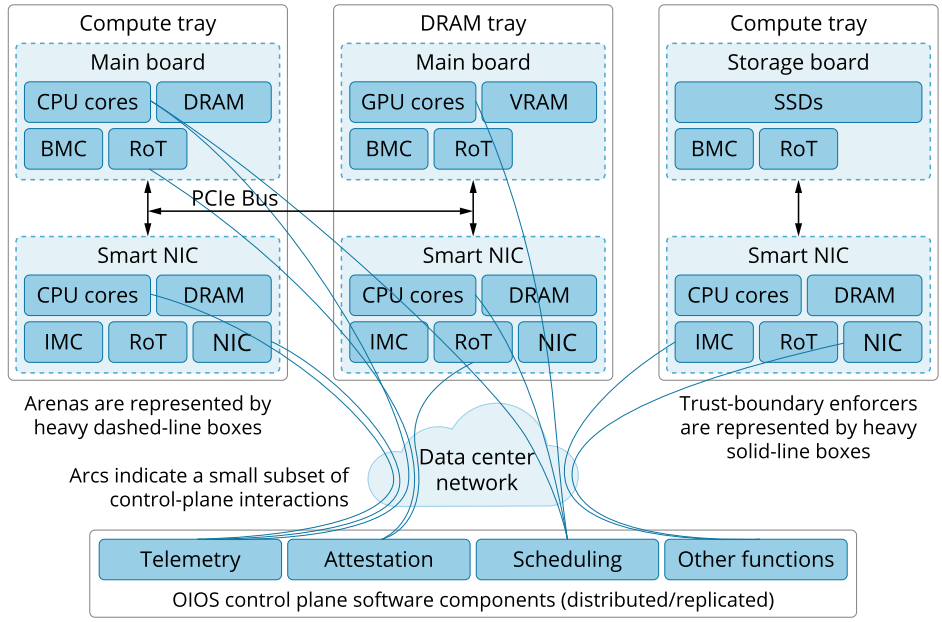
Figure 6.11 sketches a representative multi-node machine and indicates how the fleet control plane interacts with the nodes. Heavy dashed lines, denoting the arenas (hardware trust boundaries). The machine's model precisely defines what firmware within an arena must be attested via its RoT. The picture also shows the complex interactions managed by the control plane across telemetry, attestation, scheduling, etc. For simplicity we show only a subset of the arcs.

The control plane design is similar to prior approaches (e.g., software-defined networking): we aim for a clear distinction between a small set of boundary-enforcing functions local to each arena, and a broader control plane function that manages the fleet as a whole. In addition, we rely heavily on machine, rack, and network models to support the centralized control plane. For example, we use a modeling language that allows us to represent each element of the figure at multiple levels of abstraction, through entity-relationship graphs that also define reachability (which controller services can connect to which control nodes), failure domains, and attestation domains (which root-of-trusts can attest which mutable firmware storage elements).

The second half of the picture shows one use-case, namely attestation, that is managed by the job scheduler. Because workloads should only run on machines with an attested state, we chose to use our fleet-wide job scheduler as the central attestation authority. When the scheduler is asked to include a new machine in its resource pool, it asks the machine for its signed attestation policy. The policy contains a list of roots of trust in the machine and the attestations we expect to see from those RoTs. The scheduler then verifies the policy's signature and checks the policy's revocation ID against a revocation list. If the policy is invalid, the scheduler rejects the machine.

We generate a new policy (and revoke older ones) each time our management processes upgrade (or downgrade) the firmware for a machine, based on the compute elements and RoTs found in the machine's model. To avoid having the scheduler depend on another service, which can undermine availability, we store the policy for a machine on the machine itself, cryptographically signed by the certificate authority. If the policy's signature is valid, the scheduler sends RPCs to the individual arena controllers to gather attestation statements from the RoTs for each device listed in the policy. If every attestation matches its policy intent, the scheduler adds the machine to the cluster and allows jobs to be scheduled on it.

For additional details, we refer the reader to the more detailed technical paper and related contributions at OpenCompute [15], [16]. By constraining our hardware designs to meet the arena definition, and moving per-machine knowledge from code to models, we have been able to simplify reasoning about hardware trust, scale management at WSC scale to



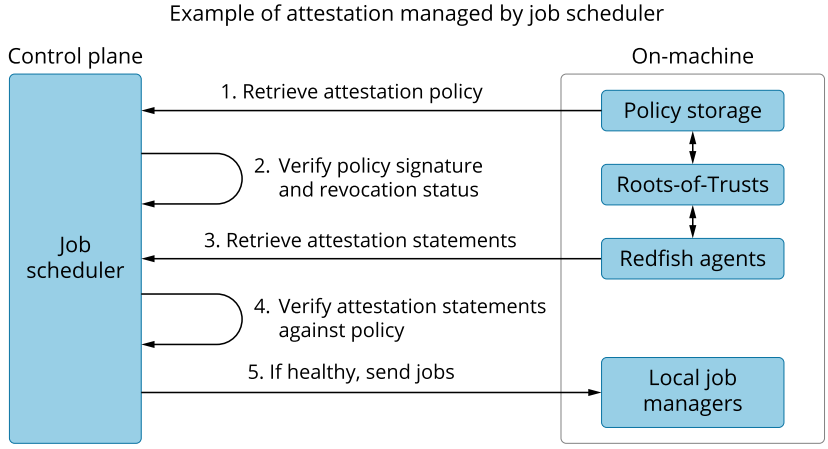


Fig. 6.11 Multi-arena architecture for servers

increasing server complexity, and greatly reduce the software engineering cost (and delays) of introducing new server configurations.

6.3 Accelerators and custom silicon

In the prior edition of the book, we discussed the introduction of specialized computing accelerators. Their adoption has since expanded exponentially, with a host of new accelerators for machine learning, video processing, networking, data movement, security, etc.

6.3.1 Specialization and the end of Moore's Law

Historically, when we designed the first WSCs, we shied away from deploying special-purpose computing. While it promised greater computing efficiencies, those benefits came at the expense of specializing to a limited number of workloads that could only benefit from them, antithetical to the fundamental WSC principles of scale and volume. Additionally, the longer lead times and higher development costs of specialized hardware risked being obsoleted by the rapid improvements of general CPUs: when CPU speeds double every year, and an ASIC takes two years to deploy, then it needs to provide a speedup of much more than 4x to amortize its development costs. However, with the slowing of traditional technology scaling – first, Dennard scaling for power efficiency, and later Moore's law scaling for cost efficiency – we are no longer seeing compounding high growth rates in general-purpose CPU performance per unit cost.

At the same time, new kinds of processing needs can emerge and become large enough to require specialization. A key motivation for building TPUs was the observation that if every Android user needed only a few minutes of voice recognition capabilities, we would have needed to double the entire computing capacity of Google at that time (2013). Optimizing hardware for large classes of workloads through special-purpose accelerators such as GPUs or TPUs was the only option to address this supply-demand gap.

6.3.2 ML accelerators and TPU systems

Deep neural networks (DNNs) and AI workloads (described in Chapter 3) are computationally intensive. They are characterized by the need for fast linear algebra operations, large-scale distributed computation, and high bandwidth memory. AI workloads encompass multiple phases with distinct computational needs. Training requires hundreds or thousands of chips to accommodate large models with many parameters and large volumes of training data, while meeting time-to-convergence requirements. Inference, or ML serving, is user-facing with strict latency deadlines but can run in smaller pods. Fine-tuning and dis-

tillation occupy a middle ground, but with unique considerations (e.g., large models with small datasets on moderate-scale systems). The rise of Large Language Models (LLMs) has amplified these demands (recall Figure 3.8 in Chapter 3).

This demand has driven the development of specialized domain-specific accelerators and systems for AI. GPUs (Graphics Processing Units), originally used primarily for graphics rendering, have emerged as a cornerstone of modern ML acceleration. Google has designed and deployed multiple generations of even more specialized accelerators for AI, called TPUs (Tensor Processing Units). Besides GPUs and TPUs, the industry has explored and introduced a range of other accelerators. Microsoft has utilized FPGAs through the Catapult and Brainwave programs [17], and later introduced the dedicated ASIC Maia [18]. Meta has developed MTIA (Meta Training and Inference Accelerator) [19], while Amazon has introduced its Inferentia and Trainium systems [20], [21].

6.3.2.1 TPUs

Google's Tensor Processing Units (TPUs) are domain-specific accelerators and systems purpose-built for AI. Their evolution (Fig. 6.12) reflects the rapidly changing landscape of AI, with each generation introducing architectural innovations to address emerging challenges and opportunities. TPUv1, launched in 2015, was focused on accelerating inference. TPUv2 added support for training, featuring a high-bandwidth torus network connecting multiple chips. TPUv3 further scaled performance with liquid cooling, enabling more powerful chips and larger systems. TPUv4 introduced reconfigurable optical interconnects through optical circuit switches (OCSes), allowing for flexible system configurations and increased resilience. It also added SparseCores, specialized dataflow processors optimized for the irregular memory access patterns of recommendation models. The TPUv5 and TPUv6 (Trillium) systems further scaled per-chip performance, memory, and maximum pod sizes.

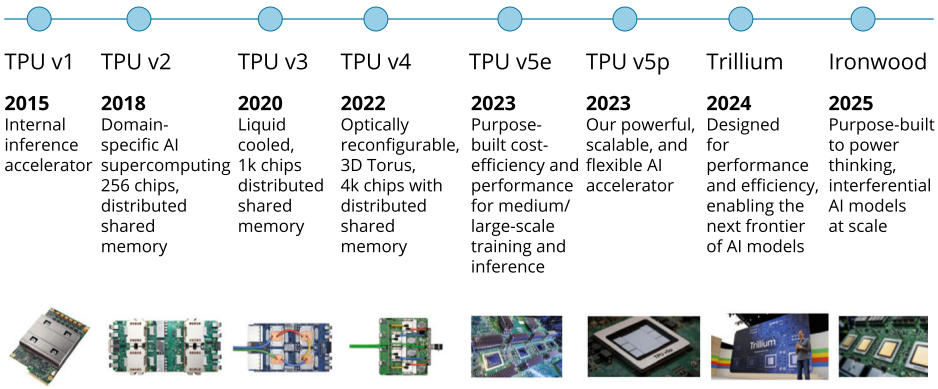


Fig. 6.12 TPU generations over the years

Figure 6.13 depicts the key blocks of a TPU chip: the TensorCore comprising a matrix multiply engine (MXU) and a vector processing unit (VPU); the SparseCore; high-bandwidth memory (HBM); and the inter-chip interconnect (ICI) router.

The matrix multiply unit (MXU) is the heart of the TPU. It employs a systolic-array architecture, a network of interconnected processing elements that efficiently execute a matrix multiply in a pipelined fashion. Each processing element performs simple operations (multiply and accumulate), forwarding the result to its neighbors in a regular dataflow. This design eliminates the need for explicit load and store instructions, which consume both cycles and energy (a memory operation consumes about 10x the energy of a floating-point operation). Instead, values flow through the systolic array in a pipelined manner, reducing control and wiring overheads. In a 256x256 systolic array, each operand is reused 256 times. This results in high throughput and energy efficiency for large matrix operations central to deep learning.

Adjacent to the MXU is the vector processing unit (VPU), a two-dimensional (128x8-wide) vector engine. This unit performs vector operations common in DNN workloads (e.g., softmax, layernorm), and is also responsible for streaming data to and from the MXU. The VPU houses a vector memory (vmem) – a large on-chip software controlled scratchpad – and a vector register file. These components are used to prefetch and stage operands from the main high-bandwidth memory (HBM), and hold outputs and partial sums from the MXU.

The SparseCore is a specialized dataflow processor; it complements the TensorCore and handles bandwidth-intensive, irregular memory operations arising from sparse tensors. The SparseCore features specialized engines that service variable-sized dynamic data streams, allowing effective handling of fine-grained scatters and gathers while operating at HBM bandwidth.

High-bandwidth memory (HBM) is essential for DNN accelerators, providing an order of magnitude more bandwidth than conventional DRAM. Finally, the inter-chip interconnect (ICI) router enables communication between multiple TPU chips, implementing a custom protocol for low-latency, high-bandwidth, and energy-efficient communication.

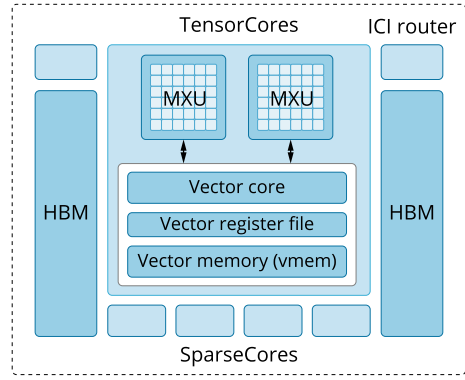


Fig. 6.13 TPU chip block diagram

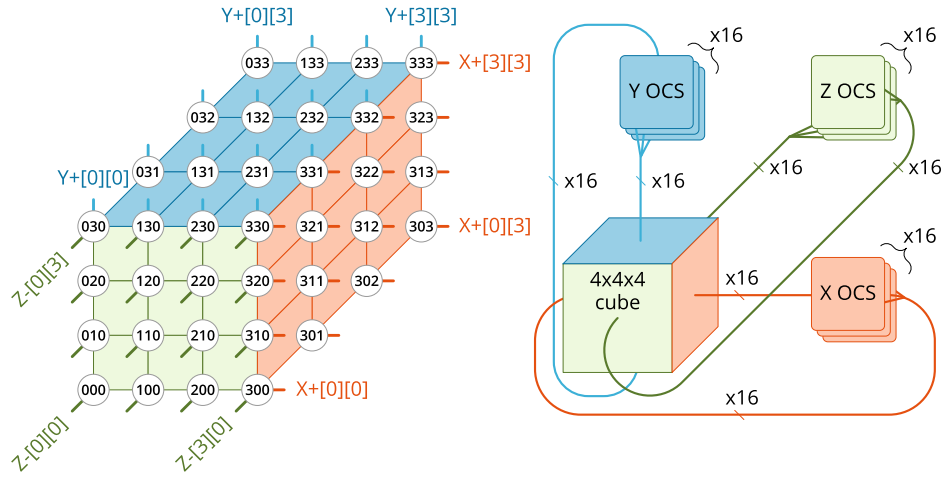


Fig. 6.14 OCS interconnect topology

A TPU supercomputer comprises a network of these TPU chips, interconnected to form a powerful and scalable parallel computing system. Early generations employed a 2D torus network, later evolving to a 3D torus for increased bandwidth and scalability. TPUv4 introduced reconfigurable optical interconnects through optical circuit switches (OCS) to enable even larger scales while maintaining high availability in the presence of failures and enabling incremental deployability (Figure 6.14). A TPUv4 system connects 64 chips using electrical links in a 4x4x4 cube configuration, which forms the minimal building block. These cubes are connected through OCS to form logical pods (of any 3D torus shape) of up to 4096 chips, or 8192 in TPUv5.

OCS improves availability in case of component failures. When any element within a 4x4x4 cube fails, the software instantly reconfigures the OCS to locate a replacement 4x4x4 cube within the data center, maintaining the same logical topology and ensuring seamless system performance for the end-user workload. OCS also facilitates modular and incremental deployment, as the smallest deployment unit is a 4x4x4 cube. Additionally, OCS allows for tailoring the topology to optimize performance for large language models (LLMs). Users can adjust the topology to match the specific type of parallelism (data, model, expert, or pipeline) required by their application. For example, a 512-chip system can be configured as 8x8x8, 4x8x16, and other arrangements to suit the application's needs.

Successive generations of TPUs have also increased peak FLOPS, bandwidth and networking capabilities, enhancing both performance and energy efficiency at large scales.

TPUs have made judicious, yet opinionated co-design bets over the years. They incorporated large systolic arrays (128x128, 256x256), betting that matrix multiplications would remain central to deep learning workloads and grow in size. Anticipating the need for

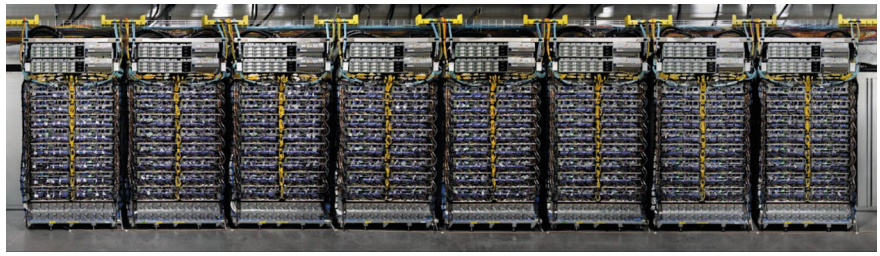


Fig. 6.15 Illustrative TPU pod (4096-chip TPU-v4 pod)

increased scale in DNN workloads, TPUs invested in unique networking capabilities to support large-scale pods (Figure 6.15).

TPUs pioneered low-precision numerics by introducing the bf16 floating point format ("Brain floating point", named after the Google Brain research team that created it). Deep learning models often don't require the high precision of 32-bit floating point. Bf16 reduces the mantissa bits (which determine precision) while keeping the same exponent bits as FP32. In contrast, the existing IEEE fp16 format reduced both mantissa and exponent, making it unsuitable for representing typical DNN weights which typically are very small. Progress in model training has further reduced the need for precision, so accelerators support even shorter formats like bf8 and bf4 today.

TPUs emphasized a robust software stack and programmability, with the XLA compiler performing key optimizations such as fusion and auto-sharding for LLMs, which are critical for performance and efficiency. Finally, TPUs are consistently optimized for improving performance and performance per watt (a rough proxy for performance/TCO [22]). These advancements and design choices have kept pace with the rapid evolution in AI, and enabled state-of-the-art recommendation systems and large language models [23] (Fig. 6.15).

6.3.2.2 GPUs

Graphics Processing Units (GPUs) are specialized processors designed for parallel processing. Each GPU chip houses numerous SIMT (Single Instruction Multiple Threads)-style cores, leveraging massive parallelism to achieve high computational throughput. Originally (in the mid-1990s) GPUs focused on accelerating graphics processing, thus their name. A decade later (in 2006), GPUs split into traditional graphics-oriented accelerators and "general purpose" GPUs (GP-GPUs) aimed at the high-performance computing market. Instead of ray tracers and pixel shaders, these GP-GPUs accelerated full 32-bit and 64-bit floating point vector operations.

A few years later, ML researchers discovered that GP-GPUs could be used to accelerate neural network training, greatly advancing the field of image recognition with the GPU-based AlexNet in 2012. Today, the general-purpose GPUs are augmented with specialized

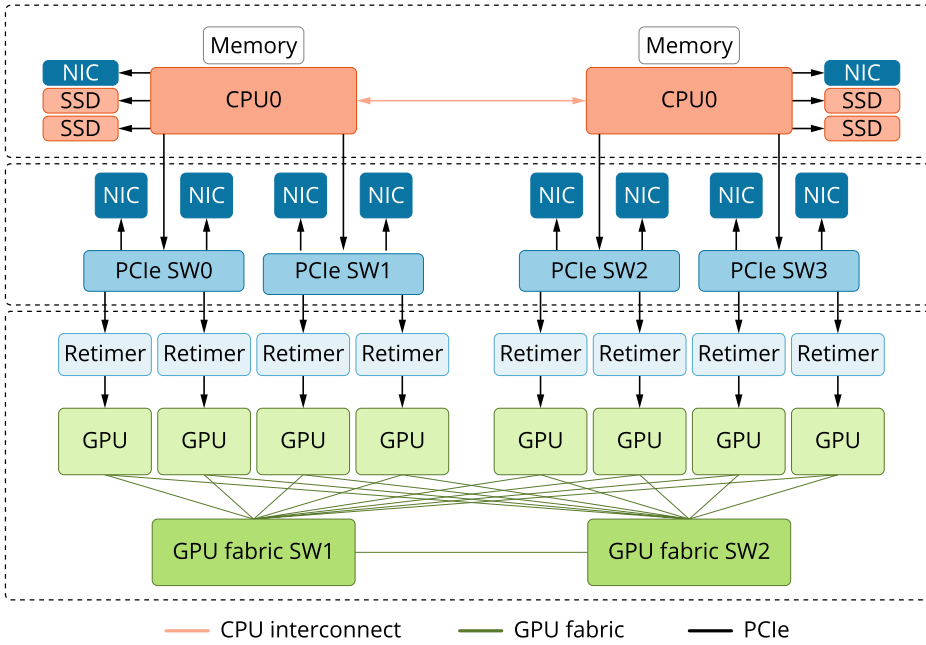


Fig. 6.16 GPUs connected to servers

matrix multiply engines that accelerate key operations in DNNs, and are now called just GPUs again.

GPU deployments for deep learning rely on multi-node configurations to maximize both computational throughput and communication efficiency. A typical system includes a CPU host connected to accelerators via high-speed PCIe links (Figure 6.16). Since GPUs can process trillions of operations per second, they require trillions of bits per second to balance the system, much more than the 100 Gbps or so provided by a standard server. Consequently, GPU deployments contain up to three separate networks. The first network (top) is the regular server network for the CPUs; typically, it's the lowest bandwidth of the three. The second network contains NICs (usually Ethernet) connected to the GPUs via PCIe, providing additional bandwidth to storage or to other GPU racks. (Unlike TPUs, GPUs don't have a specialized network spanning multiple racks.)

Each tray houses multiple GPU chips, interconnected by the third network, a high-bandwidth specialized GPU-GPU interconnect (like NVLink from Nvidia). This third network enables rapid GPU-to-GPU data exchange and minimizes communication latency, serving a role similar to the TPU's ICI network.

Additionally, some GPUs (e.g., NVIDIA's Grace-Hopper architecture) also feature a high-bandwidth path between the host CPU and the GPU, replacing the convenient but slower PCIe links previously used for that purpose .

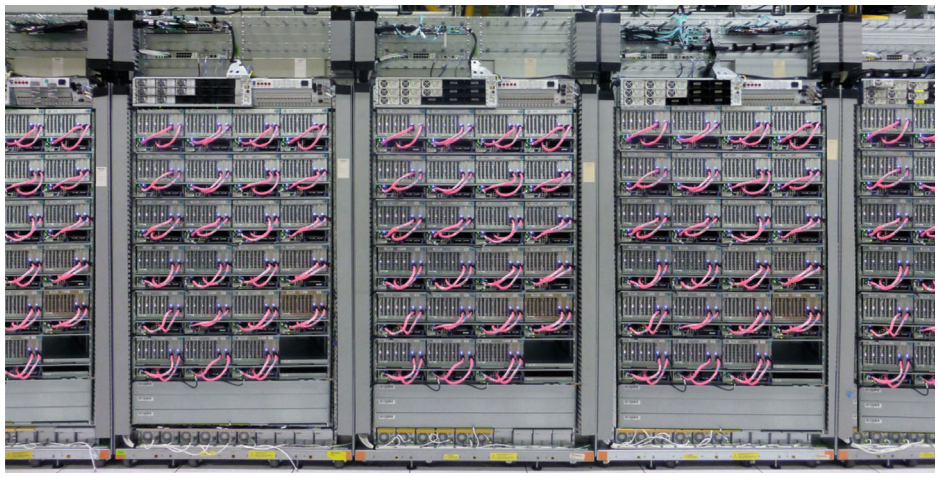


Fig. 6.17 GPUs racks with 2U server trays connected to 8U GPU trays

Beyond individual nodes, large-scale training often involves interconnecting multiple GPU-equipped system-nodes. High-performance networks, such as InfiniBand and high-speed Ethernet, play a crucial role in facilitating fast parameter synchronization across distributed systems during synchronous training. Specialized software libraries, like NVIDIA Collective Communication Library (NCCL), are further optimized for collective communication operations like all-reduce and all-gather, minimizing communication overhead and maximizing the utilization of these high-bandwidth networks. Figure 6.17 shows a network-connected pod of GPUs used for training.

Successive GPU generations have consistently increased FLOPS, memory capacity and bandwidth, and networking capabilities. Additionally, investments in lower-precision numerics (from bf16 to fp8 and fp4) improve peak FLOPS and enable larger models.

6.3.3 Video accelerators and VCU systems

Building on our successful experience introducing TPU accelerators for machine learning at Google, we focused on another foundational workload: video processing. Video is the dominant source of internet traffic [24]. Video processing underpins several important workloads on the cloud: video streaming, video conferencing, online gaming, etc. The growing popularity of live-streaming combined with higher video quality/resolution (e.g. 4K/8K videos) has correspondingly contributed to an exponential increase in the computational resources needed for video processing. In comparison, video serving (i.e., actual streaming) is computationally inexpensive since the processed video blocks are simply copied to a NIC.

Video processing poses unique challenges because of the variety in video workloads:

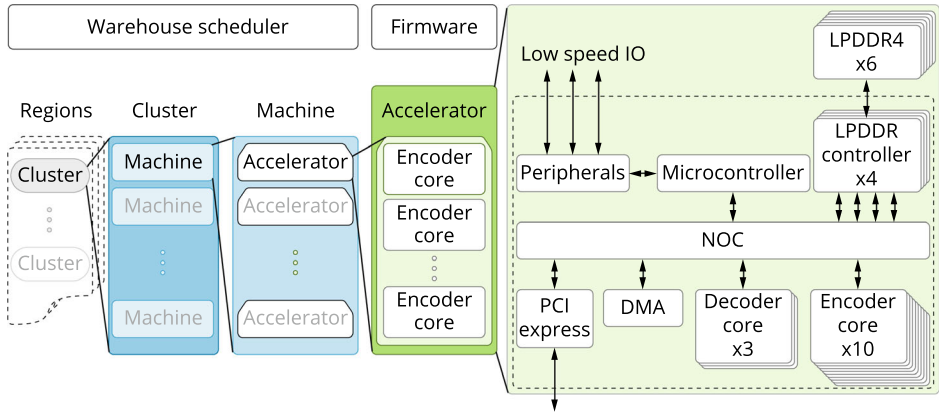


Fig. 6.18 Overview of VCU system architecture at all scales

- Diverse Formats and Resolutions: Every input video is transcoded into various output formats and resolutions to support different end-user devices and network conditions.
- Complex Algorithms: Video transcoding involves computationally intensive algorithms for decoding, scaling, and encoding, and can trade off compression cost, throughput, and perceptual quality.
- Diverse Workloads: Different video products (e.g., YouTube, Photos, Online gaming) have varying requirements for latency, storage, and playback quality.

Of the various components of video processing, video transcoding represents the most important component. Compared to solutions available for consumer devices, Cloud use cases demand higher throughput, quality, efficiency, flexibility, and reliability, but are able to run at higher power envelopes. These unique requirements lead to a fundamentally different approach for cloud-based video transcoding hardware.

A Video Coding Unit (VCU) is a hardware accelerator designed to provide flexibility and efficiency in a warehouse-scale distributed system [25]. Software-hardware co-design informed the system architecture (see Figure 6.18) from the individual hardware blocks to boards, nodes, and geographically distributed clusters, and across hardware, firmware, and distributed systems software.

The VCU SoC is optimized for throughput and density, consisting of multiple encoder and decoder cores, along with the supporting memory subsystem (see Figure 6.19). Firmware-managed queues allow concurrent operations to achieve high encoder utilization. The VCU supports frame-level operations such as encoding a frame to VP9 bitstream. Each application has a command queue along with a persistent stream state in VCU memory.

Codec-specific programming happens in user-space; the firmware is oblivious to the contents of the commands, it just sends them to the next available core. This design allows application software to be updated at the typical data center cadence (multiple times per

week), and has enabled improvements to encoding efficiency throughout the lifetime of the VCU.

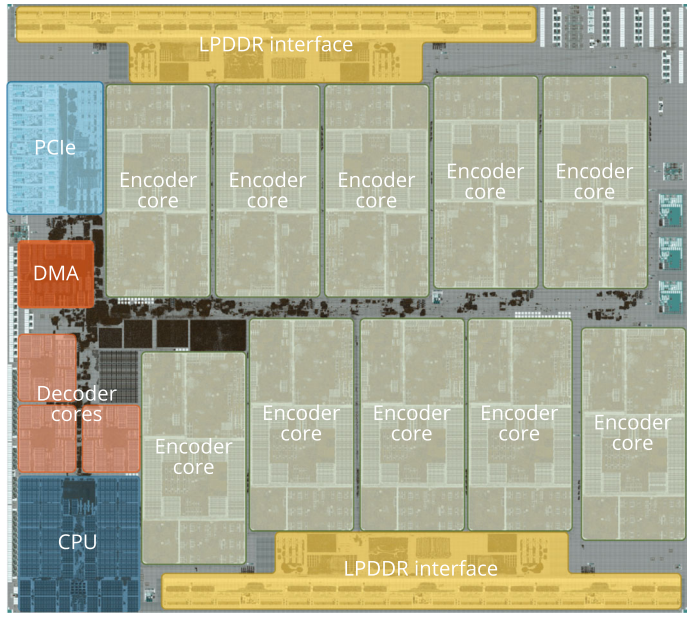


Fig. 6.19 VCU chip floorplan

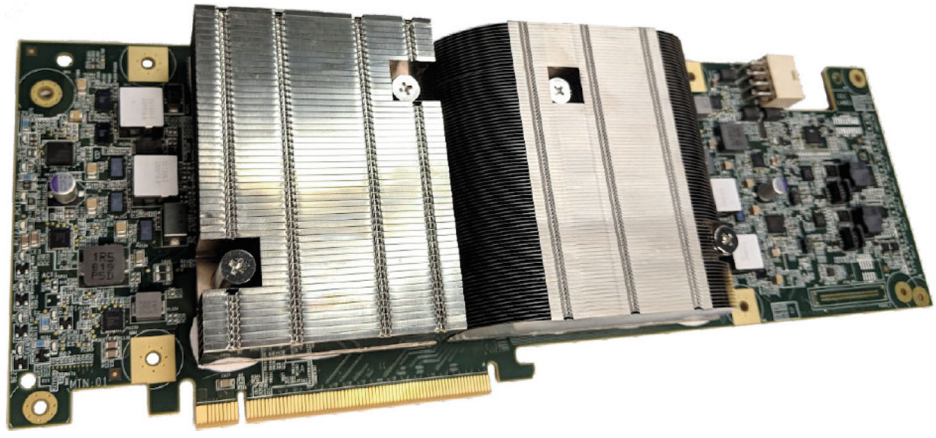


Fig. 6.20 Two VCU chips on a board

Two VCU ASICs are integrated on a PCIe board (Figure 6.20). Ten boards are fitted per system, and racks of these systems are deployed across clusters within data centers. The high VCU density per system improves performance per TCO dollar, and we maintain high reliability as VCU hardware errors do not impact the full system.

In line with the data center as a computer paradigm, our cluster scheduling system directs transcoding to VCU systems and non-transcode processing to non-VCU machines. Logical worker pools are shaped out of VCU or CPU transcoding resources, and work is scheduled onto a pool based on use case and priority. A multidimensional bin packing scheduler maps work items to workers based on resource dimensions that represent both physical resources, such as encoder cores and memory, and hard-to-attribute metrics like PCIe bandwidth. The scheduler aims to maximize load on workers, such that idle workers can be released from a logical pool to be used by another pool. In the event of a transcoding failure, the work is rescheduled within a logical pool.

VCUs have been deployed for years and benefitted products like YouTube, Google Meet, and Google Photos. VCU provides an order of magnitude improvement in efficiency relative to a well-tuned CPU based system. We use Performance per Total Cost of Ownership (Perf/TCO) as a metric for efficiency, where TCO includes capital and operational costs of a given system. Figure 6.21 compares throughput and Perf/TCO for the first-generation VCU relative to CPU and GPU processing. For offline two-pass encoding, the VCU systems provide 8x - 20x higher encoding throughput than software encoding, for H.264 and VP9, respectively. The value of hardware acceleration is more pronounced for novel video coding standards where the compute load has increased to provide higher compression gains. VCU provides a break-through in compute capacity at scale, and has enabled new capabilities such as multiple-output transcoding (MOT), which was previously cost prohibitive, especially for VP9.

The evolution of VCU enables new possibilities for advanced video processing at scale, and provides improvements to encoder quality, throughput, and efficiency. The addition of AV1, the next generation high-quality open codec, highlights the nuanced challenges of video encoding. While AV1 demands 5 - 10x more compute resources than VP9 on traditional CPUs, the VCU design space offers a broader range of trade-offs. The complexity of AV1 can be addressed by trading off chip area and power for higher throughput.

Fig. 6.21 VCU transcode throughput in VCU versus CPU and GPU systems

System	Throughput [Mpix/s]		Perf / TCO	
	H.264	VP9	H.264	VP9
Skylake	714	154	1.0x	1.0x
4xNvidia T4	2,484	—	1.5x	—
8xVCU	5,973	6,122	4.4x	20.8x
20xVCU	14,932	15,306	7.0x	33.3x

Beyond new codec support, the next generations of VCU introduce features that require cost efficiency to scale. We expand the suite of hardware-accelerated scaling and preprocessing algorithms to enable video filters that are too costly on CPUs or GPUs, including support for ML models operating at frame or subframe levels. Additionally, while new codecs like AV1 take nearly a decade to achieve widespread adoption due to their reliance on hardware decoders in client devices, ongoing algorithmic improvements to existing H.264 and VP9 codecs remain valuable throughout a VCU's lifespan. New VCU generations also include performance improvements such as redesigning the encoder core functionality to maximize encoder busy cycles, increasing the encoder clock frequency, and adopting new process technology, PCIe, and memory architectures to keep up with higher encoder throughput.

6.4 Networking

WSCs are large-scale distributed systems, across racks, clusters, data centers, campuses, and even multiple regions of the world. Thus, the networking subsystem is fundamental to the design of WSC systems.

6.4.1 Overall networking from tray to planet

The journey of a data packet starts as a simple piece of information, perhaps a snippet of an email, a fragment of a video, or a command from a user. This data originates within an application running on a host machine (often in a VM) and is handed off to the host networking stack. The prepared data is handed over to the network interface card (NIC or smartNIC) which encapsulates the data into a frame, adding headers and a checksum for error detection. At this point the packet is ready to embark on its journey. It might travel across the cluster network to a destination machine within the same cluster. Alternatively, it could continue via the campus or Wide Area Network (WAN) to a machine in a remote cluster. The packet might even go to the broader Internet through edge networks that connect with ISP providers. The networking stack encountered in this journey comprises multiple elements, often with several different configurations and generations. Below, we focus on describing the networking architecture with a focus on a circa-2023 design. More details are available in several Google papers [26], [27], [28], [29], [30], [31], [32].

6.4.2 Host networking and smartNICs

WSC hosts require high throughput and low latency, low tail latencies, traffic shaping (pacing and rate limiting), OS-bypass, stringent security and line-rate encryption, debuggability, QoS and congestion control, etc., all with low host CPU use. Public clouds further require

features to support virtualization and VM migration. These requirements can be met with two complementary approaches: onload, where the host networking software leverages host CPU to provide low latency and rich features, and offload that uses compute in NIC cards for functions such as packet processing and cryptographic functions. Example implementations include Azure's FPGA bump-in-the-wire NIC [33], Amazon's customized NIC for bare-metal support, Google's host-side traffic shaping [34], and Google's Andromeda network virtualization [35].

Below, we describe an Infrastructure Processing Unit (IPU) used at Google.

One key tenet of Google's IPU approach is to separate the server control plane from the server data plane on the same set of CPU cores. The server's control plane is responsible for monitoring the server, virtualizing hardware resources, container lifecycle, network and storage I/O processing, logging, and a growing list of other functionality unrelated to running customer workloads.

Without optimization, server control and virtualization overheads can be significant, in some cases bursting to more than 50% of overall compute capacity. This overhead must either be statically provisioned for the peak, making cost untenable, or artificially throttled near the average, limiting overall server performance. Additionally, running the control plane on the same CPU alongside customer workloads introduces security risks if malicious guests can find and exploit virtualization bugs to become root on the host CPU.

The separation of the server control plane from the server data plane leads to the system server architecture shown in Figure 6.22. In this architecture, a secure control module (DC-

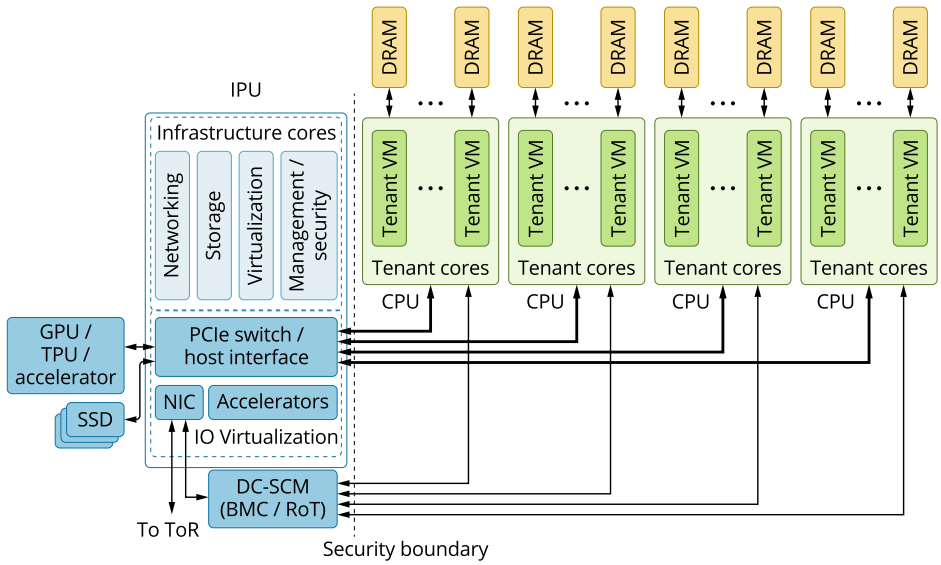


Fig. 6.22 Overview of Google's IPU integration into the host

SCM) attached to the IPU provides the root of trust (RoT) of the entire machine along with integrated RoTs of the attached CPUs. The machine is composed of the IPU itself, one or more CPU / DRAM modules, and a multitude of PCIe SR-IOV based I/O devices including NICs, SSDs, GPUs/TPUs and other accelerators for encryption, compression and other data processing. The IPU includes a PCIe switch as part of its host interface, with hardware acceleration of virtualization features such as flexible physical-to-virtual device mappings, live migration, and tenant isolation. This IPU-based server architecture allows for the fast deployment of CPUs and I/O devices from vendor OTS catalog without compromising security, isolation or performance.

Figure 6.23 shows Google's first Titanium IPU on a PCIe add-in card. The high-level block diagram is shown on the right. This first generation IPU chip, co-developed with Intel and commercially available as the Intel E2100, provides 200Gbps of network connectivity and integrates 16 Neoverse N1 ARM cores. The IPU SoC design runs the server control plane on the offload compute complex that is physically separated from the server data plane, which simultaneously provides stronger security for both customers and Google, and bare metal performance for customer workloads.

Integrating the NIC and data accelerators tightly couples the networking and storage hardware acceleration with the offload compute complex. This enables flexible interweaving

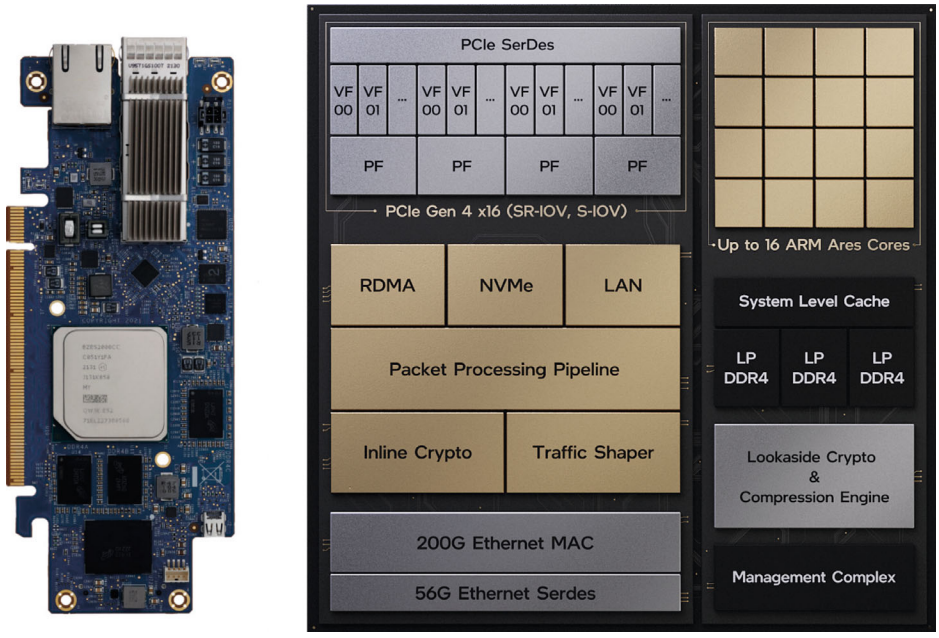


Fig. 6.23 Google's Titanium IPU: PCIe card and key architectural blocks

of hardware and software packet and command processing. This integration also enables flexible I/O virtualization and resource disaggregation, and significantly reduces overall system TCO.

6.4.3 Cluster networking

WSC clusters must interconnect thousands of servers at high speed. Unfortunately, we can't just buy (or make) an arbitrarily large switch. At any given time, the available technology limits total per-chip capacity because all chips are power- and pin-limited. For example, as of 2024, a typical merchant silicon switch chip supports a bisection bandwidth of up to 51.2 Tbps (128x 400 GbE ports) and no chips are available that can deliver more than 200 Tbps.

We can build larger switches by cascading individual switch chips, typically in the form of a fat tree or Clos network [36], [30]. Clos networks are named after Charles Clos, who first formalized their properties in 1952. The basic idea is that each leaf switch divides its ports into downward (server) facing ports and upwards (spine) facing ports. In a fully connected tree, half of the ports go each way, so if we have k-port switches, $k/2$ ports connect to the servers and the others connect to one of $k/2$ spine switches. With this topology, every leaf switch has $k/2$ paths to every other leaf switch, each path going through a different spine switch. In real networks, this concept is applied recursively: a switch may be combined inside a chassis to form a larger switch, and multiple switch chassis may be combined to form a "block" that looks like an even larger virtual switch, forming a data center network as shown in Figure 6.24.

A tree using k-port switches can support full throughput among $k^3/4$ servers using $5k^2/4$switches, allowing networks with tens of thousands of ports. However, the cost of

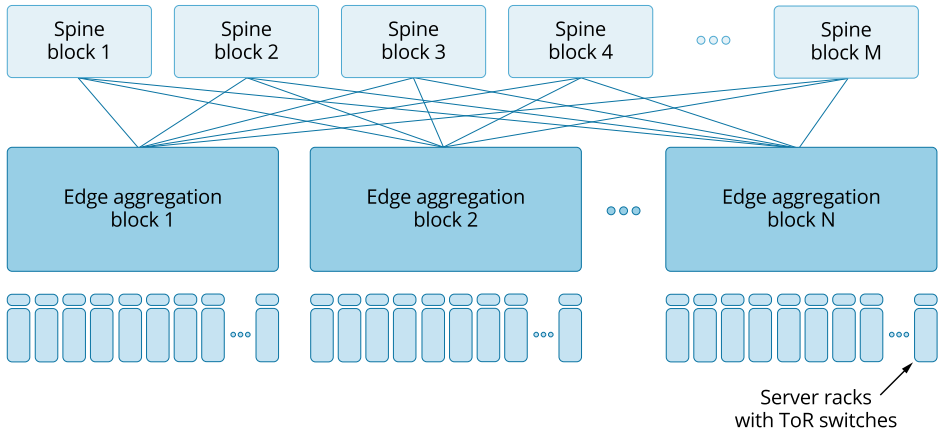


Fig. 6.24 Sample three-stage Clos topology

doing so increases significantly with the number of servers because each path to another server now involves more ports. In the simplest network (a single stage consisting of a single central switch), each path consists of two ports: switch in and switch out. The above three-stage network quintuples that to 10 ports, significantly increasing cost per server since every server port needs to be matched with 8 additional switch ports in the tree. So, as the size of the cluster (and thus the bisection bandwidth) grows, the cost per connected server grows as well.

Port costs are substantial, especially if a link spans more than a few meters, thus requiring an optical interface. Today the optical components of a 100 m 100 Gbps link can cost several hundred dollars (including the cost of the two optical ports, fiber cable, fiber termination and installation), not including the networking components themselves (switches and NICs). Thus, in modern data center networks, optics represent the largest single cost item. Since the optical links inside the data center have different reach (i.e., maximum distances), different types of optics optimize cost and power based on reach, link budget and signal-noise-ratio (SNR) requirements. Since most links inside the data center are short, optimized versions can reduce costs substantially. Even the very first iteration of Google's interconnect (Saturn) used nonstandard optics to reduce costs [37].

To further reduce networking costs per machine, WSC designers often oversubscribe the network upwards ("north") of the top-of-rack switch. Providing full bandwidth inside the rack is fairly inexpensive since the rack contains a small enough number of servers to be connected with a single switch. It's not hard to find switches with 64 ports to interconnect 64 servers at full NIC speed (say, 200 Gbps). In a full fat tree, each such switch would need the same number of ports facing the cluster fabric. All those upwards-facing links in turn require more links in the aggregation and core layers, leading to an expensive network.

In an oversubscribed network, we increase that 1:1 ratio between server and fabric ports. For example, with 2:1 oversubscription, we build a fat tree for only half the aggregate bandwidth of the servers connected to the rack switches, reducing the size of the tree and thus its cost, but also reducing the available bandwidth per server by a factor of two. Each server can still peak at 200 Gbps of traffic, but if all servers are simultaneously sending traffic, they'll only be able to average 100 Gbps. In practice, oversubscription ratios of 3 and higher are common. For example, a 64-port rack switch with 200 Gbps ports could connect 48 servers to 16 uplinks, for a 3:1 oversubscription (66.7 Gbps per server).

Figure 6.25 shows the structure of Google's Jupiter Clos network [38] first deployed more than a decade ago with a bisection bandwidth of 1.3 petabits per second. This multi-stage network fabric used low-radix switches built from merchant silicon, each supporting 16x40 Gbps ports. Each 40G port could be configured in 4x10G or 40G mode. A server was connected to its ToR switch using 10Gbps or 40Gbps Ethernet NICs. Jupiter's primary building block was the Centauri switch, a 4RU chassis housing two line cards, each with two switch chips. In an example ToR configuration, each switch chip is configured with 48x10G to servers and 16x10G to the fabric, yielding an oversubscription ratio of 3:1. Servers could also be configured with 40G mode to have 40G burst bandwidth.

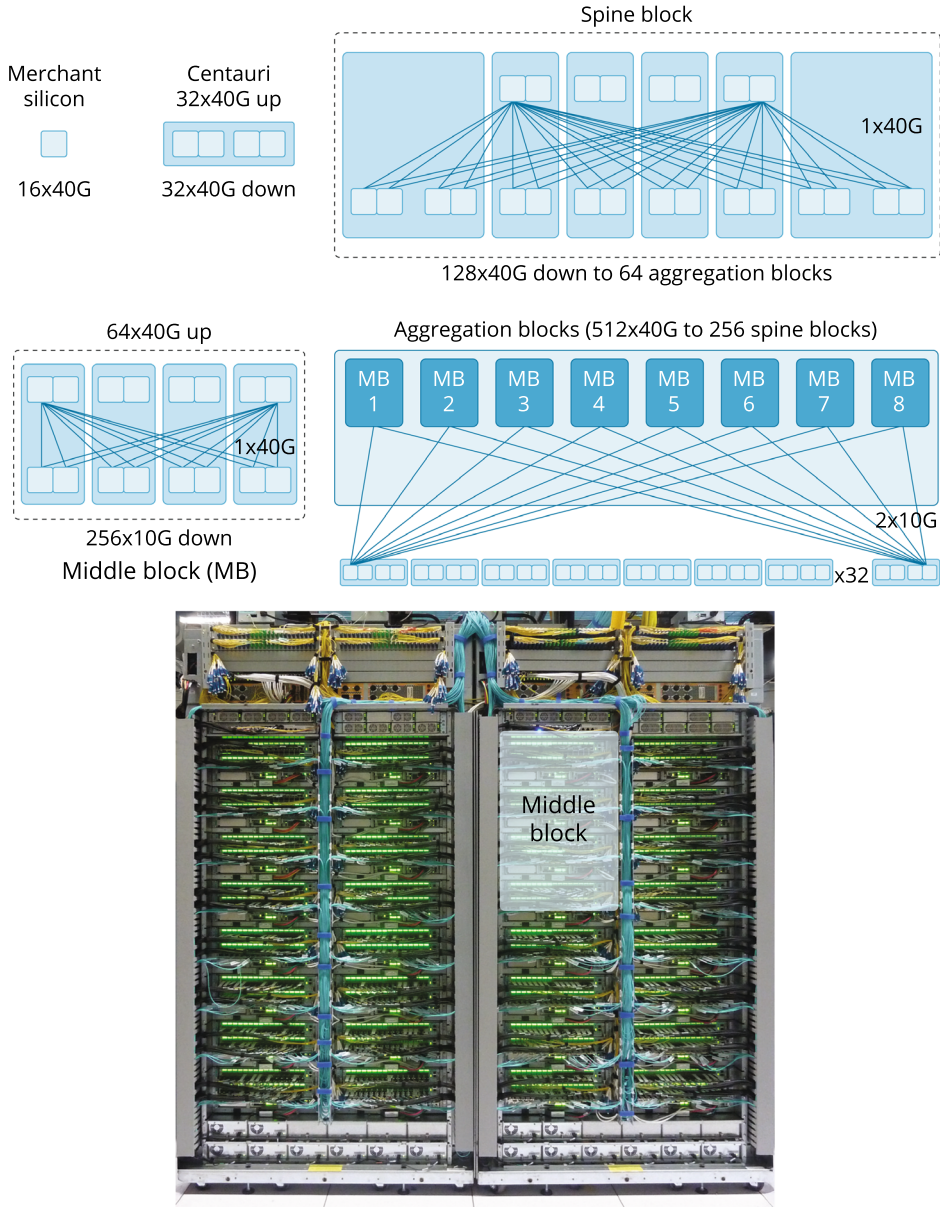


Fig. 6.25 Jupiter network topology and photo of Jupiter rack

The ToR switches connect to layers of aggregation blocks to increase the scale of the network fabric. Each Middle Block (MB) had four Centauri chassis. The logical topology of an MB was a two-stage non-blocking network, with 256x10G links available for ToR connectivity and 64x40G available for connectivity to the rest of the fabric through the spine blocks.

Jupiter employs a separate aggregation block for external connectivity, which provides the entire pool of external bandwidth to each aggregation block. As a rule of thumb, 10% of aggregate intra-cluster bandwidth is allocated for external connectivity using one to three aggregation blocks. These aggregation blocks are physically and topologically identical to those used for ToR connectivity. However, the ports normally employed for ToR connectivity are reallocated to connect to external fabrics.

With Jupiter, the intra-cluster fabric connects to the inter-cluster networking layer with Fabric Border Routers (FBRs). Multiple cluster fabrics may be deployed within the same building and multiple buildings on the same campus. The job scheduling and resource allocation infrastructure leverages campus-level and building-level locality.

Compared to WSCs, High-Performance Computing (HPC) supercomputer clusters often have a much lower ratio of computation to network bandwidth, because applications such as weather simulations distribute their data across RAM in all nodes, and nodes need to update neighboring nodes after performing relatively few floating-point computations. As a result, traditional HPC systems have used proprietary interconnects with leading-edge link bandwidths, much lower latencies (especially for common functions like barrier synchronizations or scatter/gather operations, which often are directly supported by the interconnect), and some form of a global address space (where the network is integrated with CPU caches and virtual addresses). Increasingly Ethernet, HPC optimized variants of Ethernet [39], [40], and Infiniband have started to dominate Supercomputing [41] as the performance of commodity interconnects continues to improve rapidly.

6.4.4 Spine-less networking

WSCs are not only deployed at large scale (e.g., 40 MW or more of infrastructure), but the compute, storage and accelerator devices deployed are always evolving, which requires regularly refreshing the native network interconnect speeds from 40 Gbps common just a few years ago to the 400 Gbps of today. Therefore, the WSC network needs to evolve dynamically without turning down the services hosted by the substantial infrastructure already deployed at building scale.

Unfortunately, Clos topologies in data centers need a uniform high-speed spine layer. Thus, the entire spine must be upgraded to accommodate faster port speed in any part of the cluster, which is too costly to be practical. This inflexibility makes it difficult to incrementally add faster devices to the network without disrupting the entire infrastructure.

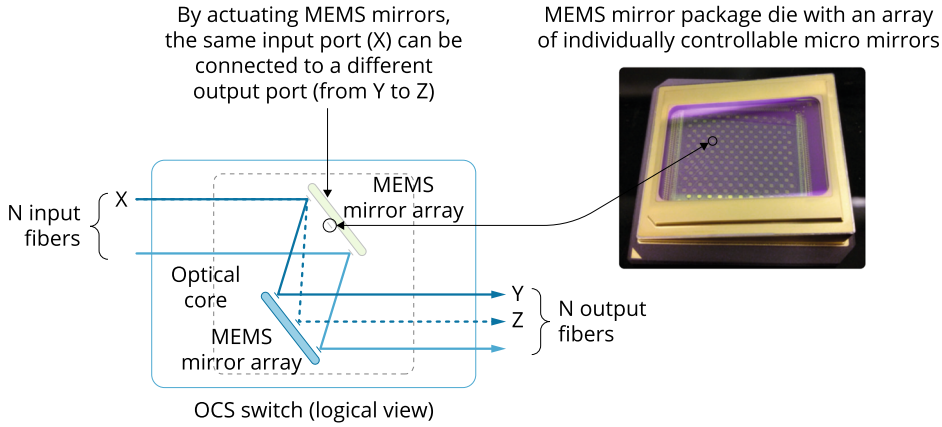


Fig. 6.26 Operating principle of an Optical Circuit Switch

Optical circuit switches (OCS) support heterogeneous speeds and also enable incremental cluster expansions. An optical circuit switch uses two sets of microelectromechanical systems (MEMS) mirrors to dynamically connect input and output fiber ports, creating flexible port-to-port connections (Figure 6.26).

A MEMS mirror array is a collection of tiny, individually controllable mirrors fabricated using microelectromechanically controlled mirrors. The individual millimeter-sized mirrors can be tilted along two axes to precisely control the direction of light beams. All incoming and outgoing fibers are very precisely aligned (collimated) so that the laser beam in each fiber hits its first mirror. To reach output port (x, y) the first mirror steers the beam to output mirror column x, and that mirror is oriented to send the light to the fiber in row y. To keep optical losses low, collimation needs to be incredibly precise because the outgoing laser beam needs to hit the output fiber head-on. Because the OCS merely steers light, it is oblivious to the network protocol, wavelengths, and speeds being used, and thus doesn't need to be upgraded when newer optics are introduced, and simultaneously handles ports with multiple speeds.

By using an Optical Circuit Switch (OCS) layer, the traditional spine layer can be removed from the data center network, enabling a direct optical connection between diverse aggregation blocks (Figure 6.27). To connect or remove a new aggregation block, the OCS simply needs to be reprogrammed to reorient its mirrors to establish the new links. Without OCS, adding a new aggregation block requires unplugging and re-plugging hundreds or thousands of fiber ports to "re-stripe" the Clos mesh, a process that is tedious and error prone.

OCS layers also enable dynamic topologies that adapt to application-specific communication patterns like "elephant flows". For example, if a large computation like an ML training run spans two aggregation blocks and needs more bandwidth, the OCS spine can be reconfigured to allocate additional ports to these two blocks. With OCS, reconfiguring network connectivity has become a standard practice, enabling topology engineering in the data center for the first time.

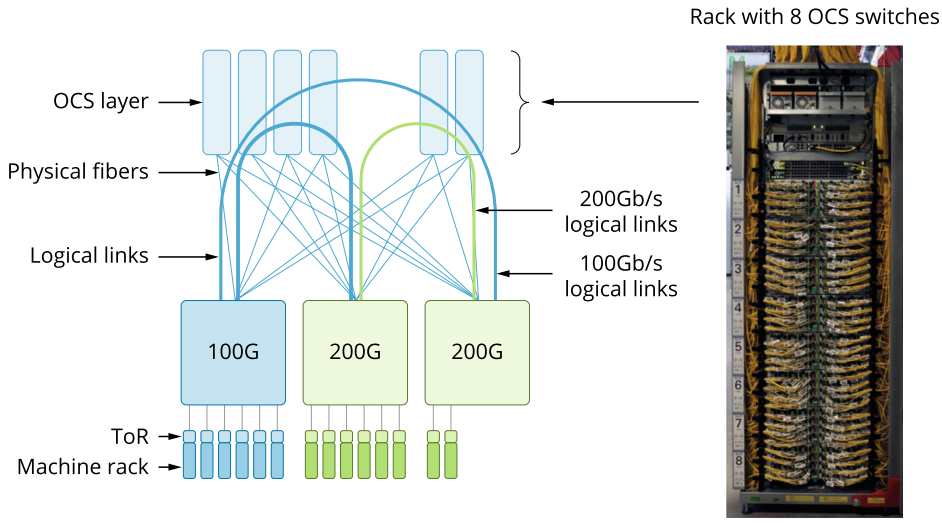


Fig. 6.27 OCS connecting aggregation blocks

6.4.5 Optics in warehouse-scale data centers

Interconnects within a data center use copper links where possible because they're cheaper and more power efficient. However, as link speeds increase, the maximum copper cable length decreases so that today, copper links are only viable inside a rack, for NIC to ToR links and switch-to-switch links within densely packed switch racks.

Active optical cables (AOC) have copper connectors on both ends. Optoelectronic modules in the connector heads convert electrical signals into light. Since the optical interfaces are fully closed, they are cheaper than optical transceivers. AOCs are typically used as alternatives to copper links to extend the reach and reduce the bulk of copper cables. They are used within a row.

Within the data center, optical links use different kinds of optics depending on distance, to optimize cost. Each switch port has a pluggable optic module (using formats such as QSFP, QSFP-DD, or OSFP) to support the necessary configurations and allow for replacement on failure. Short reach optics using parallel fiber are used for the next level optics with a reach of around 100 m. They are often used for ToR to aggregation blocks, where the reach is short and lane breakout/shuffling is needed for better fault tolerance. For longer reaches, and for cases that involve OCSs, wavelength division multiplexing (WDM) is used, where each fiber carries multiple colors of light is used to reduce fiber cost and bulk and make more efficient use of OCS switch ports.

6.4.6 Special-purpose networks

Another way to tackle network scalability is to offload some traffic to a special-purpose network. For example, we could build a separate network to connect servers to storage units. Such networks usually are called Storage Area Networks, or SANs. If that traffic is more localized (not all servers need to be attached to all storage units) we could build smaller-scale networks, thus reducing costs. Historically, all storage was networked this way: a SAN connected servers to disks, typically using FibreChannel networks rather than Ethernet. Today, Ethernet is more common since it offers comparable speeds and protocols such as FibreChannel over Ethernet (FCoE [42]), SCSI over IP (iSCSI [43]), and more recently NVMe [44] allow "converged" Ethernet networks to provide the functionality of traditional SANs. In WSCs, storage traffic may eschew traditional storage protocols entirely; for example, Google's storage layer is accessed via Colossus RPCs (see Section 4.1.2 for details). Similarly, AWS uses SRD, a proprietary protocol over Ethernet [45].

Historically, storage traffic dominated all other traffic (application RPCs) in data centers. More recently, ML traffic (GPU-GPU or TPU-TPU) has emerged as another large source of traffic. Given that these chips can perform hundreds of Teraflops per second, they consume or produce hundreds of Gbps per device, often from or to other chips. Thus, the bandwidth per chip is very high: a TPUv5 node [46] provides 4.8 Tbps of ICI bandwidth for 459 TFLOPs of 16-bit computations, and a H100 GPU supports 7.2 TBps of NVLink bandwidth for 2671 TFLOPs of 16-bit computations. In both cases, this bandwidth is provided by a separate network that's specialized for small block transfers and more closely resembles a shared-memory interconnect than an Ethernet network.

However, these special-purpose accelerator interconnects have limited sizes. As of 2025, the largest TPUv7 pod consists of 9,216 chips, and the largest NVLink pod consists of 256 H100 GPUs. Large training runs need more than that, requiring data center networks to support high-bandwidth interconnects between multiple pods at speeds of multiple hundreds of Gbps per node. Often, this inter-pod network is a separate network as well so that accelerator-to-accelerator traffic travels solely on that network. In contrast, the servers attached to the accelerators contain their own NICs that connect them to the rest of the data center fabric, and application traffic travels solely on that network.

6.4.7 WANs

In this book we focus mainly on the data center aspects of the networking architecture and thus will only briefly cover WAN and edge networking. Cloud providers have developed private WANs to connect their data centers across the globe. Because of internal demands from YouTube traffic, Google Drive, Google Ads serving, etc., Google's network is an order of magnitude bigger than that of other public cloud providers. As of 2024, Google owns or co-owns 31 submarine cables (Figure 4.5 [47]).

WANs are designed to handle massive bandwidth requirements and elastic traffic demand, and use software-defined networking (SDN) and traffic engineering to optimize performance. Cloud providers have also developed SDN-based Internet peering edge infrastructures. These infrastructures are designed to scale cost-effectively and to enable application-aware routing at Internet-peering scale. Section 4.1.4 explains how the different WAN elements fit together, and section 7.3 describes the SDN features of WANs.

These innovations have helped cloud providers improve the performance and reliability of their networks, and have also made it possible for them to deploy new networking features with high velocity. For more details on WAN and edge networking, see [48], [49], [29], [50], [51], [52], [53].

6.5 Storage

With few exceptions like debug logs or temporary files, persistent storage data needs to be accessible to any server. Thus, storage is a service, not a device type. At a low level, disk or SSD appliances make block devices available to the network, and at a higher level file systems like Colossus or GCS make storage available to applications.

Data processed by WSC workloads tends to fall into two categories: ephemeral and persistent. Ephemeral data usually is private to individual tasks and can be recreated by restarting the task. Persistent data usually is data that forms the shared state of the distributed workload. Ephemeral data tends to reside in local DRAM or local storage, is rarely replicated, and its management is simplified by virtue of its transient nature and single owner. In contrast, persistent data must be much more durable and is accessed by a large number of clients, thus requiring a much more sophisticated distributed storage system. We discuss the main features of these WSC storage systems next.

6.5.1 Storage media

Modern WSCs store vast amounts of data with diverse access bandwidth and latency requirements. These range from archival data which may be stored for decades but is rarely accessed, to scratch data that may be overwritten after only a few seconds but is a critical performance bottleneck. No single technology can meet all of these needs and as a result, WSCs use a carefully engineered combination of storage technologies.

Hard disk drives (HDDs) are the only storage technology that has survived from the early days of computing (Figure 6.28). They consist of a stack of spinning platters with a thin magnetic coating. Each platter has a head which can set or detect the magnetic polarization at any location. The head sits in an aerodynamic carriage at the end of an arm which can swing in and out to reach different rings, called tracks, on the platter. To read a particular block, the arm swings to ("seeks") the track containing that block and then the head waits

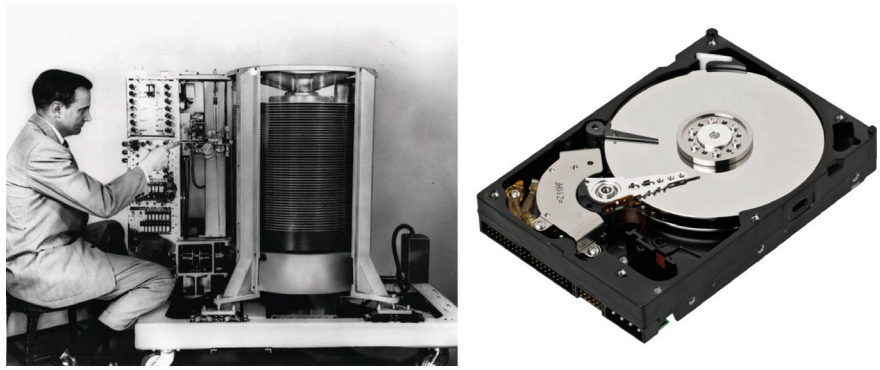


Fig. 6.28 4 MB IBM 350 hard disk in 1956, 4 TB 3.5" disk in 2013

until that block passes underneath it. Thus, the average access time is the time to move the head across half the disk, plus half the duration of a full rotation of the platters.

HDDs offer low TCO per byte with acceptable latency. However, the mechanical limitations of spinning disk technology mean that the total I/O bandwidth and access latency has remained roughly constant for decades even as capacity has grown rapidly. Consequently, available I/O per byte has declined dramatically: streaming all bytes off of a 25 TB disk at a typical maximum throughput of 200 MB/sec will take 25,000/0.2 = 125,000 seconds, or 35 hours! That trend caused famed storage pioneer Jim Gray to proclaim in 2006 that "disk is the new tape" [54].

WSC requirements for hard disk drives (HDDs) are different from enterprise requirements, needing different tradeoffs in the design of disks, tail latency, and security [55]. A decade ago, hard drives were primarily designed for enterprise servers, but today most disks go into WSCs and thus optimize for the hyperscaler use cases. The key goals for disks are high I/Os per second (IOPS, typically limited by seeks), high capacity, low tail latency, meeting security requirements, and low total cost of ownership (TCO).

In the next few years, Dual- and Multi-Actuator (DA and MA) HDDs may be introduced to WSC to increase per-drive IOPS. At the same time improvements like Shingled-Magnetic Recording (SMR) and Heat-Assisted Magnetic Recording (HAMR) have further boosted disk capacity growth, so that the overall trend of lower available IOPS per byte is expected to continue, and disks are primarily used to store cold data only, i.e., data that needs to preserved but is rarely accessed (e.g., your backed up photos from a decade ago).

Solid State Drives (SSDs) have largely replaced disks for "warm" or "hot" data. They store data directly on silicon flash memory. SSDs offer an order of magnitude more bandwidth than HDDs, and several orders of magnitude higher IOPS, but also cost several times more per byte. SSDs have limited endurance: each storage cell can only be rewritten a limited number of times, on the order of thousands of times (see Section 10.1.5). After that they lose reliability and performance, further complicating workload management and fleet optimization.

As silicon transistor miniaturization approaches its limits, SSD capacity growth and price reduction are driven by ever deeper 3-dimensional flash memory and more bits stored per memory cell. Higher byte density in particular comes at the cost of higher latency and significantly lower endurance. The TCO per byte gap between HDD and SSD continues to narrow, so more IOPS need to be offloaded from ever-larger HDDs. Thus, SSDs make up an ever-increasing portion of the storage mix in WSCs.

6.5.2 Diskless servers

In 2010-vintage WSC servers, local hard drives provided direct-attached storage and served as the boot/logging/scratch space. Network bandwidth barely reached 1 Gbps per server, and a single disk running flat out could almost saturate that bandwidth. But as network speeds rapidly increased while disk throughput remained relatively static, it became advantageous to move storage outside of servers [56]. Storage capacity and I/O bandwidth can be more easily and efficiently shared across many servers by specialized stand-alone storage trays, resulting in significant overall TCO savings for the WSC because it's now easier to supply storage capacity independently of compute capacity, reducing the stranding of idle resources on both dimensions.

Given that most of the traditional needs from the storage device are now handled by these trays, compute servers only contain a local SSD storage for special purposes, not for general storage. Google servers contain a small local SSD to serve as their boot device. The same device also temporarily stores debug logs. Additionally, some applications require extremely low and/or consistent latency which is best served by having one or more SSDs as close to the processing as possible. Typically, such local storage serves as a cache only (i.e., the data is also stored in a distributed file system) because it's hard to provide reliability and durability for storage without reinventing a distributed file system.

6.5.3 Disk appliances

Appliances consist of an appliance tray with the processor and NIC along with several HDDs, plus one or more add-on trays with minimal control hardware. This design allows changing the number of disks per appliance to adjust to local conditions or as workloads change over time. Figure 6.29 shows an example of a disk appliance used at Google that hosts tens of hard drives (43 total drives split between one appliance tray and two add-on trays in this case). The appliance tray provides power, management, mechanical, and network support for these hard drives. It runs a customized software stack that manages its local storage and responds to client requests over RPC. The limited IOPS and relatively high latency of HDDs allow even a modest processor and NIC to handle the traffic for dozens of HDDs.

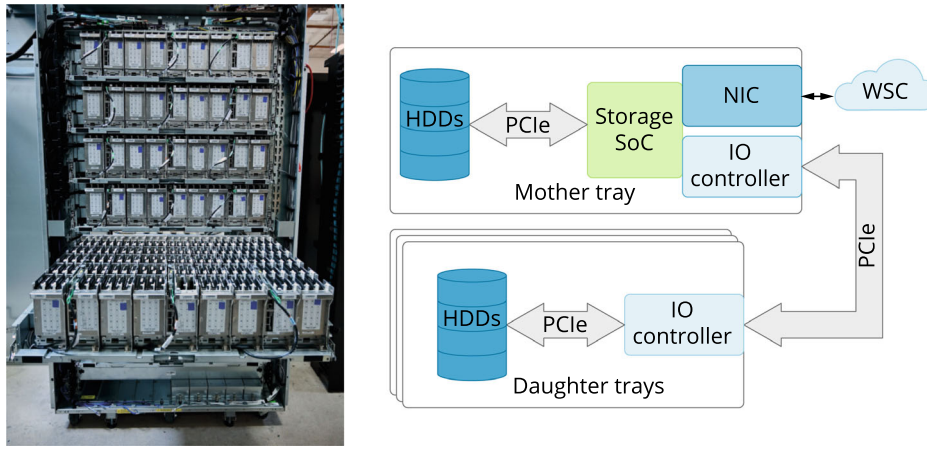


Fig. 6.29 Rack of disk appliances; block diagram of a disk tray

Since the I/O capacity of a single drive is limited to 100-200 Mbytes/sec, a typical appliance pairs a "head" server with 128 GB of RAM (for caching) with 50-100 HDDs for a total storage capacity of 1PB+ and a throughput of 50 Gbps or more. HDDs each consume 8-10W, putting the entire appliance power draw in the 1kW range. Thus, an entire rack of disk appliances draws much less power (5-10kW) than a server rack (20-40kW), let alone an accelerator rack (100kW+).

6.5.4 Flash appliances

As discussed in Section 6.5.2, disaggregated storage can reduce unit costs by improving storage utilization. This advantage must be balanced against latency and bandwidth overhead. Even for SSDs, which have much higher bandwidth and lower latency than disks, this trade-off is acceptable for most applications. As a result, most solid state storage in WSC is housed in network-attached appliances. SSD appliances are similar in concept to HDD appliances, but have a different set of constraints.

Since SSDs have an order of magnitude more I/O bandwidth than HDDs, managing a set of SSDs requires significantly more processing power, memory, and network bandwidth. SSD appliances use server-class processors, NICs, and a large amount of DRAM. These high-performance components increase the TCO overhead of SSDs appliances far above that of an HDD appliance with a similar number of drives. In an SSD appliance, the processor, network, and memory bandwidth, rather than the SSDs themselves, are typically the bottleneck in total system throughput. Since SSDs and networks continue to get faster, careful software design and implementation is more important than ever [57]. Cluster scheduling ensures that SSD traffic does not lead to network bottlenecks (see Section 7.3.2).

Fig. 6.30 An SSD appliance with 16 slots

Figure 6.30 shows a modern SSD appliance. A typical SSD appliance consists of a CPU with a fair amount of processing power, 96 - 128 GB of RAM, 12-16 SSDs of 4 - 8 TB each, and a 400 Gbps NIC. A typical SSD consumes about 20W, resulting in a total appliance power consumption of around 700W.

SSD appliances may oversubscribe the I/O bandwidth of their SSDs, so that their network bandwidth is lower than the aggregate bandwidth of the SSDs. For example, 16 SSDs capable of 4 GB/s of read/write throughput can deliver 512 Gbps of total throughput but be connected to an appliance with a 200 Gbps NIC. While this could potentially strand some performance, real-world workload mixes in large storage pools typically require less bandwidth per TB of storage than drives could provide at peak. Thus, such oversubscription lowers TCO by better amortizing the cost of the appliance per TB of SSD storage. Promising emerging technologies like Compute Express Link (CXL) and direct data cache injection should alleviate some of the current system bottlenecks and allow more SSDs to be supported in a single appliance.

Flash's most appealing characteristic with respect to disks is its performance under random read operations, which is nearly three orders of magnitude better. Note that in the worst case, writes to flash can be several orders of magnitude slower than reads, and garbage collection can further increase write amplification and tail latency if not managed carefully. The "interesting" characteristics of NAND flash (read/write asymmetry, read/write interference, garbage collection) introduce new challenges and opportunities but are beyond the scope of this book.

6.5.5 Smart storage

Traditionally, enterprise storage systems contained additional "smart" functionality beyond simple storage. For example, they provided encryption, compression, and block deduplication functionality to improve security, performance, and effective cost per byte. In WSCs, such functionality has moved to higher levels of the stack (i.e., the file system, or databases built on top of that file system). Since all data is encrypted before it reaches the actual storage device, compression at the device level no longer works. Similarly, since the file system replicates data for availability, a smart disk appliance attempting to save space by deduplicating stored data would be counterproductive.

Storage devices still provide hardware encryption because it's useful as a last layer of protection. Data may be encrypted unnecessarily multiple times but hardware acceleration has reduced the cost of encryption to a low enough level that the cost of multiple encryption operations is outweighed by modularity and security benefits. For example, an email may be encrypted with a user-specific key in Gmail's backend before it is stored in a file system which additionally encrypts it with a file key before it is written to an SSD or disk which again encrypts that data with a device specific key. This way, the user-specific key doesn't have to be exposed to the file system, providing better compartmentalization, and the default encryption provided at lower levels protects data written by systems that don't have built-in encryption, like the OS.

6.6 Cross-cutting issues

Computer architects need to choose the right combination of performance and capacity from the various building blocks that make up a WSC. In this chapter we discussed many examples of how the right building blocks are apparent only when one considers the entire WSC system. The issue of balance must also be addressed at this level, ideally by characterizing the real workloads executing on the system, while understanding how they're likely to evolve during the lifetime of the WSC. While it is tempting to optimize for current workloads, if their structure has meaningfully changed over the past 5 years then it will likely continue to evolve in the next 5 years, so the real-world TCO of a more flexible solution will beat an optimized but inflexible architecture.

The following considerations are especially important:

- Large services spending hundreds of millions of dollars per year will likely restructure their applications to take advantage of new hardware. Strong collaboration with such teams enables software-hardware co-design, while avoiding systems that are too complex to program. On the other hand, most enterprises do not have the time to optimize their applications, so new architectures need to serve existing applications well. For most of these applications, the (people) cost of optimization exceeds the possible hardware savings, so it's an economically rational choice not to optimize.

- The most cost-efficient and balanced hardware configuration may be a match with the combined resource requirements of multiple workloads and not necessarily a perfect fit for any one workload. For example, an application that is seek-limited may not need the capacity of a large disk drive, but when combined with applications needing archival storage, large disks may be the optimal solution. Even when a particular cluster initially runs a single application, that application likely won't stay in it for the full six-year lifetime of the cluster, so we need to anticipate the cost of reconfiguring it for other applications.
- Fungible resources tend to be more efficiently used: sharing improves utilization, and private resources hurt utilization. In a world where technology improvements yield just 10% a year, the difference between 50% utilization and 80% utilization equals nearly five years of hardware improvements! Thus, provided there is sufficient connectivity, software systems should prioritize flexibly utilizing resources made available by remote servers.

The right design point depends on data size and service popularity as well. For example, a service with huge data sets but relatively small request traffic may be able to serve most of its content directly from cheap disk storage. High-throughput services that either have small data set sizes or significant data locality can benefit from in-memory serving instead.

Finally, as mentioned in the first point, workload churn in this space presents a challenge to WSC architects. It is quite likely that workloads will change during a server's lifetime (typically six years). This issue is even more acute for the WSC as a whole because the lifetime of a data center facility spans several server lifetimes (20+ years). In those cases it is useful to try to envision the kinds of machinery or facility upgrades that may be necessary over the lifetime of the WSC system and take that into account during the design phase of the facility. In fact, the mistake we've probably made most frequently was to hardwire too many decisions into a design, "optimizing" initial cost/performance (e.g., because soldering a component onto a motherboard is cheaper than putting it in a swappable extension card) but "pessimizing" actual lifetime TCO as a result.

6.7 Historical retrospective

The past (and first!) twenty five years of WSCs have seen significant changes across servers, storage, and networking.

6.7.1 Servers and accelerators

Server design in WSCs started off as low-cost systems cobbled together with standard off-the-shelf components, leveraging then-CPU trends towards higher clock frequencies. This

gradually evolved into fully-customized server designs optimized specifically for Google (in some respects, setting the stage for today's custom silicon accelerators optimized for specific workloads).

WSCs were first to fully embrace (and accelerate) the multi-core era of CPU design, which naturally matched request-level parallelism and large-scale distributed systems. Starting out as search-optimized system designs, servers then evolved to address a wide range of different workloads. More recently, the slowing of Moore's law brought increasingly heterogeneous system designs, a significant departure from the original "one machine for all" philosophy of the early years. This trend manifested itself further in ISA agnostic servers (x86, ARM, RISC-V), and also in different configurations optimized for different system balance points. (In the next chapter, we will talk about how software-defined servers help manage this heterogeneity.) Modular servers address increased I/O data rates and a proliferation of form factors and configurations. Hyperscaler-driven open standards, particularly around common infrastructure for server software (drivers, qualification, testing, etc), significantly influenced the industry, which now supplies many products optimized for WSC use cases.

Concurrently, the past decade has seen an accelerating shift towards custom silicon accelerators. At Google this first started with accelerators for machine learning, TPUs. TPUs are now in their sixth generation, and are optimized for both training and inference. Similarly, GPUs have evolved from being graphics accelerators to supporting scientific computing and now machine learning workloads.

Custom chips now address other domains including security (e.g., OpenTitan, Caliptra root of trust silicon), data processing, and networking. VCUs accelerate video coding workloads, now in their third generation (and even receiving an Emmy Award in 2024!). But in many respects, we are still in the early stages of the accelerator era.

6.7.2 Networking

WSC networking has perhaps seen the most radical evolution over the last twenty five years. The very first networking designs were based on "four-post" vendor routers for scale-up networking with speeds and feeds of 1 Gbps NICs, 10 Gbps uplinks with high oversubscription, and 1 Tbps cluster fabrics. Low per-server cross-section bandwidth required server-attached storage and software optimizations for rack locality. These limitations were lifted with the first Clos networks in 2008, built from merchant silicon and improving per-server cross-section bandwidth by an order of magnitude. Software-defined networking turned the traditional network routing problem into a distributed systems problem, introducing model-driven networking design and development. Networking speeds and feeds evolved by almost two orders of magnitude in the past 15 years.

Optics were introduced at scale for the first time in large hyperscale data centers, based on cost-reduced short-range transceivers. Linux containers combined with strong resource isolation allowed large clusters to be shared across multiple large applications (search,

Gmail, data analytics, cloud computing) with efficient bin packing. Google's Global Cache was developed to serve static content and video from locations closer to users. The next wave of innovations further pushed cost-efficiency with enterprise-level merchant silicon networking technology both at the campus and WAN levels. Software-defined networking in the WAN with centralized traffic engineering and end-to-end bandwidth enforcement allowed running expensive WAN links at high utilization with a mix of high priority and low priority traffic. Networking speeds and feeds continued to evolve, hitting 1 Pbps cluster fabrics.

The last ten years have seen continued innovations. Optical circuit switching (OCS) has allowed for modular networking that can be incrementally deployed and refreshed. Spine-free networks have allowed for lower cost and power in cluster fabrics. New OS bypass techniques in the user space [58] enabled low-latency high IOPS host-to-host communication. Software-defined networking improved the management plane for reliability and feature velocity. Espresso SDN-based edge peering improved routing and traffic balancing relative to traditional BGP-based shortest-path algorithms [50]. Andromeda cloud networks introduced SDN-based overlays for cloud virtual networking [35]. More recently, topology engineering optimizations have matched the logical network topology to traffic patterns to improve efficiency [26].

Delay-based congestion control [59] brought nearly tenfold tail latency improvements. Optimizations for big packets and host-based repathing around failures have further improved efficiency and reliability [52]. Speeds and feeds now include 100-200 Gbps host NICs, 200-400 Gbps links, and 6.5 Pbps cluster networking. We are now in the era of SmartNICs (IPUs), embracing networking and storage offloads and acceleration. These designs also improved security isolation in multi-tenant networks. Additionally, networking is starting to specialize for machine learning systems, with large ML pods interconnected via OCS.

6.7.3 Storage

Similar to server compute, storage systems in WSCs had humble beginnings, starting with leveraging third party systems in an era where longitudinal recording was still the mainstream hard drive technology. They soon evolved to low-cost custom storage trays directly attached to compute machines with WSC-specific innovations for improved performance and reliability. WSCs were also at the forefront of embracing the industry transition from longitudinal to perpendicular media recording, enabling disk drive byte densities to scale dramatically.

Flash storage started with custom FPGA-based PCIe SSD devices, first with SLC NAND and later with lower-cost, higher-bit-density MLC NAND. Storage became disaggregated with the first HDD appliances, driving better utilization, maintenance, and TCO. Later, enabled by rapidly increasing network bandwidth, SSDs moved into storage appliances as well.

WSCs were early adopters of the next wave of technology advances in hard drives, the transition from air-filled to helium-filled HDDs that enabled thinner platters with lower power and higher density. SSDs transitioned to TLC and later 3D-stacked NAND. More recently, as HDD bit-density scaling has slowed down, the focus moved to enabling HSMR and energy-assist technologies. Data volumes continue to grow, considerably faster than compute, and correspondingly storage will continue to be an important component of WSC system design. New and cloud and ML workloads bring new challenges and requirements, so we expect storage systems to evolve again in the years ahead.

References

1. L. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.
2. Transaction Processing Performance Council, *TPC Benchmark C (TPC-C)*, https://www.tpc.org/tpcc/.
3. U. Hölzle, "Brawny cores still beat wimpy cores, most of the time," 2010.
4. D. G. Andersen, J. Franklin, M. Kaminsky, et al., "Fawn: A fast array of wimpy nodes," 7, vol. 54, New York, NY, USA: Association for Computing Machinery, Jul. 1, 2011, pp. 101–109.
5. L. A. Barroso, K. Gharachorloo, R. McNamara, et al., "Piranha: A scalable architecture based on singlechip multiprocessing," 2, vol. 28, New York, NY, USA: Association for Computing Machinery, May 1, 2000, pp. 282–293.
6. K. Lim, P. Ranganathan, J. Chang, et al., "Understanding and designing new server architectures for emerging warehouse-computing environments," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08, USA: IEEE Computer Society, Jun. 1, 2008, pp. 315–326.
7. HP Inc., *HP launches new class of server for social, mobile, cloud and big data*, News Release: https://investor.hp.com/news-events/news/news-details/2013/HP-Launches-New-Class-of-Server-for-Social-Mobile-Cloud-and-Big-Data/default.aspx, 2013.
8. J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services," in *Conference on Innovative Data Systems Research (CIDR'09)(January 2009)*,2009.
9. Amazon Web Services, *AWS Graviton Processor*, https://aws.amazon.com/ec2/graviton/.
10. D. DeWitt and J. Gray, "Parallel database systems: The future of high performancedatabase systems," 6,vol. 35, New York, NY, USA: Association for Computing Machinery, Jun. 1, 1992, pp. 85–98.
11. A. Vahdat and M. Lohmeyer, *Enabling next-generation AI workloads: Announcing TPU v5p and AI Hypercomputer*, 2023.
12. UALink Consortium, *Ultra Accelerator Link Consortium*.
13. D. Gibson, H. Hariharan, E. Lance, et al., "Aquila: A unified, low-latency fabric for datacenter networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1249–1266.
14. Open Compute Project, *Open Compute Project*, https://www.opencompute.org.
15. A. Lagar-Cavilla, A. Muñoz, B. Kelly, and P. J. Indranil Banerjee, *Composable security architectures*, Presentation at the 2021 OCP Global Summit: https://www.opencompute.org/events/past-events/2021-ocp-global-summit, 2021.

16. A. Lagar-Cavilla, V. Sampath, B. Pillillli, and P. Jayanna, *Composable security architectures ii*, Presentation at the 2023 OCP Global Summit: https://www.opencompute.org/events/past-events/2023-ocp-global-summit, 2023.
17. A. Putnam, A. M. Caulfield, E. S. Chung, et al., "A reconfigurable fabric for accelerating large-scale datacentre services," 11, vol. 59, New York, NY, USA: Association for Computing Machinery, Oct. 28, 2016, pp. 114–122.
18. Microsoft Azure, *Azure Maia for the era of AI: From silicon to software to systems*, Azure Blog: https://azure.microsoft.com/en-us/blog/azure-maia-for-the-era-of-ai-from-silicon-to-software-to-systems/, 2024.
19. A. Firoozshahian, J. Coburn, R. Levenstein, et al., "MTIA: First generation silicon targeting Meta's recommendation systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23, New York, NY, USA: Association for Computing Machinery, Jun. 17, 2023, pp. 1–13.
20. Amazon Web Services, *AWS Trainium*, https://aws.amazon.com/machine-learning/trainium/.
21. Amazon Web Services, *AWS Inferentia*, https://aws.amazon.com/machine-learning/inferentia/.
22. N. P. Jouppi, D. H. Yoon, M. Ashcraft, et al., "Ten lessons from three generations shaped Google's TPUv4i," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21, IEEE Press, Nov. 25, 2021, pp. 1–14.
23. G. Team, R. Anil, S. Borgeaud, et al., "Gemini: A family of highly capable multimodal models," arXiv preprint arXiv:2312.11805, 2023.
24. Sandvine, *Global internet phenomena report*, https://www.applogicnetworks.com/blog/the-2025-global-internet-phenomena-report, 2025.
25. P. Ranganathan, D. Stodolsky, J. Calow, et al., "Warehouse-scale video acceleration: Co-design and deployment in the wild," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2021, pp. 600–615.
26. L. Poutievski, O. Mashayekhi, J. Ong, et al., "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22, New York, NY, USA: Association for Computing Machinery, Aug. 22, 2022, pp. 66–85.
27. A. D. Ferguson, S. Gribble, C.-Y. Hong, et al., "Orion: Google's software-defined networking control plane," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 83–98.
28. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
29. A. Kumar, S. Jain, U. Naik, et al., "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 1–14.
30. A. Vahdat, M. Al-Fares, N. Farrington, et al., "Scale-out networking in the data center," *IEEE Micro*, vol. 30, no. 4, pp. 29–41, 2010.
31. D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.
32. D. Abts and J. Kim, *High performance datacenter networks: Architectures, algorithms, and opportunities*. Springer Nature, 2022.
33. D. Firestone, A. Putnam, S. Mundkur, et al., "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.

34. A. Saeed, N. Dukkipati, V. Valancius, et al., "Carousel: Scalable traffic shaping at end hosts," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, Aug. 7, 2017, pp. 404–417.
35. M. Dalton, D. Schultz, J. Adriaens, et al., "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 373–387.
36. R. Niranjan Mysore, A. Pamboris, N. Farrington, et al., "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09, New York, NY, USA: Association for Computing Machinery, Aug. 16, 2009, pp. 39–50.
37. Nyquist Capital, *Google's secret 10GbE switch*, Blog Post: https://nyquistcapital.com/2007/11/16/googles-secret-10gbe-switch/, Nov. 2007.
38. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: a decade of Clos topologies and centralized control in Google's datacenter network," 4, vol. 45, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
39. Ultra Ethernet Consortium, *Ultra Ethernet Consortium*, https://ultraethernet.org/.
40. D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the Slingshot interconnect," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20, IEEE Press, Nov. 9, 2020, pp. 1–14.
41. TOP500.org, *TOP500 List - June 2024*, https://top500.org/lists/top500/2024/06/, Jun. 2024.
42. P. Kale, A. Tumma, H. Kshirsagar, P. Ramrakhyani, and T. Vinode, "Fibre Channel over Ethernet: A beginners perspective," in *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, 2011, pp. 438–443.
43. J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "Internet small computer systems interface (iSCSI)," Tech. Rep., 2004.
44. NVM Express, *NVM Express Base Specification*, https://nvmexpress.org/specification/nvm-express-basespecification/.
45. L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable HPC," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, 2020.
46. Google Cloud, *Introduction to Cloud TPU v5p*, Cloud TPU Documentation: https://cloud.google.com/tpu/docs/v5p.
47. Submarine Cable Networks, *Complete List of Google's Subsea Cable Investments*, https://www.submarinenetworks.com/en/nv/insights/complete-list-of-google-s-subsea-cable-investments.
48. U. H lzle, *B4 and Google's network*, Presentation: https://www.segment-routing.net/images/hoelzle-tue-openflow.pdf.
49. S. Jain, A. Kumar, S. Mandal, et al., "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2013, pp. 3–14.
50. K.-K. Yap, M. Motiwala, J. Rahe, et al., "Taking the edge off with Espresso: scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, Aug. 7, 2017, pp. 432–445.
51. C.-Y. Hong, S. Mandal, M. Al-Fares, et al., "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, New York, NY, USA: Association for Computing Machinery, Aug. 7, 2018, pp. 74–87.
52. D. Wetherall, A. Kabbani, V. Jacobson, et al., "Improving network availability with Protective ReRoute," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23, New York, NY, USA: Association for Computing Machinery, Sep. 1, 2023, pp. 684–695.

53. A. Krentsel, N. Saran, B. Koley, et al., "A decentralized SDN architecture for the WAN," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24, New York, NY, USA: Association for Computing Machinery, Aug. 4, 2024, pp. 938–953.
54. J. Gray, *Tape is dead. disk is tape. flash is disk. ram locality is king*, https://research.microsoft.com/en-us/um/people/gray/talks/flash_is_good.ppt.
55. E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. T'so, *Disks for data centers*, https://research.google/pubs/disks-for-data-centers/, 2016.
56. G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "{Disk-locality} in datacenter computing considered irrelevant," in *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*, 2011.
57. L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.
58. M. Marty, M. de Kruijf, J. Adriaens, et al., "Snap: A microkernel approach to host networking," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, New York, NY, USA: Association for Computing Machinery, Oct. 27, 2019, pp. 399–413.
59. G. Kumar, N. Dukkipati, K. Jang, et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '20, New York, NY, USA: Association for Computing Machinery, Jul. 30, 2020, pp. 514–528.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Software-Defined Infrastructure

7

In the last two chapters we described the design of WSC hardware building blocks, and detailed how the slowing of Moore's Law drove a focus on efficient hardware design: custom silicon accelerators provide better performance per watt, and heterogeneous server platforms optimize for different system balance points and workload characteristics.

But there is another way we can counter a slowing Moore's Law, and that is to utilize the hardware we have more efficiently. In particular, the vertically integrated design of WSCs, together with the distributed systems stack discussed in Chapter 4, allow us to view WSC hardware through the lens of software that manages and abstracts this hardware for improved efficiency. We call this approach software-defined infrastructure.

Software-defined infrastructure makes hardware more flexible, better matched to applications, and easier to manage by constructing a software management and control plane that can take advantage of a global view to optimize local decisions. For example, software-defined networks relieve individual switches from making routing decisions because these can be better made globally. Software-defined servers benefit from a hardware-aware scheduler that routes the right workloads to them (i.e., the workloads that perform best on that specific hardware), without requiring users to specify hardware.

The following sections explore how software-defined infrastructure improves the performance and efficiency of compute, storage, network, and data center infrastructure.

7.1 Software-defined servers

In the last chapter, we discussed the evolution of hardware designs for servers in WSCs over the past twenty-five years. Concurrently, how these hardware designs are presented to higher layers of software – the abstraction of a server, for example – has also evolved.

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_7

In the past, Moore's Law provided such a large improvement over the prior generation that the newest machine was the best choice for all applications. Early WSCs therefore had a single "machine of the day" providing relatively homogeneous abstractions to higher layers of compute management. However, with the slowing of Moore's law, application characteristics can have a bigger performance impact than hardware. For example, if a new server is only 10–15% improved, an application with high memory bandwidth requirements might still run more efficiently on an older platform even though a newer system provides 15% more cores. Thus, since the performance variability between applications may be bigger than the variability between current-generation server types, WSC operators may concurrently buy different server types, increasing the heterogeneity of the fleet.

Since older servers continue to be price/performance competitive, server lifetimes extend, leading to additional fleet heterogeneity from multiple concurrent generations of hardware. Concurrently, workloads have also diversified, with varied compute needs, memory access patterns, latency requirements, and service-level objectives. Additionally, beyond multiple CPU architectures, WSCs have also introduced a myriad of specialized accelerators (like GPUs and TPUs) and new memory tiers (like non-volatile memory) further increasing diversity. And, finally, as discussed in Section 6.2.1, servers are evolving to disaggregated mini-distributed systems dynamically composed of various hardware resources including cores, memory, SSDs, and domain-specific accelerators.

A traditional, static, one-size-fits-all server abstraction aligned with a specific (vendor-defined) hardware design becomes inefficient in this heterogeneous world, failing to exploit the specific strengths of different hardware platforms or the unique demands of different applications.

Software-defined servers (SDS) provide an alternate adaptable abstraction, emphasizing co-design across hardware and software for dynamic optimization and better match between applications and hardware. Consequently, we can support dynamically tailoring system configurations (e.g., hardware knobs) for "SoftSKUs" optimized for specific application requirements. We can support finer-grained control over hardware resources (e.g., QoS for shared resources, non-uniform memory access) and move control from static hardware heuristics to software programmability (compiler feedback-directed optimizations like prefetching). More importantly, we can also formalize interfaces for better cluster scheduling to address hardware heterogeneity (different CPU generations, accelerators, memory tiers) to dynamically match workloads to the right hardware.

In essence, software-defined servers represent a move towards greater system intelligence and adaptability in response to the end of classical hardware scaling. By enabling software to understand and control hardware resources more deeply, software-defined servers can achieve significant improvements in performance, resource utilization, and total cost of ownership, providing continued efficiency gains in large-scale computing infrastructure even when fundamental hardware improvements are more challenging to achieve.

Software-defined servers work at multiple levels of the system stack. At the CPU or microarchitectural level, software-defined servers can enable fine-grained, dynamic tuning

of hardware features that were previously static. For example, hardware prefetchers, CPU core frequencies, or power states can be adjusted via software, often through model-specific registers (MSRs), based on the runtime behavior and needs of the application assigned to those cores. A good example is the recent Limoncello work from Google [1]. This allows configurations to move beyond generic defaults and optimize for specific workload characteristics, such as instruction mix, cache miss rates, or sensitivity to memory latency, potentially unlocking significant performance or efficiency gains previously left untapped. Specialized instruction sets (e.g., SIMD, AVX) available on different CPU generations can also be explicitly considered for optimization.

At the machine level, software-defined servers can focus on managing shared resources within a server node, particularly crucial in multi-tenant environments where applications compete for resources like last-level cache (LLC) and memory bandwidth. SDS implements Quality of Service (QoS) mechanisms, allowing the system to allocate these shared resources based on application priority, class, or measured needs [2]. This ensures more predictable performance for high-priority latency-sensitive tasks, even when co-located with resource-intensive batch jobs, by mitigating interference and enforcing allocation policies. Software-defined servers can also manage the (often complex) interplay between different resource types (e.g., limiting cache allocation might increase memory traffic) to achieve holistic optimization.

Extending beyond individual machines, software-defined servers can address cluster-level resource management and scheduling. Recognizing that different hardware platforms (e.g., different CPU generations or architectures) possess distinct performance characteristics (e.g., core count vs. frequency, memory bandwidth per core, NUMA penalties), cluster schedulers can schedule workloads specific to platforms, using performance data and workload characteristics to intelligently steer jobs onto the hardware platforms where they run most effectively [3]. This matching process maximizes overall cluster throughput and efficiency.

At a high-level, software-defined servers rely on robust abstraction and efficiency layers that extend the WSC software stack that we introduced in Chapter 2 (Figure 7.1). An abstraction layer provides a consistent interface for querying hardware capabilities and applying configurations, hiding the platform-specific implementation details from higher-level software. On the hardware side, it presents specific CPU and machine-level configurations as well-defined abstractions to the software. On the software side, it maintains the WSC abstraction for end-user applications while transparently optimizing performance. Below this, an efficiency layer incorporates monitoring infrastructure to collect performance data, modeling capabilities to understand workload-hardware interactions, and policy engines (potentially using machine learning) to generate optimal configurations and scheduling hints. Together, these layers enable automated, data-driven optimization, allowing the WSC to modify hardware behavior dynamically to meet application demands.

We use the term software-defined servers to cover a variety of optimizations. Below, we discuss some specific examples. We start with an example of compiler-driven software

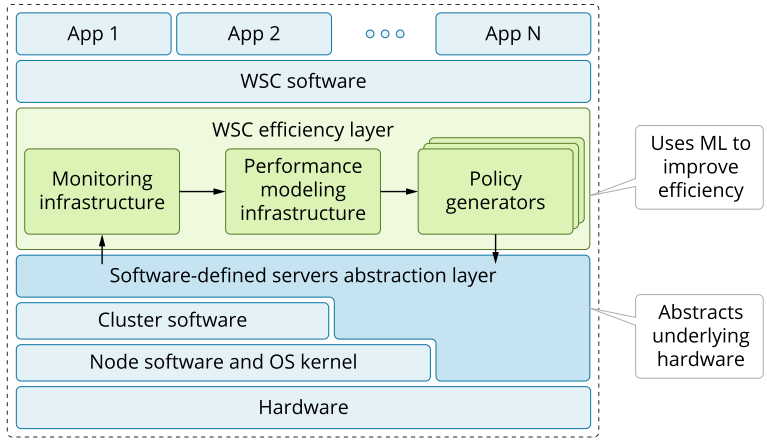


Fig. 7.1 Software-defined servers: Abstraction and efficiency layers

optimizations that proactively take workload-specific behaviour into account (using profile-guided optimization). We then discuss how we have optimized software libraries to be more hardware-aware, focusing on the example of TCMalloc optimizing TLB misses and memory footprints. Platforms-aware scheduling optimizes workloads across heterogeneous platforms, software managed tiered memory reduces total memory costs.

7.1.1 Case study: Compiler optimizations

The vertically-integrated systems stacks in WSC (discussed earlier in Chapter 4) allows for more hardware-software codesign. Specifically, given that most WSC providers build their own compilers, there are a lot of options for hardware-aware compiler optimizations. Compilers generate optimized code tailored for specific server architectures. Despite the wide variety of warehouse-scale computing (WSC) workloads, there are many common usage patterns that open up possibilities for joint hardware and software optimization. Below we discuss one such example.

At Google, we leverage a database with detailed information about instructions and basic blocks for the vast number of CPU cycles found in production binaries, to guide many optimizations [4]. The database is used to answer questions about how often instructions of type X execute, what fraction of type Y instructions have property Z, or what the most common N-grams are. This has enabled us to identify opportunities for classic microarchitectural optimizations.

For instance, the instruction footprint of WSC workloads tends to be over 100 times larger than the size of a typical L1 instruction cache and has been increasing at a rate of more than 20% annually. Consequently, these workloads experience instruction cache miss rates that

are significantly higher than the worst-case scenarios found in desktop-class applications, which are typically evaluated using SPEC CPU benchmarks.

Since processors are critically dependent on their ability to feed themselves with useful instructions, poor instruction cache (i-cache) performance can lead to significant performance losses due to front-end stalls. A cycle-level breakdown of a web search workload running on an Intel Haswell CPU revealed that 14% of its total performance potential is lost due to "Front-end latency," mainly attributed to instruction cache misses [4]. The workload has a hot steady-state instruction working set of about 4 MiB, which is much larger than the L1 and L2 instruction caches found in current server CPUs, but small enough to fit comfortably in the shared L3 cache (usually tens of MiB in size). This inspired us to investigate a compiler optimization that can automatically insert code prefetch instructions into the critical execution paths of the application binary to significantly enhance its performance.

Feedback Directed Optimization (FDO) employs runtime behavioral information to guide optimization, resulting in improvements typically ranging from 10% to 15%. However, the nature of data center applications makes it exceedingly arduous for many users to maintain a suitable benchmark that adequately represents the workload.

AutoFDO [5] automates feedback-directed optimization by utilizing hardware performance data collected from production servers (Figure 7.2). The process is lightweight and offloaded to minimize impact on production systems. It involves regular collection of performance samples, which are then symbolized, formatted, and stored in a version controlled repository.

Once the profile is available, the compiler can map it onto the intermediate representation (IR). This process involves converting a binary-level profile into a source-level profile, which is then transformed into a standard compiler format that can be mapped onto the IR. Next, the compiler can generate a control flow graph with edge and node frequency information, which contains context-sensitivity information that the compiler can use to perform selective inlining.

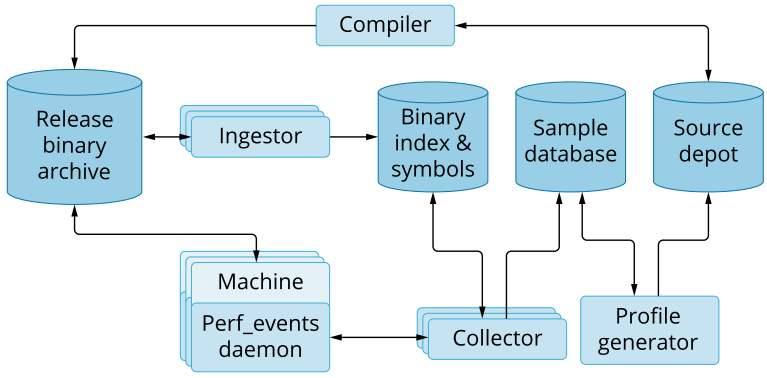


Fig. 7.2 AutoFDO end-to-end system diagram

Experience with AutoFDO has highlighted several key lessons. Rigorous testing is essential to ensure that compiler transformations don't introduce errors. While AutoFDO may not always outperform traditional feedback-directed optimization (FDO) in benchmarks, real-world performance is often comparable, even with imprecise profiles. Additionally, it's important to consider workload diversity within binaries, allowing for multiple profiles to optimize for various use cases. Finally, it must be very easy for teams to adopt AutoFDO for their binaries, so the benefits reach 100% of the fleet.

7.1.2 Case study: Library optimizations

Many software stacks share fundamental components like memory management, RPC, encryption, etc. These components, sometimes referred to as the "data center tax" [6], can consume up to 30% of processing cycles in WSC environments (Section 8.1.2), presenting a compelling case for hardware-aware optimizations for key libraries.

One such example is to optimize hugepage coverage [7] in memory allocators. Cache and Translation Lookaside Buffer (TLB) misses are major performance bottlenecks in WSCs; around 20% of cycles in WSCs are stalled due to TLB misses. Hugepages are TLB entries that cover a much larger block of memory, from several megabytes to several gigabytes. Thus, hugepages can significantly reduce TLB misses. But the larger a page, the higher the risk that internal fragmentation within that page causes physical memory to be unused. By optimizing hugepage coverage, an allocator can reduce TLB misses and cooperate with the operating system to control the placement of new allocations to reduce internal fragmentation.

TCMalloc [8], [9] is a memory allocator often used in large-scale distributed applications. Figure 7.3 shows memory organization in TCMalloc. Objects are segregated by size. First, TCMalloc partitions memory into spans, aligned to page size. Sufficiently large allocations are fulfilled with a span containing only the allocated object. Other spans contain multiple smaller objects of the same size (a size class). The "small" object size boundary is 256 KiB.

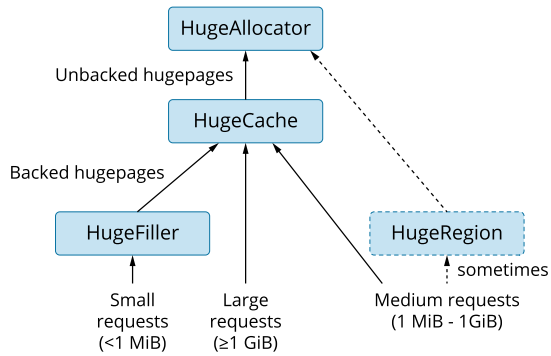


Fig. 7.3 Flow of requests by size into components in TCMalloc

Within this "small" threshold, allocation requests are rounded up to one of 100 size classes. TCMalloc stores objects in a series of caches.

The objective is to maximize the utilization of hugepages. Heuristics enable the allocator to pack objects tightly onto frequently used hugepages while also creating fully unused hugepages for return to the operating system. The main component, called HugeAllocator, deals with virtual memory and the operating system with a cache of fully-empty hugepages called the HugeCache. Additionally, a list of partially filled single hugepages, referred to as the HugeFiller, is densely filled by subsequent small allocations.

Using a memory allocator that is aware of hugepages led to a 7.7% performance improvement in WSC applications and a 2.4% decrease in RAM usage. The allocator also causes 6% fewer stalls due to TLB misses, and a 26% reduction in memory waste caused by fragmentation.

7.1.3 Case study: Platform-aware scheduling

Platform-aware scheduling is another classic software-defined server optimization, where we schedule workloads to specific platforms where they perform best for improved performance and utilization.

Workload performance can vary significantly on different platforms due to varying sensitivities to the underlying micro-architecture. For instance, memory-intensive workloads, like video transcoding, thrive on platforms with high memory bandwidth per core. In contrast, heavily multithreaded workloads, such as web search, favor platforms with simultaneous multithreading (SMT) or a large core count. Single-threaded compute-bound workloads perform best on platforms with higher clock speeds.

Figure 7.4 shows the performance of a particular binary across the five most common platforms in the fleet at the time of the measurement. The binary's performance was consistent over time and shows a clear preference for platform A, and platform D would be the worst choice to run this binary. In this graph, 100% refers to the performance that would be expected purely from the performance rating of each platform. Thus, this particular application runs about 3% slower than expected on the top two platforms but 10% slower than expected on platform D.

Platform-aware scheduling steers workloads toward the platforms that can deliver optimal performance. A data processing pipeline captures profiles of workloads running in production environments. A ranking algorithm employs data mining techniques, like hierarchical agglomerative clustering, to analyze the profiles and create a ranked list of preferred platforms for each workload. These workload-specific platform preferences are then uploaded to the cluster manager. There, the cluster manager schedules workloads onto the most suitable hardware platforms available.

Enabling platform-aware scheduling for all critical workloads improved average performance by 7–10%. Interestingly, the benefits of such an approach extend beyond general

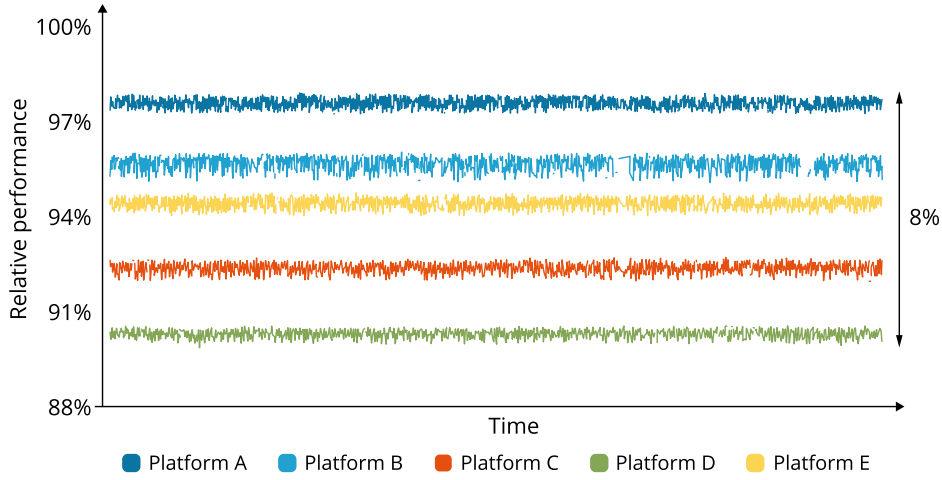


Fig. 7.4 An application's relative performance on different platforms

computing, to also accelerators and GPU-based platforms, leading to improved efficiency and utilization. In fact, implementing this approach for accelerators and GPUs has doubled the utilization of some platforms, all while maintaining a seamless experience for end-users, who remain shielded from the complexities of underlying hardware differences.

7.1.4 Case study: Software-defined memory

The final example of software-defined servers that we will discuss is around memory management.

The growing diversity of hardware platforms and workloads in data centers makes it difficult to maintain a balanced system where all resources are fully utilized. The ratio of memory bandwidth to compute capacity has been declining over time, exacerbated by the rise of memory-intensive workloads like machine learning (ML). This trend creates an imbalance across the entire data center fleet. At Google, we have deployed software-defined memory optimizations following a three-pronged approach [10]:

- Parametrize memory requirements: Identify specific bandwidth, capacity, and latency needs for each application, ensuring a more efficient and targeted allocation of memory resources.
- Deploy diverse memory technologies: Utilize a variety of memory technologies that align with each application's unique demands. For example, high-bandwidth applications can benefit from HBM or 3D stacked cache, while high-capacity applications can use larger but slower DRAM modules. Beyond hardware, we can also create diversity in

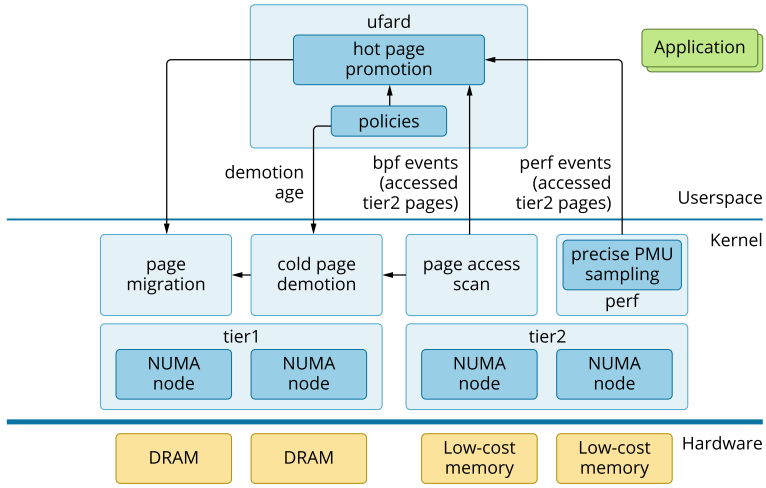


Fig. 7.5 Tiered memory architecture

memory through compression: compressed memory essentially provides the abstraction of a cheaper, albeit lower-latency, memory tier.

- Develop a tiered-memory architecture: Steer the right data to the right memory technology, automating allocation based on application needs to simplify the management of tiered memory.

Tiered memory software is structured into three layers; Figure 7.5 shows the first two layers. The bottom layer abstracts the hardware and presents a segmented physical address space across multiple memory devices. The middle layer separates the page management policy from the mechanisms used for candidate detection and page migration, which are performed by the kernel. Our past experience has shown that successful use of kernel mechanisms requires multiple design iterations to address subtle behaviors that only emerge at production scale, and to optimize for specific workload aspects. The user space policy layer allows this flexibility and agility.

The top layer (not shown in the figure) operates at the cluster-level. The Borg scheduler collaborates with the node agent, a Borglet task running on every machine, to continuously manage a queue of job requests across multiple machines. It monitors load and performance metrics on each machine and dispatches tasks to individual servers.

To migrate pages between tiers, we measure access patterns to identify "cold" pages that are infrequently accessed and "hot" pages that are frequently accessed. A kernel daemon periodically scans page access bits to determine page idle age. Demoting cold pages to lower tiers largely follows the conventional virtual memory pattern. Hot usage is determined using precise hardware sampling to enable timely detection (Fig. 7.6).

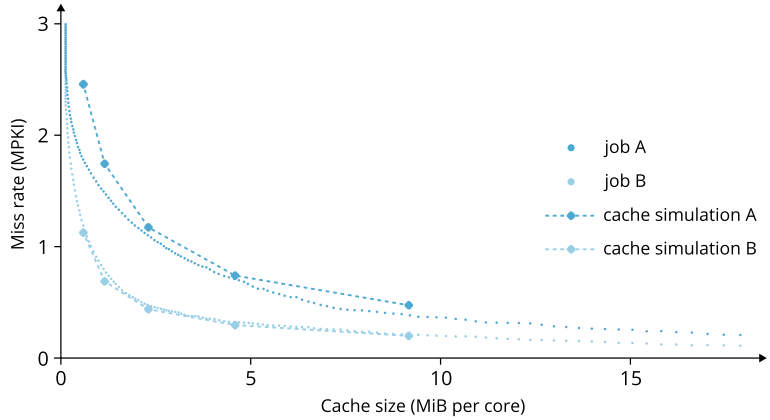


Fig. 7.6 Simulated vs actual miss rates in first tier of memory

Our high-level locality model is constructed from fleet-wide profiling data, gathered from jobs when they are serving production traffic. The statistical cache model predicts the performance of a job under different amounts of cache while measuring jobs' memory bandwidth consumption. The main building block is a miss rate curve (MRC) – a mapping of the miss-rate a job would achieve at various cache sizes. Figure 7.6 shows two sample MRCs generated using the statistical cache model. Job A and Job B are distinct workloads that exhibit varying sensitivities to cache size. Job B has a smaller working set and demonstrates limited reductions in MPKI (Misses Per Kilo Instructions) beyond 2 MiB/core, whereas Job A continues to significantly improve up to 7 MiB/core.

By combining miss-rate-curves with the hardware characteristics of upcoming memory technologies, we can predict how to improve the system balance at the cluster level. We start by estimating the reduction in last-level cache MPKI and the corresponding decrease in memory bandwidth demand for each workload. Then, we determine how many jobs can benefit from the new technologies and leverage platform-aware scheduling, discussed in the previous section, to schedule these jobs on the most suitable hardware. In summary, software-defined memory enables the use of a combination of memory technologies to enhance specific use cases and enhance the overall system balance at the cluster level.

7.2 Software-defined accelerators

The idea of software-defined infrastructure can, at a broad level, also be applied to accelerators. As one example, we discuss H2O-NAS (hyperscale hardware optimized neural architecture search) [11].

In Chapter 6.3 we discussed how ML models demand ever-increasing computational power. To fully harness the potential of ML hardware at scale, model efficiency has become

as important as building new hardware infrastructure. Neural architecture search (NAS) has shown significant promise in automatically designing ML model architectures that rival the best human-designed models. NAS approaches have four key dimensions: search strategy (e.g., one-shot vs. multi-trial), search algorithm (e,g., reinforcement learning, gradient-based algorithm, and evolution algorithm), search space (hardware-specialization and weight-sharing), and search objective (reward functions, quality and performance signals).

H2O-NAS takes this traditional NAS approach and extends it to be hardware-aware. It augments the traditional reinforcement-learning-based search algorithm with hardware optimized search spaces optimized for weight-sharing. For example, the highest-quality model (given a fixed training budget) might be just a bit too large to fit into a TPU pod, and tweaking its size and architecture could produce a revised model that trains twice as fast in practice, at a minimal loss in quality.

To better optimize for scale of hyperscale hardware and data, H2O-NAS uses a massively parallel single-step reinforcement learning (RL) search algorithm to search the model architecture and train the model weights simultaneously using real-time production traffic. More details are presented in the paper by Li et al. [11], but this approach, deployed at Google, has demonstrated significant performance and energy efficiency gains at scale with comparable or better model accuracies and quality.

7.3 Software-defined networks

Networks with many elements, whether they are in a WAN or a LAN, need to be managed (programmed) automatically. The need for a programmable network has led to much interest in OpenFlow [12], P4 [13], and software-defined networking (SDN), which move the network control plane out of individual switches into a logically centralized controller [14], [15], [16].

SDN separates data, control, and management planes, with the latter two implemented in software. Software-defined networks leverage commodity hardware components while addressing control and management requirements in software. SDN centralizes control for forwarding devices built on such merchant silicon through software, using standardized protocols to program the data plane. This decoupling enables the network to be managed as a unified entity rather than individual devices. It also allows for independent scaling, e.g., hardware can be upgraded without modifying the control algorithms. This separation also enabled the control and management planes to better integrate new algorithms to optimize the underlying hardware. Software-defined networking powers some of the largest WSC networks in the world, supporting both intra-cluster and wide-area networks.

Controlling a network from a logically centralized server offers many advantages [17]. In particular, common networking algorithms such as computing reachability, shortest paths, or max-flow traffic placement become much simpler to solve centrally. In networks where each individual router must solve the same problem, it's harder to reach a consistent state

while dealing with limited visibility (direct neighbors only), inconsistent network state (routers that are out of sync with the current network state), and many independent and concurrent actors (routers). Network management operations also become simple because a global view can be used to move a network domain, often consisting of thousands of individual switches, from one consistent state to another while simultaneously accounting for errors that may require rollback of the higher-level management operation underway. P4 [13] provides a protocol- and switch-independent high-level language that allows for programming of packet-forwarding data planes, further increasing flexibility.

Moreover, servers are easier to program and offer more powerful hardware than CPUs embedded in switches and routers. Centralizing the control plane to a few servers also makes it easier to update their software. SDN is a natural match for data center networking, since the applications running in a WSC are already managed by a central entity, the cluster manager. Thus it is natural to use a central SDN controller to also configure any network elements that applications depend on. SDN is equally attractive to manage WAN networks, where logically centralized control simplifies many routing and traffic engineering problems [14], [15].

Inspired by the theoretical advantages of SDN and early academic work at Stanford, the first real-world implementations of SDN were developed at Google. The first production SDN network, B4 [14], [15] addressed the growing video traffic and other bulk traffic on Google's network and allowed it to travel on the cheaper B4 backbone instead of the more expensive, full-feature B2 backbone. Video traffic was an ideal use case for a nascent technology like SDN because it combined high scale with a lower availability requirement. At the time, YouTube's overall availability was significantly below 99.99%, and thus carrying its (non-cached) traffic at lower reliability was acceptable. B4 performed all key management and control plane functions on servers running the SDN stack and programmed the network's Ethernet switches via OpenFlow.

Concurrently, SDN also took over data center networks [18]. Initially, centralized controllers handled only some of the work. In the 2008-era Saturn network, the centralized controllers performed configuration and helped switches understand the global state, but switch-based software programmed the switch's actual forwarding tables. The 2012-era Jupiter network implemented a five-stage Clos network topology and used Onix SDN controllers [19] using OpenFlow to push routing and configuration information to fabric switches. Current Jupiter generations provide additional functionality, having replaced OpenFlow with P4 [20].

P4 [13] evolved out of OpenFlow, recognizing that SDN requires more than programming forwarding tables. Over time, Ethernet switch chips evolved from pure packet switching (i.e., forwarding an incoming packet to an outgoing port) to sophisticated packet processing. For example, an incoming packet may be filtered (firewall functionality), rewritten for Network Address Translation (NAT), or encapsulated into a GRE tunnel. P4 lets SDN controllers specify this packet processing in a high-level, hardware-independent language. P4 compilers

then translate that specification into a chip-specific implementation, hiding vendor-specific details from the SDN controller.

SDN has also taken hold in networks composed of commercially available networking equipment, both at the optical layer and the routing layers. OpenConfig [21] allows such equipment to be programmed and managed via a vendor-agnostic standard API. The Switch Abstraction Interface (SAI) [22] provides a vendor-independent way of controlling forwarding elements, such as a switching ASIC, an NPU or a software switch. The SONiC stack [23] includes the networking software components necessary for a fully functional L3 device in a cloud data center. As a result, while WSC management and control planes are not standardized, they can take advantage of standardized components and APIs.

The next five sections illustrate how an SDN strategy plays out in practice, applied from individual fabrics to the global WAN.

7.3.1 Jupiter topology and traffic engineering

Data center networks employ Topology and Traffic Engineering [20] to optimize throughput and efficiency. Topology Engineering exploits the fact that multi-tenant and building-scale fabrics exhibit predictable traffic patterns with manageable uncertainty. Thus, we don't need to provision for non-blocking forwarding of worst-case traffic in a Clos topology, allowing cheaper solutions that still provide the same capacity. Topology Engineering removes spine blocks and creates direct-connect fabrics optimized through joint traffic and topology engineering. This approach yields shorter, more efficient paths while accommodating traffic uncertainty. Moreover, it reduces costs and power consumption associated with spine blocks and allows higher-speed direct links between newer blocks without needing to refresh the intermediate spine blocks, effectively aligning topology with long-term demand.

Traffic Engineering (TE) dynamically optimizes traffic forwarding across different paths based on real-time demand, compensating for imbalances caused by striping, failures, or management actions. Ideally, traffic would flow directly between blocks, but in practice, demand varies and doesn't perfectly match the topology. In the Jupiter network discussed in Chapter 6, TE combines direct and one-hop indirect paths to balance performance and robustness. Flow measurements from each server are aggregated into a block-level traffic matrix, with each entry indicating the bytes transferred between blocks. This matrix feeds into a predicted traffic matrix used for Weighted Cost Multipathing (WCMP) optimization, composed of peak sending rates for each pair in the last hour. This predicted matrix is updated when large changes in observed traffic occur, or periodically for freshness.

Traffic engineering and topology engineering operate at different times scales. Traffic engineering quickly adapts to topology and traffic changes, while topology engineering is a slower, planned process for implementing new network structures. To address potential load balancing issues across parallel links, host-based Multipathed Network Load Balancing adaptively distributes traffic based on congestion at the RTT level.

7.3.2 Network aware scheduling

Network aware scheduling provides proactive and reactive mechanisms for placing compute and storage jobs, enhancing centralized schedulers like Borg with insights into network traffic patterns and sustained hotspots [24]. These hotspots, often caused by workload fluctuations or network failures, can impact performance. Network aware scheduling proactively bin-packs jobs within a cluster, considering not only compute and storage utilization but also network bandwidth. In addition, reactive mechanisms within network-aware scheduling can relocate demanding jobs away from latency-sensitive workloads in the event of network disruptions or changes in traffic patterns. This ensures optimal resource utilization and minimizes the impact of network congestion. Through proactive and reactive schemes, network aware scheduling complements per-flow congestion control mechanisms, which focus on managing individual flows to maximize link utilization. Together, they form a robust system for maintaining an available and predictable network experience for all users.

7.3.3 Bandwidth enforcer

Google's Bandwidth Enforcer (BwE) [25] optimizes bandwidth allocation and enforcement in real time across the WAN. BwE is crucial for supporting distributed computing and large-scale data transfers. It distributes WAN capacity amongst competing applications, prioritizing them based on business needs while maintaining high overall network utilization, and accounting for a global view of bandwidth and failure conditions. BwE uses global knowledge of network topology and link utilization as input to a hierarchy of bandwidth enforcers, ranging from a global enforcer down to rate enforcers on each host. It actively controls the rate of traffic transmission at the source host, before traffic enters the network. It utilizes telemetry data from both hosts and switches to understand traffic patterns and identify potential bottlenecks, ensuring optimal network performance.

Host-based per-flow congestion control mechanisms complement BwE by providing a rapid response to congestion on a per round-trip time (RTT) basis [26]. Fast congestion detection keeps the network loss and delay low in the face of congestion even at RTT-scale, while BwE ensures isolation between applications on a broader timescale, preventing them from interfering with each other and maintaining a predictable user experience.

7.3.4 B4 traffic engineering

B4 Traffic Engineering (TE), a centralized traffic engineering service [15] optimizes network flow paths and tunnels across inter-data center WANs. This approach yields superior resource utilization compared to traditional distributed routing methods. Traffic Engineering continuously recalculates paths in response to changing network conditions and acts as

an overlay, with traditional routing serving as a fallback in case of failures. The centralized TE server receives real-time network topology, link state updates, and flow demands, then computes a globally optimized path allocation to achieve near 100% utilization, while splitting application flows among multiple paths to balance capacity against application priority/demands. To further enhance availability, host-based mechanisms (Protective ReRoute [27]) adaptively reroute flows from end-hosts. If a flow encounters a black hole due to a network failure, path rerouting quickly switches to an alternative working path within a few round-trip times. Such rapid repathing keeps the network available for user traffic in the face of failures even at short timescales (multiple RTTs) while TE provides high-quality routes at longer timescales.

Espresso [28] extends SDN to Google's global peering edge, and was developed to address the underutilization of peering links caused by traditional Internet Protocols. Espresso dynamically selects serving endpoints based on real-time network performance rather than static assignments mapping users based on IP addresses. Additionally, Espresso centralizes traffic management logic, moving it from individual routers to a distributed system. This system leverages WSC's computing infrastructure together with application signals to analyze aggregated network data and optimize individual flows based on the end user's experience. Overall, Espresso outperforms router-centric protocols, delivering higher availability and performance.

7.3.5 DDoS attack mitigation

Distributed denial of service (DDoS) attacks attempt to disable a service by overwhelming it with synthetic network traffic, either at the network layer (by exhausting the capacity of the site's internet links) or the application layer (with requests designed to overload the application's front or back end capacity).

Cloud Armor [29] globally monitors and detects DDoS attacks. It allows network operators to manually push network Access Control Lists (ACLs) when necessary to block specific traffic patterns, and can automatically generate ACLs to thwart incoming attacks.

Some "attacks" are just genuine traffic spikes that cause a temporary bottleneck. A core objective of virtualized networks is to provide consistent availability and performance, preventing one Virtual Machine (VM) from negatively impacting another's network experience. However, "noisy neighbors" where one VM causes congestion that affects another VM's performance, poses a similar challenge. While the host's LAN link may not be oversubscribed, upstream links (inside a building, or across zones, or out of a region) will be, and thus a user suddenly sending or receiving lots of traffic may affect other users and thus needs to be contained.

Centralized systems often have delayed response times and limited information to restore isolation, while a purely host-based system can lead to excessive communication overhead from distributing state. To address these issues, a hybrid solution combines fast, reactive

receiver-driven DoS detection at the host level with a centralized DoS Server. The server receives congestion signals from hosts and utilizes additional telemetry data to accurately calculate and distribute enforcement rates, ensuring a fast mitigation for large classes of network disruptions.

7.3.6 What's next for SDN

The groundwork laid by SDN helps address the unique challenges posed by emerging AI/ML workloads. Unlike loosely coupled distributed applications, ML training jobs demand simultaneously healthy compute and networking supercomputer resources. These jobs also necessitate unprecedented amounts of accelerators and networking, leading to significantly shorter mean time between failures. In shared clusters, the competition for supercomputer resources requires dynamic reconfiguration and rebalancing of resources. Starting with TPU v4 [30], SDN has been instrumental in managing high bandwidth inter-chip interconnect (ICI), using optical circuit switching to dynamically reroute traffic around failures and increase job availability. Looking ahead, SDN's continued evolution is essential to deliver a highly available, scalable compute-networking substrate to support the ever-growing demands of AI/ML workloads [31].

Also, recent work [32] has found plausible alternatives to the standard centralized ways of implementing SDN. In this decentralized SDN (dSDN), every router runs an operator-defined dSDN controller. Each dSDN controller constructs a global network view via a simple flooding-based dissemination protocol and then locally runs a traffic engineering algorithm to compute capacity-aware paths. For simple consensus-free path selection dSDN uses source routing (a router is the sole decision maker for paths that originate at that router).

The dSDN architecture eliminates any dependency on components external to the data plane and on legacy protocols. Despite these simplifications, dSDN retains the benefits of traditional centralized SDN while restoring the fate-sharing of traditional protocols that provides resiliency. When a router goes down, the damage to the control plane is aligned with the damage to the data plane, unlike with centralized SDN. While dSDN imposes a higher computational load on the router CPU (to run TE) and additional packet header state (to carry source-routes), both are easily accommodated by modern routers.

7.4 Software-defined storage

Google optimizes storage efficiency via software-defined storage providing fungibility across various data centers and storage devices. Conventional storage systems are often deployed specifically for particular workload types. For instance, a storage system might prioritize performance by placing database workloads on high-end SSD devices, while an archival cloud service might select the largest-capacity HDD drives available leaving IOPS

unused. Given the extensive range of WSC services and workloads discussed in Chapter 3, maximizing the efficiency of the deployed data center footprint requires storage resources to be as fungible as possible.

At Google, we employ a unified global layered storage system. This system is designed to accommodate diverse workloads and shapes. To achieve maximum efficiency and interchangeability, various storage service types, such as block, database, object, key-value, and file, are all built upon a single, modular distributed storage system (Colossus). This architecture allows for seamless integration and optimization of storage resources, resulting in an efficient and globally-scalable storage solution.

7.4.1 Storage workload diversity

A global-scale storage system must accommodate a diverse range of workload types while optimizing device utilization. Storage devices combine a fixed amount of storage (bytes) with a fixed amount of access to that storage, limiting both the number I/O operations per second and the total data transfer bandwidth.

Each workload requires a different ratio of bytes vs IOPS. The simple way to match disk sizes to workloads deploys large HDDs for cold workloads and small HDDs (or SSDs) for hot workloads. Even then, it's unlikely that the workload perfectly matches available devices, leaving at least one of the device's capabilities underutilized. Additionally, applications change over time, so it's unlikely that the best fitting device today will still be the right choice a year from now.

For instance, if we were to deploy storage solely for archival storage, we would likely opt for the largest possible HDDs to get the lowest cost per byte for this "cold" workload. In this scenario, we would maximize byte efficiency but probably leave a significant portion of IOPS remaining unused. Conversely, if we were to deploy storage just for databases, we might opt for SSDs, at a much higher cost per byte.

With software-defined storage, we can combine these workloads together onto shared devices and right-size the overall fleet mix for the overall ratio of IOPS/TB. It's much easier to optimize capacity for large pools than for individual applications. For example, storage for low-IOPS archival workloads can be spread across continents or even world-wide to fill devices that are close to their IOPS limits but have free bytes.

Figure 7.7 shows the wide range of I/O needs of Google workloads that use disk storage. For each byte stored, we calculate the theoretical HDD size that it could fit on while running at 100% utilization (i.e., maximum seek rate). For example, the coldest workloads could use disks in excess of 300 TB, ten times larger than the largest actual disks available in 2024. Thus, if served by a private storage fleet, those workloads would run even the largest disks available at <10% of their available I/O rates. Conversely, the hottest bytes need much smaller disks. However, the unique workloads of Google comfortably fit on disks; even at the 80th percentile, data fits drives that are larger than what's commercially available today. In

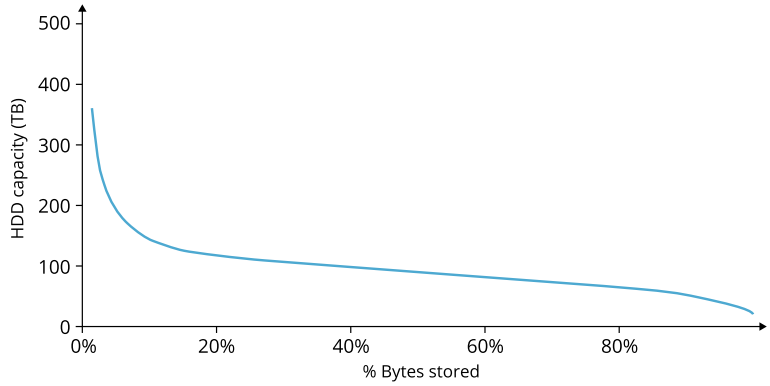


Fig. 7.7 Theoretical disk sizes (bytes versus IOPS)

that regard, Google's storage needs differ from that of typical enterprises, which do not have the large semi-archival storage demands stemming from storing people's rarely accessed old photos and videos. (At the right end of the graph, disks can no longer sustain the required IOPS and data needs to be stored on SSD, but we've omitted that portion of the graph.)

To help plan and correctly reflect actual costs, the internal billing system charges for the cost of IOPS and cost of bytes separately [33]. This cost model allows us to incentivize developers to correctly trade off between bytes and IOPS to achieve an economic balance.

To support the sharing of I/O among varied workloads, the storage API enables workload class separation. Latency-sensitive workloads are isolated from latency-tolerant workloads through distinct storage service categories. This approach ensures bounded latency for service-oriented workloads such as Gmail and YouTube, while accommodating latency-tolerant tasks like data analytics and data movement. As a result, a batch analytics job can access disks "in the background" while the storage system prioritizes services paying for latency-optimized IOPS.

7.4.2 Transparent SSD caching

With the growth in the size of deployed HDDs, the demand for IOPS has surpassed the IOPS provided by the HDDs from deployed bytes. Meeting the collective IOPS demand using current HDD sizes would require significantly more HDDs, leading to stranded bytes. To achieve an optimal balance between HDDs deployed and IOPS utilization, we leverage a global shared storage cache implemented on SSDs. By leveraging a cache behind the unified storage API, we can seamlessly balance the system without requiring modifications to existing workloads. This storage cache enables us to effectively utilize all deployed HDD bytes and IOPS capacity.

7.4.3 Trading data center space between compute and storage

Data centers face varying demands for compute and storage, affecting the ratio between them. Additionally, workloads differ in their requirements for IOPS versus bytes. To optimize resource allocation, we can leverage location-independent storage. A significant portion of disk-based storage is "cold" and thus can be placed anywhere in a particular region or continent, or even globally. Since cold storage requires little bandwidth per Tbyte of storage, placement isn't constrained by network bandwidth but only by jurisdictional requirements for data locality.

Fungible storage allows us to balance demand and capacity across data centers. For instance, if a data center has excess space and power, unoccupied by compute, we can allocate a flexible amount of cold storage capacity to fill that space over time. Moreover, if there's an imbalance between IOPS and bytes, we can relocate the cold portion of location-independent storage to that data center, utilizing available bytes that would otherwise go stranded.

7.4.4 Other software-managed hardware optimizations

While we have discussed a few notable examples above, the design space for software-managed hardware optimizations for storage is fairly large. For example, SmartFTL (Flash Translation Layer) optimizations use application-provided write hints to manage media placement to reduce write amplification [34]. Similarly, hybrid SMR HDD drives incorporate software control of how data is managed across CMR and SMR (conventional and shingled magnetic recordings) [35].

7.5 Software-defined data center

7.5.1 Software-defined power

Power oversubscription boosts data center efficiency and reduces costs by deploying more servers in a data center than its power supply can nominally support. If all servers peaked at the same time in an oversubscribed data center, they would trip breakers, or overload cooling, or overload generators, or all of the above. Thus, oversubscribed data centers need a power capping mechanism to drop a subset of load if critical power draw approaches the physical limits of the data center. (We will discuss oversubscription in more detail in Section 9.5.)

Fortunately, data centers run both high-priority serving tasks and lower-priority batch workloads. Power capping systems throttle the latter but try not to throttle the former, and thus they typically don't oversubscribe the power needed by high-priority serving tasks. With

a reasonable workload diversity (where not all loads peak at the same time) combined with significant batch loads, oversubscription can increase available capacity by tens of percent.

Software-defined power employs a hardware-agnostic power capping system [36] to allow for safe power oversubscription with minimal disruption to workloads. The system implements a two-threshold, randomized unthrottling/multiplicative decrease control policy that balances power safety with performance optimization. In addition, the system utilizes the CPU bandwidth control feature of the Linux kernel to enable task-level Quality of Service (QoS) aware throttling. This ensures that the system can efficiently allocate power resources while maintaining low latency for latency-sensitive tasks.

The software architecture depicted in Figure 7.8 includes the system's key modules. The meter watcher module is responsible for polling power readings from meters at a rate of one reading per second. These readings are then passed to the power notifier module and stored in a power history datastore for later reference.

The power notifier module serves as the central control module, implementing both reactive and proactive capping logic. When power readings are available, the module uses them for reactive capping logic. If readings are unavailable, it queries the risk assessor module for proactive capping logic. The risk assessor utilizes historical power information from the power history datastore to assess the risk of breaker trips. Thus, the system can maintain robust performance even in situations where some power telemetry data is unavailable.

If either logic determines that capping is necessary, the power notifier module passes the appropriate capping parameters to the machine manager module. The machine manager then sends remote procedure call (RPC) requests to the node controller of individual machines to reduce power consumption concurrently. The power topology datastore is a repository containing critical information about the power delivery topology. This includes details such as the protected power limits and the specific machines within a given power domain. These

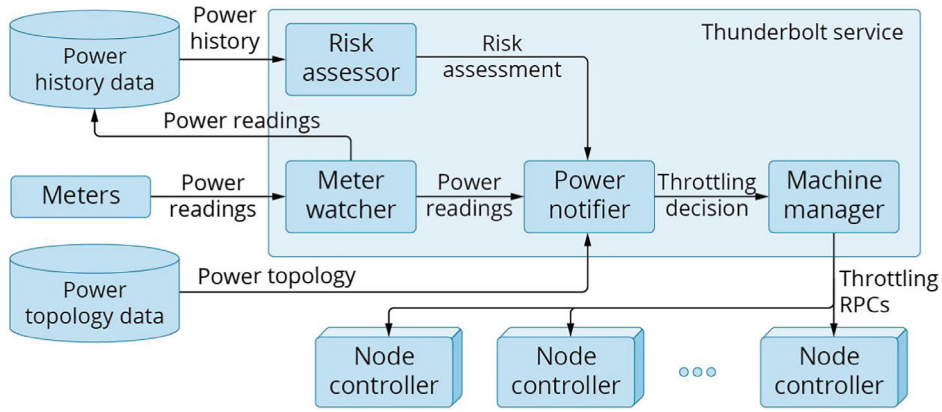


Fig. 7.8 Power capping architecture

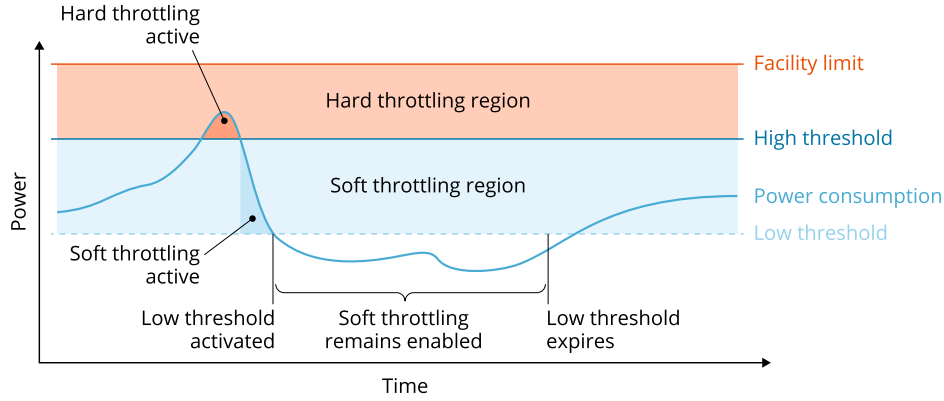


Fig. 7.9 Power throttling response over time

power domains can vary in scale, with some being as small as a few megawatts while others can be as large as tens of megawatts.

Two capping thresholds (high and low) strike a balance between responsiveness for power safety and efficiency for minimizing performance impact. If load exceeds the high threshold, hard throttling starts and sets the CPU allocation of "victim" tasks to near zero, thus effectively stopping them (Fig. 7.9). Once actual power subsequently drops below the high threshold, soft throttling takes over and progressively increases CPU allocations while attempting to keep usage below the high threshold. This soft-throttling region prevents load oscillations that would otherwise be caused by intermittent hard throttling. Below the low threshold there is no throttling. The low threshold is active only when the high threshold was reached previously, so if power briefly spikes over the low threshold but doesn't reach the high threshold, no throttling will take place. After the load has stayed below the low threshold for a certain amount of time, the low threshold timer expires and soft throttling deactivates.

This approach to software-defined power has been successfully implemented at scale in multiple production clusters, resulting in power oversubscription gains of between 9% and 25%.

7.5.2 Software-defined fleet

Software-defined infrastructure provides the convenience of automating capacity management. For example, Autocap is a comprehensive internal tool that automates capacity management for production services by taking into account long-term planning and upcoming growth needs, and redundancy targets ($N + 1$ or $N + 2$). With Autocap, cluster selection and job placement are automated.

On the other hand, another internal tool, Autopilot [37], makes short-term adjustments by using the current CPU utilization to scale jobs in real time. It can adjust both the number of concurrent tasks in a job (horizontal scaling) and the CPU/memory limits for individual tasks (vertical scaling).

Both services rely on machine learning algorithms that analyze historical data about prior executions of a job, in combination with finely-tuned heuristics. These features help maintain a balance between responsiveness and efficiency, providing optimal capacity management for production services.

7.6 Self-driving systems

Machine learning is driving computing demand, and systems are being redesigned for ML workloads. But machine learning can also be used to better design systems, specifically in the context of software-defined infrastructure to improve performance, efficiency, and resilience. We discuss some representative examples below.

Machine learning can automate the identification of suitable accelerators for large codebases by analyzing code patterns, automatically pinpointing potential acceleration targets, and suggesting optimal hardware/software co-design strategies. This accelerates development and fosters innovative interfaces and ecosystems.

Similarly, ML-driven generative design can be used to automate chip and PCB layouts, optimizing for performance, power efficiency, and reliability while accommodating heterogeneity [38], [39]. This automation enables rapid innovation and customization across diverse applications, overcoming the limitations of traditional rigid and time-consuming design processes. It's conceivable that in the future, a dozen people could design a chip in three months that today takes a hundred people a year to finish, while also optimizing to a better design.

ML can also be used in most control systems at the node level. For example, ML models can analyze system-wide telemetry data to dynamically adjust various parameters and optimize resource allocation based on real-time workload characteristics. This can lead to enhanced quality of service (QoS), efficiency, and resource utilization across a wide range of vendors, platforms, jobs, clusters, and applications.

ML can also be used in large-scale distributed systems. For example, ML-based approaches can model intricate interactions within disaggregated systems like networks, storage, and clusters. By optimizing power/cooling distribution, intelligently scheduling tasks across clusters, and minimizing tail latency, ML can enable efficient scaling, fault tolerance, and overall performance improvement in these complex environments.

While these examples show the promise of ML, we are still in the early days of using machine learning for systems. ML is proving to be an indispensable tool for addressing the diverse challenges faced by future WSC, and will likely be a much bigger part in WSC designs going forward.

References

1. A. Jain, H. Lin, C. Villavieja, et al., "Limoncello: Prefetchers for scale," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24, vol. 3, New York, NY, USA: Association for Computing Machinery, Apr. 27, 2024, pp. 577-590.
2. D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 450-462.
3. S. Dev, D. Lo, L. Cheng, and P. Ranganathan, "Autonomous warehouse-scale computers," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20, IEEE Press, Nov. 18, 2020, pp. 1-6.
4. G. Ayers, N. P. Nagendra, D. I. August, et al., "AsmDB: Understanding and mitigating front-end stalls in warehouse-scale computers," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19, New York, NY, USA: Association for Computing Machinery, Jun. 22, 2019, pp. 462-473.
5. D. Chen, D. X. Li, and T. Moseley, "AutoFDO: Automatic feedback-directed optimization for warehousescale applications," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, ser. CGO '16, New York, NY, USA: Association for Computing Machinery, Feb. 29, 2016, pp. 12-23.
6. S. Kanev, J. P. Darago, K. Hazelwood, et al., "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 158-169.
7. The Linux Kernel Archives, *HugeTLB Pages*, Kernel Documentation: https://www.kernel.org/doc/html/v4.18/admin-guide/mm/hugetlbpage.html.
8. A. H. Hunter, C. Kennelly, P. Turner, et al., "Beyond malloc efficiency to fleet efficiency: A hugepage-aware memory allocator," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 257-273.
9. Z. Zhou, V. Gogte, N. Vaish, et al., "Characterizing a memory allocator at warehouse scale," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24, vol. 3, New York, NY, USA: Association for Computing Machinery, Apr. 27, 2024, pp. 192-206.
10. A. Lagar-Cavilla, J. Ahn, S. Souhlal, et al., "Software-defined far memory in warehouse-scale computers," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, New York, NY, USA: Association for Computing Machinery, Apr. 4, 2019, pp. 317-330.
11. S. Li, G. Andersen, T. Chen, et al., "Hyperscale hardware optimized neural architecture search," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023, New York, NY, USA: Association for Computing Machinery, Mar. 25, 2023, pp. 343-358.
12. Open Networking Foundation, *OpenFlow Standard (Archived)*, https://www.openflow.org/.
13. P4.org, *P4 Language Consortium*, https://p4.org/.
14. U. Hölzle, *B4 and Google's network*, Presentation: https://www.segment-routing.net/images/hoelzle-tue-openflow.pdf.
15. S. Jain, A. Kumar, S. Mandal, et al., "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2013, pp. 3-14.
16. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special*

Interest Group on Data Communication, ser. SIGCOMM '15, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183-197.
17. M. Casado, M. J. Freedman, J. Pettit, et al., "Ethane: Taking control of the enterprise," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2007, pp. 1-12.
18. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," 9, vol. 59, New York, NY, USA: Association for Computing Machinery, Aug. 24, 2016, pp. 88-97.
19. T. Koponen, M. Casado, N. Gude, et al., "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10, USA: USENIX Association, Oct. 4, 2010, pp. 351-364.
20. L. Poutievski, O. Mashayekhi, J. Ong, et al., "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22, New York, NY, USA: Association for Computing Machinery, Aug. 22, 2022, pp. 66-85.
21. A. Shaikh and J. George, *SDN in the management plane: Openconfig and streaming telemetry*, https://research.google/pubs/sdn-in-the-management-plane-openconfig-and-streaming-telemetry/, 2015.
22. Open Compute Project, *SAI - Switch Abstraction Interface*, GitHub Repository: https://github.com/opencomputeproject/SAI.
23. sonic-net, *SONiC - Software for Open Networking in the Cloud*, GitHub Repository: https://github.com/sonic-net/SONiC.
24. H. H. Bazzaz, Y. Bi, W. Pang, et al., "Preventing network bottlenecks: Accelerating datacenter services with Hotspot-Aware placement for compute and storage," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, Philadelphia, PA: USENIX Association, Apr. 2025, pp. 317-333.
25. A. Kumar, S. Jain, U. Naik, et al., "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 1-14.
26. N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," 2, vol. 60, New York, NY, USA: Association for Computing Machinery, Jan. 23, 2017, pp. 58-66.
27. D. Wetherall, A. Kabbani, V. Jacobson, et al., "Improving network availability with Protective ReRoute," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23, New York, NY, USA: Association for Computing Machinery, Sep. 1, 2023, pp. 684-695.
28. K.-K. Yap, M. Motiwala, J. Rahe, et al., "Taking the edge off with Espresso: scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, Aug. 7, 2017, pp. 432-445.
29. Google Cloud, *Cloud Armor DoS Protection & WAF*, https://cloud.google.com/security/products/armor.
30. N. Jouppi, G. Kurian, S. Li, et al., "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23, New York, NY, USA: Association for Computing Machinery, Jun. 17, 2023, pp. 1-14.

31. Y. Zu, A. Ghaffarkhah, H.-V. Dang, et al., "Resiliency at scale: Managing Google's TPUv4 machine learning supercomputer," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 761-774.
32. A. Krentsel, N. Saran, B. Koley, et al., "A decentralized SDN architecture for the WAN," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24, New York, NY, USA: Association for Computing Machinery, Aug. 4, 2024, pp. 938-953.
33. E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. T'so, *Disks for data centers*, https://research.google/pubs/disks-for-data-centers/, 2016.
34. C. Sabol and S. Desai, *SmartFTL architecture for SSDs*, https://www.youtube.com/watch?v=3O3zDrpt3uM, 2021.
35. Open Compute Project, *OCP Hybrid SMR Product Requirements*, https://www.opencompute.org/files/OCP18-Hybrid-SMR-Product-Requirements.pdf, 2018.
36. S. Li, X. Wang, F. Kalim, et al., "Thunderbolt: Throughput-optimized, quality-of-service-aware power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI20)*, 2020, pp. 1241-1255.
37. K. Rzadca, P. Findeisen, J. Swiderski, et al., "Autopilot: Workload autoscaling at Google," in *Proceedings of the 15th European Conference on Computer Systems*, ser. EuroSys '20, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2020, pp. 1-16.
38. A. Mirhoseini, A. Goldie, M. Yazgan, et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207-212, 2021.
39. DeepMind, *How AlphaChip transformed computer chip design*, DeepMind Blog: https://deepmind.google/discover/blog/how-alphachip-transformed-computer-chip-design/.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Performance and Costs

8

8.1 Performance

8.1.1 Broad system performance characterizations

8.1.1.1 System balance for performance and efficiency

Computer architects are trained to find a balanced combination of performance and capacity from the various building blocks that make up a WSC. Below, we discuss some examples of how the right building blocks are apparent only when one considers the entire WSC system. Balance must also be addressed at this level.

Three important observations help to better frame the available options:

- Workloads aren't immutable, but they have inertia. Developers can restructure key services to better match a new design alternative. At the same time, there's a high cost (and opportunity cost) to such changes, so such optimizations tend to happen for just the most resource intensive services. Additionally, we need to be careful not to arrive at machines that are too complex to program even via a compiler. This principle influences custom accelerators like TPUs or VCUs discussed earlier in Chapter 6.
- The most cost-efficient and balanced hardware configuration may not be a perfect fit for any one workload, but it may still be the best match for the combined resource requirements of all workloads. For example, an application that is seek-limited may not fully use the capacity of a very large disk drive but could share that space with an application that needs space mostly for archival purposes. This observation motivates the software-defined infrastructure discussed in Chapter 7.
- Fungible resources tend to be more efficiently used. When comparing two alternatives, do not assume that both will achieve the same utilization! Systems that can flexibly utilize resources across multiple pools of servers will often be more efficient. This observation is frequently underappreciated.

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_8

Finally, workload churn in this space presents a challenge to WSC architects. It is likely that the software base evolves fast enough for a server design to become suboptimal during its lifetime of six years or more. For example, increasing the initial cost of a server by adding spare DIMM slots may reduce lifetime TCO by allowing later memory upgrades. But perhaps such spare slots will be superfluous because future DIMMs will double the capacity of each DIMM so that no additional slots are needed? Or perhaps double-capacity DIMMs will happen but we don't know how expensive they will be.

This issue is even more important for the WSC as a whole because the lifetime of a data center facility generally spans several server lifetimes, two decades or so. In those cases it is useful to consider the kinds of machine or facility upgrades that may be necessary over the lifetime of the WSC system and take that into account during the design phase of the facility.

8.1.1.2 Data/storage hierarchy in WSCs

Figure 8.1 shows a programmer's view of the data/storage hierarchy of a hypothetical WSC. The server consists of one or two processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and access to a NIC and an optional SSD. Racks come in several other flavors, including accelerator racks and disk or SSD racks. (Racks may not be homogeneous, so a single rack may contain different kinds of trays.)

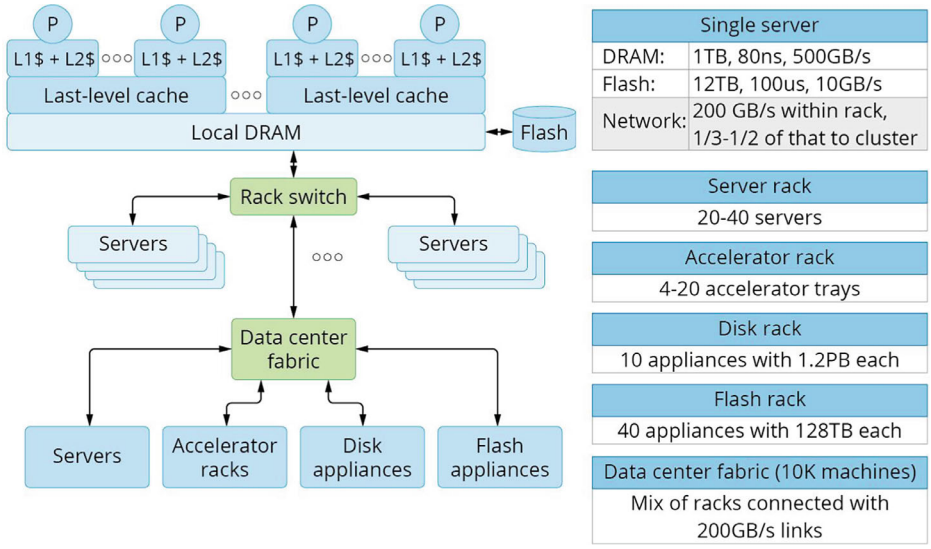


Fig. 8.1 Storage hierarchy of a WSC

All resources in a cluster are accessible via the cluster-level switch fabric. The relative balance of the resources depends on the needs of target applications and changes over time.

8.1.1.3 Quantifying latency, bandwidth, and capacity

Figure 8.2 quantifies the latency, bandwidth, and capacity characteristics of a WSC. For illustration, we assume a cluster with a 200 Gbps network fabric that is oversubscribed by 3:1. Network latency numbers assume a TCP/IP transport, and networking bandwidth values assume that each server behind an oversubscribed set of uplinks is using its fair share of the available cluster-level bandwidth.

The graph shows the relative latency, bandwidth, and capacity of each resource pool. For example, the bandwidth available from 2-4 local SSDs is about 10 GB/s. Disks are only shown at the cluster level since their performance is independent of their locality.

A large application that requires many more servers than can fit on a single rack must deal effectively with these large discrepancies in latency, bandwidth, and capacity. These discrepancies are much larger than those seen on a single machine, making it harder to program a WSC. A key challenge for architects of WSCs is to smooth out these discrepancies in a cost-efficient manner. Conversely, a key challenge for software architects is to build cluster infrastructure and services that hide most of this complexity from application developers. For example, NAND flash, originally developed for portable electronics, now is common in WSC systems and can bridge the cost and performance gap between DRAM and disks, as displayed in Figure 8.2. But intermediate non-volatile memory layers like flash also present new performance and efficiency challenges and opportunities, as traditional system design and software optimizations lack support for their microsecond-scale latencies. A new set of hardware and software technologies are needed to provide a simple programming model to achieve high performance [1].

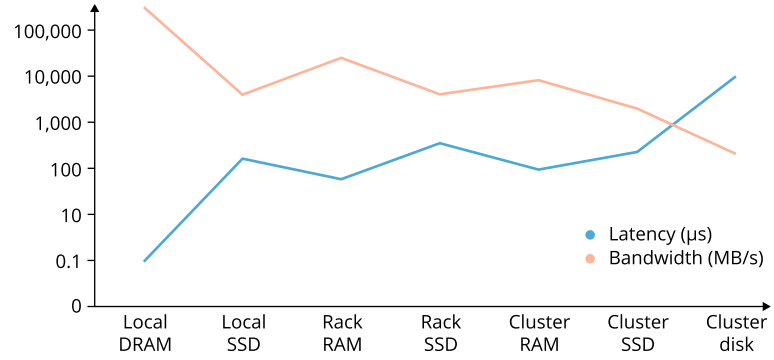


Fig. 8.2 Latency and bandwidth of WSC storage hierarchy levels

8.1.1.4 Latency numbers every engineer should know

Within Google, our colleague, Jeff Dean, maintains a summary of key latency numbers that "every engineer should know" [2]. These rough operation latencies help engineers reason about throughput, latency, and capacity within a first-order approximation. We have updated the numbers in Figure 8.3 to reflect current hardware speeds as of 2025.

8.1.2 Fleetwide workload characterization

WSCs run a wide range of workloads, with no dominant binaries (see Section 3.2). While a few products make up a large fraction of the total load (e.g., Google Cloud and Search), and some infrastructure services are really big (e.g., Spanner and Colossus), each of them consists of many different binaries that implement the microservices within that larger system. Thus, it makes sense to look across all of these services to see what they have in common.

Figure 8.4 shows a profile (generated using GWP, discussed in Chapter 4) of all workloads at Google over time, split across common functions that are used by all services. We termed these common functions the "data center tax" [3] because they're performing functions that

Operation	Time
L1 cache reference	1 ns
Branch misprediction	3 ns
L2 cache reference	4 ns
Uncontended mutex lock/unlock	16 ns
L3 cache reference	25 ns
DRAM reference (low utilization)	80 ns
DRAM reference (high utilization)	250 ns
Decompress 1 KB with Snappy	500 ns
"Far memory" (CXL) reference	1,000 ns (1µs)
Compress 1 KB with Snappy	2,000 ns (2µs)
Read 1 MB sequentially from memory	3,000 ns (3 µs)
Read 1 MB sequentially from 400Gbps network	20,000 ns (20 µs)
SSD random read	80,000 ns (80 µs)
Read 1 MB bytes sequentially from local SSD	250,000 ns (250 µs)
Read 1 MB sequentially from disk	500,000 ns (500 µs)
Disk seek	10,000,000 ns (10 ms)
Send packet California→Netherlands→California	150,000,000 ns (150 ms)

Fig. 8.3 Latency numbers that every WSC engineer should know

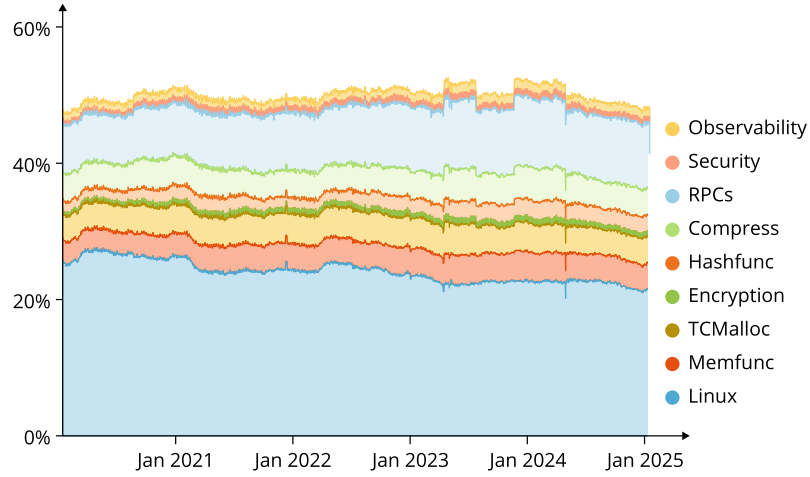


Fig. 8.4 Fleetwide cycle spent on "data center tax"

the service developer doesn't really think about because it's not code that they wrote, but rather "tax" functions that are essential for distributed large-scale WSC computing.

Most cycles are spent in the kernel, reflecting the memory and I/O intensive nature of most services. "Memfunc" denotes memory functions like memory moves and copies, and TCMalloc is the optimized memory allocator described in Section 7.1.2. RPC-related functions are the largest contributor to the rest of the "tax".

Despite many person-years of optimization, the overall fraction of cycles spent in this set of functions has stayed surprisingly stable over the years. Some have grown (esp. RPCs), and some have shrunk (esp. Kernel). This stability doesn't imply that optimization didn't work, just that the optimization gains were roughly offset by increasing usage. In fact, without continued optimization, overall cycles would have increased considerably. Still, the stable consumption highlights opportunities for building accelerators to offload the cycles consumed, freeing up CPU cycles for other general purpose tasks [4].

The same profiling system also tracks resource consumption of application-specific libraries such as video-related functions and linear algebra (for ML). Given the investment in ASICs for these two categories, their footprint on CPUs is low.

8.1.3 Performance modeling

Hyperscale infrastructure designers benefit from a deep understanding of the in-house applications running on their infrastructure. This application knowledge can provide critical feedback on the architecture and design of various hardware components that go into a typical warehouse scale computer. To benefit from this knowledge, many hyperscale infrastruc-

ture providers have started designing in-house SoCs and hardware accelerators. A robust performance modeling and simulation infrastructure enables architecture pathfinding during the design stage, workload performance estimation during the development stages, and performance validation of the final hardware after tapeout.

8.1.3.1 Simulating silicon to cluster

Performance simulation infrastructure is a key requirement for building custom silicon and can aid in many aspects of system design. Given the inhouse knowledge of applications and the vertical integration that hyperscale infrastructure designers benefit from, they typically rely on a tiered performance simulation methodology (Figure 8.5) that can model and simulate the performance characteristics: from the microarchitecture or core level simulation using traces to make early estimations of node-level performance, all the way to full cluster-level simulation of many different applications to determine production-level performance. The next section summarizes the high-level characteristics of each of these levels of simulation:

The core simulator models the proposed CPU core with cycle-level accuracy and facilitates exploration and pathfinding for core architecture features, microarchitecture optimizations, and trade-off and sizing studies.This simulator also helps evaluate the functional and performance readiness of RTL and of parts needed early during silicon bring-up, allowing simulated power-on booting before actual hardware is available.

The SoC simulator allows for the design and exploration of the architecture space by sweeping through various options and their performance and cost benefits. It provides input to system and platform designers on selecting various IPs (core, uncore, memory agents, I/O interfaces, etc.) that should comprise the system to meet workload performance requirements while balancing the total cost of system operation. Furthermore, this simulator enables exploration and pathfinding of various SoC components, such as the last level cache, mesh fabric, inter-chiplet communication, and memory controller.

The node simulator provides a full system-level emulation, simulation, and/or mock environment for studying future hardware. It enables realistic trade-offs around provisioning networking and memory bandwidth and memory capacity for representative workloads. This simulator also provides a projection methodology for workloads to evaluate performance

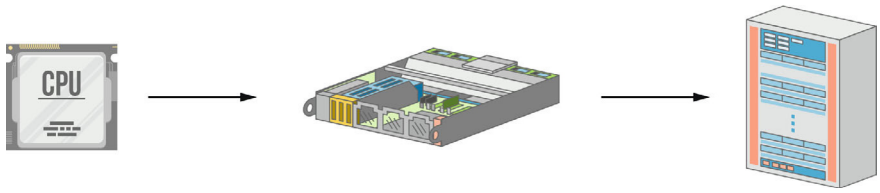


Fig. 8.5 Tiered performance simulation approach

and cost benefits compared to existing machines. Additionally, it allows for the evaluation of industry and partner devices and technology and their communication interfaces and protocols by implementing and studying preliminary datasheets or specifications within the simulation environment.

Finally, the cluster simulator enables the characterization of warehouse applications and services. It combines cluster-level scheduling with machine-level behavior to simulate real production clusters. This simulator also enables cluster-level performance projection for heterogeneous workloads and heterogeneous hardware types.

8.1.3.2 Benchmarking and tracing

Benchmarking and tracing play critical roles in performance modeling and evaluation of new hardware platforms. They help evaluate early platform performance and provide critical input for optimizations, and help identify and diagnose performance regressions caused by system software changes (BIOS/kernel updates) or platform differences. They also enable architects to analyze the performance and cost of future platforms by making data-driven decisions about architecture, shape, and features.

Warehouse scale computer workloads have fundamentally different characteristics than those captured by traditional benchmarks like SPEC. Achieving optimal efficiency for warehouse scale workloads requires changes to modern computer architecture and design. For example, Google workloads have data and instruction footprints that go beyond the capacity of modern CPU caches, such that the CPU spends a significant portion of its time waiting for code and data (Section 7.1.1). Thus, while SPEC benchmarks are useful, we complement them with other data sources to allow better performance evaluations of current or future hardware.

Fleetbench [5] is a benchmarking suite for Google workloads consisting of a curated set of microbenchmarks for hot functions across Google's fleet. The data set distributions it uses for executing the benchmarks are derived from data collected in production. It is meant to be used by chip vendors, compiler researchers, and others interested in making performance optimizations beneficial to warehouse scale workloads similar to Google's. More recently, MLPerf has been used in the industry to benchmark performance of ML systems. Large users like Google often supplement MLPerf with other internal benchmarks that more accurately represent their workloads.

Real-world workload traces from warehouse scale applications [6] are yet another tool in the arsenal that play a critical role in helping system designers better understand WSC workloads. They also enable research on processor front-ends, on-die interconnects, caches and memory subsystems, etc. — all areas that greatly impact WSC workloads. For example, memory traces are a key input to AsmDB (Section 7.1.1), which is an always-on, fleet-wide monitoring system that tracks front-end bottlenecks and enables system designers to design new optimizations.

8.1.3.3 Production performance modeling

Performance evaluation of warehouse scale infrastructure requires lightweight and non-intrusive tools that can capture and represent the characteristics of warehouse scale applications in a real production environment. For example, WSMeter [7] evaluates WSCs using actual job behaviors in live production environments. It addresses the two major challenges associated with WSC performance: first, the lack of a holistic metric that incorporates thousands of workloads and summarizes their performance, and second, the high costs and risks of conducting an evaluation in a live environment. At Google, we address these challenges using a new WSC performance metric (internally referred to as GCUs or Google Compute Units) that uses the quota-weighted sum of the MIPS (Million Instructions Per Second) rating of each job. This weighted sum captures the variety and distribution of jobs, allowing for a more accurate and cost-effective evaluation of performance in live production environments.

To address the costs of evaluation at WSC scale, a statistically-grounded sampling strategy based on independence among jobs and the Central Limit Theorem runs a constrained optimizer across a few carefully selected samples to get accurate estimates at low overheads (sometimes less than 1% of the fleet). Another optimization, Sage [8], uses unsupervised ML models to circumvent the expense of trace labeling, and learns the dependencies between various services in a warehouse scale application. Sage correctly identifies the root causes of performance issues across a diverse set of microservices using clever techniques to understand how the different microservices affect each other and test "what-if" scenarios without actually disrupting the live service. It can pinpoint the specific microservice and resource (like CPU or network) causing the issue and help fix it, leading to more predictable, performant, and efficient WSCs. Tools like Sage not only model application performance characteristics but also serve as valuable tools for use in production to debug, root-cause and correct performance and QoS regressions quickly.

8.2 Costs

8.2.1 Total cost of ownership model

As described in Chapter 2, one of the defining characteristics of warehouse-scale computers (WSCs) is their emphasis on cost efficiency at scale. To better understand this, let us examine the total cost of ownership (TCO) of a data center. At the top level, costs split into capital expenses (Capex) and operational expenses (Opex). Capex refers to investments that must be made upfront and that are then depreciated over a certain timeframe. Examples are the construction cost of a data center or the purchase price of a server. Opex refers to the recurring monthly costs of actually running the equipment, excluding depreciation: electricity costs, repairs and maintenance, salaries of on-site personnel, and so on. Thus, we have:

$$TCO = DC\ depreciation + DC\ opex + server\ depreciation + server\ opex$$

We focus on top-line estimates in this chapter, simplifying the models where appropriate. For academic purposes, our simplified model is accurate enough to model all major costs; the primary source of inaccuracy compared to real-world data centers will be the model input values, such as the cost of construction.

8.2.1.1 Capital costs

Data center construction costs vary widely depending on design, size, location, and desired speed of construction. Not surprisingly, adding reliability and redundancy makes data centers more expensive, and small data centers tend to be more expensive because fixed costs cannot be amortized over many watts.

As of 2023, construction costs varied between $6/W and $13/W depending on location [9] for large data centers. These costs have risen over the past decade due to inflation and cover all of the building shell and fitout but omit site-level costs such as the cost of land, electrical substations, or utility connections for power and water. Such costs can add another $1-2/W depending on local circumstances.

Costs vary significantly between geographic regions. In particular, labor cost and productivity can vary significantly; e.g., smaller markets may not have enough qualified electricians, requiring the project to recruit electricians from far away, which may require paying higher wages or providing subsidized housing. As a rule of thumb, about 50% of a data center's construction cost are labor costs.

Additionally, taxes and construction standards (e.g., earthquake risks) influence costs. In smaller markets, the cost index may also be influenced by the customer mix. For example, banks building custom data centers vs hyperscalers building standardized designs.

Characterizing cost in terms of dollars per watt makes sense for larger data centers (where size-independent fixed costs are a relatively small fraction of overall cost) because all of the data center's primary components—power, cooling, and space—roughly scale linearly with watts. Typically, approximately 60-80% of total construction cost goes toward power and cooling infrastructure, and the remaining 20-40% toward the general building and site construction.

Cost varies with the degree of desired redundancy and availability, and thus we always express cost in terms of dollars per critical watt; that is, watts that can actually be used by IT equipment. For example, a data center with 20 MW of generators may have been built in a $2N$ configuration and provide only 6 MW of critical power (plus 4 MW to power chillers). Thus, if construction costs $120 million, it costs $20/W, not $6/W. Industry reports often do not correctly use the term critical power, so our example data center might be described as a 20 MW data center or even a 30 MW data center if it is supplied by an electrical substation that can provide 30 MW. Similarly, power oversubscription introduced in the last chapter (and discussed more in Section 9.4) also distinguishes between provisioned watts and critical watts. If a 30 MW data center that costs $240M can be oversubscribed to 40 MW, it costs $6/W versus $8/W.

Fig. 8.6 Bitcoin mining data hut in Alberta, Canada

With sufficiently low reliability standards, it's possible to reduce construction costs dramatically. For example, crypto mining data centers can be built at $1-2/W. In exchange for that much lower cost (which also comes with a much shorter construction time) these data centers typically don't have any redundancy, are optimized for a single use case (high density ASIC or GPU racks), and tolerate higher ambient temperature variations and higher equipment failure rates. Miners use many different designs, using direct air cooling for minimal cost, or sometimes immersion cooling for specialized chips. For example, the containerized data center huts in Figure 8.6 are open to the environment and have a capacity of up to 1.2MW each, with a list price of $1M [10]. In addition to the container costs, a deployment also includes additional costs for external transformers and power lines, site work (grading and gravel), fencing, etc, so the final cost will be more than $1/W.

Sometimes, construction cost is quoted in dollars per square foot, but that metric is less useful because it cannot adequately compare projects and is used even more inconsistently than cost expressed in dollars per watt. In particular, there is no standard definition of what space to include or exclude in the computation, and the metric does not correlate well with the

primary cost driver of data center construction, namely critical power. Thus, most industry experts avoid using dollars per square foot to express cost.

The monthly depreciation cost (or amortization cost) that results from the initial construction expense depends on the duration over which the investment is amortized (related to its expected lifetime) and the assumed interest rate. Most data center equipment is depreciated over periods of 15-25 years and up to 40 years for a building shell. Under U.S. accounting rules, it is common to use straight-line depreciation where the value of the asset declines by a fixed amount each month. For example, if we depreciate a $12/W data center over 12 years, the depreciation cost is $0.08/W per month. If we took out a loan to finance construction at an interest rate of 8%, the associated monthly interest payments add an additional $0.05/W, for a total of $0.13/W per month. Interest rates vary over time, but many companies use a cost of capital rate in the 7–12% range.

Server costs are computed similarly, except that servers have a shorter lifetime. In recent years, all major cloud providers have switched to using 6 years for server depreciation, based on the increased observed lifetimes of servers. Here, lifetime refers to the economic lifetime (how long is it economically viable to run the server) vs technical lifetime (how long can the server run before it breaks). Lifetime has increased not because servers have higher quality and thus last longer, but because Moore's Law has slowed down. To see why, let's assume server performance doubles each year while prices stay the same. After three years, an old server has just 1/8th of the performance and thus it's no longer economical to run the server even if it is fully depreciated ("free") because the cost of running this server (power, space, etc) is more than 1/8th of that of the newest server. In contrast, if server price/performance improves by only 10% a year, even a five year old server is still price/performance competitive.

To normalize server and data center costs, it is useful to characterize server costs per watt as well, using the server's peak real-life power consumption as the denominator. For example, a $16,000 server with a peak power consumption of 1100 W costs $14.5/W. Depreciated over 6 years, the server costs $0.20/W per month and 8% annual interest adds another $0.05/W per month, for a total of $0.25/W per month.

As discussed in earlier chapters, with the slowdown of Moore's law, WSCs are increasingly turning to hardware accelerators to improve performance per watt. The Capex of internally developed accelerators also includes non-recurring engineering (NRE), the cost of designing and fabricating the ASIC, as well the surrounding infrastructure. If developing and deploying 100,000 accelerators has a one-time cost of $50M, the NRE cost is $500 per unit, so each chip is $500 more expensive than its pure manufacturing cost. Thus, custom hardware typically only makes economic sense if it's produced in high volumes and provides a significant cost/performance advantage, otherwise the development costs and risks outweigh the benefits.

8.2.1.2 Operational costs

Data center operational expense (Opex) depends heavily on operational standards (for example, how many security guards are on duty at the same time or how often generators are tested and serviced) as well as on the data center's size. Larger data centers are cheaper (per MW) because fixed costs are amortized better. Costs can also vary depending on geographic location (climate, taxes, salary levels, and so on) and on the data center's design and age. For simplicity, we will break operational cost into a monthly charge per watt that represents items like security guards and maintenance, and electricity. Typical operational costs for large data centers in the United States range from $0.05 to $0.10/W per month, excluding depreciation, interest, and the actual electricity costs. (Note that these numbers reflect costs, not market prices charged to customers.)

Similarly, servers have an operational cost. Server maintenance costs vary greatly depending on server type and maintenance standards (for example, response times for four hours versus two business days) as well as data center size.

In traditional IT environments, server costs are very different because the bulk of operational cost lies in the applications; that is, software licenses and the cost of system administrators, database administrators, network engineers, and so on. We are excluding these costs here because we are focusing on the cost of running WSC infrastructure, but also because application costs vary greatly depending on the situation. In small corporate environments, it is not unusual to see one system administrator per a few tens of servers, resulting in a substantial per-machine annual cost [11].

8.2.2 TCO breakdowns and case studies

Given the large number of variables involved, it is best to illustrate the cost factors by looking at scenarios that represent different kinds of deployments.

First, we consider a typical new multi-megawatt corporate data center in the United States (something close to the Uptime Institute's Tier III classification, discussed in Chapter 5). The remaining base case parameters chosen are as follows:

- The cost of electricity is the 2024 average U.S. industrial rate of 8 cents/kWh.
- The interest rate a business must pay on loans is 8%, and we finance the servers with a 6-year lease. This assumption favors the on-premise case since most companies use higher rates for their cost of capital.
- The cost of data center construction is $10/W, amortized over 20 years and financed at the same interest rate of 8%.
- Data center Opex is $0.05/W per month.
- The data center has a power usage effectiveness (PUE) of 1.5, the current industry average
- Server lifetime is 6 years, and server repair and maintenance is 5% of Capex per year.
- The server's average power draw is 75% of peak power.

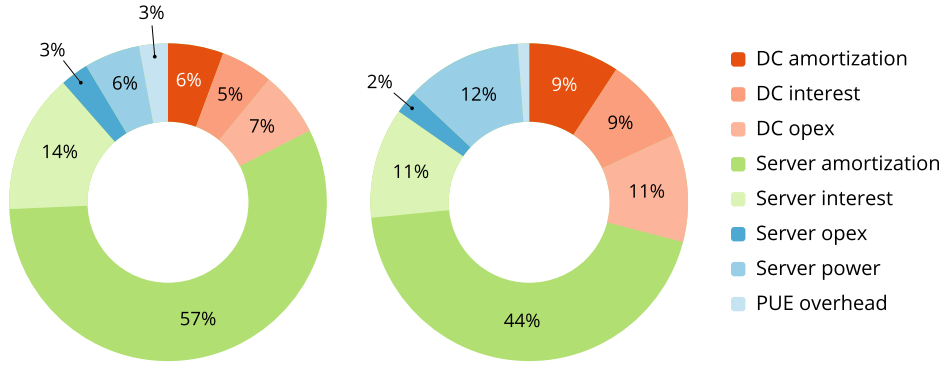


Fig. 8.7 Cost breakdown for case study A (left) and B (right)

These values, and the values below, are intended to be broadly representative of real-world costs.

For case A, we choose servers that are what a typical enterprise company would buy: a two-socket server with two fast 16-core CPUs, 512 GB of RAM, and 2 TB SSD. Let's say this server draws 950 W at peak and costs approximately $29,000 as of 2024. We'll assume that the data center is fully populated (an unrealistic assumption, but hold that thought).

The left side of Figure 8.7 shows a breakdown of the yearly TCO for case A, split into data center and server-related Opex and Capex components. In this example, typical of classic data centers, the high server capital costs dominate overall TCO, with 71% of the monthly cost related to server purchase and an additional 11% to server opex.

Commodity-based lower-cost (and perhaps lower-reliability) servers, or higher power prices, can change the picture quite significantly. For case B (Figure 8.7 right), we assume a cheaper, faster, higher-powered server consuming 1100 W at peak and costing only $16,000 in a location where the electricity cost is $0.10/kWh and the PUE 1.1. In this case, data center-related costs rise to 29% of the total, and energy costs rise to 12%. In other words, the server's hosting cost (that is, the cost of all infrastructure and power to house it) is starting to approach the cost of purchasing the server.

Note that even with the assumed higher power price and higher server power, the absolute 6-year TCO in case B is lower than in case A ($36,000 versus $51,000) because the server is cheaper.

8.2.3 Real world costs: utilization matters!

In fact, real-world data center costs are significantly higher than modeled so far. The models presented so far assume that the data center is 100% full and the servers are fairly busy (75% of peak power corresponds to a CPU utilization of about 50%; see Chapter 9). In

reality, such high utilization is uncommon, both for data centers and servers. For example, because data center space takes a while to build, we may need to keep some empty space to accommodate future growth. In addition, server layouts often assume overly high (worst case) power consumption. A server may consume up to 1100 W with all options installed (maximum memory, disk, PCI cards, and so on), but the actual configuration deployed may only use 500 W. If the server layout assumes the nameplate rating of 1100 W, we will reach a utilization factor of only 45% and thus the actual data center cost per server more than doubles. Thus, in reality, the actual monthly cost per server is often considerably higher than shown in cases A and B because the data center-related costs increase inversely proportional to data center power utilization.

As discussed later in Chapter 9, reaching a high data center power utilization is not as simple as it may seem. Even if the vendor provides a power calculator to compute the actual maximum power draw for a particular configuration, that value will assume 100% CPU utilization. If we install servers based on that value and they run at only 30% CPU utilization on average (consuming 400 W instead of 1100 W), we just stranded more than half of the data center's power, and more than doubled the cost! However, if we installed servers based on the average value of 400 W and at month's end the servers actually run at near full capacity for a while, our data center may overheat or trip a breaker. Similarly, we may choose to add additional RAM or disks to servers at a later time, which would require physical decompaction of server racks if we had left no slack in our power consumption calculations.

Thus, in practice, private data center operators leave a fair amount of slack space to guard against these problems. Reserves of 20–50% are common, which means that real-world data centers rarely run at anywhere near their rated capacity. In other words, a data center with 10 MW of critical power will often consume a monthly average of just 4–6 MW of actual critical power (plus PUE overhead).

To model a partially filled data center, we scale the Capex and Opex costs (excluding power) by the inverse of the occupancy factor. For example, a data center that is only half full has a 100% higher unit costs per watt. Taking case B above but with a 50% occupancy factor, data center costs represent now 45% of TCO (Figure 8.8), up from 29%. This case is more typical of real-world costs.

Partially used servers affect operational costs in a positive way because the servers use less power. Of course, these savings are questionable because the applications running on those servers are likely to produce less business value. Our TCO model cannot capture this effect because it is based on the cost of the physical infrastructure only and excludes the application running on this hardware. To measure this end-to-end performance, we can measure a proxy for application value (for example, the number of bank transactions completed or the number of web searches) and divide the TCO by that number. For example, if we had a data center costing $1 million per month and completing 100 million transactions per month, the cost per transaction would be 1 cent. On the other hand, if traffic is lower at one month and we complete only 50 million transactions, the cost per transaction doubles to 2 cents. In this

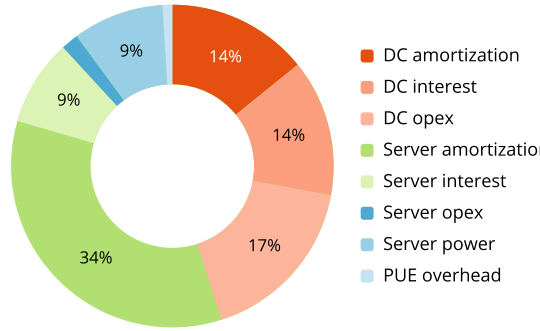


Fig. 8.8 TCO case study C (50% filled facility)

chapter, we have focused exclusively on hardware costs, but it is important to keep in mind that, ultimately, software performance and server utilization matter just as much. Such issues are also exacerbated in the context of accelerators that deliver higher value but also incur additional costs for the software ecosystem support.

8.2.4 Cost advantages of public clouds

Instead of building their own data centers and servers, enterprises can rent a virtual machine (VM) from a public cloud provider such as Google's Compute Engine or Amazon's EC2. The server used in our prior example is roughly comparable to a GCE c3d-highmem-60 instance, assumed at $3.67 per hour as an on-demand instance, $1.47 per hour as a spot instance, or $1.65 per hour with a three-year commitment, ignoring instance storage costs. Accounting for the 2 TB SSD in our example, our per-hour cloud instance costs become $3.94 (on-demand), $1.74 (spot), or $1.92 (three-year commitment). All these numbers are current as of mid-2024.

Before we compare these with our cost model, consider the three very different pricing plans. On-demand pricing optimizes for "pay-as-you-go" flexibility. You can start and stop a VM at any time, so if you need one for only a few days a year, on-demand pricing will be vastly cheaper than other alternatives. For example, you may need two servers to handle your peak load for four hours per day on weekdays, and one server during the rest of the time. With an on-demand instance, you pay for only 20 hours per week versus 168 hours if you owned the server, for an 8:1 savings.

On-demand instances are more expensive: at $3.94 per hour, using one for three years would cost $103,500. But steady-state usage isn't what on-demand VMs are used for; using the above example where the on-demand server is needed for only 20 hours per week, total on-demand costs for three years are $12,300.

If you need a server continuously for three years, public cloud providers will discount the hourly price in exchange for this long-term commitment. With a three-year commitment, GCP discounts the example server by over 55%, resulting in a three-year cost of $50,500.

Why are on-demand instances relatively expensive? Since the cloud provider doesn't know whether you're going to need a server, it must keep additional servers ready in case someone wants them. Thus, the utilization of the server pool used for on-demand instances is substantially lower, and thus the revenue per month is lower as well. For example, if the typical on-demand instance covers the six hours a day when traffic peaks, their utilization will be 25%, and thus their cost per hour is four times higher than that of a "base load" instance that runs 24 hours a day.

Since on-demand instances aren't utilized all day, most cloud providers offer their idle capacity at reduced rates as "spot" or "preemptible" VMs. These VMs aren't guaranteed to be available (if you try to obtain one, you may fail to get one) and they can be preempted (i.e., terminated) on short notice, but they are cheap. Of course, the workload must tolerate preemption in this form; for example, many batch jobs will.

Let's compare the cloud VM to an on-premises server using a hypothetical but realistic example. As of August 2024, a comparably configured Dell PowerEdge R7525 lists for $29,000 (which would be discounted substantially for large customers) and an equivalent SuperMicro AS-1125HS-TNR server is $11,000. Adding interest, data center and power costs over the server life time of 6 years results in raw total costs of $49,000 and $26,900, respectively.

Thus, at first glance, the on-premises cost looks a lot cheaper: six years cost about the same as three years on GCP (for Dell), or about half as much (SuperMicro). But the reality is a bit more complicated, because public cloud isn't the same as server hosting. We'll walk through that in more detail, starting with hardware differences:

- The data center costs for the on-premises servers assumed a PUE of 1.5 (excellent for a large corporate data center) and an unrealistic 100% data center utilization. Using a more realistic PUE of 1.8 and a data center utilization of 50% adds about $10,500 over six years.
- When hardware fails, a replacement VM is available immediately in the cloud. On-premises, a service contract with a four-hour response (not resolution!) SLA costs 5-10% of the equipment cost per year. To be conservative we'll discount this to $5,000 for 6 years.
- The local SSD of the on-premises server is a raw device with no redundancy. The equivalent "persistent disk" in GCP is more akin to a network-attached RAID SSD with high availability and durability. Most enterprise applications need the higher reliability, which at least doubles the on-premises storage cost (assuming a second SSD with mirroring). The actual equivalent would be an SSD NAS (which also supports snapshots for backups), which is significantly more expensive. However, to be conservative, we'll assume that this isn't needed and only add another $4,000 of lifetime cost for a second SSD and basic backup software. We'll additionally assume that failures are free because they're covered by the hardware maintenance contract above, and there are no follow-on costs from lost data. (A bold assumption!)

- On-premises servers, once bought, incur costs whether they're used or not. So if a company has intermittent computing demands, an on-premises setup offers few good alternatives, whereas in a public cloud, three-year commitments can be mixed with on-demand or spot usage to match the workload demands. Similarly, a cloud VM is easier to repurpose over its lifetime, without additional costs. For example, it could be moved to a different zone to become the failover server for another server, something that will cost hundreds of dollars of time and materials to do on-premises. To be conservative, we will assume that the on-premise setup is highly stable, so that on-premise underutilization costs just 10% of total costs. (While the concept of hosting the baseload capacity and "bursting" into the cloud has been around for a while, in practice, it is too complicated to do because the infrastructure stacks are too different.)
- Additionally, the cloud server only requires a three-year commitment vs the six-year on-premises ownership, so it's much less likely to become obsolete. How likely is the on-premise server to be a great fit for the future workloads of 4-6 years from now? To be conservative, we will assume that the obsolescence risk during the second three-year phase is only 33%. That is, the average on-premise server is just as ideal during the first five years as it is initially, and unused in the sixth year only.

Thus, after accounting for hardware differences and realistic data center utilization, the three-year on-premises cost conservatively rises to $49,000 and $31,000, respectively, compared to three years of GCP at $50,000.

Now, let's look at the software differences because the GCP VM equivalent to an on-premises server doesn't come with just hardware and hosting, it comes with a rich service stack: virtualization, networking, load balancing, network-attached storage, machine image distribution, centralized console logs, basic security features, managed OS images, monitoring, and so on. Each of these services would cost extra in an on-premises scenario:

- A typical virtualization license costs about $50/core/year, or about $3000/yr for our example server. Even if open source software was used, an employee needs to install, configure, and upgrade it, also costing thousands of dollars per year. We'll conservatively assume $2500/yr for virtualization.
- The cloud solution also includes network and storage management, centralized logging, load balancing and firewalls, and basic security and backup functions, all at no or very low cost. There are many on-premises products in those spaces, with widely ranging costs; it's not uncommon for enterprises to spend more on such functionality than on server hardware. We'll assume another $2000/yr for these costs (a very conservative estimate).

Accounting for just this first layer of software and service, the commodity on-premises server is only slightly cheaper than a GCP VM (Fig. 8.9), despite several optimistic assumptions that all favor the on-premise case. In particular, most enterprises will see much higher costs from underutilization and obsolescence, and their capital cost is closer to 11-12% vs

	On-prem	Cloud
Three-year hardware TCO (realistic utilization / PUE)	$23,000	$50,500
Hardware repair and maintenance	$2,500	—
On-prem underutilization from missing on-demand VMs	$6,000	—
On-prem obsolescence costs beyond first three years	$2,000	—
Virtualization software	$7,500	—
All other system software	$6,000	—
Total	$47,000	$50,500

Fig. 8.9 On-premises vs. cloud comparison

the 8% we assumed. Thus, most companies will already see cloud savings at this point; predicting server needs six years ahead of time is very hard. (It is telling that none of the major clouds offer commitments longer than three years, suggesting there is little demand for them.) Additionally, for all but the largest companies, people costs (for systems and database administrations) will be significantly higher than we assume here.

However, we are not done with the comparison. Public cloud provides seamless access to a large catalog of services from the cloud provider and third parties: managed databases, data analytics, a global WAN, firewalls, managed backups, etc. In an on-premises environment, each of these requires significant additional staffing to buy, install, and maintain. On top of that effort, on-premises implementations incur opportunity costs. If additional computing, storage, or networking is needed, procuring and installing it may take weeks or months, during which the business need driving the upgrade is unmet. Additionally, public clouds provide SaaS applications for almost any vertical. For enterprises, this additional functionality and its "as a service" management and integration is the additional decisive factor that makes a cloud approach much cheaper. Since it's harder to quantify these benefits in terms of dollars per VM we've mentioned them last, but for many enterprises they come first. In fact, most enterprises don't want to buy servers or VMs, they prefer managed PaaS services (like Cloud Run) where they can reduce their development and maintenance costs even further.

The key insight from this cost analysis is that public cloud computing isn't the same as server hosting. If the only requirement is managed hardware of a fixed fleet, a colocation provider is the cheaper solution because it's mostly a hardware solution. But few companies need just server hosting: for enterprise uses where significant functionality is needed, and where data center and server utilizations of even 50% are hard to achieve, it is hard to beat public cloud economics. And even though this book focuses on the data centers and hardware layer, public clouds are primarily providing value via a software stack and ecosystem, not server hosting.

References

1. L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.
2. J. Dean, "Designs, lessons and advice from building large distributed systems," *Keynote from LADIS*, vol. 1, 2009.
3. S. Kanev, J. P. Darago, K. Hazelwood, et al., "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 158–169.
4. S. Karandikar, A. N. Udipi, J. Choi, et al., "CDPU: Co-designing compression and decompression processing units for hyperscale systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23, New York, NY, USA: Association for Computing Machinery, Jun. 17, 2023, pp. 1–17.
5. A. Abel, Y. Li, R. O'Grady, C. Kennelly, and D. Gove, "A profiling-based benchmark suite for warehouse-scale computers," in 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2024, pp. 325–327.
6. V. Lee, *Workload traces for google warehouse-scale computers*, Google Cloud Blog: https://cloud.google.com/blog/topics/systems/workload-traces-for-google-warehouse-scale-computers.
7. J. Lee, C. Kim, K. Lin, et al., "WSMeter: A performance evaluation methodology for Google's production warehouse-scale computers," 2, vol. 53, New York, NY, USA: Association for Computing Machinery, Mar. 19, 2018, pp. 549–563.
8. Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Practical and scalable ML-driven cloud performance debugging with Sage," *IEEE Micro*, vol. 42, no. 4, pp. 27–36, 2022.
9. Turner & Townsend, *Data Centre Cost Index 2023*, https://reports.turnerandtownsend.com/dcci-2023/data-centre-cost-trends, 2023.
10. Bitfury, *BlockBox AC - Crypto Infrastructure*, https://bitfury.com/crypto-infrastructure/blockbox.
11. E. Bauer, *The total cost of ownership of open source software: A report for the UK Cabinet Office supported by OpenForum Europe*, https://evanbauer.com/essays/LinuxTCO.pdf, 2005.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Energy and Sustainability

9

9.1 The importance of energy in WSCs

Warehouse-scale computers have a singular purpose: to convert input energy in the form of electricity into output in the form of useful services, such as public clouds, web search, video streaming, social media, personalized assistants, data analytics, and so on. As a side effect of processing, all that energy turns into heat. Thus, WSC designers have two big efficiency opportunities: reduce the amount of processing energy, or the amount of cooling energy.

Two decades ago, the state of the industry was still poor: the average real-world data center and the average server were both inefficient, mostly because efficiency was historically neglected and took a backseat relative to reliability, performance, and capital expenditures. As a result, the average data center wasted two thirds (!) or more of its energy: inside most servers, at least a third of the plug power got lost in inefficient power conversions before it reached actual chips, and the building used as much power for its own systems as did the servers themselves.

This kind of inefficiency was expensive. Why was it tolerated? Interestingly, most of the time nobody knew about it. Three factors contributed to five decades of inefficiency:

- At the beginning of commercial computing in the 1960s, mainframe computers were fickle and very expensive per watt, and thus no expense was spared to keep them optimally cool. Additionally, mainframe power consumption was small.
- Data center operators knew that the key to professional success was "no outages," and thus weren't incentivized to look for improvements: any change to the tried and true way of running data centers represented a risk, and risk was seen as bad. As a result, while servers hosted in data centers were much sturdier and much cheaper per watt, data centers barely changed.

© The Author(s) 2026
L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_9

- Nobody saw the TCO (total cost of ownership) of computing: the bill for new servers went to the IT department, but the bill for power and cooling went to the facilities department. Thus, IT buyers didn't demand server efficiency because they didn't know that slightly more expensive, efficient servers would actually be cheaper.

The upside of this history of neglect was that sizable improvements were easy to obtain twenty years ago. An overall factor of two in efficiency improvements was possible, without much risk, by simply applying best practices to data center and server designs. Thus, over a short period of 2-3 years starting 2005, state-of-the-art PUEs radically improved to 1.2, and since then it has gradually improved further to about 1.1.

Today, state-of-the-art WSCs are fairly close to optimum efficiencies in their power and cooling systems, operating within 10-15% of the theoretically optimal efficiency. Thus, greater opportunities for gains in energy efficiency from now on will need to come from computer scientists and engineers (for example, by optimizing software), and less so from mechanical or power conversion specialists.

We cover several key themes in this chapter:

- Power and energy must be well managed to minimize operational cost. Higher power results in higher provisioning costs, electricity usage costs, and environmental impact.
- Improving energy efficiency at low utilizations is a tremendous opportunity. Low utilization is common for IT loads, and energy efficiency tends to be poor at lower utilizations.
- Energy optimization is a complex end-to-end problem, requiring intricate coordination across all layers of the stack. A decision that is optimal in one layer of the stack can ruin the efforts of a different layer. Real progress requires an end-to-end view of energy efficiency.
- Hardware specialization is a valuable tool to improve energy efficiency. The era of free efficiency improvements from general purpose CPUs is over, whereas ASICs and specialized workload-customized designs offer better performance and energy efficiency at the cost of lower generalizability.
- Improving server utilization improves both energy efficiency and cost efficiency: if you pay for all those servers, they better be doing something useful.

9.2 Energy efficiency of data center infrastructure

The broadest definition of WSC energy efficiency would measure the energy used to run a particular workload (say, to sort a petabyte of data). Unfortunately, no two companies run the same workloads and, as discussed in Chapter 3, real-world application mixes change all the time, so it is hard to benchmark WSCs this way. Thus, even though such benchmarks have been contemplated [23], they haven't yet been widely used. However, it is useful to

view energy efficiency as the product of three factors that we can independently measure and optimize, as shown in the equation below:

$$\begin{aligned} Efficiency &= Computation/Total\ Energy \\ &= \underbrace{\frac{1}{PUE}}_{(a)} \cdot \underbrace{\frac{1}{SPUE}}_{(b)} \cdot \underbrace{\frac{Computation}{Energy\ To\ Components}}_{(c)} \end{aligned}$$

In this equation, the first term (a) measures facility efficiency, the second (b) measures server power conversion efficiency, and the third (c) measures the server's architectural efficiency. We discuss these factors in the following sections.

9.2.1 PUE

Power usage effectiveness (PUE) reflects the quality of the data center building infrastructure itself [1], and captures the ratio of total building power to IT power (the power consumed by the computing, networking, and other IT equipment). IT power is also referred to as "critical power." We can easily measure PUE by adding electrical meters to the lines powering the various parts of a data center.

$$PUE = \frac{Facility\ Power}{Equipment\ Power}$$

Historically, the PUE for the average data center has been embarrassingly poor. According to a 2006 study [2], 85% of data centers were estimated to have a PUE greater than 3.0. In other words, the building's mechanical and electrical systems consumed twice as much power as the actual computing load. Only 5% had a PUE of 2.0 or lower. A subsequent EPA survey of over 100 data centers reported an average PUE of 1.91 [3]. A 2012 Uptime Institute survey of over 1100 data centers covering a range of geographies and sizes reported an average PUE value between 1.8 and 1.89 [4].

Figure 9.1 shows the distribution of results from a 2012 study. Large facilities reported the biggest improvements, and about half of small data centers (with less than 500 servers) were still not measuring PUE.

A 2016 report from LBNL noted PUEs of 1.13 for hyperscale data centers (warehouse-scale computers) and 1.6 to 2.35 for traditional data centers [5]. The most recent survey from the Uptime Institute in 2022 [6] shows the average PUE largely stagnant from 2018 onwards (Figure 9.2).

In contrast, WSC operators rapidly improved the PUE of their designs, starting with Google in 2005, and many started to report their PUEs. At scale, it is easy to justify the importance of efficiency; for example, Google has saved billions of dollars to date from energy efficiency measures. Google's initial PUE report showed a world-wide average of

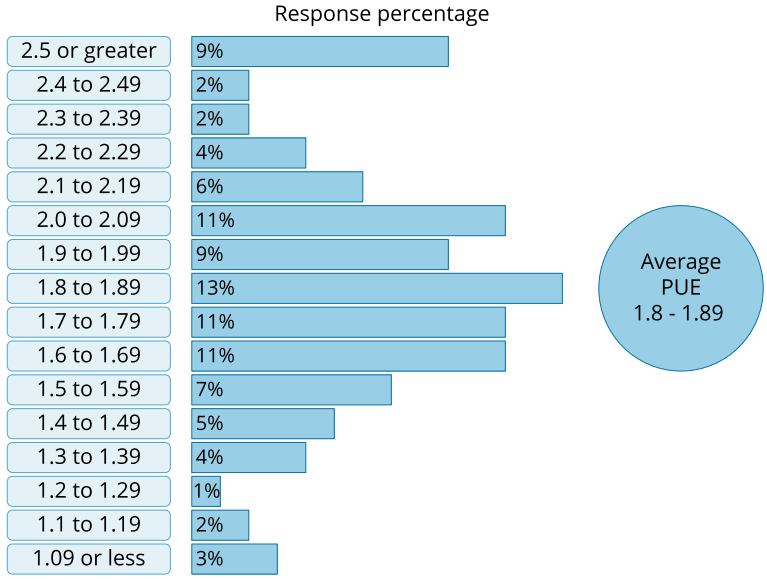


Fig. 9.1 2012 survey of PUE for 1100+ data centers

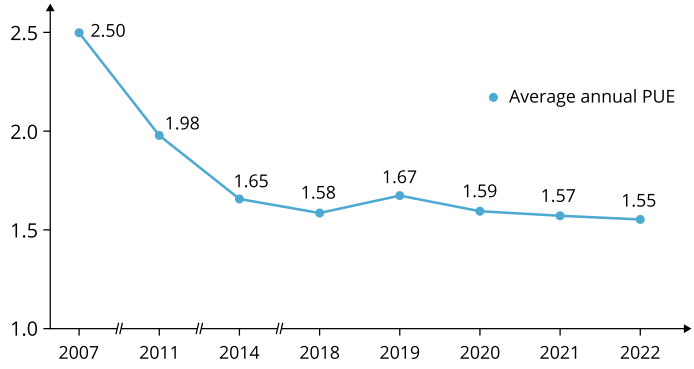


Fig. 9.2 Uptime Institute survey tracking PUE over time

1.2 in 2008 and subsequently improved to 1.09 over the years (Figure 9.3) [7]. Annual peaks in PUE values correspond to summer months when the cooling overhead is higher.

9.2.2 Issues with the PUE metric

Although The Green Grid publishes detailed guidelines on how to measure and report PUE [8], many published values aren't always comparable, and sometimes PUE values are used in marketing documents to show best-case values. The biggest factors that can skew PUE values are as follows:

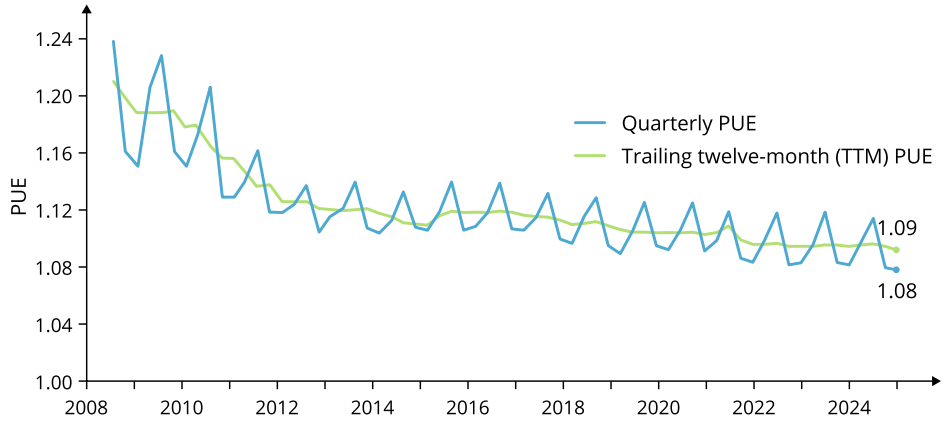


Fig. 9.3 Average global PUE for Google data centers over time

- Not all PUE measurements include the same overheads. For example, some may include losses in the primary substation transformers or in wires feeding racks from PDUs, whereas others may not. Google is reporting its fleet-wide PUE of 1.09 using a comprehensive definition of overhead that includes all known sources, but could have reported a PUE of 1.06 with a more "optimistic" definition of overhead [7]. For PUE to be a useful metric, data center owners and operators should adhere to The Green Grid guidelines [8] in measurements and reporting, and be transparent about the methods used in arriving at their results.
- Instantaneous PUEs differ from average PUEs. Over the course of a day or a year, a facility's PUE can vary considerably. For example, on a cold day, it might be low, but during the summer it might be considerably higher. Generally speaking, annual averages are more useful for comparisons.
- Some PUEs aren't real-world measurements. Often vendors publish "design" PUEs that are computed using optimal operating conditions and nominal performance values, or they publish a value measured during a short load test under optimal conditions. Typically, PUE values provided without details fall into this category.
- Some PUE values have higher error bars because they're based on infrequent manual readings, or on coarsely placed meters that force some PUE terms to be estimated instead of measured. For example, if the facility has a single meter measuring the critical load downstream of the UPS, PDU and low-voltage distribution losses will need to be estimated.

In practice, PUE values should be measured in real time. Not only does this provide a better picture of diurnal and seasonal variations, it also allows the operator to react to unusual readings during day-to-day operations. For example, someone may have left on a

set of backup pumps after a periodic test. With real-time metrics, the operations team can quickly correct such problems after comparing expected versus actual PUE values.

The PUE metric has also been criticized as not always indicating better energy performance, because PUEs typically worsen with decreasing load. For example, assume a data center's PUE is 2.0 at a 5 MW load versus 1.5 at a 10 MW load. If it's possible to run the given workload with a 5 MW load (for example, with newer servers), that clearly is more energy efficient despite the inferior PUE. However, this criticism merely points out that PUE is just one of three factors in the efficiency equation shown earlier in this chapter, and overall the widespread adoption of PUE measurements has arguably been the driver of the biggest improvements in data center efficiency in the past 50 years.

9.2.3 Breakdown of energy inefficiencies

The section on data center power systems in Chapter 5 describes the efficient transformation of power as it approaches the data center floor. The first two transformation steps bring the incoming high-voltage power (110 kV and above) to medium-voltage distribution levels (typically less than 50 kV) and, closer to the server floor, to low voltage (typically 480 V in North America). Both steps should be very efficient, with losses typically below half a percent for each step.

Inside the building, conventional double-conversion UPSs cause the most electrical loss, operating at 88–94% efficiency under optimal load, significantly less if partially loaded (which is the common case). High-efficiency UPSs can reach efficiencies of about 97%, and in-rack UPSs and batteries can improve that even further. The final transformation step in the PDUs accounts for an additional half-percent loss, and 1–3% of power can be lost in the cables feeding low-voltage power (110 or 220 V) to the racks. (Recall that a large facility can have a raised floor area that is over 100 meters long or wide, so power cables can be quite long.) Most modern WSC designs thus use bus bars to deliver power along the server row, with only short cables connecting the bus bar to the rack.

Figure 9.4 shows the typical distribution of energy losses in a conventional, low efficiency data center with a PUE of 1.5. For clarity, this breakdown omits additional losses due to server fans or electrical resistance on server boards.

The most significant portion of data center energy inefficiencies stems from cooling overhead, with chillers being the largest culprit. In an average conventional data center, cooling losses are about three times greater than power losses, presenting the most promising target for efficiency improvements: if all cooling losses were eliminated, the PUE would drop to 1.18, whereas a zero-loss UPS system would yield a PUE of only 1.35. Typically, the higher a facility's PUE is, the higher the percentage of the total loss comes from the cooling system [9]. Intuitively, there are only so many ways to mishandle a power distribution system, but many more ways to mishandle cooling.

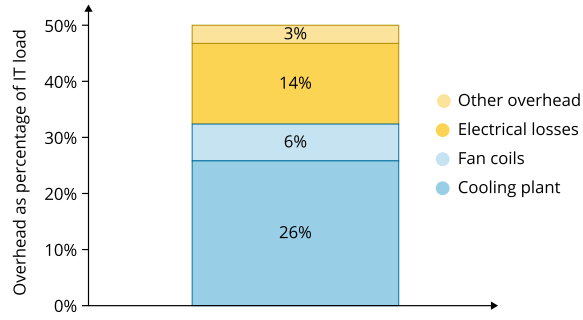


Fig. 9.4 Energy losses in an average low-efficiency data center

9.2.4 Improving data center efficiency

As discussed in the previous chapter, careful design for efficiency can substantially improve PUE [10], [11]. To summarize, the key steps are:

- Careful air flow handling: Isolate hot air exhausted by servers from cold air, and keep the path to the cooling coil short so that little energy is spent moving cold or hot air long distances.
- Elevated temperatures: Keep the cold aisle at 27–30°C rather than 18–20°C. Higher temperatures make it much easier to cool data centers efficiently. Virtually no server or network equipment actually needs intake temperatures of 20°C, and there is no evidence that raised aisle temperatures in this range cause more component failures [12], [13], [14].
- Free cooling: In most moderate climates, free cooling can eliminate the majority of chiller runtime or eliminate chillers altogether.
- Better power system architecture: Select higher-efficiency gear to greatly reduce UPS and power distribution losses, as discussed in Chapter 5.

In April 2009, Google first published details on its data center architecture, including a video tour of a container-based data center built in 2005 [15]. In 2008, this data center achieved a state-of-the-art annual PUE of 1.24, yet differed from conventional data centers only in the application of the first four principles listed above. Today, large-scale data centers commonly feature PUEs below 1.2, especially those belonging to cloud operators. Even in unfavorable climates, today's PUEs are lower than the state-of-the-art PUEs in 2008. For example, Google's data center in Singapore, where the average monthly temperature rarely falls below 25°C, still achieves an annual PUE of 1.13 [16].

The four steps are easy to implement, but more sophisticated approaches can unlock significant additional efficiency. In particular, despite promising early results, AI-controlled cooling is not yet common. Machine Learning is a natural fit for cooling optimization because there are many ways to control the operation of the data center's cooling infrastructure. The cooling overhead has a non-linear relationship with many system parameters and environmental factors, such as the total system load, the number of chillers operating, and the outside wind speed. For example, if we have 10 cooling towers and are running at 40% load, should we run all of them at 40% speed, run four of them at 100%, or something else? It's difficult to intuit the relationship between these variables and the total cooling overhead. On top of that, WSCs are dynamic: workloads change and equipment is being moved in or out every day. The large amount of data being collected regularly from the network of sensors used to operate the control loop for data center cooling allows machine learning systems to discover building- and workload-specific optimizations to find additional PUE efficiencies. One such system reduced summertime cooling overhead in Google data centers by 40% [17]. As these results suggest, ML represents a promising area for further work.

9.2.5 Beyond PUE to SPUE

Recall the energy efficiency formula from the beginning of this section:

$$\begin{aligned} Efficiency &= Computation/Total\ Energy \\ &= \underbrace{\frac{1}{PUE}}_{(a)} \cdot \underbrace{\frac{1}{SPUE}}_{(b)} \cdot \underbrace{\frac{Computation}{Energy\ To\ Components}}_{(c)} \end{aligned}$$

So far we've discussed the term (a), facility overhead. The second term (b) accounts for overheads inside servers or other IT equipment using a metric analogous to PUE: server PUE (SPUE). SPUE consists of the ratio of total server input power to its useful power, where useful power includes only the power consumed by the electronic components directly involved in computation: CPUs, DRAM, accelerators, NICs, I/O cards, SSDs, and so on. Substantial amounts of power may be lost in the server's power supply, voltage regulator modules (VRMs), power traces on the motherboard, and cooling fans. As discussed in Chapter 5, the losses inside the server can exceed those of the entire upstream data center power train.

SPUE measurements aren't standardized yet like PUE but are fairly straightforward to define. Almost all equipment contains two transformation steps: the first step transforms input voltage (typically 110-220 VAC) to local DC current (typically 12 or 48/54 V), and in the second step VRMs (Voltage Regulator Modules, small DC-DC transformers mounted on motherboards) transform that down to much lower voltages used by a CPU or DRAM. (The first step requires an additional internal conversion within the power supply, typically

to 380 VDC.) SPUE ratios of 1.6–1.8 were common at the turn of the century: many server power supplies were less than 80% efficient, and many motherboards used VRMs that were similarly inefficient, losing more than 25% of input power in electrical conversion losses. In contrast, commercially available AC-input power supplies today achieve 94% efficiency, and VRMs achieve 96% efficiency (see Chapter 5). Thus, a state-of-the-art SPUE is 1.11 or less. For example, instead of the typical 12V DC voltage, Google uses a 48 VDC voltage rack distribution system [18], [19], which further reduces energy losses by over 30%.

Figure 9.5 shows the SPUE overheads of a typical Google server circa 2024. Unlike in the data center, electrical losses dominate cooling here: only 5% goes to server fan power, but 19% is lost during power delivery. These losses are harder to optimize because the electrical currents involved are high. For example, voltage regulators (VRs) placed right next to the CPU transform the board's input voltage (54 V for this server) to the CPU supply voltage (0.9 V), leading to currents exceeding 500 A. Some CPUs have on-chip VRMs to step down an external supply voltage of 1.8 V to the actual internal voltage (possibly variable, depending on CPU frequency), so some of the power transformation losses may happen on-chip and should be included in the SPUE.

Over the past decade, CPU supply voltages have decreased, greatly improving energy efficiency as measured in Joules per operation. Unfortunately, VR losses have increased because lower output voltages make it harder to design highly efficient voltage regulators. Additionally, significant power losses may be incurred on the printed circuit board (PCB) itself as high low-voltage currents flow through narrow power traces.

The product of PUE and SPUE constitutes an accurate assessment of the end-to-end electromechanical efficiency of a WSC, the true (or total) PUE (TPUE). Around 2005, a typical TPUE stood at more than 3.2 for the average data center. That is, for every productive watt, at least another 2.2 W were consumed. By contrast, a modern facility with an average PUE of 1.11 and an average SPUE of 1.23 achieves a TPUE of 1.35. Close attention to cooling and power system design in combination with new technology has provided an order of magnitude reduction in overhead power consumption.

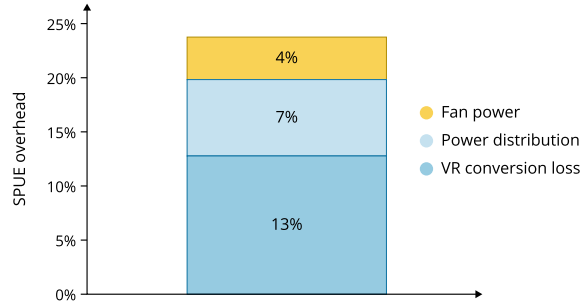


Fig. 9.5 SPUE overhead in a 2024 Google server

9.3 Energy efficiency of computing

So far we have discussed efficiency in electromechanical terms, the (a) and (b) terms of the efficiency equation, and largely ignored term (c), the server's architectural efficiency, which measures how efficiently the electricity delivered to electronic components is actually translated into useful work. In a state-of-the-art facility, the electromechanical components have a limited potential for improvement: Google's TPUE of approximately 1.2-1.3 means that even without any electromechanical overheads, the total energy efficiency would improve by only 20-25%. In contrast, the energy efficiency of computing has doubled approximately every 1.5 years in the last half century [20]. Though such rates have declined significantly during the past decade due to CMOS scaling challenges [21], they are still outpacing any electromechanical efficiency improvements. The rest of the chapter focuses on the energy and power efficiency of computing.

9.3.1 Measuring energy efficiency

Ideally, we want to measure the energy consumed to produce a certain result like a database query or a Google search. Many industry benchmarks try to measure energy efficiency at this level. In high-performance computing (HPC), the Green 500 [22] benchmark ranks the energy efficiency of the world's top supercomputers using LINPACK. Similarly, Joulesort [23] measures the total system energy to perform an out-of-core sort. SPECpower [24] focuses on server-class systems and computes the performance-to-power ratio of a system running a typical business application on an enterprise Java platform.

For machine learning, MLCommons defines a set of ML performance benchmarks for inference and training [25]. These benchmarks provide a consistent methodology with clear metrics for comparison across a wide variety of ML workloads, with broad support from both academia and industry. Building upon this foundation, the MLPerf Power benchmark compares the performance measurement from MLPerf with energy measurements (or estimates thereof) for running the benchmark. For comparability, the power measurements must follow stringent criteria such as measurement duration, sampling frequency, and accuracy. For training workloads, the energy efficiency score is based on the MLPerf Training benchmark and is reported as Samples/J. For inference, score is based on MLPerf Inference with the result reported as 1/J.

In a relatively new field like ML, annual efficiency improvements are orders of magnitude higher than in a mature field like general compute. The MLPerf Power authors observe that benchmarks have improved energy efficiency by over 100x over a three-year period, mostly driven by model improvements [26]. On the hardware side, architectural improvements have delivered significant efficiency benefits with each generation in GPUs and TPUs, suggesting that rapid progress will continue in the near future.

9.3.2 Improving energy efficiency: Energy proportionality

9.3.2.1 Fleet utilization statistics

Figure 9.6 shows the average CPU utilization of two Google clusters in 2013 and 2024, each cluster having over 20,000 servers. The cluster on the left is representative of a typical Google cluster mixing several types of workloads. Such WSCs tend to have lower average utilization because online services experience diurnal variations in load. The cluster on the right represents one of Google's most highly utilized clusters, where large continuous batch workloads run. Workloads of this class can be scheduled efficiently and reach high average utilization.

Over the years, continuous work on improving utilization has yielded results. In 2013 clusters were spending most of their time in the 10–50% CPU utilization range, but by 2024 that had increased to 40–60%.

Across a larger sample of clusters running Google-internal workloads in 2024, we see a similar distribution that now more closely resembles a normal distribution with an average CPU utilization of 50% (Figure 9.7). The clusters represented in the figure comprise 11% of the total fleet, and utilization was measured over a three month period. Thus, while Figure 9.6 showed two individual clusters with fairly different utilizations, Figure 9.7 is a good proxy for the fleet average.

How good is an average utilization of 50%? The maximum achievable utilization is limited by several factors:

- If a load exhibits a diurnal pattern that corresponds to user activity, the daily peak-to-trough ratio limits the achievable utilization. For example, if the daily trough is 60% of

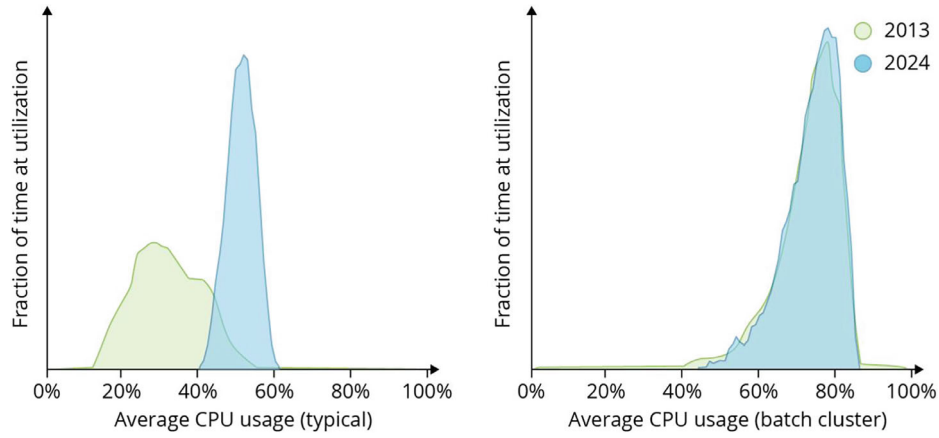


Fig. 9.6 CPU utilization of Google clusters in 2013 and 2024

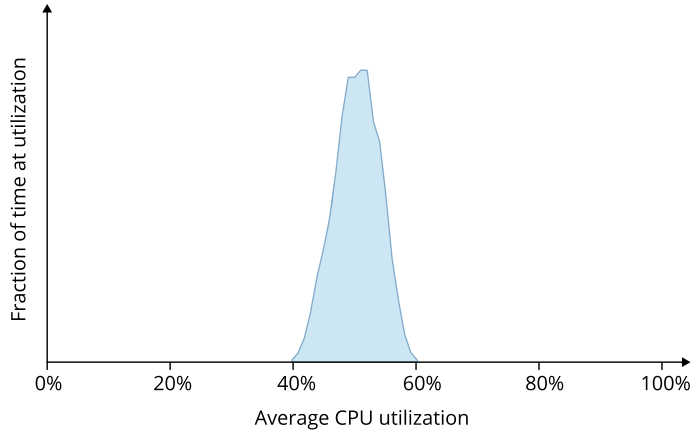


Fig. 9.7 CPU utilization of Google fleet sample

the daily peak, and usage falls (or drops) evenly between peak and trough, the maximum utilization would be 80%, assuming we run at 100% at peak.

- Services often have latency targets, and latency will increase at higher utilization because of request queuing and core or memory saturation. The more stringent the latency requirements, the lower the CPU utilization at which the latency becomes too high. Well-optimized services can run at 70-90% utilization, so a good rule of thumb assumes 80% as a limit.
- To handle localized outages, services run at $N + M$ provisioning; at a minimum, $N + 1$ (i.e., $M = 1$, one service instance can go down without causing a global overload). For a service provisioned at $N + 1$ with 10 service instances, the maximum utilization will be 90% if outages are rare.

Using the assumptions above, the maximum realistically achievable utilization drops to 80% * 80% * 90% = 58%. Batch workloads can further improve utilization. For example, if batch loads represent 20% of capacity and run for 8 hours at night, utilization can improve by an additional 20%/3 = 6.6%. Thus, for large consumer services, an average monthly utilization of 65-70% is close to an upper bound. Therefore, with Google's internal services averaging 50%, utilization can probably be improved by another factor of 1.2x, but not much more.

In contrast, typical IT loads run at substantially lower utilizations. Figure 9.8 shows the utilization of a sample of GCE zones containing customer workloads. These workloads average a significantly lower utilization of 33.5%. While this number looks low, it is not unusual: other studies (enterprise data center utilization, Alibaba, AWS, Azure) report similarly low utilization rates ranging from 6%-25%. [28–30].

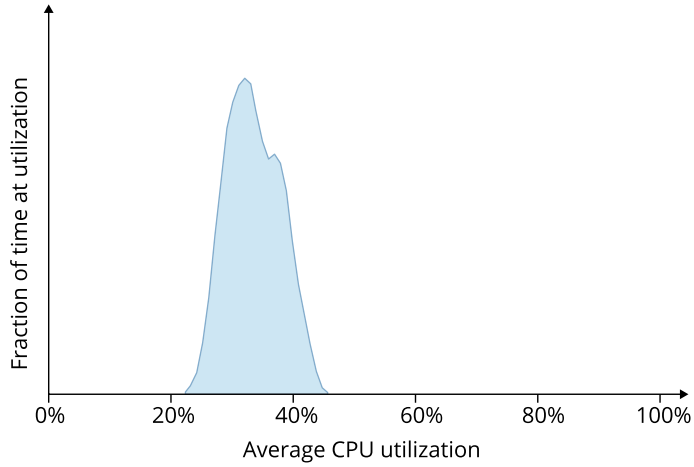


Fig. 9.8 CPU utilization across Google Cloud zones

Nobody likes to waste money, so why are enterprise utilization rates so low? First, most enterprise workloads aren't scaled: they have few users and run on a single VM. For example, a company may have a payroll application that is used only a few times a day. Second, enterprise applications often have higher availability requirements, so the payroll application may have not just a low-utilization primary server but also a completely unutilized backup server for disaster recovery. Finally, driving up utilization is people-intensive, and for small applications the cost of that effort may be higher than the server costs saved.

As a result, the activity profile of enterprise servers turns out to be a mismatch with the energy efficiency profile of modern servers in that the WSC servers spend most of their time in the load region where they are most inefficient.

One tempting idea for reducing energy consumption is to put servers in a sleep state when they're idle. However, servers are rarely completely idle; they just run at low utilization. For hyperscale services like Gmail, this is dictated by diurnal usage patterns: even at the daily low, people still use Gmail. Enterprise workloads are more likely to be completely idle; for example, the application serving the internal phone directory for a small company probably receives no requests outside of business hours. However, long periods of true idleness are hard to manufacture: even at night, a VM contains many system processes that run occasionally, and would require manual intervention (like scheduling regular shutdowns) to idle completely. The Tickless kernel project [31] provides examples of how difficult it is to create and maintain idleness. Public cloud workloads make this even more difficult, since a physical server hosts dozens or hundreds of VMs. Even if they're all relatively low utilization, some computation will happen every millisecond and thus the server cannot just enter into a sleep state.

This situation improves if developers can move away from a VM-based model to a PaaS (Platform as a service) or FaaS (Functions as a service) model; examples are Google Cloud

Run, Cloud Functions, or AWS Lambda. Section 9.3.2.8[1] will discuss software-based efforts to improve utilization.

9.3.2.2 Utilization and energy efficiency

Clearly, the same application binary can consume different amounts of power depending on the server's architecture and, similarly, an application can consume more or less of a server's capacity depending on software performance tuning. Furthermore, systems efficiency can vary with utilization: under low levels of utilization, computing systems tend to be significantly less efficient than when they are exercised at maximum utilization.

Figure 9.9 shows the SPECpower benchmark results for a Dell PowerEdge R7725 server as of November 2024 under varying utilization [27]. Bars indicate energy efficiency in ops/W, and the line indicates power consumption. Both are plotted for a range of utilization levels. The server's efficiency (performance-to-power ratio) drops appreciably as the target load decreases because system power decreases more slowly than does performance. For example, at 30% load, power consumption is above 50%, and thus energy efficiency is more than 1.6x worse than at peak load. Moreover, when the system is idle, it is still consuming 138 W, or 16% of peak power consumption.

These results show best case efficiency since the server hardware and firmware settings have been carefully tuned for the benchmark. For example, systems may disable memory prefetching, turbo modes, and speculative execution, which all increase peak performance but not necessarily in an energy optimal way. Real-world servers may not be able to use such settings because of their performance or latency impact. As a result, their power draw at near-idle load is significantly higher than 16%, often as high as 25–30%.

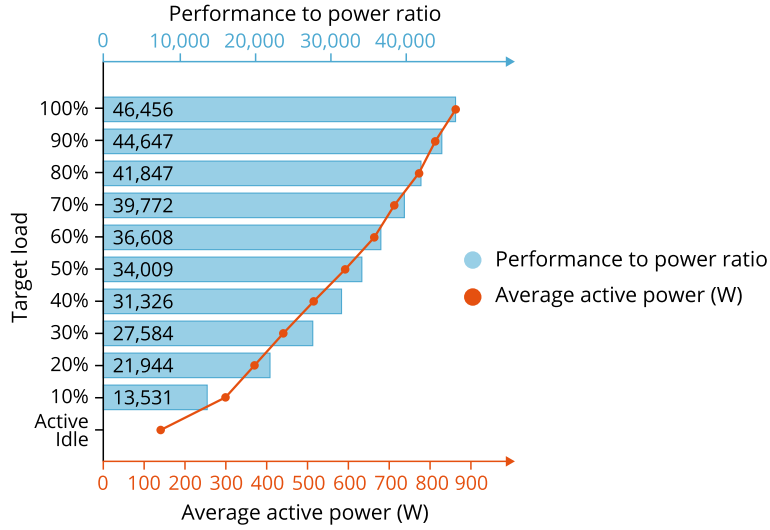


Fig. 9.9 Example benchmark result for SPECpower_ssj2008

9.3.2.3 Energy proportional computing

The mismatch between server workload profiles and server energy efficiency behavior must be addressed at the hardware level. Thus, energy proportionality should be added as a design goal for computing components [32]. Ideal energy-proportional systems will consume almost no power when idle (particularly in active idle states where they are still available to do work) and gradually consume more power as the activity level increases. A simple way to reason about this ideal curve is to assume linearity between activity and power usage, with no constant factors. Such a linear relationship would make energy efficiency uniform across the activity range, instead of decaying with decreasing activity levels.

Note, however, that linearity is not necessarily the optimal relationship for energy savings. As shown in Figure 9.8, since servers spend relatively little time at high activity levels, it might be fine to decrease efficiency at high utilizations, particularly when approaching maximum utilization. However, doing so would increase the maximum power draw of the equipment, thus increasing facility costs.

Figure 9.10 illustrates the possible energy efficiency of three hypothetical systems: a low-proportionality server with 45% idle power (red), a more energy-proportional server with 10% idle power (green), and a hypothetical sublinearly energy-proportional server with 10% idle power (blue). Solid lines represent power normalized to peak power, and dashed lines represent efficiency as a percentage of power efficiency at peak utilization.

The efficiency at peak utilization is the same for all three servers, but real world efficiency of an energy-proportional server is much better if the server spends most of its time at 30-50% utilization.

The potential gains from energy proportionality in WSCs were evaluated by Fan et al. [33] in a power provisioning study that used traces of activity levels of thousands of machines

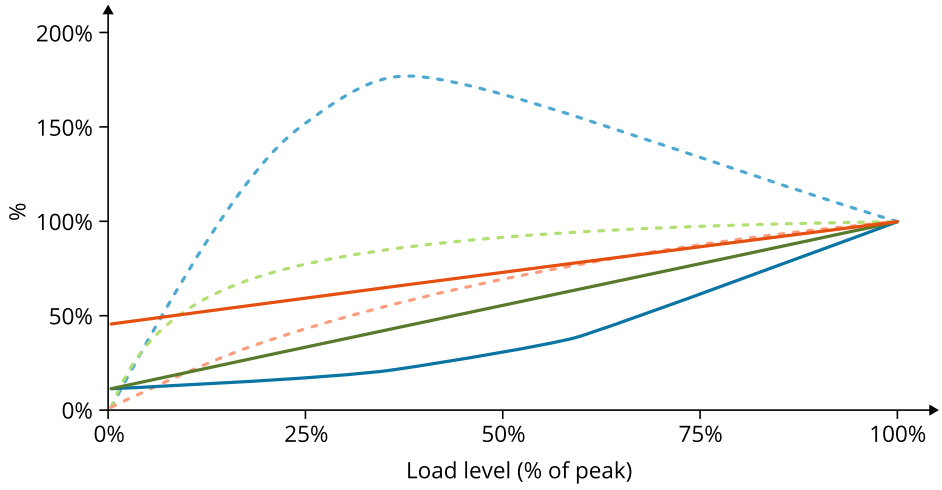


Fig. 9.10 Power and efficiency of three hypothetical systems

over six months to simulate the energy savings gained from using more energy-proportional servers. Assuming two servers, an efficient one with idle consumption at 10% of peak (similar to the green curves in Figure 9.10), and an inefficient one with 50% power at idle (such as the corresponding red curve). Their models suggest that energy usage would be halved through increased energy proportionality alone because the two servers compared had the same peak energy efficiency.

This insight from the original call for energy proportionality [32] spurred the industry to prioritize energy proportionality. Today, it is common for scale-out servers to idle at 20-25% of peak power, and at least partially scale proportionally from there. Energy efficiency at low utilization still isn't perfect (as seen in Figure 9.9), but at 60% utilization it's 75% as efficient as at peak efficiency.

9.3.2.4 Breaking down energy proportionality

Although CPUs historically have a bad reputation regarding energy usage, they are not necessarily the only culprit for poor energy proportionality. Over the last decade, CPU designers have paid more attention to energy efficiency than their counterparts for other subsystems. Despite the end of energy efficiency improvements from feature size scaling (Dennard scaling), the switch to multicore architectures (versus continuing to push single-core higher clock frequencies and larger levels of speculative execution) has helped continue improve CPU efficiencies. Further improvements came from techniques like FinFET, integrated voltage regulators, finer-grained clock gating in sleep states, and adding frequency management through APIs such as RAPL [34].

The relative contribution of the memory system to overall energy use has decreased over the last five years with respect to CPU energy use, reversing a trend of higher DRAM energy profile throughout the previous decade. Newer DDR5 DRAM is more efficient, and DRAM chip voltage levels have dropped from 1.8 V (DDR2 in 2008) to 1.1 V (DDR5 in 2022). Concurrently, new CPUs use more energy as more aggressive binning processes and temperature-controlled "turbo" modes allow CPUs to run closer to their thermal envelope, and today's systems have a higher ratio of CPU performance per GB as a result of DRAM technology scaling falling behind that of CPUs (the "memory wall").

Figure 9.11 shows the power usage of a 2024 two-socket AMD server optimized for low latency and peak performance. Contrary to the server shown in Figure 9.9 which idled at 16% of peak power consumption, this server consumes 30% of peak power at idle (completely empty server) and 36% at very low utilization (almost but not completely idle). What causes this stark difference to SPECpower?

The two servers differ in their low-level settings, not their hardware. The SPECpower benchmark is a throughput benchmark, and thus the system configuration uses the most aggressive settings for sleep states, even when resuming from such a state incurs delays of hundreds of microseconds. Additionally, the SPECpower configuration trades peak per-

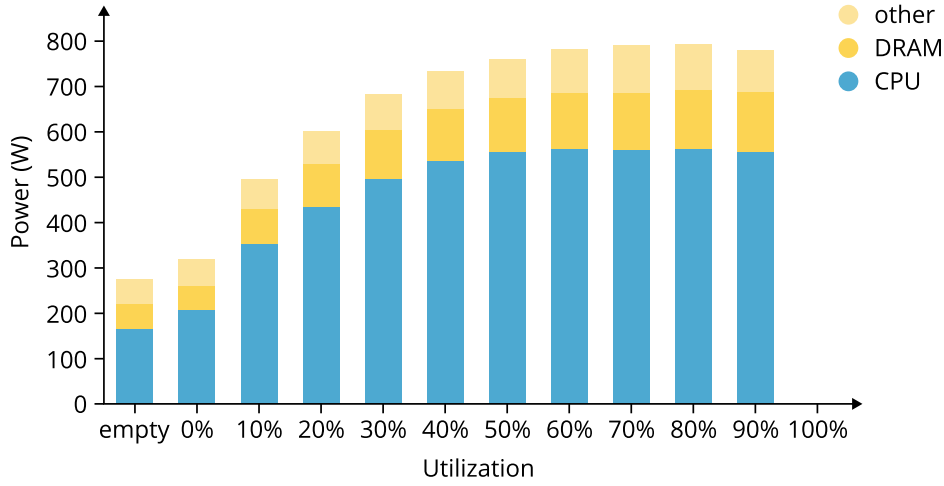


Fig. 9.11 Power usage in an x86 server optimized for performance

formance for throughput, limiting features like cache prefetching and CPU turbo modes for increased energy efficiency. In comparison, the real-world use case can tolerate long wake-up latency when completely empty (saving an extra 5%) but not in normal operation.

More than half of the idle power attributed to the CPU in Figure 9.11 is "uncore" power, i.e., power drawn by elements outside the actual CPU cores, because the cores themselves are idle and thus consuming relatively little power. PCIe and DRAM busses have limited idle modes even today; i.e., they only support "full idle" modes that allow powering down the entire subsystem that require significant latency to power up again. Hence, these are typically disabled for performance sensitive workloads. These systems do not support "active idle" models, like lower clock speeds. (The idle power picture would look even worse if we included off-server networking, i.e., allocated the power consumed by the network port the server is connected to.)

Thus, full energy proportionality at the system level cannot be achieved through CPU optimizations alone, but instead requires further improvements across all components. Unfortunately, scaling voltages or frequencies up or down very rapidly in systems using relatively long wires is challenging, and a better approach is to vary the number of channels that are active. For example, PCIe 6.0 introduces a new L0p power state which allows for dynamic power down on a subset of lanes in a link while keeping the remainder fully active. Both bandwidth and power vary linearly with the number of active lanes, offering a path to energy proportionality.

9.3.2.5 Improving energy proportionality

Added focus on energy proportionality as a figure of merit has resulted in notable improvements for server-class platforms. A meaningful metric of the energy proportionality of a server for a WSC is the ratio between the energy efficiency at 30% and 100% utilizations. A perfectly proportional system will be as efficient at 30% as it is at 100%. In the first edition (early 2009), that ratio for the top 10 SPECpower results was approximately 0.45, meaning that when used in WSCs, those servers exhibited less than half of their peak efficiency. As of June 2018, that figure has improved almost twofold, reaching 0.80. However, since 2018 efficiency improvements have largely plateaued.

Figure 9.12 shows increasing proportionality in Intel reference platforms between 2007 and 2022, based on data from spec.org. While not yet perfectly proportional, newer systems are dramatically more energy proportional than their predecessors.

While processor energy proportionality has improved, it's harder to improve the energy proportionality of bulk storage. Disk drives use most of their power to keep the disk spinning at a constant speed, and the power consumption of a disk drive has hardly changed over the past 20 years. However, disk drives have greatly increased storage efficiency. A drive using about 10 W stored 80 GB in 2005 and 32 TB in 2024, reducing energy per byte by a factor of 400 over that period.

SSDs, having no mechanical parts, are more energy proportional. Typical 4TB SSDs consume 5W when idle and up to 15W at maximum read/write loads. Compared to disks, energy per TB stored is higher, but energy per access is several orders of magnitude lower.

LAN networking is only mildly energy proportional, with both switch chips and optical transceivers idling at a high percentage of full power. A 2024-era switch with 400 Gbps ports consumes about 10W per port for switching, and another 10-15W per port for the

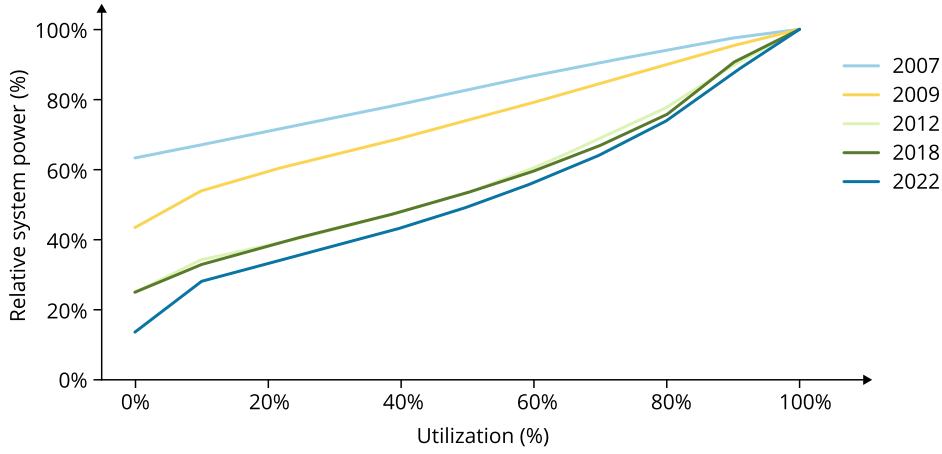


Fig. 9.12 Normalized system power of servers from 2007 to 2022

optical transceiver. However, energy efficiency (in pJ/bit) improves with every generation, since switch (and port) bandwidth increases faster than power consumption.

Finally, energy-proportional behavior is not only a target for electronic components but for the entire WSC system, including power distribution and cooling infrastructure.

9.3.2.6 Low power modes

As discussed earlier, long idleness intervals allow achieving higher energy proportionality by using various kinds of sleep modes. We call these low-power modes inactive because the devices are not usable while in those modes, and typically a sizable latency and energy penalty is incurred when load is reapplied. Inactive low-power modes were originally developed for mobile and embedded devices, and they are very successful in that domain. However, most of those techniques are a poor fit for WSC systems, which would pay the inactive-to-active latency and energy penalties too frequently. The few techniques that can be successful in this domain have very low wake-up latencies, as is beginning to be the case with CPU low-power halt states (such as the ACPI C1E state).

Unfortunately, these tend to be the low-power modes with the smallest degrees of energy savings. Larger energy savings are available from inactive low-power modes such as spun-down disk drives. Most CPUs allow powering down "uncore" functions like the PCIe or DRAM channels, reducing power consumption considerably. However, waking up from such states takes on the order of 0.1 ms, which may introduce too much latency (i.e., performance jitter) for some applications.

Active low-power modes save energy at a performance cost while not requiring inactivity. CPU voltage-frequency scaling is an example of an active low-power mode because it remains able to execute instructions, albeit at a slower rate. Unlike inactive modes, active modes are useful even when the latency and energy penalties to transition to a high-performance mode are significant. Because active modes are operational, systems can remain in low-energy states for as long as they remain below certain load thresholds. Given that periods of low activity are more common and longer than periods of full idleness, the overhead of transitioning between active energy savings modes amortizes more effectively.

Very low power inactive modes with high-frequency transitions have been proposed as a way to achieve energy proportionality [35], [36]. The systems proposed, PowerNap and IdleCap, assume that subcomponents have no useful low power modes other than full idleness and modulate active-to-idle transitions in all subcomponents in order to reduce power at lower utilizations while limiting the impact on performance. The promise of such approaches hinges on system-wide availability of very low power idle modes with very short active-to-idle and idle-to-active transition times, a feature that seems within reach for processors but more difficult to accomplish for other system components. In fact, Meisner et al. [37] analyze the behavior of online data-intensive workloads (such as web search) and conclude that existing low power modes are insufficient to yield both energy proportionality and low latency.

9.3.2.7 Case study: Turbo on Demand

We have argued that hardware components must undergo significant improvements in energy proportionality to enable more energy-efficient WSC systems. However, more intelligent power management and scheduling software infrastructure plays an important role too. For some component types, achieving perfect energy-proportional behavior may not be a realizable goal. Designers will have to implement software strategies for intelligent use of power management features in existing hardware, using low-overhead inactive or active low-power modes, and implementing power-friendly task scheduling to enhance energy proportionality of hardware systems. For example, if the activation penalties in inactive low-power modes can be made small enough, techniques like PowerNap [35] could be used to achieve energy-proportional behavior with components that support only inactive low-power modes.

This software layer must overcome two key challenges: encapsulation and performance robustness. Energy-aware mechanisms must be encapsulated in lower-level modules to minimize exposing additional infrastructure complexity to application developers; after all, WSC application developers already deal with unprecedented scale and platform-level complexity. Additionally, in large-scale systems, completion of an end-user task also tends to depend on large numbers of systems performing at adequate levels. If individual servers begin to exhibit excessive response time variability due to mechanisms for power management, the potential for service-level impact is fairly high and can lead to the service requiring additional machine resources, resulting in minimal net improvements.

Turbo Boost (to dynamically adjust clock speeds) adds new dimensions (CPU activity) for trading off power and performance. The behavior of Turbo Boost is highly dependent on the number of active cores and the compute intensity of the workload. For example, CPU core frequency can vary by as much as 85% on Intel Skylake server CPUs [38]. This phenomenon can also manifest when wider vector instructions, such as AVX-512, cause large drops in CPU core frequency because their high power consumption reduces the thermal budget available to cores. On one hand, these techniques enable higher peak performance for certain workloads, but on the other hand they increase performance variability across the WSC. Dynamic frequency scaling decisions made in hardware present a set of new challenges in achieving performance robustness, and software designers must be cognizant of such effects in hardware and to handle the resulting performance variation.

In fact, Turbo Boost creates a significant mismatch between performance and CPU utilization. At low CPU utilizations, the CPU has more headroom to boost to higher frequencies, which often is unneeded performance. Conversely, at high loads, when throughput requirements are at their peak, the available boost is far less. Such behavior makes sense in end-user multi-core systems like laptops and phones: when a single-core application runs, boosting that core's frequency improves the user experience. But in a WSC this behavior is less useful, since most of the cycles are spent in the largest applications, and these applications all scale horizontally and thus run on all the cores.

Incorporating end-to-end metrics and service level objective (SLO) targets helps overcome this dilemma, reducing performance variability challenges while moving the efficiency needle. The key insight is that during periods of low utilization, latency slack exists between the (higher latency) SLO targets and the currently achieved latency. This slack represents power saving opportunities, as the application is running faster than needed, and thus it could run at a lower clock frequency while still maintaining its SLO.

Lo et al. [39] describe the Pegasus system that combines hardware power actuation mechanisms (Intel RAPL [40]) with software control policies. The system uses end-to-end latency metrics to drive decisions on when to adjust CPU power in response to load shifts. By combining application-level metrics with fine-grained hardware actuation mechanisms, the system makes overall server power more energy proportional while respecting the latency SLOs of the WSC application. Furthermore, load-testing is typically done at high utilizations, where the CPU frequency cap is limited to the all-core-active boost frequency. Thus, capping the frequency to the all-core-active frequency is safe, as performance and latency will be no worse than during the load test.

At Google, Pegasus has evolved into Turbo on Demand (ToD), which combines the insights of Pegasus to encompass all workloads, not just those with a latency SLO. By tuning the turbo frequency caps to match the performance needs with the available performance from the CPU, ToD boosts efficiency by up to 30% without impacting the rated capacity of the platform. Figure 9.13 shows the improvement in energy proportionality of a system before and after the roll-out of Turbo on Demand in 2014.

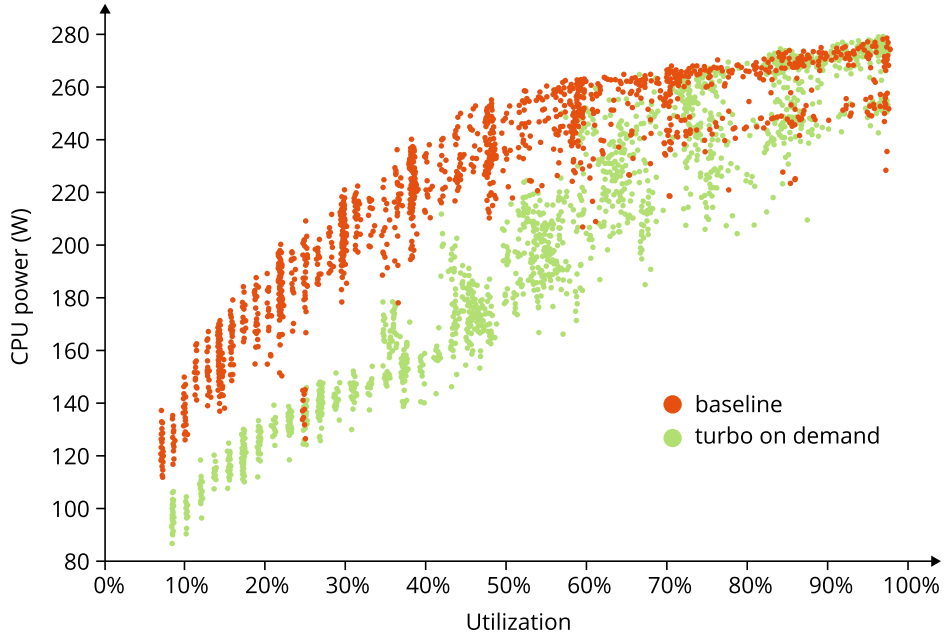


Fig. 9.13 Efficiency impact of Turbo on Demand

9.3.2.8 Improving enterprise workload utilization with software

While energy proportionality and other hardware efforts can help, the biggest driver of underutilization in the enterprise (and thus in public clouds) is the underlying operational model that assumes that each workload runs on its own VM. Since the capacity of even a single-core VM is still fairly high, the resulting CPU utilization is very low for applications with low traffic volumes.

A first level of optimization attempts to improve utilization by oversubscribing resources while still keeping the VM-based model. Specifically, instead of allocating a full hardware core to a low-utilization VM, such approaches allocate a fractional core, thus packing more VMs onto a physical server and improving utilization. For example, AWS T3 instances are burstable general-purpose instances that provide a baseline level of CPU performance with the ability to burst CPU usage [41]. On GCP, E VMs are shared-core [42] that time-share a physical core between vCPUs. GCP also allows customer-defined oversubscription on sole-tenancy servers [43] to let the customer pick the appropriate balance between oversubscription and performance.

The next level of optimization improves utilization by moving to a container-based model (see section 4.3.2) so that the overhead of the VM is shared across multiple applications. Containerization requires an application to be reconfigured and repackaged, but often requires few changes to the application code. Especially for smaller applications, the OS overhead can be a substantial fraction of total resource usage, so the savings can be substantial. In particular, requiring less memory per application allows more applications to be packed onto a server, since memory capacity often limits oversubscription before CPU capacity does.

Containerization also reduces administration costs, since OS management and container infrastructure management are provided as a service (e.g., by Google Kubernetes Engine, GKE). The container abstraction allows even simpler operational models like Cloud Run [44] that eliminate most administrative tasks and just run containers as a service so the user is not exposed to Kubernetes administration (and, in fact, Cloud Run doesn't even run on Kubernetes proper). Additionally, such services provide auto-scaling, including scale-to-zero: after a defined period where no requests have been received, the last container instance is shut down so the application uses zero live resources while idle. Conversely, when the next request comes in, Cloud Run automatically restarts the application container. For enterprise applications that get only a few requests per day (like the example payroll service), scale-to-zero can greatly reduce costs and drops energy consumption to almost zero.

9.3.2.9 Interference and noisy neighbors

Cluster schedulers aim to run the maximum number of tasks on the minimum number of machines by bin-packing tasks with varying shapes (e.g., different-size VMs), possibly oversubscribing physical resources. This is a net win for energy efficiency but also an economic win since fewer servers are needed to handle the given workload.

However, as server utilization is pushed higher, performance degradation from shared resource contention becomes a bigger issue. For example, if two workloads that would each completely saturate DRAM bandwidth are co-located on the same server, then both workloads will suffer significantly degraded performance compared to running each workload in isolation. With workload agnostic scheduling, the likelihood of this scenario occurring increases with larger CPU core counts.

To counter the effects of interference, service owners tend to increase the resource allocations of sensitive workloads to ensure that their jobs will have sufficient compute capacity even in the face of resource contention. This extra padding lowers server utilization, thus also negatively impacting energy efficiency. Contention-aware scheduling avoids this pitfall and further raises utilization while maintaining strict application-level performance requirements [45], [46], [47]. While the specific mechanisms differ for each system, they all use historical performance metrics to make better scheduling and resource allocation decisions. Such performance-aware systems can lead to significantly more resource-sharing opportunities, increased machine utilization, and ultimately energy efficient WSCs that can sidestep poor energy proportionality.

Higher utilization also makes it harder to handle "noisy neighbors," tasks with temporary spikes in their resource utilization. On a lightly loaded system, such spikes are absorbed by idle capacity so that other tasks running on that system may not be affected. A heavily loaded system has no such slack, requiring either strong performance isolation (hard capping the resources available to the noisy task) or very fast task migration to free up resources, or a combination of the two.

9.3.2.10 AI power fluctuations

Today's ML workloads require synchronized computation across tens of thousands of accelerator chips, together with their hosts, storage, and networking systems; these workloads often occupy one entire data-center cluster — or even multiples of them. Power consumption rises and falls between idle and peak utilization levels much more steeply, since the entire cluster's power usage is dominated by a few large ML workloads where all worker nodes behave in highly synchronized ways, stopping or peaking at the same time. Many synchronized events cause large fluctuations: when a workload launches or finishes, or during alternating compute- and networking-intensive phases of the workload within a training step.

Figure 9.14 shows a batch-synchronous ML workload running on dedicated ML clusters, with observed power fluctuations of 15 MW. Compared to a traditional load profile, the ramp speed can be almost instantaneous, repeat as frequently as every few seconds, and last for weeks or even months.

Fluctuations of this kind pose the following risks:

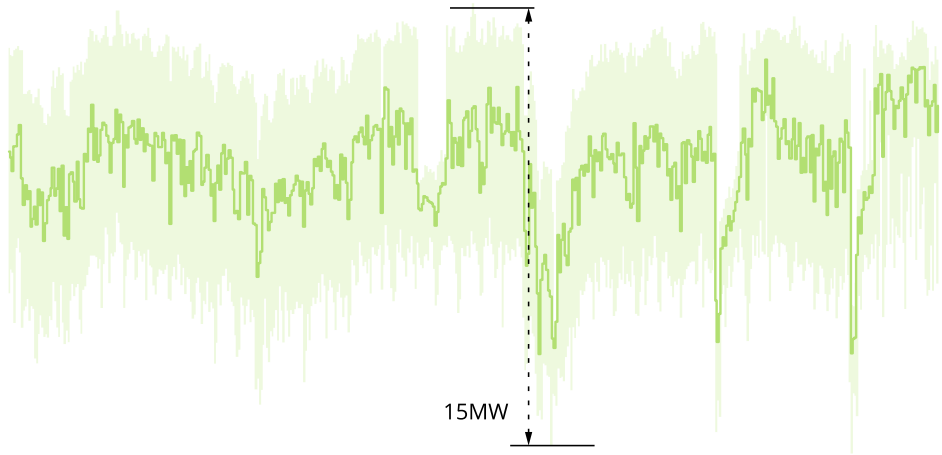


Fig. 9.14 Power fluctuations in a synchronized ML workload

- Reduced reliability of rack and data center equipment, including rectifiers, transformers, generators, cables and busways.
- Reduced reliability of chips, motherboards, and other server components.
- Damage, outage, or throttling at the upstream utility, including, in some cases, mismatches with expectations of the utility about power usage profiles.
- Unintended and frequent triggering of the uninterrupted power supply (UPS) system from large power fluctuations, resulting in shortened lifetime of the UPS system.

At the hardware level, the thermal fluctuations caused by frequent power fluctuations trigger various forms of stress including warpage, degradation of thermal interface material properties, electromigration in semiconductor materials, and more. In fact, rapid thermal cycling is one of the key techniques for Accelerated Life Testing (ALT) which aims to predict component or system lifetimes based on their (shorter) lifetime under stressful conditions.

To reduce this problem, it is necessary to proactively shape workload power profiles to reduce power fluctuations. Profile-guided optimization starts with the TPU compiler instrumenting binaries to monitor workload signatures linked to power fluctuations (e.g., global synchronization points). Then, the compiler aims to dynamically balance the activities of major TPU compute blocks to smooth out their utilization over time. This rebalancing reduces power and thermal fluctuations with little performance overhead. (In the future, we may also apply this approach to workload start and finish phases to enable a gradual, rather than abrupt, change in power levels.)

Figure 9.15 shows the system's total power consumption and a single chip's hotspot temperature with and without optimization [48]. In this real-world workload, power fluctuations dropped by almost 50%. Temperature fluctuations dropped from approximately 20°C to approximately 10°C (note that the temperature graph shows temperature over a

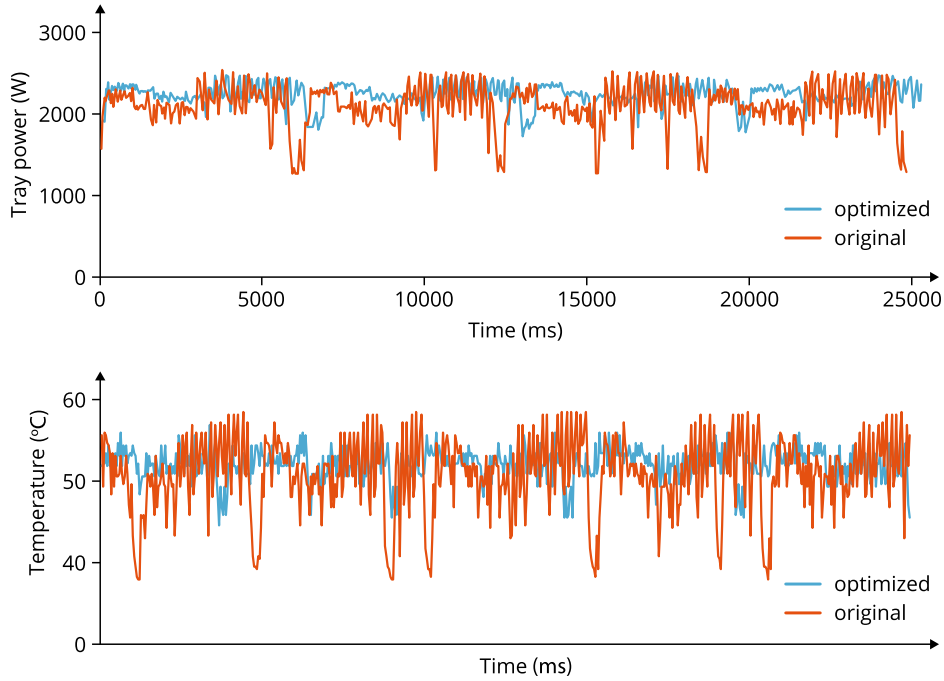


Fig. 9.15 Unoptimized and optimized power and temperature variation

longer time interval). With proper tuning, these benefits impact average power and total performance by less than 1%.

9.4 Improving energy efficiency via specialization

So far we have discussed traditional WSCs: collections of servers, each with CPUs, DRAM, networking, and disks, and with all computation handled by general purpose CPUs. However, recapping the discussion in Chapter 6, Dennard scaling has now ended due to fundamental device limitations, and Moore's Law is well on its way to being sunset. Looking forward, general purpose CPUs face a daunting task when trying to further improve energy efficiency. This issue is orthogonal to energy proportionality, as it is about improving energy efficiency at peak compute load.

While CPU efficiency at peak load still improves (slowly) over time, the demand for compute is growing at a steady rate. Currently, this demand is being driven by artificial intelligence and machine learning, which require large amounts of compute commensurate with large model sizes and consume large amounts of data during training. While general-purpose CPUs are capable of performing the operations necessary for AI, they are not optimized to run these kinds of workloads.

Specialized accelerators are designed for running one particular class of workloads well. Their significant performance advantages over CPUs cause significant efficiency advantages. For example, the Tensor Processing Unit (TPU) greatly improves the energy efficiency of both computations and data accesses. Computations become more efficient since they are lower-precision (e.g., 8-bit floating point vs 32-bit). The TPU's systolic array allows it to fetch a piece of data once and then reuse it multiple times, reducing the access rate to large SRAM arrays that consume significant amounts of power. (Perhaps surprisingly, fetching the inputs of a multiply-add operation consumes an order of magnitude more energy than the actual computation.)

In addition, compared to a modern superscalar out-of-order CPU, the control logic for a TPU is relatively simple and thus much more energy efficient. Since parallelism in machine learning applications is easier to extract, TPUs have no need for the complicated and energy hungry control hardware found in CPUs. These and other design decisions for the TPU have unlocked a vast improvement in energy efficiency.

Figure 9.16 shows how the original TPUs were orders of magnitude more energy efficient for inference tasks compared to a contemporary server CPU of its time (Intel Haswell, 2015). At a system level (accelerator plus supporting CPU), a GPU performed twice as well as a CPU when comparing a weighted mean across benchmarks, and a TPU performed 34 times better. Comparing the performance per watt of just the chips, the GPU outperformed the CPU by 2.9x and the TPU outperformed by 83x. GM and WM are the geometric and weighted means. The figure also shows the performance of TPU' - a hypothetical TPU die scaled to the same process technology and GDDR5 as the comparative GPUs. In this case, the relative total performance/Watt/die of TPU' increases to 31x-86X over CPUs and 25X-41X over the GPUs.

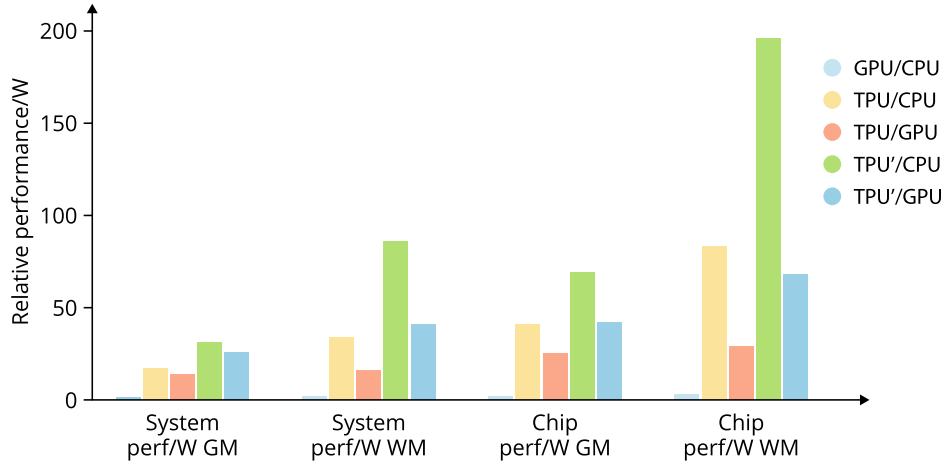


Fig. 9.16 Relative performance of GPUs, TPUs, and CPUs

TPU and GPU performance has improved by 2-4x per generation since then, based on improved architectural features, better compilers, and improved chip manufacturing. As a result, over the past decade, ML performance per chip has improved by well over 100 times!

Nevertheless, not all workloads can run on specialized hardware. The nature of the workload itself may be hard to accelerate; general purpose CPUs can be viewed as accelerators for complex, branchy, and irregular code. Or, more frequently, the use case isn't big enough to justify the fixed upfront investment in specialized hardware. Thus, it is still important to improve the overall energy efficiency of the entire data center, general purpose servers included, along with improving energy efficiency of accelerators.

9.5 Data center power provisioning

Energy efficiency optimizations reduce electricity costs. In addition, they reduce construction costs. For example, if free cooling eliminates the need for chillers, then we don't have to purchase and install those chillers, nor do we have to pay for generators or UPSs to back them up. Such construction cost savings can double the overall savings from efficiency improvements.

Fully using the provisioned power of a facility is equally important. If a facility operates at 50% of its peak power capacity, the effective provisioning cost per watt used is doubled. This incentive to fully use the power budget of a data center is offset by the risk of exceeding its maximum capacity, which could cause outages.

9.5.1 Power stranding

How many servers can we install in a 1 MW facility? This simple question is harder to answer than it seems. First, server specifications usually provide very conservative values for maximum power consumption. Some vendors, such as Dell and HP, offer online power calculators [49], [50] to provide better estimates, but it may be necessary to measure the actual power consumption of the dominant applications manually.

Second, actual power consumption varies significantly with load (thanks to energy proportionality), and it may be hard to predict the peak power consumption of a group of servers. While any particular server might temporarily run at 100% utilization, the maximum utilization of a group of servers probably isn't 100%. In a real deployment, it's easy to end up with an underutilized facility even when paying attention to correct power ratings. For example, a facility typically needs to accommodate future growth, but keeping space open for such growth reduces utilization and thus increases unit costs. Various forms of fragmentation can also prevent full utilization. Perhaps we run out of space in a rack because low-density equipment used it up, or we can't insert another server because we're out of network ports, or we're out of plugs on the power strip, or simply because things don't divide evenly. For example, a 2.5 kW circuit supports only four 520 W servers (stranding 420 W), limiting

utilization to 83% on that circuit. Since equipment flows in and out of the facility over time, it's difficult to plan for 100% utilization, and most organizations don't.

To do better, we'd need to understand the correlation between the simultaneous power usage of large groups of servers. The larger the group of servers and the higher the application diversity, the less likely periods of simultaneous very high activity will become. In typical enterprise data centers, this data either isn't available or the relatively small number of servers doesn't provide robust statistical guarantees, and thus such data centers rarely use oversubscription. WSCs can remove both limitations.

9.5.2 Power oversubscription: The concept

Thus, getting high facility utilization is harder than it looks. We need to estimate the power consumption of the equipment in the data centers to decide how much power to provision, but often these estimates are too conservative, leading to underutilized capacity. Data centers routinely run at low utilization, so this is a real problem in practice [51].

In response, we could either provision less power or build out more servers than the conservative calculations would support. In both these cases, we incur a risk that we'll occasionally exceed the available amount of power; that is, we'll oversubscribe facility power [52]. A successful implementation of power oversubscription increases the overall utilization of the data center's power budget while minimizing the risk of overload situations.

Fan et al. [33] studied the potential opportunity of oversubscribing facility power by analyzing power usage behavior of clusters with up to 5,000 servers running various workloads at Google during a period of six months. One of their key results is summarized in Figure 9.17, which shows the cumulative distribution of power usage over time for groups of 80 servers (Rack), 800 servers (PDU), and 5,000 servers (Cluster).

Power is normalized to the peak aggregate power of the corresponding group. Oversubscription benefits increase with the size of power domains: a pool of 100,000 servers sees lower spikes (relative to the average) than a pool of 100 servers. For example, the figure shows that although rack units spent about 80% of their time using less than 65% of their peak power, they did reach 93% of their peak power at some point during the six month observation window. For power provisioning, this indicates a low oversubscription opportunity at the rack level because only 7% of the power available to the rack was stranded. However, with larger machine groups, the situation changes. In particular, the whole cluster never ran above 72% of its aggregate peak power. Thus, had we allocated a power capacity to the cluster that corresponded to the sum of the peak power consumption of all machines, 28% of that power would have been stranded. This means that within that power capacity, we could have hosted $1/(1 - 0.28) = 1.39x$ more machines.

This study also evaluated the potential of more energy-proportional machines to reduce peak power consumption at the facility level. It suggests that lowering idle power from 50% to 10% of peak can further reduce cluster peak power usage by more than 30% and thus increase hosting capacity by 40%. Mixing different workloads within a cluster increases

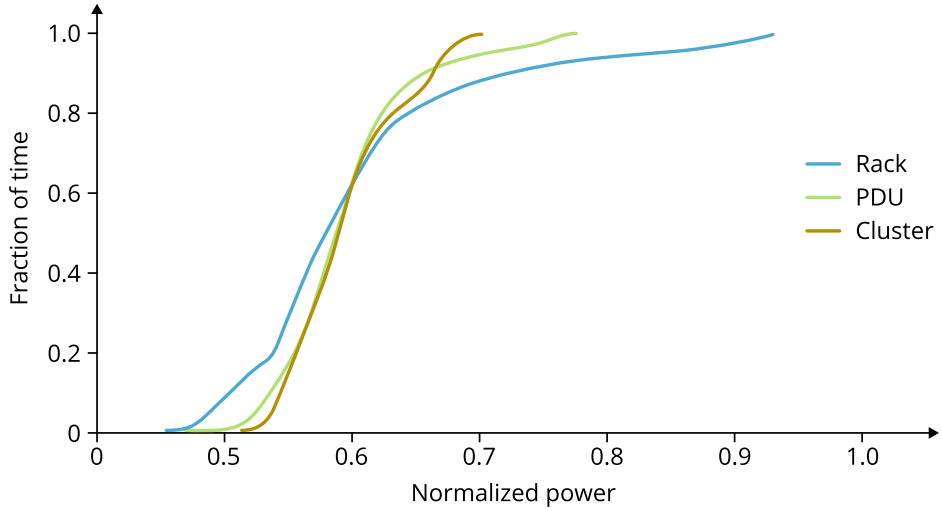


Fig. 9.17 Cumulative distribution of time spent below a given power level

opportunities for power oversubscription by reducing the likelihood of synchronized power peaks across machines.

Once oversubscription is applied, we need a safety mechanism for workload spikes exceeding the data center capacity. Stopping low-priority jobs is an attractive way to instantaneously lower data center load. A typical cluster scheduling system, such as Borg, manages and runs hundreds of thousands of jobs, from thousands of different applications, across a number of clusters, each with thousands of machines. The cluster runs a heterogeneous workload with two basic application tiers (recall the discussion of workload classes in Chapter 3):

- Production tier: production services that should rarely go down (because it would cause user visible outages) and are assumed to have a high availability target. Examples include customer VMs, databases, Gmail, etc.
- Non-production tier: jobs that are much less sensitive to short-term interruptions and typically have looser availability requirements. Examples include test runs, batch jobs, ML training jobs, and chip simulations.

The non-production tier allows increasing the oversubscription rate (OSR) aggressively by taking advantage of the weak availability requirements for these workloads. High oversubscription is tolerable as long as the power consumed by production workloads never exceeds electrical limits. Once total load approaches the electrical limit of a choke point, power capping software pauses or aborts non-production workloads to reduce facility load.

Provisioning should not be so aggressive as to require this mechanism to be triggered often, which might be the case if oversubscription is applied at the rack level, for example.

To help researchers interested in this area, Google released power traces [53]. The traces cover one month of total and production workload power data across 10 clusters and 57 power domains (PDUs).

9.5.3 Oversubscription variants

Oversubscription techniques differ along two dimensions: the resource being oversubscribed, and the granularity at which the oversubscription occurs. So far, we primarily focused on server power (the resource) oversubscription at a cluster (spatial granularity) over a long-term period (temporal granularity). Naturally, oversubscription can also be applied to other resources and other granularities as well to unlock additional saving opportunities. Each new combination requires mitigation strategies to handle the rare scenarios where actual usage exceeds the provisioned capacity.

In the resource dimension, the most prominent application is power oversubscription, but other resources like cooling capacity and backup power generation can also be oversubscribed. Infrastructure oversubscription works best if it is aware of the different availability requirements of workloads, to right-size infrastructure pieces to maximize individual utilization and overall TCO. For example, if 20% of total power is consumed by batch jobs that could be deferred by up to 8 hours, then we have multiple options to further boost safe oversubscription. Such workloads can temporarily be suspended for a few seconds if the actual power draw on a bus bar or PDU exceeds safe limits, or for a few minutes during a utility outage when the backup generators are needed to power uninterruptible production jobs. If utilization is really high on the hottest day of the year, then deferring some of those batch loads to later in the day will allow the cooling system to maintain temperatures safely.

Oversubscription isn't necessarily static, even though all the approaches discussed so far determine a statically safe oversubscription value. For example, we could oversubscribe cooling more aggressively during winter to accelerate maintenance schedules. Similarly, oversubscription is workload-dependent: not every cluster runs the same workloads, and different power domains within a cluster can have different peaking behaviors. Additionally, these values can change over time as workloads evolve. Temporal and spatial awareness can be combined into temporary oversubscription, where the extra headroom created by workload relocation can be used to perform maintenance that would normally violate the cluster's oversubscription limits.

In a redundant design, every power domain has two feeds: a transformer to step down a higher-voltage utility feed into a lower-voltage busbar feed, and a generator that can provide power if the utility feed fails. One form of variable oversubscription boosts the oversubscription factor when at least one utility feed is available (which is the case almost all

the time) because power domains usually are limited by generator capacity, not transformer capacity.

At first blush this situation sounds unlikely: in a well-designed facility, shouldn't paired transformers and generators have matching capacities? Usually, their rated design capacity is not matched exactly, since transformers are cheaper to oversize. More importantly, they don't have an immediate, hard upper capacity limit: a 2 MW transformer running at 2.2 MW of load gets warmer (and has a shorter lifetime) but it won't fail. In contrast, a 2 MW generator will trip offline within seconds when faced with a 2.2 MW load. Thus, it's possible to allow short excursions beyond the rated power limit when running on utility power, but not when running on backup generators.

Oversubscription can also be applied at even larger domains; for example, at the campus level, where the aggregate behavior of different clusters could provide even more statistically stable behavior. For example, a campus with 120 MW of provisioned data center capacity across four buildings may not need a utility feed capable of handling the full load, because it's very unlikely that any building is running at 100% of its capacity, let alone all four running at 100% simultaneously.

9.5.4 Case study: Power oversubscription at Google

This case study describes power oversubscription at Google [54], where many data centers achieve a 25% oversubscription ratio. Two key techniques enable high oversubscription. First, a larger power domain size (tens of MW per domain) allows more statistical multiplexing and sharing for increased oversubscription. Second, workload availability awareness, in combination with fine-grained yet scalable power capping actuators operating on each server, enables responsive power control in the event of an excursion.

Our case study discusses a medium voltage power plane (MVPP) that operates at around 15 kV and creates power sharing (and oversubscription potential) across tens of MWs of equipment. (Figure 5.7 in Chapter 5 describes it in more detail.)

In an MVPP architecture, the generator farm limits overall capacity. Thus, the power capping infrastructure is generator-aware, reducing oversubscription when utility power fails. Additionally, it can handle generator failures. When operating on generator power and a backup generator fails, the maximum allowable power is reduced by a commensurate amount. The power capping controller notifies the node controller on every server in the domain. The node controllers react by suspending or killing jobs to reduce power, starting with the non-production tier of jobs. Power capping needs to react quickly: via the engine's momentum, a generator can "ride through" a small amount of excess load for just a few seconds before it stalls.

Power capping requires accurate measurements of the current power draw. Figure 9.18 shows the high-level software architecture of the power capping service employed in Google's data centers [54]. Every second, a meter watcher module continuously polls several fast-response power meters located at the utility feeds and generator farm. These measure-

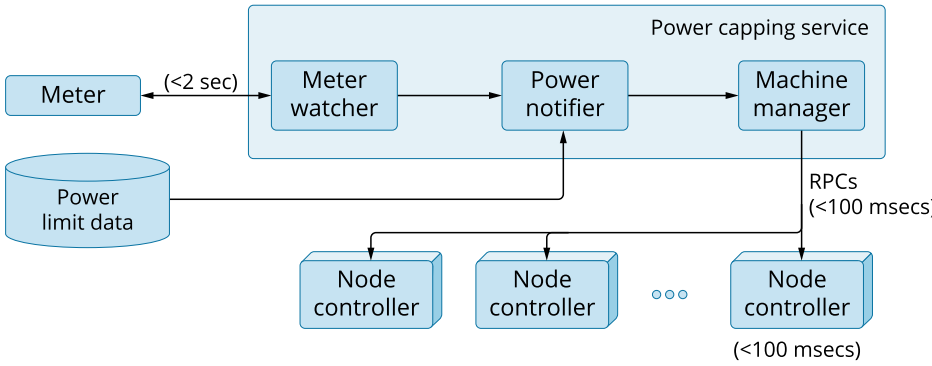


Fig. 9.18 Block diagram for the power capping service

ments are fed to the power notifier module that aggregates readings. Whenever the power capping threshold is exceeded, the notifier contacts the machine manager module to send RPCs to individual node-level schedulers on the power plane. These in turn suspend low-priority tasks using a SIGSTOP signal. Each node scheduler will prevent low priority tasks from running as long as power remains above threshold. Once power drops sufficiently, the node scheduler will issue a SIGCONT signal, allowing tasks to resume.

To characterize the impact on non-production tier unavailability as a function of OSR even in hypothetical situations, we use a time-domain Monte Carlo simulation to simulate single-cluster behavior at different OSR values using the historical power utilization time series of actual clusters. The simulation employs conservative assumptions about maintenance and failure rates, and utility power outage frequency and duration. Over centuries of simulated time, the simulations generate a probabilistic series of outcomes incorporating three types of events: single utility outage/feed maintenance, dual utility outage, and dual utility outage with generator failures.

Figure 9.19 shows the worst case scenario, the fraction of time when non-production workloads cannot run due to power capping in a given cluster, assuming that utility power is unavailable and thus power capping must enforce the lower generator limits. The plot shows the range of non-production workload unavailability across clusters as a function of OSR value while running on generators (which is at most a few hours per year). Marks correspond to individual clusters and bars indicate the first and third quartiles of cluster distribution.

Most clusters will see the impact to non-production workload for only a few percent of the generator run time at most OSR rates. However, some outlier clusters may see significant curtailing of loads: at an OSR of 25%, outlier clusters may cap non-production workloads for more than 50% of generator time. That may still be acceptable, especially for batch jobs, since most high-voltage utility failures last for just a few minutes.

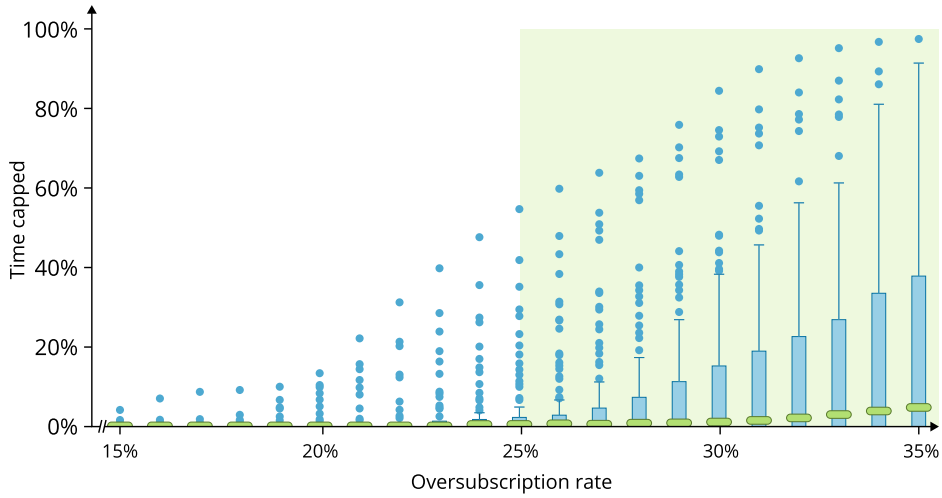


Fig. 9.19 Nonproduction power capping while on generator

9.5.5 Intermittent energy supply

Large workloads that are interruptible create additional opportunities for optimization. In particular, they may be able to run in regions where the electrical grid doesn't have additional peak supply but has available supply most of the time.

This isn't a rare case because grids are built to handle peak usage, not to have high utilization. In other words, electrical grids prioritize availability over cost, because they are expected to handle the load of the highest-demand day of the year (e.g., the hottest summer day when air conditioning demand is highest). Such peaks may occur less than once per year, on average, due to extreme weather events. As a result, the bulk of the year sees demand levels well below that peak, leaving substantial headroom in installed capacity. As a result, the average annual utilization lies in between 30% and 60% for most grids.

Thus, a large load that is interruptible during peak demand periods can help stabilize the grid because they can be curtailed at peak periods. They also reduce energy costs because the grid's fixed costs are prorated over a larger consumption volume.

ML training runs are ideal workloads for intermittent energy supply: they are large (tens or hundreds of MW in a single location), they can be stopped quickly (within less than a minute), and they can equally quickly restart from a saved snapshot. While interruptions are not desirable because they idle expensive ML accelerators, they are tolerable if the total duration of such interruptions is small (e.g., 0.5% or 1% of the year).

A 2025 study of the US grid [55] suggested that moderate amounts of intermittency would enable load growths that greatly exceed the forecasted demand from AI workloads. Specifically, 76 GW of new load—equivalent to 10% of the US aggregate peak demand—could be integrated with an average annual load curtailment rate of 0.25%.

9.5.6 Other energy optimizations

Several studies [56], [57], [58] propose using energy stored in the facility's backup systems (such as UPS batteries) to optimize facility performance or reduce energy costs. Stored energy could be used to flatten the facility's load profile (using less utility power when it's most expensive), mitigate supply variability in a wind-powered facility, or manage short demand peaks in oversubscribed facilities (using stored energy instead of capping the load, or to extend the time available to enact capping).

To our knowledge, no such power management systems have yet been used in production systems. Besides the control complexity, the additional cost of batteries would be significant. We couldn't just reuse the existing UPS capacity for power management, as doing so would make the facility more vulnerable in a real outage: if the UPS SLO requires that a facility can run on batteries for N minutes, then that capacity must be available at all times, not just some of the time. While some have argued that expanded UPSs would be cost-effective [58], the economic case in real-world deployments has been more complex. Power microgrids (discussed in Chapter 5) could make these optimizations more relevant.

Managing energy, peak power, and temperature of WSCs have also been the target of many research studies. Chase et al. [59], G. Chen et al. [60], and Y. Chen et al. [61] consider schemes for automatically provisioning resources in data centers, taking energy savings and application performance into account. Femal and Freeh [62], [63] focus specifically on the issue of data center power oversubscription and describe dynamic voltage-frequency scaling as the mechanism to reduce peak power consumption. Managing temperature is the subject of the systems proposed by Heath et al. [64] and Moore et al. [65]. Pedram [66] provides an introduction to resource provisioning and summarizes key techniques for dealing with management problems in the data center. Incorporating application-level knowledge to safely save power by re-shaping its latency distribution through DVFS is studied by Lo et al. [39], Kasture et al. [67], and Hsu et al. [68].

Raghavendra et al. [69] studied a five-level coordinated power management scheme, considering per-server average power consumption; power capping at the server, enclosure, and group levels; as well as employing a virtual machine controller (VMC) to reduce the average power consumed across a collection of machines by consolidating workloads and turning off unused machines. Such intensive power management poses nontrivial control problems. For one, applications may become unstable if some servers unpredictably slow down due to power capping. On the implementation side, power capping decisions may have to be implemented within milliseconds to avoid tripping a breaker.

Wu et al. [70] studied Dynamo, a dynamic power capping system then in production at Meta. Dynamo makes coordinated power decisions across the entire data center to safely oversubscribe power and improve power utilization. Using Intel RAPL as the node-level enforcement mechanism to cap machine power, the system is workload-aware to ensure that high priority latency-sensitive workloads are throttled only as a measure of last resort. As a result of deploying Dynamo, the authors note a significant boost in power capacity utilization

at their data centers through increased use of dynamic core frequency boosting; namely, Intel Turbo Boost [71], which can run CPU cores at higher frequencies given sufficient electrical and thermal headroom. Much like our case study system described earlier, Dynamo combines application-specific knowledge with fine-grained hardware knobs to improve the realizable compute capability of the WSC while respecting application performance boundaries.

9.6 Environmental sustainability for data centers

Up to now we've mostly examined the electricity consumption of data centers. While energy use and sustainability are related, environmental sustainability is a broader topic, including resources beyond energy (such as fresh water) and greenhouse gas emissions, both direct and indirect, of the data center. Addressing these sustainability concerns has an increasingly significant impact towards reducing humanity's environmental footprint in a world that is increasingly stressed for resources.

9.6.1 Emissions: Scope 1, 2, 3

Greenhouse gas concentrations in the atmosphere have increased substantially since the start of the industrial revolution, with CO2 concentrations increasing 51% from an average of 280 ppm in the 1700s to 422ppm in 2024. Human activities have contributed significantly to this increase, through activities such as agriculture, deforestation, and fossil fuel energy generation [72].

Environmental sustainability has been a priority for many hyperscalers and covers areas as broad as greenhouse gas (GHG) emissions, water use, waste disposal, and hazardous materials.

GHG emissions fall into three categories or scopes:

- Scope 1 emissions are direct emissions that a company controls directly as part of their operation, such as fossil fuel consumption by a company fleet of vehicles, employee travel, office heating, or backup generators.
- Scope 2 emissions are indirect emissions that result from the energy a company buys from utilities. For data centers, scope 2 often is the primary category of GHG emissions.
- Scope 3 emissions are indirect emissions that occur in a company's value chain, i.e., are embedded in the products and services the company buys. For WSCs, examples are the GHGs emitted to produce concrete and steel for data centers, or chips and other electronic components.

For WSCs, scope 1 emissions are minimal: in 2023, Google's Scope 1 emissions accounted for 1% of its total emissions [73].

The vast majority of scope 2 emissions come from fossil fuel-based electrical generation by utilities. Thus, reducing scope 2 emissions comes primarily from reducing the electricity generated by fossil fuel sources by increasing the percentage of carbon-free energy (CFE). One method to achieve this is via market actions, such as buying carbon-free electricity from utilities or independent power producers. Starting in 2017, Google has matched its total annual energy consumption with 100% renewable energy. (The official scope 2 footprint still isn't zero because the GHG Protocol currently does not count out-of-region renewable energy purchases.) More recently, that goal has expanded to achieving 100% carbon-free energy 24/7, in each location, at all times. As of 2023, Google has reached 64% hourly CFE, and more than ten data center locations achieve 90% hourly CFE [73].

Given the intermittency of most forms of renewable energy (the wind does not blow continuously and the sun doesn't shine at night), guaranteeing 100% renewable energy generation is a challenge. One way to solve this is to match energy consumption patterns with the availability of renewable energy, such as shifting the execution of time-insensitive workloads to periods of time where renewable energy availability is predicted to be high, as described in the next section.

Scope 3 emissions (sometimes called "embedded or embodied emissions") represent all other indirect emissions caused by the organization; for example, during server manufacturing or data center construction. The IT equipment hosted in WSCs contribute to the majority of scope 3 emissions of WSC operators, followed by the emissions embodied in the buildings themselves. Since these emissions come from suppliers, they are harder to reduce. One action a WSC operator can take to reduce scope 3 emissions is to encourage or require suppliers to decarbonize their operations. For electronic equipment, around half of the embedded emissions stem from energy consumption during the manufacturing process, so that boosting renewable energy in manufacturing countries can significantly reduce such emissions.

Comparing scope 2 and scope 3 emissions is difficult, since GHG accounting rules treat them differently. Scope 2 emissions are incurred over the lifetime of the server, so if a server consumes one sixth of its total energy in year 1, only that energy's carbon is counted against the year's emissions. In contrast, all of the scope 3 emissions embedded in a server is accounted for in year 1. For data centers, with their lifetime of 20-30 years, this distortion is even more pronounced since all embedded carbon is counted as an emission in year 1. While this is technically correct (the building was indeed completed that year and all its embedded emissions have been emitted), it makes it hard to directly compare scope 2 and scope 3 emissions. For example, even though scope 3 outnumbered scope 2 by 3:1 in Google's 2023 emissions, this ratio doesn't imply that a server's total carbon footprint is dominated by embedded emissions. An actual lifecycle footprint analysis of TPUs, shows that scope 2 emissions represent 70% of total emissions when the TPU runs in a grid with average carbon intensity [74].

A direct role that WSC operators can take in reducing scope 3 emissions is to reduce the amount of new IT equipment being deployed, either through increased utilization and

oversubscription, or by extending the operational lifespan of servers. Thus, the efficiency improvements discussed in the first half of this chapter not only reduce costs but also reduce scope 2 and 3 emissions. Similarly, extending the lifetime of components via resale reduces scope 3 emissions. For example, an old DIMM may be too slow for continued server use but still be attractive for upgrading older PCs. Similarly, recycling steel or copper from obsolete racks or network cables reduces emissions.

For building emissions, Google's internal analysis indicates that concrete and steel are the largest carbon contributors to data center GHG emissions. The building superstructure contains most of this embodied carbon. While cement typically forms only 7-15% of a concrete mix, it can contribute up to roughly 80% of the embodied carbon in concrete. Cement production is highly carbon intensive because it requires high temperatures during production, usually achieved with fossil based fuels, and extraction and transportation of raw materials is dependent on fossil fuels. Additionally, the chemical reaction that produces cement releases carbon dioxide to the atmosphere.

However, lower-carbon cement alternatives are starting to emerge. Recently, all major hyperscalers called for more work towards low-carbon concrete, including using fly ash and slag with less impact on carbon emissions, more carbon-friendly materials such as limestone calcined clay cement (LC3), and longer-term approaches like storing carbon in concrete [75].

Similarly, low-carbon steel or steel alternatives like mass timber (also called CLT or GluLam) can reduce carbon emissions in WSC data centers. These engineered wood products provide excellent strength and fire resistance and have started to displace steel and concrete for commercial and residential high-rise buildings [76]. While data centers demand even more strength (both for wind resistance and because of much heavier equipment loads), mass timber is likely to at least partially displace steel and concrete for data centers too [77]. The OpenCompute foundation also has a track on data center sustainability including metrics for carbon accounting and telemetry.

Many large corporations, particularly those that operate WSCs, have made commitments to substantially reduce their greenhouse gas footprint and emissions. Since 2017 Google has matched 100% of its electrical consumption with renewable energy generation [78]. By 2030, Google aims to operate completely on carbon-free energy 24/7 and reduce the combined scope 1, scope 2, and scope 3 emissions by 50% compared to its 2019 baseline [79]. Other large hyperscalers have also made similar commitments and progress: Microsoft is targeting near zero scope 1 and scope 2 emissions by 2025, and a 50% overall reduction in scope 1, 2, and 3 emissions by 2030 [80]. Meta has achieved net zero emissions for Scope 1 and Scope 2 using a market-based methodology that includes purchasing Renewable Energy Certificates (RECs), with a target for net zero emissions across its entire value chain by 2030 [81].

9.6.2 Carbon-aware scheduling

The ambitious carbon targets described above need new approaches by WSC operators. To completely decarbonize electricity consumption for every hour of every day (24/7), we need to accurately match carbon-free energy demand and supply at each location. One such optimization is to shift compute tasks between different data centers, based on regional hourly carbon-free energy availability. Such carbon-intelligent schedulers use day-ahead predictions of energy mixes to shift computing across the globe, favoring regions where there's more carbon-free electricity [82]. In some respects, using the terminology in Chapter 7, this can be viewed as software-defined carbon efficiency. Like other software-defined approaches, appropriate policies ensure that workload SLOs defined in Chapter 3 continue to be met.

Beyond shifting compute tasks across locations to match the availability of clean energy, we can also shift compute tasks across time to use more electricity when carbon-free energy sources like solar and wind are plentiful. Many batch jobs have some flexibility (for example, "must run after midnight and complete by 9am" instead of "run at midnight") and can thus be time-shifted as needed.

9.6.3 Water

Data centers consume large amounts of energy in the form of electricity, with virtually all of it converted into waste thermal energy that then needs to be removed from the facility. Water is an integral part of this heat removal process: from being used as a heat transfer medium in heat exchanger loops, to directly being evaporated in cooling towers (see Chapter 5) as a way of transferring heat to the environment via vaporization. Water cooling involves a trade-off between water usage and energy usage: water cooling (compared to air-cooling) can reduce the energy usage of a data center and its associated carbon emissions by 10% [83].

Typically, fresh water from municipal sources is used in data center operations. Fresh water is a precious resource in drought-stressed areas, where careful water management is a critical consideration. In some cases, sources other than fresh water, such as reclaimed ("grey") water from water treatment plants or even seawater can also be used as well. Twenty-five percent of Google's data center campuses use such water sources [83].

The Green Grid proposed an industry standard metric to measure water efficiency in the form of the Water Usage Effectiveness metric, or WUE. WUE is defined as the annual amount of water consumed by the data center in liters, divided by the energy usage of the data center in kWh [84].

$$WUE = \frac{water\ usage(l)}{energy\ usage(kWh)}$$

WUE is a good start at quantifying and tracking the efficiency of water use for data centers. As a metric, however, there is still room for improvement:

- The current WUE metric weighs reclaimed or non-potable water the same as freshwater use and thus does not accurately reflect the impact of the WSC's water usage on municipal water supply.
- Similarly, it does not consider the re-use of discharged WSC wastewater for other uses (e.g., for irrigation) that would otherwise have used potable water.
- Finally, it does not include the water consumed by the power plants supplying the data center, such as cooling water for thermal power plants. Such "upstream" water consumption can be as high or higher than the on-site data center consumption (but would be almost zero if the data center sourced all its power from solar and wind generation).

For these reasons, the official WUE metrics don't accurately represent the water impact of a data center, though many in the industry report WUE. The industry average WUE is estimated to be 1.8L/kW-hr [5]. Large WSC operators have achieved much lower WUE measures: Meta has a WUE of 0.26 [85], and AWS has a WUE of 0.25 [86]. Google does not report WUE but does report total water use from all sources.

While water-cooled data centers consume large amounts of water (millions of liters per day), water use is small relative to other uses. For example, Google's entire 2023 global water consumption is roughly equivalent to that of 41 golf courses in the dry US Southwest, compared to an estimated 39,000 golf courses globally. Thus, unlike energy consumption, data center water consumption does not play a significant role globally. However, it still needs to be managed carefully locally, where a data center might impact regional water supply.

Many WSC operators have committed to being water positive, i.e., producing more water than they consume. Google has made a commitment to be water positive by 2030, with a target of replenishing local water sources by 120% of water consumption across its data centers and offices [87]. Meta and AWS also have similar targets to be water positive by 2030 [88], [86]. Typical water replenishment projects consist of expanded wastewater treatment plants that treat waste- or stormwater and inject it into the local aquifer.

9.6.4 Other sustainability considerations

Greenhouse gas emissions and water usage are easily quantifiable physical impacts that WSCs have on the environment, but WSC owners and operators also need to consider other factors. Data centers do not exist in a vacuum: they are part of a community, and data centers can impact that community in both positive and negative ways as a neighbor. WSCs can provide economic opportunities by employing data center staff to support operations and

construction workers being built, but it is up to the WSC to ensure that the local workforce can acquire the skills to realize these opportunities.

Data centers also generate waste, much of it e-waste, that needs to be recycled to minimize pollution and recover precious metals and other valuable resources that can be reused. Proper e-waste processing is difficult and must happen with strong environmental controls. Finding secondary (i.e., resale) markets for older equipment can often extend its useful life span.

WSC operators also need to take care to minimize pollution generated on-site: for example, ensuring proper storage and containment procedures for diesel fuel used by backup generators, and treating discharged cooling to remove contaminants. WSCs can also play a role in driving broader change due to their purchasing power. For example, Google requires components that are RoHS compliant (Restriction on Hazardous Substances) even in jurisdictions that don't make that mandatory. RoHS restricts the amount of toxic substances such as lead, mercury, cadmium, PBBs, and other harmful substances that can be used in the manufacture of various types of electronic equipment.

9.7 Energy usage of AI

While we have thus far talked about WSC energy efficiency in general, in this section, we will discuss the energy usage of AI/ML workloads. With the increased prominence of AI workloads, a growing number of academic papers have highlighted the energy intensity of machine learning (ML) workloads, including some sensational claims. Any workload that is growing exponentially at a rate of several hundred percent a year indeed is reason for concern, and there is value in elevating the discussion around carbon emissions in ML, but the real picture is nuanced.

Google's energy use has been growing by 15-20% annually for more than a decade, driven by the success of its search engine and, later, Google Cloud. Within that usage, the percentage for ML held steady for 2019-2021, representing <15% of total energy use [89]. Inference represented about 60% of total ML energy consumption in 2021, owing to the many billion-user services that use ML. As early as 2018, Google reported using more "ML cycles" than conventional CPU cycles. Despite this high number of operations per second, CPU cycles represent the majority of the annual energy consumption because ML operations are dominated by 4-, 8-, and 16-bit arithmetic whereas CPUs usually process 64-bit data.

To estimate overall AI power consumption, we can use publicly available data about GPU shipments. Specifically, NVIDIA was estimated to ship 3.5-5 million data center GPUs in 2024, translating to about 2-2.5 GW of real-world power consumption when assuming an average power consumption of 60-70% of peak. If all other ML accelerators (including all other NVIDIA models, GPUs from competitors like AMD, and TPUs) add another 1-1.5 GW, this would represent 3-4 GW of AI growth in 2024. Compared to the total power consumed by data centers (see Section 9.8.1), this estimate suggests that AI increased overall data center power by 15-20% in 2024.

However, some studies suggest immense future growth that would triple or quadruple total data center energy consumption in just a few years (Figure 9.20 [90]). A primary reason for these very high AI energy estimates stems from the assumption that technology remains constant but problem size increases exponentially, leading to an equivalent exponential increase in energy. In reality, technology has evolved extremely rapidly in this space, both in software and hardware. For example, over a period of just four years, the efficiency of training a transformer model improved by a factor of 83 [89]. Similarly, a new GPU introduced in 2024 improved the energy required to train the same model by almost a factor of four relative to the previous GPU [91]. The inference cost for a system performing at the level of GPT-3.5 dropped over 280-fold between November 2022 and October 2024 [92].

With these rapid advances in software, algorithms, and hardware, the energy efficiency of AI has been improving at an astonishing rate, offsetting much of the demand growth. Additionally, once models become small enough, they can run on end-user devices like phones, dropping their data center energy use to zero. Thus, it is very hard to accurately predict the energy consumption of AI in data centers, even if we were able to predict AI use correctly.

At the same time, Jevons' paradox [93] suggests that great improvements in efficiency can also greatly increase demand, offsetting the efficiency savings. However, economic realities also prevent AI energy use from growing exponentially for too long unless it becomes profitable, and thus Jevon's paradox can't play out fully.

Consider this back-of-the-envelope calculation: building a 1GW data center and filling it with GPUs costs roughly 30-40 billion dollars, the majority of that for GPUs (assuming a discounted price of $15,000 per 500-750W GPU). On top of that, running those GPUs for six years will cost another 10-20 billion dollars for power, staffing, maintenance, and so

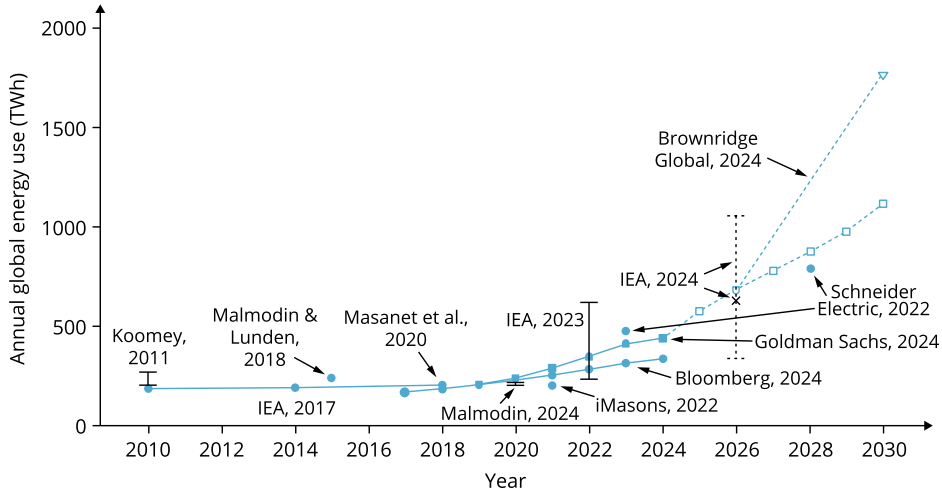


Fig. 9.20 Estimates of global data center energy use

on, yielding a total six-year cost of around 40-50 billion dollars per GW of ML capacity. Assuming a minimum gross margin of 50% to pay for development, sales, and marketing of the products consuming this AI capacity requires six-year revenue of 100 billion dollars per GW of AI capacity installed, or 16 billion dollars in annual revenue.

To put that number into perspective, today (circa 2024), fewer than 100 companies worldwide generate more than 100 billion dollars of revenue per year, so a hypothetical AI application requiring "just" 6 GW of ML capacity would be one of the world's largest companies, yet use <0.1% of the world's electricity (the world's installed electrical generation capacity is about 7500 GW). Thus, while it's conceivable that AI will one day use 1% of the world's electricity, it would require AI-based applications to generate revenues rivaling those of the world's largest companies, suggesting that it will not happen quickly. David Cahn of Sequoia Capital published a similar analysis in 2023 and 2024, using different assumptions but arriving at similar numbers [94].

Additionally, the magnitude of these costs will ensure continued focus by large AI providers on improving the efficiency of AI training and inference. Given the relatively immature state of the field, order-of-magnitude improvements are possible, and thus it is hard to forecast the true extent of AI energy usage over the next few years. The next edition of this book will have a more accurate answer, but for now we predict that the high-end forecasts will not materialize because both efficiency improvements and economic constraints will moderate AI energy growth. Some industry experts come to similar conclusions, based on similar analyses [95].

In some ways, the concerns about AI energy use parallel those expressed about internet energy use during the early internet boom around 2000. For example, several papers estimated the internet would consume 8-10% of total US electricity in 1999 and predicted that this would soon rise to 30%, whereas later studies corrected these numbers and showed much lower baselines and much lower growth [96]. Similarly, video streaming was projected to use enormous amounts of energy, up to 51% of global electricity by 2030 [97], whereas current estimates suggest a figure closer to 0.5% (see next section).

Why did energy consumption turn out to be much smaller than feared? In both cases, rapid efficiency improvements (i.e., greatly reduced energy per byte processed, stored, or transmitted) mostly compensated for exponentially growing internet usage, and that trend remained stable for two decades. While it's too early to be sure that the same pattern will repeat for AI energy usage, the parallels remain true.

AI's footprint includes not just the energy consumed while using AI hardware (discussed above), but also the energy consumed while manufacturing. An in-depth study of five generations of TPUs shows that manufacturing energy is relatively small compared to operational energy [98]. While exact numbers depend on the energy mix in manufacturing and data center locations, a rule-of-thumb from that study suggests that embodied emissions represent about 10% and operational emissions about 90% of an AI system's lifetime emissions.

9.8 Energy usage of the internet

WSC data centers are a critical component of the internet's physical infrastructure, and in the internet's highly decentralized design they are one of the very few visible examples of structures that have scale. As a result, they have risen from obscurity and have become a point of discussion, especially where their power demands become significant vis-a-vis the electricity grids that serve them.

9.8.1 Data center energy usage

Data center energy use has shifted considerably over the past decade. Globally, in 2010, more than 80% of the total energy consumed by data centers was in small data centers, not WSCs [99], [100]. Just eight years later, more than half of that demand had shifted to WSCs, with only about one-third of usage remaining in small data centers. Over the same time period, computation became much more efficient: compute instances increased by 550% but total energy usage increased by only 6%. Readers of previous sections will know why: WSC PUE is much better, Moore's Law reduced energy per computation, and WSC-based cloud services run at much higher utilization. Total global data center energy consumption in 2018 was 205 TWh, or 23.4 GW of average annual power draw. For comparison, Google's 2018 electricity consumption was 10.1 TWh or about 5% of the total [101], and Microsoft's was 8 TWh [102].

A 2024 study about US data center energy consumption estimated them at 176 TWh in 2023, or 4.4% of US electricity consumption [90]. For comparison, Google's 2023 electricity consumption was 25 TWh [73] or about 14% of the total, and Microsoft's was 24 TWh. Driven by the continued migration of on-premise IT to the cloud, the energy share of these two companies increased from 10% to 27% over this period.

For comparison, 560 GW of carbon-free energy (CFE) was installed in 2023. While this number is large relative to data center demand, it's not a given that CFE can power all new data centers. First, 60% of the 560 GW were concentrated in China. Second, the number refers to peak capacity, not average generation. For solar plants, typical capacity factors range from 15% to 25%, so 560 GW of peak capacity translates to about 90-140 GW of average generation capacity. Some of this generation replaces existing coal or gas plants, decarbonizing the grid but not necessarily expanding it. Additionally, electrification of households (heat pumps, electric cars) and industries create load growth well in excess of what data centers need, so the additional generation is needed elsewhere as well. Still, ample new carbon-free energy is being constructed, and hyperscalers have been able to significantly reduce the carbon intensity of their energy supply over the past five years.

Globally, according to the International Energy Agency (IEA), data center energy use has remained at 1-2% of global electricity demand [103]. In 2020, the IEA estimated data

center usage at 223 TWh [104], for a growth of 9% since 2018; in 2024, 2022 consumption was estimated at 240-340 TWh, or around 1-1.3% of global final electricity demand [103].

9.8.2 Cryptocurrency energy usage

A discussion of data center energy use also needs to consider a special class of new data centers for cryptomining. Such data centers may be WSC-like in scale but do not provide any of the services typically provided by WSCs because they merely host specialized hardware, providing power, cooling, and hardware servicing.

Several cryptocurrencies, most notably Bitcoin, use a "proof of work" approach to validate transactions and "mine" new currency. The "proof of work" adds blocks of transactions to a blockchain by solving complex cryptographic puzzles that require significant computational power. In exchange for adding blocks to the blockchain, miners are rewarded with transaction fees in the form of new cryptocurrency coins. About every four years, the reward for validating new Bitcoin blocks is cut in half. This is done to control the supply of Bitcoin; without it, hardware performance improvements would lead to an ever increasing supply. Miners use specialized ASICs or GPUs to perform their proof of work.

By design, Bitcoin mining involves expensive computations, which in turn create a large energy footprint. The U.S. Energy Information Administration (EIA) estimates that, as of 2023, mining for bitcoin and other digital currencies accounts for 0.6-2.3% of electricity use in the US [105]. Actual data is scarce since industry associations have fought efforts to disclose energy consumption in courts.

The Cambridge Bitcoin Electricity Consumption Index (CBECI [106]) estimates the power demand from operations mining for Bitcoin and Ethereum on a daily and annual basis. As of April 2024, CBECI estimates an annualized consumption of 175 TWh, with a lower bound of 80 TWh and an upper bound of 500 TWh. The IEA estimated Bitcoin energy use at 110 TWh in 2022 and 130 TWh in 2023 [107]. Thus, a conservative estimate of Bitcoin energy consumption is comparable to that of all public cloud data centers combined, and a higher estimate would make it bigger than total data center demand.

Some cryptocurrencies, most notably Ethereum, use a process known as "proof of stake" instead of "proof of work." To validate transactions, participants stake a portion of the blockchain's native tokens as collateral. These validators are rewarded if the transaction is completed successfully or penalized if not (which would indicate an attempt to perform illegitimate transactions). "Proof of stake" cryptocurrencies use significantly less energy, and as a result CBECI puts Ethereum's total energy use at 0.005% of Bitcoin's.

9.8.3 End-to-end internet energy usage

An analysis of the 2023 IEA report about energy use in 2020 (Figure 9.21) shows user devices and global WAN networks as the primary consumers of IT-related energy [108]. Data centers overall use about 25%, with 8.5% attributed to WSCs.

At first glance this result is surprising: naturally we expect most of the energy use of cloud computing to be in the data centers that host the cloud. The IEA study shows the detailed calculations behind their estimates, but a brief back-of-the-envelope calculation helps to illustrate the relative ratios:

- In 2023, Google's average global power draw was about 3 GW, including GCP and consumer services like search and YouTube [73]. Assuming for simplicity that this infrastructure serves 3 billion users, we get 1 W of continuous power draw per user.
- A typical cable or fiber modem consumes 10 W or more.
- A Wi-Fi access point uses 5-20 W of power, depending on speed.
- A cell phone uses about 1-3 W of power, including the charging overheads. A typical wireless phone charger consumes 0.5 W of standby power [109].

In other words, two wireless phone chargers that are not in use consume as much electricity as all Google services for that user! These estimates are consistent with other studies; for example, the International Telecommunications Union (ITU) estimates the energy use of smartphones at 23 TWh in 2022, about a quarter of the 85 TWh consumed by all cloud providers [110]. Larger end-user devices require even more energy: PC energy consumption is estimated at 392 TWh, an order of magnitude higher than phones, and almost five times the consumption of the data centers that host the cloud.

A large study by the World Bank and ITU came to similar overall conclusions about IT energy use, stating that "at least 1.7 percent of global emissions stem from the ICT sector" [110]. Since ICT companies represent 60% of global corporate renewable energy purchases,

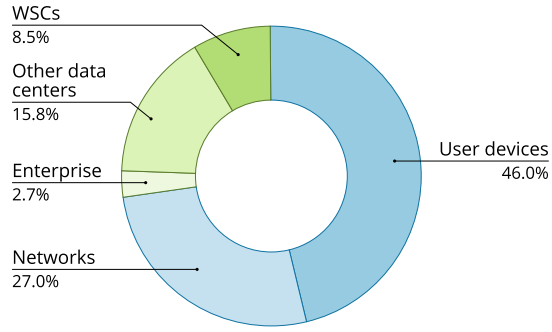


Fig. 9.21 Global IT energy usage across categories

IT's share of emissions is smaller than its share of electricity consumption. The study's split of IT energy across data centers, networks etc closely resembles that of the IEA report above.

Two other studies calculate the energy consumption per hour of video watched, arriving at values consistent with the numbers above. The IEA estimates the end-to-end power consumption of video streaming at 37-77 W in 2019, depending on the device displaying the video [111]. When displayed on a TV or laptop, screen energy consumption dominates consumption, whereas on a phone, network transmission dominates consumption. Wi-Fi and WAN networks have become significantly more energy efficient in the years since 2019 because information coding efficiency has at least quadrupled since then, so network energy estimates likely would be much smaller today.

A 2021 study by Carbon Trust [112] arrives at numbers roughly twice as high, primarily because they include the home Wi-Fi router in the calculation and assume that it uses as much energy as the average screen (71 W). Despite these high estimates, the resulting carbon impact per hour of video is quite low. The study puts the European average footprint at 55g CO_2 per hour of video streaming. For comparison, the emissions from microwaving a bag of popcorn for four minutes is about 16g CO_2 and the average European car emits 220 g CO_2 per km driven.

The discrepancy between these two studies demonstrates why it's easier to estimate energy per sector (for example, energy per year consumed by all networking devices) than to estimate energy per user activity (for example, energy per hour of video). For example, suppose we know with certainty that the power consumption of a Wi-Fi router is 10 W. This router is running all day, consuming 240 Wh per day. If the user watches three hours of video per day, how much does the router contribute to the energy consumption of an hour of video: 0 Wh, 10 Wh, or 80 Wh? All three numbers can be justified: 0 Wh is correct if we define the Wi-Fi router energy as baseload that would be consumed anyway, even if no video is displayed: watching the video does not increase the daily energy consumption by the Wi-Fi router. 10 Wh is correct if we attribute this baseload consumption to an activity while it is happening. 80 Wh is correct if we assume that the Wi-Fi router exists only to display videos, and thus attribute its entire daily energy consumption to the three hours of video watching.

As a further caveat, there are significant methodological challenges to accurately estimating energy consumption in all categories [113], [114], [115]. For example, calculations of overall data center energy consumption are based on the number of servers (data center class CPUs) sold over time. To translate these (fairly accurate) volumes into energy consumption requires accurately estimating their average utilization and their lifetimes. Thus, we estimate that such studies have an accuracy of no better than 25%, that is, the true numbers may differ by a factor of 1.25x or more. Figure 9.22 illustrates how three studies of 2020 consumption differ quite strongly in their estimates of how energy consumption is divided between the three categories [116]. Estimate A by Andrae and Edler (2015) [117] uses their

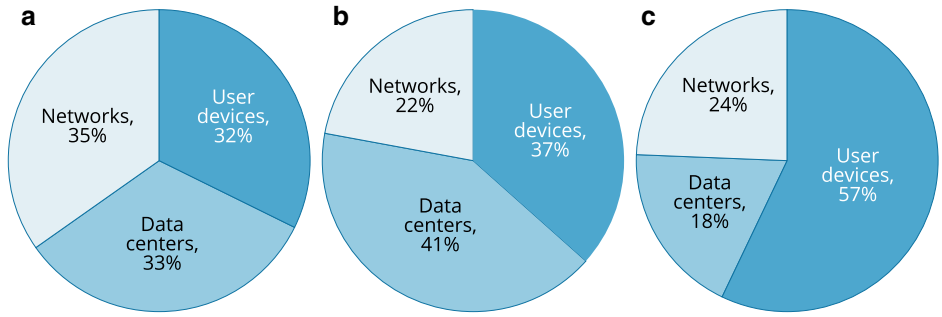


Fig. 9.22 Differing analyses of ICT's carbon footprint, excluding TVs

best case (total of 623 $MtCO_2e$). Estimate B by Belkhir and Elmeligi (2018) [118] is based on their average case of 1,207 $MtCO_2e$ (twice as much as A). Estimate C is by Malmodin and Lunden (2018) [119], based on a total of 690 $MtCO_2e$. Some of the estimates were adjusted by Freitag et al to make them comparable [116].

Another reason for estimates of global IT energy consumption to differ quite widely is the definition of "IT" or "ICT." Most studies include the core definition of IT, that is, corporate data centers and other equipment, including employee devices, networking gear, printers, and so on. When expanding to the consumer side, though, assumptions vary widely. For example, the three studies above exclude TVs, which are treated as entertainment devices, not IT devices. We expect future studies to include all home devices, since TV content may be watched on phones, and YouTube content on TVs, and games are on all devices.

Despite these differences in assumptions, data sets, and methodology, all studies generally agree that the total client-side carbon footprint is at least as large as, and likely larger than, the server-side footprint (Figure 9.23 [116]).

In summary, internet energy consumption is dominated by consumer devices and other electronics in the household, followed by data centers. While Bitcoin mining isn't usually included in these calculations, its energy consumption is in the same ballpark as that of data centers.

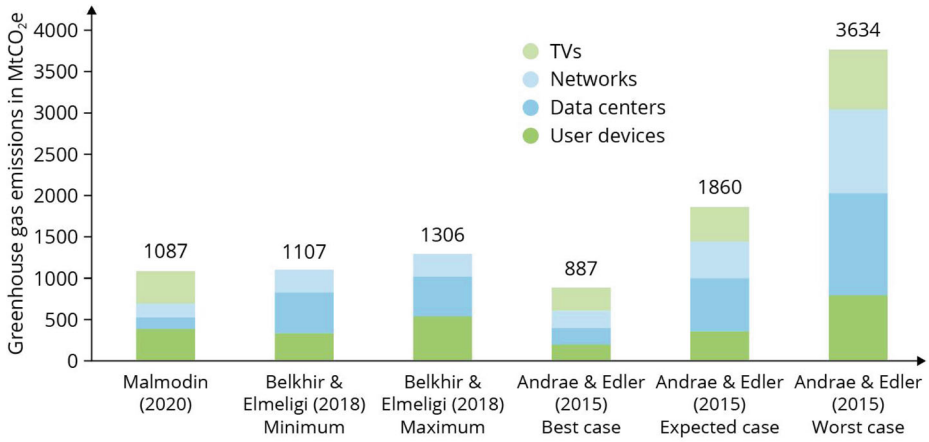


Fig. 9.23 Estimates for global ICT's carbon footprint in 2020

References

1. The Green *Grid, WP#49 - PUE: A Comprehensive Examination of the Metric*, https://www.thegreengrid.org/en/resources/library-and-tools/237-WP#49---PUE:-A-Comprehensive-Examination-of-the-Metric.
2. C. Malone and C. Belady, "Metrics to characterize data center IT equipment energy use," in *Proceedings of the Digital Power Forum, Richardson*, TX, vol. 35, 2006.
3. The Green Grid, *Recommendations for Measuring and Reporting Overall Data Center Efficiency*, https://www.energystar.gov/ia/partners/prod_development/downloads/DataCenters_GreenGrid02042010.pdf, 2010.
4. Uptime Institute, *2014 Data Center Industry Survey*, Uptime Institute Journal: https://journal.uptimeinstitute.com/2014-data-center-industry-survey/, 2014.
5. A. Shehabi, S. Smith, D. Sartor, et al., *United States data center energy usage report*, https://eta.lbl.gov/publications/united-states-data-center-energy, 2016.
6. Uptime Institute, *2022 Data Center Industry Survey*, https://uptimeinstitute.com/uptime_assets/6768eca6a75d792c8eeede827d76de0d0380dee6b5ced20fde45787dd3688bfe-2022-data-center-industry-survey-en.pdf, 2022.
7. Google, *Efficiency: Google Data Centers*, https://www.google.com/about/datacenters/efficiency/internal/index.html#measuring-efficiency.
8. J. Haas, J. Froedge, J. Pflueger, and D. Azevedo, "Usage and public reporting guidelines for the Green Grid's infrastructure metrics (PUE/DCiE)," *White paper*, vol. 22, 2009.
9. C. Belady and C. Malone, "Preliminary assessment from Uptime Institute: IDC data center of the future US server power spend for 2005," *Data Center Power Projection to*, vol. 2006, 2014.
10. D. Nelson, M. Ryan, S. DeVito, et al., "The role of modularity in data center design," 2009.
11. S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt, "Best practices for data centers: Lessons learned from benchmarking 22 data centers," *Proceedings of the ACEEE summer study on energy efficiency in buildings in Asilomar, CA. ACEEE, August*, vol. 3, pp. 76–87, 2006.

12. E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, ser. FAST '07, San Jose, CA: USENIX Association, 2007.
13. B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," 1, New York, NY, USA: Association for Computing Machinery, Jun. 15, 2009, pp. 193–204.
14. N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: Why some (might) like it hot," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12, New York, NY, USA: Association for Computing Machinery, Jun. 11, 2012, pp. 163–174.
15. G. Cloud, *Google container data center tour*, https://www.youtube.com/watch?v=zRwPSFpLX8I, 2009.
16. Google, *Efficiency at Scale - Google Data Centers*, https://www.google.com/about/datacenters/efficiency/.
17. DeepMind, *DeepMind AI reduces Google data centre cooling bill by 40%*, DeepMind Blog: https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/.
18. X. Li and S. Jiang, *Google 48V rack adaptation and onboard power technology update*, https://www.youtube.com/watch?v=aBkz2JR4UVs, 2019.
19. Maxim Integrated (now Analog Devices), *48V Rack Power Architecture for Hyperscale Data Centers*, https://www.maximintegrated.com/content/dam/files/products/power/switching-regulators/48v-rack-power-architecture-for-hyperscale-data-centers.pdf.
20. J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, no. 3, pp. 46–54, 2011.
21. S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, vol. 44, no. 1, pp. 31–38, 2011.
22. TOP500.org, *The Green500 List*, https://top500.org/lists/green500/.
23. S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza, "Models and metrics to enable energyefficiency optimizations," *Computer*, vol. 40, no. 12, pp. 39-48, 2007.
24. Standard Performance Evaluation Corporation, *SPECpower_ssj2008 Benchmark*, https://www.spec.org/power_ssj2008/, 2008.
25. MLCommons, *MLPerf Benchmarks*, https://mlcommons.org/benchmarks/.
26. A. Tschand, A. T. R. Rajan, S. Idgunji, et al., "MLPerf Power: Benchmarking the energy efficiency of machine learning systems from Microwatts to Megawatts for sustainable AI," arXiv:2410.12032 [cs.AR].
27. Standard Performance Evaluation Corporation, *SPECpower_ssj2008 Result ID 01464*, https://www.spec.org/power_ssj2008/results/res2024q4/power_ssj2008-20241007-01464.html, 2024.
28. H. Liu, "A measurement study of server utilization in public clouds," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011, pp. 435–442.
29. B. Everman, N. Rajendran, X. Li, and Z. Zong, "Improving the cost efficiency of large-scale cloud systems running hybrid workloads-a case study of Alibaba cluster traces," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100 528, 2021.
30. E. Cortez, A. Bonde, A. Muzio, et al., "Resource Central: understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2017, pp. 153–167.

31. S. Siddha, V. Pallipadi, and A. Ven, "Getting maximum mileage out of tickless," in *Proceedings of the Linux Symposium*, vol. 2, 2007, pp. 201–207.
32. L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33-37, 2007.
33. X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07, New York, NY, USA: Association for Computing Machinery, Jun. 9, 2007, pp. 13–23.
34. H. Zhang and H. Hoffmann, *A quantitative evaluation of the RAPL power control system*, https://newtraell.cs.uchicago.edu/files/tr_authentic/TR-2014-11.pdf, 2014.
35. D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating server idle power," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV, New York, NY, USA: Association for Computing Machinery, Mar. 7, 2009, pp. 205–216.
36. A. Gandhi, M. Harchol-Balter, R. Das, J. O. Kephart, and C. Lefurgy, "Power capping via forced idleness," in *Proceedings of Workshop on Energy Efficient Design*, Citeseer, 2009, pp. 1–6.
37. D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proceedings of the 38th annual International Symposium on Computer Architecture*, ser. ISCA '11, New York, NY, USA: Association for Computing Machinery, Jun. 4, 2011, pp. 319–330.
38. *Intel, Intel® Xeon® Platinum 8164 Processor*, Intel ARK: https://ark.intel.com/content/www/us/en/ark/products/120503/intel-xeon-platinum-8164-processor-35-75m-cache-2-00-ghz.html.
39. D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st annual International Symposium on Computer architecuture*, ser. ISCA '14, IEEE Press, Jun. 14, 2014, pp. 301–312.
40. *IPU - Intel RAPL interface advisory*, Intel advisory, https://www.intel.com/content/www/us/en/securitycenter/advisory/intel-sa-00389.html, 2020.
41. Amazon Web Services, *T3 Instances - Amazon EC2*, https://aws.amazon.com/ec2/instance-types/t3/.
42. Google Cloud, *Shared-core machine types - General-purpose machine family*, Compute Engine Documentation: https://cloud.google.com/compute/docs/general-purpose-machines#sharedcore.
43. Google Cloud, *CPU overcommit for sole-tenant VMs*, Compute Engine Documentation: https://cloud.google.com/compute/docs/nodes/overcommitting-cpus-sole-tenant-vms.
44. Google Cloud, *Cloud Run - Serverless Container Platform*.
45. J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44, New York, NY, USA: Association for Computing Machinery, Dec. 3, 2011, pp. 248–259.
46. D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 450–462.
47. C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, New York, NY, USA: Association for Computing Machinery, Feb. 24, 2014, pp. 127–144.

48. Google Cloud, *Mitigating power and thermal fluctuations in ML infrastructure*, Google Cloud Blog: https://cloud.google.com/blog/topics/systems/mitigating-power-and-thermal-fluctuations-in-ml-infrastructure.
49. Dell, *Enterprise Infrastructure Planning Tool (EIPT)*, https://www.dell.com/calc.
50. Hewlett Packard Enterprise, *HPE Power Advisor*, https://poweradvisorext.it.hpe.com/?Page=Index.
51. LooseBolts, *Chiller side chats – the capacity problem*, Blog Post: https://loosebolts.wordpress.com/2009/06/02/chiller-side-chats-the-capacity-problem/, Jun. 2009.
52. P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," 2, vol. 34, New York, NY, USA: Association for Computing Machinery, May 1, 2006, pp. 66–77.
53. Google, *Google Cluster Data - Power Data 2019*, GitHub: https://github.com/google/cluster-data/blob/master/PowerData2019.md, 2019.
54. V. Sakalkar, V. Kontorinis, D. Landhuis, et al., "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, New York, NY, USA: Association for Computing Machinery, Mar. 13, 2020, pp. 497–511.
55. T. H. Norris, T. Profeta, D. Patino-Echeverri, and A. Cowie-Haskell, *Rethinking load growth: Assessing the potential for integration of large flexible loads in us power systems*, Nicholas Institute for Energy, Environment & Sustainability, Duke UniversityNicholas Institute for Energy, Environment & Sustainability, Duke University, https://hdl.handle.net/10161/32077.
56. S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned data centers," in *Proceedings of the seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII, New York, NY, USA: Association for Computing Machinery, Mar. 3, 2012, pp. 75–86.
57. D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy, "Energy storage in data centers: What, where, and how much?" In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12, New York, NY, USA: Association for Computing Machinery, Jun. 11, 2012, pp. 187–198.
58. V. Kontorinis, L. E. Zhang, B. Aksanli, et al., "Managing distributed UPS energy for effective power capping in data centers," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12, USA: IEEE Computer Society, Jun. 9, 2012, pp. 488–499.
59. J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the eighteenth ACM Symposium on Operating systems principles*, ser. SOSP '01, New York, NY, USA: Association for Computing Machinery, Oct. 21, 2001, pp. 103–116.
60. G. Chen, W. He, J. Liu, et al., "Energy-aware server provisioning and load dispatching for connectionintensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08, USA: USENIX Association, Apr. 16, 2008, pp. 337–350.
61. Y. Chen, A. Das, W. Qin, et al., "Managing server energy and operational costs in hosting centers," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '05, New York, NY, USA: Association for Computing Machinery, Jun. 6, 2005, pp. 303–314.

62. M. E. Femal and V. W. Freeh, "Safe overprovisioning: Using power limits to increase aggregate throughput," in *Power-Aware Computer Systems: 4th International Workshop, PACS 2004, Portland, OR, USA, December 5, 2004, Revised Selected Papers 4*, Springer, 2005, pp. 150–164.
63. M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *Second International Conference on Autonomic Computing (ICAC'05), IEEE*, 2005, pp. 250–261.
64. T. Heath, A. P. Centeno, P. George, et al., "Mercury and Freon: Temperature emulation and management for server systems," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XII, New York, NY, USA: Association for Computing Machinery, Oct. 20, 2006, pp. 106–116.
65. J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma, "Making scheduling "cool": Temperatureaware workload placement in data centers.," in *USENIX annual technical Conference, General Track*, vol. 4, 2005, pp. 61–75.
66. M. Pedram, "Energy-efficient data centers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1465–1484, 2012.
67. H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48, New York, NY, USA: Association for Computing Machinery, Dec. 5, 2015, pp. 598–610.
68. C.-H. Hsu, Y. Zhang, M. A. Laurenzano, et al., "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 271–282.
69. R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII, New York, NY, USA: Association for Computing Machinery, Mar. 1, 2008, pp. 48–59.
70. Q. Wu, Q. Deng, L. Ganesh, et al., "Dynamo: Facebook's data center-wide power management system," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, IEEE Press, Jun. 18, 2016, pp. 469–480.
71. Intel, *Intel®Core™Processors*, https://www.intel.com/content/www/us/en/products/details/processors/core.html.
72. US Environmental Protection Agency, *Climate Change Indicators: Atmospheric Concentrations of Greenhouse Gases*, https://www.epa.gov/climate-indicators/climate-change-indicators-atmospheric-concentrations-greenhouse-gases.
73. Google, *Google 2024 Environmental Report*, https://sustainability.google/reports/google-2024-environmental-report/, 2024.
74. Google Cloud, *TPUs improved carbon efficiency of AI workloads by 3x*, Google Cloud Blog: https://cloud.google.com/blog/topics/sustainability/tpus-improved-carbon-efficiency-of-ai-workloads-by-3x.
75. Infrastructure Masons (iMasons), *Greener concrete for digital infrastructure: An open letter and call to action*, Press Release: https://imasons.org/press-releases/greener-concrete-for-digital-infrastructure-an-open-letter-and-call-to-action/.
76. Google, *Google's first mass timber building takes root in silicon valley*, Google Blog: https://blog.google/outreach-initiatives/sustainability/mass-timber-building/.
77. Net Zero data centers, *Projects*, https://www.netzerodatacenters.com/projects.
78. Google Sustainability, *Our third decade of climate action: Realizing a carbon-free future*, https://sustainability.google/commitments/carbon/.

79. Google, *Google supports climate action at COP27*, Google Blog: https://blog.google/outreach-initiatives/sustainability/cop27-google-climate-action/, Nov. 2022.
80. B. Smith, *Microsoft will be carbon negative by 2030*, Microsoft Blog: https://blogs.microsoft.com/blog/2020/01/16/microsoft-will-be-carbon-negative-by-2030/, Jan. 2020.
81. Meta, *Climate - Meta Sustainability*, https://sustainability.fb.com/climate/.
82. A. Radovanovic, R. Koningstein, I. Schneider, et al., "Carbon-aware computing for data centers," 2021. arXiv:2106.11750 [cs.DC].
83. U. Hölzle, *Our commitment to climate-conscious data center cooling*, 2022, https://blog.google/outreach-initiatives/sustainability/our-commitment-to-climate-conscious-data-center-cooling/.
84. D. Azevedo, S. C. Belady, and J. Pouchet, "Water usage effectiveness (WUE): A green grid datacentre sustainability metric," *The Green Grid*, vol. 32, 2011.
85. Meta, *Data Centers - Meta Sustainability*, https://sustainability.fb.com/data-centers/.
86. Amazon Press Center, *AWS makes water positive commitment to return more water to communities than it uses by 2030*, Press Release: https://press.aboutamazon.com/2022/11/aws-makes-water-positive-commitment-to-return-more-water-to-communities-than-it-uses-by-2030, Nov. 2022.
87. Google Sustainability, *Replenishing 120% of the water we consume by 2030*, https://sustainability.google/commitments/water/#water-at-google.
88. Meta, *Restoring more water than we consume by 2030*, Meta Newsroom: https://about.fb.com/news/2021/08/restoring-water/, Aug. 2021.
89. D. Patterson, J. Gonzalez, U. Hölzle, et al., "The carbon footprint of machine learning training will plateau, then shrink," *Computer*, vol. 55, no. 7, pp. 18–28, 2022.
90. A. Shehabi, A. Hubbard, A. Newkirk, et al., *2024 United States data center energy usage report*, https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report_1.pdf, 2024.
91. Fortune, *Nvidia's monstrous new Blackwell chip is here to solve AI's carbon footprint problem-not make it worse*, Fortune Magazine: https://fortune.com/2024/03/19/nvidia-new-blackwell-chip-ai-carbon-footprint-problem/, Mar. 2024.
92. Stanford Institute for Human-Centered Artificial Intelligence (HAI), *Artificial Intelligence Index Report 2025*, https://hai.stanford.edu/ai-index/2025-ai-index-report, 2025.
93. Wikipedia contributors, *Jevons paradox*, Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Jevons_paradox.
94. Sequoia Capital, *AI's $600B question*, https://www.sequoiacap.com/article/ais-600b-question/.
95. A. Lovins, *Artificial Intelligence meets natural stupidity: Managing the risks*, https://integrative-design-for-radical-energy-efficiency.stanford.edu/sites/extreme_energy_efficiency/files/media/file/data-centersaiel-dr-16-10-may-2025.pdf, 2025.
96. J. Koomey, "Woldwide electricity use in data centers," *Environ. Res. Lett*, vol. 24983, pp. 220–236, Jul. 2008.
97. A. S. G. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015, issn: 2078-1547.
98. I. Schneider, H. Xu, S. Benecke, et al., *Life-cycle emissions of ai hardware: A cradle-to-grave approach and generational trends*, https://arxiv.org/abs/2502.01671, 2025. arXiv:2502.01671 [cs.AR].
99. E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.
100. N. Lei and E. R. Masanet, "Global data center energy demand and strategies to conserve energy," in *Data Center Handbook: Plan, Design, Build, and Operations of a Smart Data Center*. 2021, pp. 15–26.

101. Google Sustainability, *Google Environmental Report 2019 - Data Centers*, https://sustainability.google/reports/environmental-report-2019/#data-centers, 2019.
102. Microsoft, *Designing and Building Sustainable Data Centers*, https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE3455q.
103. International Energy Agency (IEA), *Data Centres and Data Transmission Networks*, https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks.
104. J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, "ICT sector electricity consumption and greenhouse gas emissions-2020 outcome," *Telecommunications Policy*, vol. 48, no. 3, p. 102 701, 2024.
105. U.S. Energy Information Administration (EIA), *Electricity demand from data centers could double by 2026*, https://www.eia.gov/todayinenergy/detail.php?id=61364, Apr. 2024.
106. Cambridge Centre for Alternative Finance, *Cambridge Bitcoin Electricity Consumption Index (CBECI)*, https://ccaf.io/cbnsi/cbeci.
107. International Energy Agency, *Analysis and forecast to 2026*, IEA Report, https://www.iea.org/reports/electricity-2024, 2024.
108. D. Castro, M. McLaughlin, and E. Chivot, "The carbon footprint of artificial intelligence," Social Science Research Network (SSRN), Tech. Rep. 4424264, 2023.
109. D. Patterson, J. M. Gilbert, M. Gruteser, et al., "Energy and emissions of machine learning on smartphones vs. the cloud," 2, vol. 67, New York, NY, USA: Association for Computing Machinery, Jan. 25, 2024, pp. 86–97.
110. World Bank, *Data Centers and Decarbonization: Unlocking Flexibility in Power Grids*, https://documents1.worldbank.org/curated/en/099121223165540890/pdf/P17859712a98880541a4b71d57876048abb.pdf, 2023.
111. G. Kamiya, *The carbon footprint of streaming video: Fact-checking the headlines*, https://www.iea.org/commentaries/the-carbon-footprint-of-streaming-video-fact-checking-the-headlines, 2020.
112. A. Stephens, C. Tremlett-Williams, L. Fitzpatrick, et al., *Carbon impact of video streaming*, =https://www.carbontrust.com/our-work-and-impact/guides-reports-and-tools/carbon-impact-of-video-streaming, 2021.
113. J. Koomey and E. Masanet, "Does not compute: Avoiding pitfalls assessing the internet's energy and carbon impacts," *Joule*, vol. 5, no. 7, pp. 1625–1628, 2021.
114. C. Bremer, G. Kamiya, P. Bergmark, et al., *Assessing energy and climate effects of digitalization: Methodological challenges and key recommendations*, https://www.networkdee.org/publications/assessing-energy-and-climate-effects-of-digitalization%3A-methodological-challenges-and-key-recommendations, 2023.
115. C. Freitag, M. Berners-Lee, K. Widdicks, et al., "The climate impact of ICT: A review of estimates, trends and regulations," *arXiv preprint* arXiv:2102.02622, 2021.
116. C. Freitag, M. Berners-Lee, K. Widdicks, et al., "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations," *Patterns*, vol. 2, no. 9, 2021.
117. A. S. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.
118. L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018, issn: 0959-6526.
119. J. Malmodin and D. Lundén, "The energy and carbon footprint of the global ict and e&m sectors 2010–2015," *Sustainability*, vol. 10, no. 9, 2018, issn: 2071-1050.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Trusted Computing: Reliability, Availability, Security, Privacy

10

10.1 Fault tolerance at scale

10.1.1 Introduction to fault tolerance at scale

Large-scale enterprise and consumer cloud usage requires trust that services will be always available. This expectation translates into high reliability requirements for building-sized computers. Fundamentally, the appropriate level of reliability is a tradeoff between the cost of failures (including repairs) and the cost of preventing them. For traditional enterprise-class servers, the cost of failures is thought to be high, and thus designers go to great lengths to provide more reliable hardware by adding redundant power supplies, fans, error correction coding (ECC), RAID disks, and so on. Many legacy enterprise applications were not designed to survive frequent hardware faults, and it is hard to make them fault-tolerant after the fact. Under these circumstances, making the hardware very reliable is a rational design decision.

In warehouse-scale computers (WSCs), however, hardware reliability alone cannot deliver sufficient availability primarily due to its scale. Suppose that a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days). Even with these highly reliable servers, a cluster of 10,000 servers will see an average of one server failure per day. Thus, any application that depends on the availability of the entire cluster will see an MTBF no better than one day.

Moreover, large and complex internet services are often composed of several software modules or layers that are not bug-free and can themselves fail at even higher rates than hardware components. Consequently, WSC applications must work around failed services as well, either with code in the application itself or via functionality provided by middleware, such as a provisioning system for virtual machines that restarts a failed VM on a spare node. Some of the implications of writing software for this environment are discussed by

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_10

Hamilton [1], based on his experience designing and operating some of the largest services at MSN and Windows Live. For additional in-depth discussions of reliable systems, we also recommend the SRE book [2], and the software reliability book [3].

Some public cloud workloads add an additional twist to the reliability equation. Enterprise cloud workloads often are moved from on-premise environments to the cloud with little modification ("lift and shift"). As a result, they have little or no fault tolerance built in, and thus depend on reliable VMs. Thus, whereas a modern large service running on load-balanced servers can easily tolerate a single machine failure, a legacy enterprise application running on a single cloud VM cannot.

To provide high reliability despite this sensitivity to server outages, public clouds use two primary defenses. First, they can quickly reschedule a VM onto a new server when a hardware failure occurs. This ensures that mean time to repair (MTTR) is low. Second, most of these applications are stateless and store their data in a database. Usually, that self-managed database is migrated to a managed cloud database like GCP's Cloud SQL, which typically provides better uptime and performance than the on-premise installation.

AI training workloads pose a different challenge: model training is a highly synchronous computation where each node contains a fraction of the model data and exchanges data with other nodes after every training step. (In many ways this is similar to supercomputing applications like weather forecasting.) As a result, if a single node goes down, the entire computation stops. Thus, a training job running on 9,000 TPUs that each have an MTBF of 5 years (1800 days) will see an outage about every 4 hours! For example, during a 54-day Llama3 training run, the 16K-GPU cluster encountered about 320 hardware failures that reduced training run throughput by about 10% [4]. As discussed in Chapter 6, optical circuit switching can help to rapidly swap in replacement TPUs and restart from the latest checkpoint, but managing the reliability of large synchronous computations remains a challenge.

Before we continue, it's important to understand the difference between availability, unavailability, and failure. (We'll discuss the difference between availability and reliability in Section 10.1.3.1.) A system's availability is the fraction of time during which it is available for use; conversely, its unavailability is the fraction of time during which the system isn't available for some reason. Failures are one cause of unavailability, but are often much less common than other causes such as planned maintenance for hardware or software upgrades. Thus, a system with zero failures may still have availability of less than 100%, and a system with a high failure rate may have better availability than one with low failures, if other sources of unavailability dominate.

10.1.2 Fault tolerance in hardware vs software

Fault-tolerant software is more complex than software that assumes fault-free operation. As much as possible, a WSC should try to implement a fault-tolerant software infrastructure layer that can hide much of the complexity of dealing with failures from application-level software.

Once hardware faults can be tolerated without undue disruption to a service, computer architects have some leeway to choose the level of hardware reliability that maximizes overall system cost efficiency. In addition, this model can lead to simplifications in common operational procedures. For example, to upgrade low-level system software in a cluster, a server can be "drained" of the workloads running on it. Many workloads are short-lived, enabling rapid drains; the remaining long-lived workloads can be migrated to another host. Once the server is idle, the invasive upgrade (e.g., one requiring a reboot) can be performed. Hardware upgrades can follow a similar procedure. Basically, the same fault-tolerant software infrastructure mechanisms built to handle server failures could have all the required mechanisms to support a broad class of operational procedures. By choosing opportune time windows and rate-limiting the pace of restart actions, operators can still manage the desired number of planned service-level disruptions.

Unlike traditional server environments, hyperscale applications no longer require keeping individual servers running at all costs. This simple requirement shift affects almost every aspect of the deployment, from machine and data center design to operations, often enabling optimization opportunities that would not be on the table otherwise. For instance, let us examine how this affects the recovery model. A system that needs to be highly reliable in the presence of unavoidable transient hardware faults, such as uncorrectable errors caused by cosmic particle strikes, may require hardware support for checkpoint recovery so that upon detection the execution can be restarted from an earlier correct state. A system that is allowed to go down upon occurrence of such faults may choose not to incur the extra overhead in cost or energy of checkpointing.

This requirement shift works really well for hyperscale applications that have built-in redundancy that's a simple result of their scale. To handle large request volumes, these applications already need to scale horizontally, which results in many replicas of the same server, which in turn provides fault tolerance. For enterprise applications which often are single-server applications, it doesn't work, and thus different solutions are needed. Typical approaches include offering explicitly coordinated maintenance (e.g., performing a kernel upgrade during a maintenance window on weekends), implicitly coordinated maintenance (e.g., waiting for the VM to terminate or restart), and VM migration (transparently moving the customer VM to a new server so the previous one can undergo maintenance).

Another common approach involves "hitless" upgrades that upgrade code while it is running. For example, the Linux kernel offers live patching [5], and many network switches provide hitless upgrades by strictly separating their data plane code and data from their control plane code. During an upgrade, the data plane (usually implemented in hardware)

continues to forward packets while the control plane restarts. Such systems can be complex to maintain and test, but often cannot be avoided.

Cloud services, even for enterprise use cases, can benefit from the shift in responsibility for reliability since they themselves are hyperscale services. Foundational cloud services like storage, networking, or monitoring are provided "as a service" so that hardware failures are handled in these services and are not visible to the customer. Similarly, higher-level services like analytics or database backups also run serverless and thus relieve the customer from having to worry about machine failures. Even virtual machines themselves can be more reliable than their underlying hardware. For example, if a server starts showing uncorrectable DRAM errors but these errors haven't affected a particular VM yet, the cloud provider can transparently migrate the VM to a new server and then repair the defective server.

A system that can tolerate failures at the software level requires that the hardware layer detects and reports faults to software in a timely manner. It is not necessary for that hardware to transparently correct all faults. However, when hardware error correction (or better error detection) is possible with a reasonable cost or complexity, it may well pay to support it. For example, DRAM provides powerful error correction at a reasonable additional cost, and hardware DRAM error detection would be hard to replace with software.

Relaxing the requirement that hardware errors be detected would require every software component to be burdened with the need to check its own correct execution. Usually, the added software complexity greatly exceeds the cost of hardware error detection.

For example, early Google servers lacked DRAM parity checking. Producing a web search index consists essentially of a very large shuffle/merge sort operation, using several machines over a long period. In 2000, one of the then monthly updates to Google's web index failed pre-release checks when a subset of tested queries was found to return seemingly random documents. After some investigation, a pattern was found in the new index files that corresponded to a bit being stuck at zero at a consistent place in the data structures; a bad side effect of streaming a lot of data through a faulty DRAM chip. Consistency checks were added to the index data structures to minimize the likelihood of this problem recurring, and luckily no further problems of this nature were reported. The next machine generation at Google did include memory parity detection. Once the price of memory with ECC dropped to competitive levels, all subsequent generations have used ECC DRAM.

However, many WSC-scale systems still contain software error checks, and they are essential, because hardware error detection may not be strong enough. For example, Ethernet frames have a 32-bit CRC field, and TCP adds a 16-bit checksum on top of that, but with networks running at petabits per second, some corrupted packets will not be detected by these hardware checks. An undetected bit error caused a global outage of the AWS S3 storage system in 2010 [6]. Silent data corruption in CPUs (see section 10.1.10 below) presents another source of headaches to WSC designers, lending a new meaning to Intel founder Andy Grove's motto "only the paranoid survive".

10.1.3 Categorizing faults

An efficient fault-tolerant software layer must be based on a set of expectations regarding fault sources, their statistical characteristics, and the corresponding recovery behavior. Software developed in the absence of such expectations can suffer from two risks: being prone to outages if the underlying faults are underestimated, or requiring excessive overprovisioning if faults are assumed to be much more frequent than they actually are.

Providing an accurate quantitative assessment of faults in WSC systems is challenging given the diversity of equipment and software infrastructure across different deployments. Instead, we will attempt to summarize the high-level trends from publicly available sources and from our own experience.

10.1.3.1 Reliability vs availability

Reliability and availability represent distinct but intertwined system characteristics fundamental to fault-tolerant systems. While both contribute to overall system robustness, they quantify different aspects of a system's operational profile and thus are related but distinct measures. Reliability quantifies the probability of failure-free operation over a period, while availability quantifies the probability of a system being operational at a random point in time.

Reliability is formally defined as the probability that a system will execute its specified function without failure within a given time interval under defined operational conditions. Quantitatively, reliability is often characterized using Mean Time Between Failures (MTBF) and Failure Rate. MTBF, calculated as the average time elapsed between successive failures, provides a statistical expectation of a system's operational longevity before failure. Conversely, the Failure Rate represents the expected number of failures per unit time. A system exhibiting high reliability will demonstrate a high MTBF and a correspondingly low Failure Rate.

Availability, on the other hand, measures the probability that a system is operational and accessible to fulfill its intended function at any given point during a defined period. This metric quantifies the proportion of time the system is in a functional state, ready for use. Availability is primarily impacted by the system's downtime, encompassing both the frequency and the duration of outages. Metrics associated with availability typically include an Availability Percentage, often expressed in terms of "nines" of availability (e.g., "five nines" = 99.999%), and the Mean Time To Repair (MTTR). MTTR represents the average time required to restore the system to an operational state following a failure. Hence, a system with high availability minimizes downtime, achieving this through a combination of failure prevention (contributing to reliability) and rapid recovery mechanisms (minimizing MTTR).

A system can demonstrate high availability even with comparatively lower reliability. This scenario arises when the system experiences frequent failures but possesses mechanisms for swift restoration, resulting in minimal downtime.

In system design, the choice of emphasizing reliability or availability depends on the application's criticality. High reliability is paramount for systems where failures have catastrophic consequences, such as in aerospace or medical equipment control systems. High availability, however, fits scenarios where temporary or partial service interruptions are acceptable but prolonged outages are undesirable, such as in cloud infrastructure or telecommunication networks.

Because reliability and availability aren't the same, an availability of 99.99% can represent very different experiences. It could represent a system with a single outage per year that lasts about an hour, or weekly one-minute outages, or it could never have a complete outage but always fail to process 0.01% of requests.

Reliability and availability demands of enterprise customers are different. For consumer services, a single average metric often is sufficient since all users are similar. For enterprise services, some customers may be a thousand times more important than others (as measured by revenue, usage, brand prominence, etc). "99.99% of our customers aren't seeing any problems" may be a good state to be in, or a terrible one, if the problems are concentrated in the most important customers. Additionally, most enterprise customers build high-reliability paths to their cloud providers, and thus end-to-end reliability is dominated by actual service reliability, whereas for consumers it may still be dominated by ISP reliability.

For internet services, perfect availability is not a realistic goal because the internet itself has limited availability characteristics. Back in 2001, Chandra et al. [7] reported that internet endpoints may be unable to reach each other between 1% and 2% of the time due to a variety of connectivity problems, including routing issues. That translates to an availability of less than "two nines." In other words, even if a 2001-era internet service was perfectly reliable, users would, on average, perceive it as being no greater than 99.0% available. Over time, however, internet reliability has improved. In 2014, Google measurements (as seen from the perspective of Chrome users) indicated end-to-end availability to Google servers of 99.6%-99.9% but with wide variations per ISP. Some areas of the world still experience significantly lower availability, caused by less robust infrastructure and intentional shutdowns [8]. End-to-end availability in 2024 has improved to better than 99.95% across ISPs in developed countries, and 99.8-99.9% in developing countries. These numbers don't include large but rare outages and thus may overestimate actual availability. For example, a dual cable cut on May 12, 2024 caused prolonged outages in several countries including Kenya, Tanzania, Rwanda, and Uganda.

In addition to reliability and availability metrics, two other metrics focus on outage recovery. The Recovery Time Objective (RTO) is the maximum acceptable time a system can be offline, dictating how fast services must recover. It answers, "How long can we be down?". In contrast, the Recovery Point Objective (RPO) is the maximum acceptable amount of data loss, measured by the time between the last good backup and the state at

the outage. RPO dictates the frequency of back ups. Essentially, RTO is about tolerance for downtime, while RPO is about tolerance for data loss.

10.1.3.2 Severity

Hardware or software faults can affect internet services to varying degrees, resulting in different service-level failure modes. The most severe modes may demand high reliability levels, whereas the least damaging modes might have more relaxed requirements that can be achieved with less expensive solutions. We broadly classify service-level failures into the following categories, listed in decreasing degree of severity:

- Corrupted: Committed data is lost or corrupted, and is impossible to regenerate.
- Unreachable: Service is down or otherwise unreachable by users.
- Degraded: Service is available but in a degraded mode (e.g., some functionality may be unavailable).
- Masked: Faults occur but are hidden from users by fault-tolerant software and hardware mechanisms.

Acceptable levels of robustness will differ across those categories. We expect most faults to be masked by a well-designed fault-tolerant infrastructure so that they are effectively invisible outside of the service provider. It is possible that masked faults will impact the service's maximum sustainable throughput capacity, but a careful degree of overprovisioning can ensure that the service remains healthy.

If faults cannot be completely masked, their least severe manifestation is one in which there is some degradation in the quality of service. Here, different services can introduce degraded availability in different ways. One example of such degraded service is when a web search system uses data partitioning techniques to improve throughput but loses some systems that serve parts of the database [9]. Search query results will be imperfect but probably still acceptable in many cases. Graceful degradation due to faults can also manifest as reduced freshness. For example, a user may access their email account, but new email delivery is delayed by a few minutes, or some fragments of the mailbox could be temporarily missing. Although these kinds of faults also need to be minimized, they are less severe than complete unavailability. Internet services need to be deliberately designed to take advantage of such opportunities for gracefully degraded service. In other words, this support is often application-specific and not easily hidden within layers of cluster infrastructure software.

One particularly damaging class of failures is the loss or corruption of committed updates to critical data, particularly user data, critical operational logs, or relevant data that are hard or impossible to regenerate. Consumer and enterprise expectations of data durability are high: it is much more critical for services not to lose data than to be perfectly available to all users. At the same time, some data is less critical: for example, copies of the web and their corresponding index files are voluminous and important data for a search engine, but can

ultimately be regenerated by recrawling the lost partition and recomputing the index files. Thus, the best design point for storage durability will differ between systems depending on the value of the underlying data.

10.1.3.3 Root causes

In WSCs, it is useful to understand faults in terms of their likelihood of affecting the health of the whole system, such as causing outages or other serious service-level disruption. A classic 2003 paper by Oppenheimer et al. [10] studied three internet services, each consisting of more than 500 servers, and tried to identify the most common sources of service-level failures. They conclude that operator-caused or misconfiguration errors are the largest contributors to service-level failures, with hardware-related faults (server or networking) contributing to 10–25% of the total failure events.

Oppenheimer's results are somewhat consistent with the seminal work by Gray [11], which doesn't look at internet services but instead examines field data from the highly fault-tolerant Tandem servers between 1985 and 1990. Gray also found that hardware faults are responsible for a small fraction of total outages (less than 10%). Software faults (~60%) and maintenance and operations faults (~20%) dominate the outage statistics.

More recent studies come to similar conclusions: failures are predominantly software related, and hardware failures are comparatively rare. For example, an analysis of Google postmortems between 2010 and 2017 showed that the top two triggers of outages were configuration and binary pushes, representing 68% of all outages [2]. The top root causes were software bugs (41%), development process failure (e.g., missing tests, 20%), and complex system behavior (unanticipated interactions between systems, i.e., "deep" bugs, 17%).

At first, it is surprising to see hardware faults contributing to so few outage events in these two widely different systems. Rather than making a statement about the underlying reliability of the hardware components in these systems, such numbers indicate how successful the fault-tolerant techniques have been in preventing component failures from affecting high-level system behavior. In Tandem's case, such techniques were largely implemented in hardware, whereas in cloud services, we can attribute it to the quality of the fault-tolerant software infrastructure. Whether software- or hardware-based, fault-tolerant techniques do particularly well when faults are largely statistically independent, which is often (even if not always) the case in hardware faults. Software-, operator-, and maintenance-induced faults have a high impact on outages because they are more likely to affect multiple systems at once, thus creating a correlated failure scenario that is much more difficult to overcome.

Factors other than hardware equipment failure dominate service-level disruption because it is easier to architect services to tolerate known hardware failure patterns than to be resilient to general software bugs or operator mistakes. A study by Jiang et al. [12], based on data from over 39,000 storage systems, concludes that disk failures are in fact not a dominant contributor to storage system failures. That result is consistent with analysis by Ford et al. [13]

of distributed storage systems availability. In this study, conducted using data from Google's Colossus distributed file system, planned storage node reboot events are the dominant source of node-level unavailability. The study also highlights the importance of understanding correlated failures (failures in multiple storage nodes within short time windows), as models that don't account for correlation can underestimate the impact of node failures by many orders of magnitude. Fifteen years later, the study's insights are still valid.

10.1.4 Machine-level failures

10.1.4.1 Restarts and downtime

To design fault-tolerant distributed systems one needs to understand availability at the server level. First, let's consider overall downtime, whatever the cause (such as planned maintenance, operating system bugs, or hardware failure).

Google's machine-level failure and downtime statistics are summarized in Figure 10.1 and Figure 10.2. The data is based on a six-month observation of all machine restart events and their corresponding downtime, where downtime corresponds to the entire time interval where a machine is not available for service, regardless of cause. These statistics cover all of Google's machines. For example, they include machines that are in the repairs pipeline, planned downtime for upgrades, and all kinds of machine crashes.

Figure 10.1 shows the distribution of machine restart events in 2018. 50% or more machines restart at least once a month, on average. The tail is relatively long (the figure truncates the data at 11 or more restarts) due to the large population of machines in Google's fleet. Approximately 5% of all machines restart more than once a week.

Several effects, however, are conflated by such large-scale averaging. For example, we typically see higher than normal failure rates during the first few months of new server

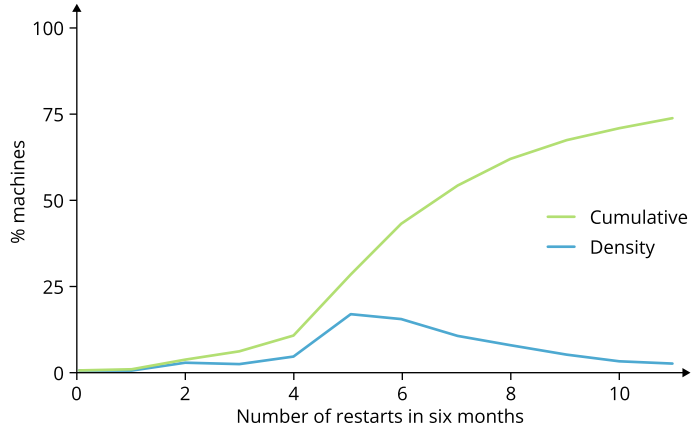


Fig. 10.1 Machine restarts at Google over six months in 2018

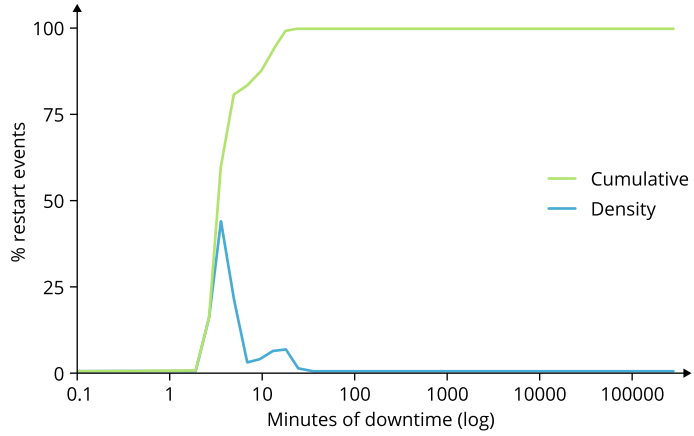


Fig. 10.2 Machine downtime observed over six months

product introduction. The causes include manufacturing bootstrapping effects, firmware and kernel bugs, and occasional hardware problems that become noticeable only after a large number of systems are in use. Similarly, the installation and "burn-in" phase of new machines involves several reboots.

If we exclude non-production machines from the analysis, the average restart rate changes to approximately one month between restarts, largely driven by Google's firmware and kernel release process that supports bug resolution, feature enhancements, and security-related upgrades. We also note that machines with frequent restarts are less likely to be in active service for long.

In another internal study conducted in 2018, planned server unavailability events were factored out from the total machine unavailability. The remaining unavailability, mainly from server crashes or lack of network reachability, indicates that a server can stay up for many years on average. This data confirms the intuition that most restarts are due to planned events, such as software and hardware upgrades.

Frequent upgrades are necessary to keep kernels up to date and to enable Google to prudently react to urgent security issues. For example, when the Spectre and Meltdown CPU security flaws were discovered in 2018, Google was able to update CPU microcode without requiring customer action, so that all customers were already secure by the time the vulnerability was disclosed.

As discussed earlier, Google Cloud's Compute Engine offers live migration to keep the virtual machine (VM) instances running by migrating the running instances to another host in the same zone rather than requiring VMs to be rebooted. As a result, the reboots of physical host servers are largely invisible to customers. However, live migration doesn't work well for customers with highly latency sensitive services, or when using bare metal instances. While some maintenance like CPU microcode upgrades can't be hidden from

customers, the impact and frequency of provider-scheduled disruptions can be reduced via customer-specified maintenance windows (so the disruption happens at a more convenient time), by reducing the code required to run a VM (thus reducing the need for updates), by moving such code off the main CPU to specialized processors, and by designing the software to support hitless upgrades.

Restart statistics are key parameters in the design of fault-tolerant software systems, but the availability picture is complete only once we combine it with downtime data—a point articulated earlier by the Berkeley ROC project [14]. Figure 10.2 shows the 2018 downtime distribution from the same population of Google servers. The average annualized restart rate across all machines is 12.4, corresponding to a mean time between restarts of just less than one month. The x-axis displays downtime, against both density and cumulative machine distributions. The data includes both planned reboots and those caused by miscellaneous hardware and software failures. Downtime includes all the time a machine stopped operating until it has rebooted and basic node services were restarted. In other words, the downtime interval ends not when the machine finishes rebooting, but when the machine is able to take on load again.

Approximately 55% of all restart events last less than 3 minutes, 25% of them last between 3 and 30 minutes, with most of the remaining restarts finishing in about a day. Long outages usually correspond to some combination of physical repairs and automated file system recovery from crashes. Approximately 1% of all restart events last more than a day, usually corresponding to systems going into repairs. Average downtime is just over 10 minutes. The resulting average machine availability is 99.93%.

10.1.4.2 Machine reliability

Relatively little published field data exists on server availability. A 1999 study by Kalyanakrishnam et al. [15] finds that Windows NT machines involved in a mail routing application had an MTTF of approximately 22 days, or an annualized machine failure rate of more than 1,600%. In 2007, Schroeder and Gibson [16] studied failure statistics from high-performance computing systems at Los Alamos National Laboratory and found a failure rate of 0.3 faults per year per CPU. For a dual-socket server, this would imply an annualized machine failure rate of 60% or an MTTF of approximately 20 months. This rate of server failures is more than 25 times lower than the one observed by Kalyanakrishnan.

More recent data puts the crash rate of mature servers (those that survived infant mortality) comfortably below a 5% annual failure rate. Figure 10.3 shows the distribution of failure events on Google's machines during a six-month period in 2024 (note the logarithmic Y axis). Over 97% of all machines experienced no crashes, 2% experienced one crash, and a small tail population of machines experienced multiple crashes. Clearly, single-machine reliability of cost-efficient servers has made great strides, from an annual failure rate of over 1000% in 1999 to 60% in 2009 to less than 5% today. That trend isn't accidental, since the

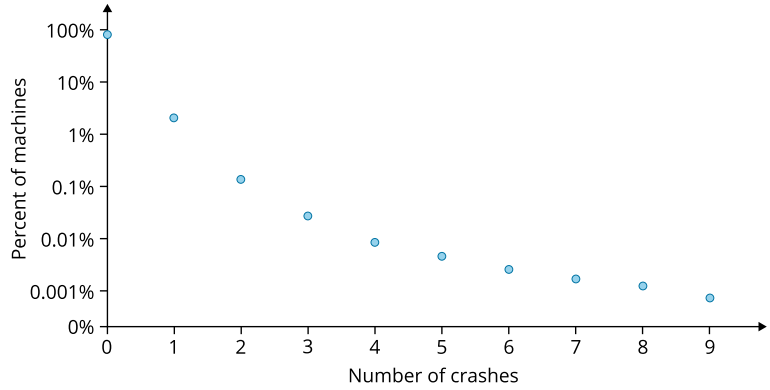


Fig. 10.3 Distributions of machine crashes at Google over six months in 2024

advent of public cloud services with their different reliability requirements has increased the attention to single-machine reliability.

However, these reliability improvements still are no match for scale. With a server crash rate of 5% per year (or a server MTBF of 19.4 years), a service using 2,000 servers needs to plan for a machine crash approximately every 3-4 days.

The reliability of VMs can significantly exceed the reliability of the underlying hardware if the system can migrate VMs off of failing servers. In particular, many hardware failures are memory failures where frequent parity errors can presage an actual data loss event. Even if memory actually fails with an unrecoverable error, VMs that don't have any of their memory mapped to the failing memory cells can be migrated to a healthy machine. As a result, GCE VMs routinely reach average theoretical lifetimes of 10 years or more. That is, if VMs were left undisturbed by the customer, they would run for more than a decade without experiencing a failure. In reality, most VMs are restarted much earlier because customers upgrade their guest kernel or restart the VM for application changes.

10.1.4.3 What causes machine crashes

When considering the cause of an individual machine crash, it can be tempting to ask whether it was due to hardware or software. Certainly, some operating system bugs can cause machine crashes and reside strictly in the realm of software. However, modern hardware includes a large volume of firmware and microcode that blurs this distinction, and even defects that are etched in silicon are often guarded by logic that allows a software update to control their use.

We find a more useful distinction to be whether a machine crash is caused by behavior that is specific to an individual unit of hardware or one that is widespread across a hardware fleet. In the former case, the cause is usually a manufacturing defect or late-life wear-out, and the appropriate response is to repair the machine by replacing the component. In the

latter case, no amount of like-for-like component replacement will help, and we are instead dealing with a bug that we must identify and remediate.

Reliably identifying culprits for machine crashes is generally difficult because, in many situations, transient hardware errors can be hard to distinguish from operating system or firmware bugs. We rely on component-level diagnostics to identify behavior that is occurring outside of specification.

These diagnostics have improved greatly over the past decade, which is relevant for two reasons. First, improved diagnostics allow repair pipelines to identify and replace the faulty component with higher accuracy, meaning that we see fewer repeat failures and a lower overall failure rate. Second, improved diagnostics have allowed us to identify the following three trends in machine crash rate across Google's fleet.

Historically, memory faults were the most common reason for machine crashes. Faults usually are characterized using FIT (Failure In Time), a unit that represents many failures occurring every 10^9 hours. A 2009 study [17] evaluated DRAM errors for the population of servers at Google and found FIT (Failure In Time) rates substantially higher than previously reported (between 25,000 and 75,000) across multiple DIMM technologies. That translates into correctable memory errors affecting about a third of Google machines per year. Because of ECC protection, however, only about 1.3% of all machines ever experience uncorrectable memory errors per year. A subsequent 2012 study [18] found that a large fraction of DRAM errors could be attributed to hard (non-transient) errors and suggested that simple page retirement policies could mask a large fraction of DRAM errors in production systems while sacrificing only a negligible fraction of the total DRAM in the system.

In the past decade, DRAM has become more reliable. A 2023 study [19] collected data from tens of thousands of servers across multiple data centers at Google. Server DRAM used chipkill ECC [20] and observed FIT rates of 70-100 per device, much improved from the rates seen years earlier. Interestingly, transient errors caused by cosmic radiation striking a DRAM and flipping the charge level of a bit accounted for about a third of all errors.

At the same time, failures caused by the boot drive hosting the machine's operating system boot disk have also fallen dramatically. Originally built on disk drives and co-resident with data storage, loss of the boot disk was a large source of machine crashes. Rather than continue with this architecture, a transition to "storage-less" compute servers, built with small, lightly utilized SSDs has nearly eliminated this class of failure.

With these sources of failures addressed, the proportion of failures from processing units – CPUs and offload processors – began increasing. In response, improvements in diagnostics to direct individual repairs and improvements in load testing and at-scale debugging have kept the processing unit failure rate in check.

What we are left with is a balanced failure decomposition, where we see roughly equivalent contributions from CPU failures, RAM failures, NIC and offload processor failures, and operating system failures. Not all failures can be diagnosed, and this remains an area of active improvement. We consider the balance to be reflective of a mature technology ecosystem where outsized sources of failure have been incrementally reduced.

ML training workloads are even more sensitive to failures than cloud workloads, since they involve highly synchronous computations where a failing machine brings the entire training run to a halt. Even with automated failure handling, these failures impact available throughput. For example, during a 54-day Llama3 training run, the 16K-GPU cluster encountered about 320 hardware failures that reduced training run throughput by about 10% [4].

10.1.5 Storage errors

Storage devices can fail, either entirely (the entire device becomes unusable) or partially (some data cannot be read anymore). Three classic 2007 studies based on data from NetApp and the University of Wisconsin [21], Carnegie Mellon [22], and Google [23] examined the failure characteristics of disk drives. Hard failure rates (measured as the annualized rate of replaced components) ranged between 2% and 4% in these large field studies. Bairavasundaram et al. [21] looked specifically at the rate of latent sector errors—a measure of data corruption frequency. In a population of more than 1.5 million drives, they observed that less than 3.5% of all drives develop any errors over a period of 32 months.

The effect of ambient temperature on the reliability of disk drives was studied by Pinheiro et al. [23], and El Sayed et al. [24]. Both of these field studies suggested that most disk errors appeared to be uncorrelated with temperature as long as the disk temperature was below 45°C. Newer disks, as well as SSDs, appear to be more sensitive to temperatures, however.

SSDs are relatively unreliable at the raw bit storage layer but are highly reliable at the block layer since they employ extensive error detection and correction logic. However, SSD reliability and durability are threatened by two fundamental limitations. First, SSD bits can be written a limited number of times only: before writing new data, a destructive block erase operation resets the bits, and then a high-voltage "program" cycle sets the new bits. Each program/erase (P/E) cycle degrades the storage media slightly, so that after a certain number of cycles the raw bit error rate begins to increase (i.e., new writes may fail to set all bits correctly) [25]. SSD erase blocks are fairly large, between 1 and 100 Mbytes. To allow writes smaller than entire blocks, each block is subdivided into pages that are at least 4-16 Kbytes large.

The number of P/E cycles before this degradation becomes significant is the SSD lifetime or endurance [26]; an alternate metric is drive writes per day (DWPD), i.e., how many times each byte can be written per day for the drive to last for the given warranty period. Unfortunately, endurance degrades with density: lower-capacity (SLC) SSDs provide 10,000+ erase cycles but the highest-capacity SSDs (currently TLC and QLC) provide about 3,000 and 1,500 P/E cycles, respectively. Thus, if a 4TB TLC drive receives 12TB of write traffic per day (140 Mbytes/sec), or 3 DWPD, it will start to fail after 1000 days. Fortunately, higher-density SSDs contain more bytes, so an 8TB SSD with an endurance of 1,500 cycles can

absorb the same average write rate as a 4TB SSD with an endurance of 3,000 cycles. Thus, in practice, endurance is a solvable problem [27], [28], [29].

Write amplification occurs when physical writes to SSDs are larger than logical writes. Since SSDs can only be rewritten in pages, the smallest physical write is a page write. If an application updates a single byte in a file stored on an SSD, the device will still write a full physical page (and eventually erase the old page). Thus, a one-byte logical write turns into a 4 or 16 Kbyte physical write, for a write amplification factor of 4-16,000! Consequently, SSDs are best matched to systems that use log-structured storage that coalesces writes into an append-only write log.

The rise of SSDs brought additional published studies about their reliability. A 2015 study [30] provided some insights of Google SSD reliability at the level of raw NAND error patterns. A 2019 Alibaba study looked into correlated SSD failures out of nearly 1M SSDs [31], quite similar to [13] but targeting SSD rather than hard drives.

Lastly, SSD data degrades even when the drive is powered off! NAND SSDs store data in electrons that are placed into a "gated" area. Unlike DRAM, that gate is designed to retain its charge for a long time. During active operation (programming), higher temperatures are good: they make the silicon more conductive, thus increasing the flow of current during program/erase operations. This causes less stress on the tunnel oxide, which improves its capability to hold the electrons inside the gated area, and thus the endurance of the cell improves.

But when powered off, as temperature rises, conductivity again increases, and the charge stored in the floating gate leaks faster. Eventually, enough leakage will change the voltage state of the cell and render data unreadable [25], [32], [33]. The rate of degradation follows Arrhenius' Law so that every 10°C temperature increase results in (k) times faster degradation, with (k) being a technology-specific factor. Reality is more complicated [34], [25]. Enterprise NAND SSDs typically guarantee power-off durability of at least three months at room temperature, but at adversarially high temperatures retention times may be significantly shorter.

Overall, SSD reliability does not dominate the reliability of the overall storage system. At the system level, a 2018 study of Azure outages [35] tried to categorize 112 reported incidents (visible to users or system level monitors). Most incidents were deemed to have software root causes which were the focus of the paper, and hardware failures accounted for less than 5% of the reported incidents. While that study wasn't specific to storage systems, hardware failures indeed contribute a relatively small contributor to storage system failures, and we don't expect this to change in the foreseeable future. Fundamentally, the degree of system reliability is a function of how fast we must innovate and change. The hardware/platform abstractions (block devices; NVMe etc) are relatively stable and thus reliability frameworks (redundancy codes, standardized monitoring and failure handling) that worked well in the past will likely continue to work well. Meanwhile, on the software and operator front, we must continuously evolve and change to serve new business needs, workloads, APIs, and

other requirements. That velocity in innovation ultimately represents the biggest limit to overall reliability.

10.1.6 Predicting and localizing faults

Predicting whether a possibly-faulty component will cause a future application failure allows the fault to be masked from the users, especially if we predict the failure early enough. These predictions involve short time-horizons—predicting with 100% accuracy that a memory module will fail within the next 10 years is not particularly useful.

Machine learning models can identify early indicators of an application-impacting failure and have replaced traditional heuristics. Models can be calibrated to trade off the costs of a false positive and false negatives. Additionally, they can surface novel signals and adapt to newer hardware generations.

Fault prediction can be valuable even when the model's accuracy is less than perfect, which is often the case. For example, a model developed at Google to detect faulty CPUs that cause Silent Data Corruption [36] has been very successful at directing costly screening where it has the greatest return. Faulty machines are rare (less than 1 in 1000), so even a model with only 30% precision significantly improves screening. The model has replaced cumbersome poor-recall human-crafted rules.

Another model at Google predicts DRAM failures based on observed spatial and temporal patterns of correctable errors. Despite the high false positive rate, this model outperforms the legacy heuristic solution. The low prediction accuracy (~50%) of this model doesn't provide a strong enough signal to send a machine directly to repairs, but machines predicted as faulty are good candidates for migrating workloads from and then running highly intrusive memory scans to confirm a bad DIMM.

Pinheiro et al. [23] described one of Google's earlier attempts to create predictive models for disk drive failures based on disk health parameters available through the Self-Monitoring Analysis and Reporting Technology (SMART) standard. They concluded that such models are unlikely to predict individual drive failures with sufficient accuracy, but they can be useful in reasoning about the expected lifetime of groups of devices which can be useful in optimizing the provisioning of replacement units. A more recent collaboration between Google and Seagate reached high accuracy (98%) with an ML-based model, but predicted only 35% of total failures [37]. Academic papers [38], [39], [31], [40] came to similar conclusions: predicting individual disk failures is a needle-in-a-haystack problem. On the other hand, identifying how many disks failed (annualized failure rates) is an easier problem, and still useful because storage redundancy scheme choices are based on population-level failures.

10.1.7 Repairing faults

An efficient repair process is critical to the overall cost efficiency of WSCs. A machine in repair is effectively out of operation, so the longer a machine is in repair, the lower the overall availability of the fleet. Also, repair actions are costly in terms of both replacement parts and the skilled labor involved. Last, repair quality—how likely a repair action will actually fix a problem while accurately determining which (if any) component is at fault—affects both component expenses and average machine reliability.

Broken machines eventually must be repaired. Traditional scenarios, where a repair must happen immediately, require costly staging of replacement parts and additional costs to bring a service technician on site. When we can batch repairs, we can lower these costs per repair. For reference, service contracts for IT equipment that provide on-site repair within 24 hours typically come at an annual cost of 5–15% of the equipment's value; a 4-hour response time can double that cost.

In comparison, repairs in large server farms are cheaper. To illustrate the point, assume that a WSC has enough scale to keep a full-time repairs technician busy. Assuming one hour per repair and an annual failure rate of 5%, a system with 40,000 servers would suffice. In reality, that number will be considerably smaller because the same technician can also handle installations and upgrades. Let us further assume that the hourly cost of a technician is $100 and that the average repair requires replacement parts costing 10% of the system cost. Both of these assumptions are generously high. Still, for a cluster of servers (hypothetically) costing $10,000 each, we arrive at an annual cost per server of $5\% * (\$100 + 10\% * \$10,000) = \$55$, or 0.55% per year. In other words, keeping large clusters healthy can be quite affordable.

In addition, when many thousands of machines are in operation, massive volumes of machine health data can be collected and analyzed to create automated systems for health determination and diagnosis. Google's system health infrastructure, illustrated in Figure 10.4, is an example of a monitoring system that takes advantage of this massive data source. It constantly monitors every server for configuration, activity, environmental, and error data. This information is stored as a time series in a scalable repository where it can be used for various kinds of analysis, including an automated machine failure diagnostics tool that uses machine learning methods to suggest the most appropriate repair action.

Besides performing individual machine diagnostics, the system health infrastructure is useful in other ways. For example, it monitors the stability of new system software versions and has helped pinpoint a specific batch of defective components across a large fleet of machines. It has also been valuable in longer-term analytical studies, such as the disk failure study mentioned in the previous section, and a data center-scale power provisioning study [41].

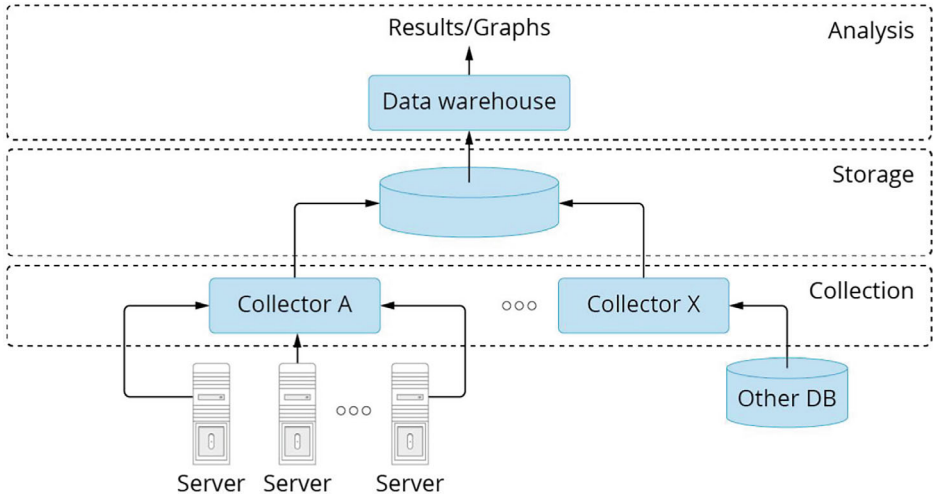


Fig. 10.4 Google's system health monitoring and analysis infrastructure

10.1.8 Tolerating faults

In systems descriptions, we often use N to denote the number of servers required to provide a service at full load, so $N + 1$ describes a system with one additional replica for redundancy. Common arrangements include N (no redundancy), $N + 1$ (tolerating a single failure), $N + 2$ ("concurrently maintainable," tolerating a single failure even when one unit is offline for planned maintenance), and $2N$ (mirroring every unit).

Systems with large amounts of internal replication to increase capacity (horizontal scaling) provide redundancy at a very low cost. For example, if we need 100 replicas to handle the daily peak load, an $N + 2$ setup incurs just 2% overhead for high availability. Hyperscale services that are distributed across multiple regions often are configured at $N + 1$ or $N + 2$ regionally, i.e., they can tolerate the failure of a region while a second region is undergoing maintenance (typically, a software upgrade). Since regions are highly reliable, service owners may choose an $N + 1$ configuration and perform upgrades outside peak traffic times, when lower request levels put the service in an $N + 2$ state (since N is lower during daily troughs).

Such systems can tolerate failures so well that an operator might be unaware of how much internal slack remains, or in other words, how close to the edge one might be. In those cases, the transition from healthy behavior to meltdown can be abrupt: one second everything seems fine (100% of requests are successful) and the next second everything blows up. Thus, it's essential to monitor masked faults and slack capacity, both at the service level and the machine infrastructure level, so that faults can be well tolerated and yet visible to operators.

Good monitoring enables prompt corrective action before the amount of internal redundancy approaches the limits of what the fault-tolerant software layer can handle.

10.1.9 Zonal and regional failures

While hyperscale services like Google Search run globally in many regions, most smaller applications run in a single region, and some enterprise applications may even run in a single zone. One reason for this is operational cost: small services don't need multiple replicas for scale, and thus keeping multiple replicas for availability becomes expensive. Another reason is development cost: building multi-regional services is more complicated, especially for enterprise applications that were originally built for a single on-premise data center.

To support these use cases, a low-level system stack needs to provide sufficient reliability and durability. It's hard to provide high single-zone availability (e.g., 99.99%) since a single software or physical failure (e.g., a faulty update or a malfunctioning circuit breaker) can bring down a zone. It's also nearly impossible to provide high single-zone durability since data stored in a single building could be lost in a localized disaster like a fire or flood. Thus, applications with high availability or durability requirements should not run in a single zone. At the same time, the lowest-level zonal system stack should aim to minimize the impact of such failures, such that only the most significant events would cause significant outages or data loss.

To address this goal, zonal infrastructure typically provides resource allocation and job scheduling with certain availability guarantees, and eliminates planned outages as a source of overhead by allowing concurrent maintenance.

Figure 10.5 shows a schematic of such a cluster stack [42], [43], [44].

The base of the stack delivers power and cooling to machines. Often, the design target is a fault tolerant and concurrently maintainable physical architecture for power and cooling.

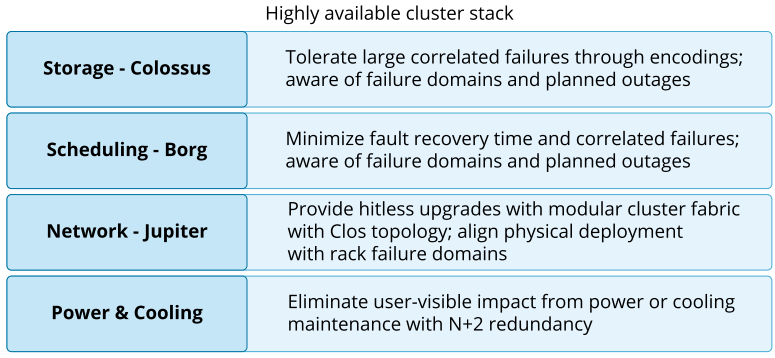


Fig. 10.5 Highly available cluster architecture

For high availability, cluster scheduling spreads jobs across these units of failure. Similarly, the required redundancy in storage systems is in part determined by the fraction of a cluster that may simultaneously fail due to a power or cooling event. Hence, larger clusters lead to lower storage overhead and more efficient job scheduling while still meeting availability and durability requirements.

A typical cluster fabric, like the Jupiter design described in Chapter 6, achieves high availability by redundancy in fabric and physical diversity in deployment. It also focuses on robust software for the necessary protocols and a reliable out-of-band control plane. Given the spreading requirement for the cluster scheduler as previously discussed, Jupiter also supports uniform bandwidth across the cluster and resiliency mechanisms in the network control plane in addition to data path diversity.

Applications that run on Borg (Google's large-scale cluster management system) are expected to handle single-machine failure events using techniques such as replication, storing persistent state in a distributed file system, and (if appropriate) taking occasional checkpoints. Borg mitigates the impact of failure events by automatically rescheduling failed tasks (on a new machine if necessary), and reduces correlated failures by spreading applications across failure domains such as machines, racks, and power domains.

Last but not least, the overall storage system is highly available and deals with errors as previously described in Section 10.1.5 and in Chapter 6.

Together, these capabilities provide a highly available cluster suitable for a large number of big or small services. In cloud platforms such as Google Cloud Platform (GCP), availability is exposed to customers using the concept of zones and regions [45]. Single-zone applications will have the lowest reliability (about 99.7% [46]), whereas dual-region applications can reach 99.998% availability.

Region and zone definitions are very similar across major public clouds, except that Azure has a number of regions with two zones only.

Google internally maintains a map of clusters to zones such that the appropriate product Service Level Agreements (SLAs) can be satisfied (for example, Google Compute Engine SLAs [48]). Cloud customers are encouraged to utilize the zone and region abstractions in developing highly available and fault-tolerant applications on GCP.

Figure 10.6 shows a representative mapping of compute zones and regions. A service like Compute Engine may exist at the regional level, offering functionality like regional load balancing between zones or starting batch jobs in any zone. It is also provided as a zonal API, performing functions specific to a single zone. This distinction allows customers to choose their reliability posture based on their specific needs. For example, a cloud native application may delegate handling zone failures to the cloud provider by using regional load

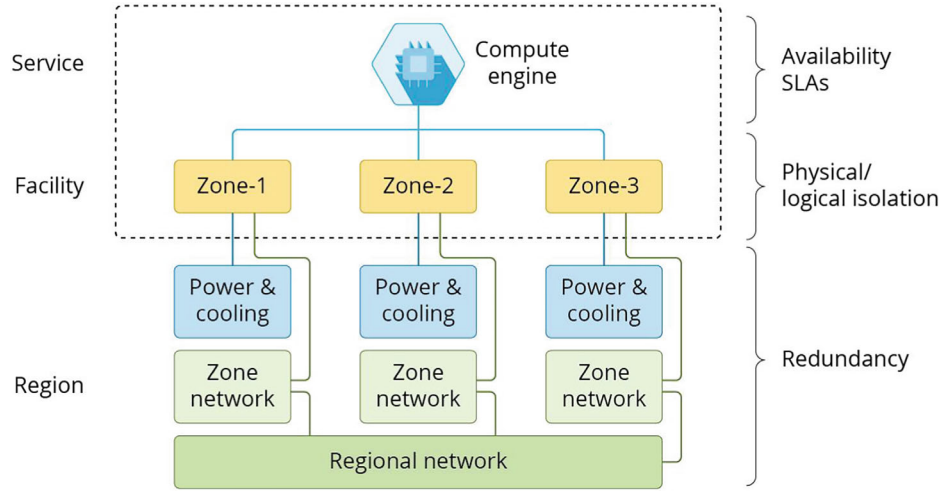


Fig. 10.6 Structure of a cloud service with zones and regions

balancing and a regional database service. Another customer running a high availability application with a primary and secondary instance in two zones can avoid being exposed to any outages of the cloud provider's regional API by only using zone specific APIs.

10.1.10 Silent data corruptions and corrupt execution errors

We are accustomed to thinking of computers as fail-stop and correct in their computations, and most system software implicitly relies on that assumption. For example, if we add two numbers, we implicitly assume that the result will indeed be the correct sum.

Unfortunately, while hardware almost always lives up to these expectations, very rarely (e.g., once in a quadrillion instructions) it may not. Chip manufacturing isn't perfect but a sophisticated array of manufacturing tests ensure that they operate within specifications. For decades, such manufacturing tests proved reliable, resulting in products with only a few dozen defective parts per million.

However, the slowing of Moore's Law described in Chapter 6 has also led to new and insidious reliability problems. Specifically, with increasingly smaller feature sizes and more elaborate computational structures, and increased silicon specialization to improve performance, ephemeral computational errors that are not detected during manufacturing tests have become more common [36], [47], [49], [50]. These defects cannot always be mitigated by techniques such as microcode updates, and are very hard to diagnose. For example, they may be correlated to specific components within the processor, allowing small code changes to effect large shifts in reliability. Worse, these failures are often "silent": the only symptom

is an erroneous computation. (Silent errors are possible because not all processor paths are protected by error detection or correction logic.)

Wider investigation at Google has revealed multiple different kinds of such silent corrupt execution errors, at rates much higher than software engineers expect [36]. These are not just incremental increases in the background rate of manufacturing hardware faults; they can sometimes manifest long after initial installation. Beyond the risks inherent in uncaught errors, such silent data corruptions lead to significant software toil in debugging such errors. In environments where workloads migrate across machines or distributed systems where communication "blast radius" is high, these can also lead to cascading failures.

These increasing silent data corruption errors need a corresponding increased emphasis on new WSC techniques for reliable computation. First, we need to prevent the occurrence of these errors more successfully. Device-level analysis [49] suggests that they are caused by standard device defects (marginal transistors) but that they aren't caught by existing manufacturing tests if the marginal transistor still works but introduces timing errors in a higher-level data flow. Second, since many of these bugs are evident only at WSC-scale deployments and transistors may fail only after some time as the device ages, we need improved in-situ testing, software to monitor in-field aging and wearout and to proactively detect and respond to reliability excursions at scale with low overheads. Additionally, we need to consider new WSC system-level optimizations across all aspects of WSC design: from hardware (design, validation, manufacturing, screening, and burn-in) to applications (computational and data invariants like checksums and replicated compute, data redundancy, sampled invariants, etc). More details are discussed in [51].

10.1.11 Operational practices

So far, this section has focused on physical and technical aspects of reliability. Often, operational practices have an even larger impact on reliability than hardware or software features designed to improve it. For a broad discussion of this area we refer interested readers to the SRE Book [2] but we will briefly highlight some key aspects below.

Change management probably influences reliability more than any other operational practice. Most outages are caused by changes: a new binary has a bug not discovered during testing, a new switch was cabled incorrectly, a device loses power while a nearby new device is installed, a technician opens the wrong circuit breaker in preparation for a repair, a configuration change drops servers because of a truncated configuration file, and so on.

Because of these risks, traditional IT practices discourage changes, but hyperscalers need to support high product velocity, with daily or weekly changes to many services and applications. Additionally, hyperscale requires operational changes that are constant and frequent: every day, thousands of servers may be installed or repaired, buildings or network paths are added or retired, peering ports are turned up, and so on. Thus, change management needs to be automated, with reliability in mind.

Five key practices are common [52].

1. **Failure domain containment**. System designers should identify failure domains (e.g., groupings of physical entities which share power supply, cooling, software control, etc.) and enforce as much as isolation as possible between these domains. Containment minimizes the risk that a failure's "blast radius" will extend beyond domain boundaries. Failure domain isolation also reduces the risk of lockout during disaster recovery, since proper isolation requires systems to avoid cyclic dependencies involved in bootstrapping the failure domain (e.g., the cluster network service cannot depend on a distributed storage service which in turn depends on the cluster network service). In practice, system designers need to strike a balance between minimizing blast radius and minimizing system complexity and cost.
2. **Progressive, supervised change rollout**. Faulty operations can lead to service disruptions or violations of failure domain containment. To mitigate these risks, changes should be rolled out gradually (e.g., zone by zone). Progressive rollout alone does not improve safety; the health of the system needs to be closely supervised throughout the rollout, otherwise it remains vulnerable to "slow wrecks" where service behavior gradually but continuously degrades. To keep up with the volume of operations needed to sustain the business, the system must strike a good balance between safety and velocity (e.g., level of concurrency) of operations.
3. **Monitoring and upholding invariants**. Safety invariants should be monitored during change rollout to maintain system health. Black-box metrics (those that do not assume any knowledge of the internal implementation details of the network, and therefore capture the user's perspective) are effective at detecting failures regardless of root cause as soon as the service behavior begins degrading since they capture the exact conditions that the user experiences, but they are not effective at detecting the degradation before it occurs. White-box invariants, i.e. those that leverage implementation details of their subsystems (e.g., "at least 2 replicas should be responsive") can detect problems early but are hard to generalize so they cover a wide diversity of system interactions.
4. **Provisioning capacity headroom**. Provisioning more capacity than strictly needed to accommodate traffic improves safety. For example, headroom ensures that when a management operation (especially those that require involvement from human technicians) becomes stuck, the operation can remain in a paused state without risking congestion on the unaffected portions of the network.
5. **Fast recovery**. Fast recovery limits outage duration. In practice, recovery often requires humans to understand the situation and determine a recovery strategy. In a system with many ongoing changes, it is not always obvious what needs to be rolled back. Systems can also fail in ways that cannot be recovered by rollback, e.g., if the outage wipes out data, or prevents access to the system itself.

Such practices can be enforced via training and standard practices like change reviews, and they can be supported by software systems. For example, Google's CAPA system [52] employs a central proxy: automated management systems cannot directly enact changes but instead submit them to the proxy that will execute them. Being the sole mutator of production state, the proxy can enforce invariants like gradual rollouts.

Similarly, physical changes need automation to become reliable [53]. In hyperscale networks, many physical changes occur concurrently because many installs, turn-ups, upgrades, and repairs will happen across many locations. Without detailed models to accurately describe the millions of devices in a network, safe changes would be impossible.

10.2 Security at warehouse scale

Cloud users depend on the WSC provider's ability to be a responsible steward of user and customer data. Hence, most providers invest considerable resources into a sophisticated and layered security posture. The scope is as broad as the scale is large, and serious WSC infrastructure requires serious information security expertise.

10.2.1 Data center security: Building and systems

At the physical level, security entails securing the physical data center premises to prevent physical tampering or theft of equipment. WSC providers employ technologies such as biometric identification, single-person locks, metal detection, cameras, computer vision, vehicle barriers, laser-based intrusion detection systems, and other technical means to detect or prevent unauthorized presence. Additionally, they also constrain the personnel even permitted onto the data center floor [54]. Google's data centers define six separate layers of access [55]. For example, to ensure that each person entering a sensitive area is authenticated and authorized, circular single-person doors ensure that nobody can tailgate into the area behind an authorized person (Fig. 10.7).

To ensure a high level of physical security, best practices require regular testing of operational performance. Such tests can be desktop exercises (observing the response to a simulated event) or adversarial ("red team") exercises where an experienced third party is paid to attempt to gain access to a data center by any means necessary (excluding violence, of course).

Additionally, providers need to secure the control systems that operate the facility, to prevent disruptions by electronic attacks that disable or damage equipment. Data center campuses contain thousands of embedded controllers, each of them essential for the proper functioning of a piece of equipment: transformers, electric switchgear and breakers, backup generators, cooling towers, pumps, fans, and so on. Additionally, a Building Management Systems (BMS) allows operators to observe and control the overall operation of the building.

Fig. 10.7 Circular single-person door to prevent tailgating

A BMS and its connected controllers typically are on a segmented network that is "air gapped" from the internet, reducing the likelihood of an external attacker gaining access to it. Air-gapping by itself does not guarantee security though: famously, Iran's Natanz facility was attacked by the Stuxnet virus brought in on a USB stick which then infected the BMS and manipulated the control parameters of the centrifuges in a way that led to their destruction [56].

Unfortunately, current industrial control systems, including systems used in WSCs, lag the state of the art for security practices, and significantly so. In many cases these issues are readily preventable by adopting best current practices: consistently and reliably applying security patches; responsibly managing authentication and authorization; implementing appropriate network segmentation. Recent events and customer demand have increased the industry's attention to security but as of today, the overall security posture of such systems still remains poor.

10.2.2 Security of servers

Most servers in WSCs are custom built for this purpose, and place a high emphasis on security. For example, Google designs the server boards and most networking equipment in a cluster. This includes vetting the component vendors and choosing components with care, and working with vendors to audit and validate the security properties that are provided by the components. Google also designs custom chips, including a hardware security chip (called Titan, discussed further in the next section), that we deploy on servers, devices, and peripherals. These chips let us identify and authenticate legitimate Google devices at the hardware level and serve as hardware roots of trust for boot security and credential provisioning. Variants of the Titan hardware chip are also used in Pixel devices and the Titan Security Key.

Each server in the data center has a unique identity. This identity can be tied to the hardware root of trust and the software with which the machine boots. This identity is used to authenticate API calls to and from low-level management services on the machine. This identity is also used for mutual server authentication and transport encryption. For example, Google's Application Layer Transport Security (ALTS) [57] system secures remote procedure call (RPC) communications, similar to open-source mTLS [58] but with improved scalability. Machine identities can be centrally revoked to respond to a security incident. In addition, their certificates and keys are routinely rotated, and old ones revoked.

Automated systems provide the following guarantees:

- Ensure that servers run up-to-date versions of their software stacks (including security patches).
- Detect and diagnose hardware and software problems.
- Ensure the integrity of the machines and peripherals with verified boot and implicit attestation.
- Ensure that only machines running the intended software and firmware can access credentials that allow them to communicate on the production network.
- Remove or re-allocate machines from service when they're no longer needed.

Services deployed on this infrastructure use encryption for inter-service communication; establish systems for service identity, integrity, and isolation; implement access control mechanisms on top of identity primitives; and ultimately carefully control and audit access to end user or customer data.

Individual challenges, such as authenticating end user identity, give rise to a rich set of security and privacy challenges on their own. Authenticating users in the face of password reuse and poor password practices at scale, and otherwise preventing login or account recovery abuse often require dedicated teams of experts.

Data at rest must always be encrypted, often with both service- and device-centric systems with independent keys and administrative domains. Storage services face a form of cognitive dissonance, balancing durability requirements against similar requirements to be able to truly, confidently assert that data has been deleted or destroyed. Storage services are bandwidth intensive, requiring additional computational resources to ensure the network traffic is suitably encrypted.

Internal network security must be managed alongside suitable infrastructure to protect internet-scale communication. The same reliability constraints that motivate workload migration infrastructure also require network security protocols that accommodate the model of workloads disconnected from their underlying machine. One example is Google application layer transport security (ALTS) [57]. The increasing ubiquity of SSL/TLS, and the need to respond appropriately for ever-present DoS concerns, require dedicated planning and engineering for cryptographic computation and traffic management.

Realizing the full benefit of best-practices infrastructure security also requires a significant and ongoing operational security effort. Intrusion detection, insider risk, securing employee devices and credentials, and enforcing best practices for safe software development all require non-trivial investment.

As information security begins to truly enjoy a role as a first-class citizen in WSC infrastructure, some historical industry norms are being reevaluated. For example, it may be challenging for a WSC to entrust critical portions of its infrastructure to an opaque or black-box third-party component. The ability to return machines to a known state—all the way down to the firmware level—to reassign it from one to the next workload is becoming a must-have.

10.2.3 Root of trust

To trust a system, one must trust that the software running on that system is the correct software. Validating software versions requires transitive trust in the software layer below. For example, if an OS has been tampered with, it can lie about application integrity and undermine any attempts by the application to checksum itself. Similarly, malicious lower-level firmware can tamper with memory or attached devices to undermine the OS itself.

To provide an unambiguously trustworthy starting point, modern servers use a silicon root of trust (RoT). The silicon RoT can help ensure that the hardware infrastructure and the software that runs on it remain in their intended, trustworthy state by verifying that the critical system components boot securely using authorized and verifiable code. The silicon RoT can provide many security benefits by helping to:

- Ensure that a server or a device boots with the correct firmware and hasn't been infected by low-level malware.

- Provide a cryptographically unique machine identity, so an operator can verify that a server or a device is legitimate.
- Protect secrets, like encryption keys, in a tamper-resistant way even for people with physical access (e.g., while a server or a device is being shipped).
- Provide authoritative, tamper-evident audit records and other runtime security services.

The silicon RoT can be used in server motherboards, network cards, client devices (e.g., laptops, phones), consumer routers, IoT devices, and more. For example, Google has relied on a custom-made RoT chip, Titan [59], to help ensure that machines and their peripherals such as accelerators in Google's data centers boot from a known trustworthy state with verified code.

When a device using a RoT is powered on, the RoT is the first processor to start running, before the CPU or any other device. That allows the RoT to validate the signatures of all software that is part of the boot process. This validation happens before the RoT powers up the main processor and points it to the first instruction of the first boot image. Subsequent elements of the boot chain then validate higher-level elements before executing them. For example, the OS image may not be stored locally on the machine, so the bootloader will load it from storage or from the network and validate its cryptographic signature before starting it.

The RoT plays a similarly critical role during updates to firmware images. In a RoT based system, no device (not even the CPU running the OS) can update firmware images. Thus, even if an OS were to become compromised, an attacker attempting to persist their access beyond a reboot by modifying firmware to leave a backdoor will be either thwarted or detected. This kind of write protection usually is ensured in hardware: the write pin of the firmware EPROM is connected to the RoT, and the RoT only.

The RoT does not trust the local CPU and thus doesn't have an API that could be used by a local process to update firmware. Instead, the RoT securely communicates with an off-server control system which is responsible for distributing updates. Thus, an attacker must compromise this small central control system to compromise server firmware, and can't succeed by compromising one or more of the regular servers in the fleet.

Recognizing the importance of anchoring trust in silicon, RoTs receive much attention during design and manufacturing. For example, Google's Titan chip contains defenses against physical tampering (either during manufacturing or when in operation), against supply chain attacks (inserting "fake" Titan chips into a shipping box), and against software tampering. For example, the initial software image of the Titan chip is physically installed in a secure facility via a one-time mechanism, and authenticated cryptographically. After that initial imaging, an on-chip fuse is physically destroyed so that no further physical installation is possible, i.e., all future updates require a private key known only to Google.

To further increase trust in the integrity of Google's cloud infrastructure, the OpenTitan RoT open-sources a full-silicon RoT with a high-quality implementation created by Google and partners [60]. Its goal is to spread the benefits of reliable silicon RoT chips to the rest of

the industry. Google also contributes to the Caliptra open source RoT project [61]. Unlike a stand-alone chip like OpenTitan, Caliptra consists of IP and firmware for an integrated RoT block that is part of a larger chip. Caliptra targets data center-class SoCs like CPUs, GPUs, DPUs, and TPUs. It contains the specification, silicon logic, ROM and firmware for implementing a Root of Trust for Measurement (RTM) block inside an SoC. A Caliptra integration provides the SoC with Identity, Measured Boot, and Attestation capabilities. The combination of a discrete external RoT with updatable storage and internal RoT logic in application processors enables designers to raise the bar for physical tamper-resistance at the whole-machine level, significantly increasing adversary cost.

10.2.4 Encryption

Encryption has long been used to protect important data, but its widespread use is relatively recent. For example, in 2010 Gmail was the first email service to use SSL by default, and in 2014 it was the first service to make it mandatory. In part, this widespread use was enabled by improved hardware support for encryption. The biggest adoption hurdle had in fact been the computational expense of establishing and maintaining SSL/TLS connections.

Encryption-at-rest protects data stored in any storage system. Often, data is in fact encrypted multiple times, with different keys, at different levels of the storage stack. For example, a disk drive may use an onboard key and encryption engine to encrypt data before it's written. Thus, if the disk is stolen, the data on it won't be readable unless the key is stolen as well. However, since the embedded software isn't available for audit and may contain bugs, the storage software layer also encrypts all data, perhaps with a different key for each storage volume, or each storage user [62]. Google encrypts data before it is written to a database storage system or hardware disk. Encryption is inherent in all our storage systems, rather than added afterward.

All of Google's storage systems use a similar encryption architecture, though implementation details differ from system to system. Data is broken into subfile chunks for storage; each chunk can be up to several gigabytes in size. Each chunk is encrypted at the storage level with an individual data encryption key (DEK). Two chunks won't have the same DEK, even if they are owned by the same customer or stored on the same machine.

If a chunk of data is updated, it is encrypted with a new key, rather than by reusing the existing key. This partitioning of data, each using a different key, limits the risk of a potential data encryption key compromise to only that data chunk.

Each data chunk has a unique identifier. Access control lists (ACLs) help ensure that each chunk can be decrypted only by Google services that operate with authorized roles, which are granted access only at that point in time. This access limitation helps prevent access to the data without authorization, strengthening data security and privacy.

Each chunk is distributed across the storage system and is replicated in encrypted form for backup and disaster recovery. An attacker who wants to access customer data would

need to know and be able to access two things: all of the storage chunks that correspond to the data that they want, and all of the encryption keys that correspond to the chunks.

The following diagram shows how customer data is uploaded and then broken into encrypted chunks for storage (Fig. 10.8).

Google uses the AES algorithm to encrypt data at rest. All data at the storage level is encrypted by DEKs which use AES-256 by default. AES is widely used because both AES-256 and AES-128 are recommended by the National Institute of Standards and Technology (NIST) for long-term storage use, and AES is often included as part of customer compliance requirements.

Encryption in transit protects data that is being sent over a network [63]. For example, encryption prevents communications from being intercepted while data moves between a customer site and the cloud provider or between two services. This protection is achieved by encrypting the data before transmission; authenticating the endpoints; and, on arrival, decrypting and verifying that the data was not modified. For example, Transport Layer Security (TLS) is often used to encrypt data in transit for transport security, and Secure/Multipurpose Internet Mail Extensions (S/MIME) is used for email message encryption.

Just like with encryption at rest, data may be encrypted multiple times at multiple layers. For example, a router may encrypt all outgoing packets with a link-specific key [64], and a network connection using that link may use TLS to encrypt the entire connection, and that connection is used to send an encrypted email message. While these multiple levels of encryption are theoretically undesirable, they enable proper layering, defense in depth, and (via ubiquitous hardware acceleration) are fairly inexpensive.

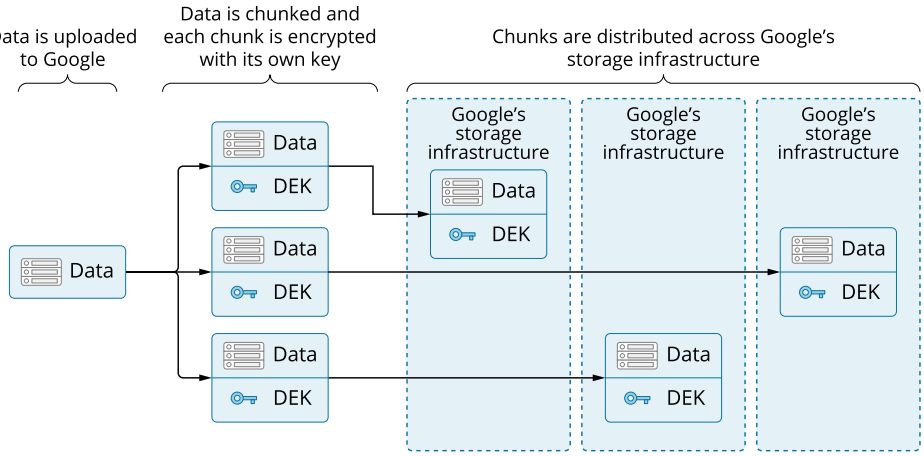


Fig. 10.8 Data encryption at rest in Google storage

Encryption, and security in general, is a fascinating area that deserves a book all its own. For example, even "just" encryption in transit involves many more systems than described here. We refer the interested reader to a high level overview of Google Cloud encryption for an example of that [63]. For in-depth advice on building secure and reliable systems, we recommend the eponymous book [3].

10.2.5 Confidential computing

Confidential VMs can protect the confidentiality of data in the cloud by encrypting data-in-use while it's being processed. With confidential computing, all data is encrypted when it leaves the chip package, even in DRAM, so that an attacker cannot compromise confidentiality even when they have full physical access to a running server. Attacks directly on CPU internals would be required. Confidential VMs take advantage of security features in newer-generation CPUs from AMD, Intel, ARM, and others. With Confidential Computing, customers can be confident that their data will stay private and encrypted even while being processed in the cloud.

Confidential VM instances offer two key benefits. First, isolation: encryption keys are generated by—and reside solely in—dedicated on-chip hardware, inaccessible even to the hypervisor. Thus, even the most privileged service on the machine (the hypervisor) cannot access unencrypted data. Second, attestation: customers can verify the identity and the state of the VM, to make sure that key components haven't been tampered with (no "man-in-the-middle" attack). This type of hardware isolation and attestation is known as a Trusted Execution Environment (TEE) [65]. Cell phones also contain TEEs, e.g., for biometric authentication.

Each vendor implements confidential computing features differently. In 2016, AMD Secure Encrypted Virtualization (SEV [66]) was the first implementation to ship and offers hardware-based memory encryption through an AMD Secure Processor, and boot-time attestation via an external TPM. High-performance encryption inside the CPU decrypts or encrypts data exchanged between the on-chip cache and main memory. The performance difference between an SEV Confidential VM and a standard VM usually is minimal.

While SEV allows for high performance, it doesn't protect against some sophisticated attacks where a malicious hypervisor compromises the confidential environment via encrypted data replay and memory remapping. AMD Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP [67]) expands on SEV, adding hardware-based integrity protections. The basic principle of SEV-SNP integrity ensures that if a VM is able to read a private (encrypted) page of memory, it must always read the value it last wrote. The downside of this additional protection is additional overhead, especially for I/O intensive applications.

Intel Trust Domain Extensions (TDX [68]) is another hardware-based TEE. TDX creates an isolated trust domain within a VM, and uses hardware extensions for managing and encrypting memory. TDX protects against many forms of attacks that use physical access to

DRAM, including capturing, modifying, relocating, splicing, and aliasing memory contents. However, it does not protect against replay attacks. ARM's CCA [69] protects data and code within hardware-isolated secure environments called Realms, shielding workloads from even privileged software like the operating system or hypervisor. Similar to SEV-SNP, CCA is designed to thwart memory replay attacks.

Complementing these CPU features, cloud providers also offer external key management (EKM) that allows customers to store their encryption keys in hardware security devices hosted outside of the cloud provider's space [70]. Using mutual attestation of their identities, CPUs with TEEs can obtain keys directly from these EKMs without risk of tampering or key access by the cloud provider.

10.2.6 CPU vulnerabilities

In addition to silicon roots of trust that verify firmware and software integrity, server security depends on the reliability of the hardware itself, especially the bug-free performance of CPU instructions. The correct implementation of memory protection is especially important: all operating systems rely on virtual memory to isolate the address spaces of separate processes, such that it is impossible for one process to read the contents of another process' memory without explicitly being granted permission. Without memory protection, it becomes difficult or impossible to efficiently run mutually distrusting workloads on a single CPU.

However, consisting of billions of circuits, complex chips like CPUs do contain bugs despite the vendors' best efforts. Such bugs usually are discovered early in the rollout of a new chip and are described in "errata" documents. Most modern chips contain on-chip software in the form of microcode, and often such bugs can be fixed by updating that microcode (with such updates being themselves protected by the silicon root of trust).

Until 2017, such bugs were fairly rare and weren't considered a top security concern. But in a watershed moment in June of 2017, researchers at Google's Project Zero and in academia almost simultaneously found two bugs that were later described as "possibly the worst CPU flaws ever found", not just because of the severity of the two bugs themselves but because it was immediately apparent that they represented the first examples of a new class of CPU bugs and that there would be many more, and that they affected virtually every modern CPU [71].

The first few such bugs (Spectre [72], Meltdown [73], and Foreshadow [74]) demonstrated the dangers of side channel attacks created by shared CPU resources. A side channel attack is an attack that doesn't directly attack the target functionality (e.g., memory protection) but instead uses a "side channel" that is correlated with the target to learn secret information about the target. For example, suppose a password validation function compares two strings, the password that's entered and the password that's stored. If that string comparison terminates early when it discovers a mismatch, and if we can measure the execution time of the password check with high enough precision, then we can guess the correct password via this timing

side channel. To do so, we pick a random password starting with 'a' and then 'b' etc until we discover an execution that took longer than the others. That execution was one where the first iteration of the string comparison succeeded and the second one didn't, and thus we now know the password's first character.

In reality, things aren't that simple. However, all CPUs provide highly accurate timers (they're needed for performance tuning and other functions), and they also have many instances where physical CPU tables serve multiple processes. That combination creates side channels that can be exploited, especially in CPUs that additionally use speculative execution. These exploits are fairly complex so we won't describe them here, but the two original bugs have been documented extensively and give a good indication of both the complexity of discovering them, but also the complexity of mitigating them.

At the same time, the Spectre and Meltdown attacks illustrated the potential security advantages of WSCs and thus public clouds. In particular, Google knew of the security bug a full six months before it became public, and was able to make critical contributions to mitigations. More generally, WSC providers can be expected to maintain highly skilled teams to find and assess risks and develop or deploy mitigations. WSC providers employ coordinated management systems that can deploy CPU microcode, system software updates, or other patches in a rapid but controlled manner. Some (like Google Cloud) even support live migration of customer VMs to allow the underlying machines to be upgraded without disrupting customer workload execution. As a result, when Spectre and Meltdown became public, Google Cloud customers were already protected against them because the infrastructure had been patched without requiring customers to take action.

10.2.7 Secure service deployment

As discussed earlier, a cluster orchestration service controls the services that are running directly on the WSC infrastructure. This section describes the additional security measures for services deployed at Google; other WSC providers may have similar measures.

The infrastructure does not assume any trust between the services that are running on the infrastructure. This trust model is called a zero-trust security model. A zero-trust security model implies that no devices or users are trusted by default, whether they are inside or outside of the network.

Because the infrastructure is designed to be multi-tenant, data from different customers (consumers, businesses, and Google's own data) is distributed across shared infrastructure. This infrastructure is composed of tens of thousands of machines. The infrastructure does not segregate customer data onto a single machine or set of machines, except in specific circumstances, such as when a customer is using sole-tenant nodes for Compute Engine [75].

10.2.7.1 Service identity, integrity, and isolation

To enable inter-service communication, applications use cryptographic authentication and authorization. Authentication and authorization provide strong access control at an abstraction level and granularity that administrators and services can understand.

Services do not rely on internal network segmentation or firewalling as the primary security mechanism. Ingress and egress filtering at various points in the network helps prevent IP spoofing. Customers can configure additional security mechanisms such as VPC Service Controls [76] and Cloud Interconnect [77].

Each service running on the infrastructure has an associated service account identity. A service is provided with cryptographic credentials that it can use to prove its identity to other services when making or receiving RPCs. These identities are used in security policies. The security policies ensure that clients are communicating with the intended server, and that servers are limiting the methods and data that particular clients can access.

Various isolation and sandboxing techniques help protect a service from other services running on the same machine. These techniques include Linux user separation, language-based (such as the Sandboxed API [78]) and kernel-based sandboxes, application kernel for containers (such as gVisor [79]), and hardware virtualization. In general, we use more layers of isolation for riskier workloads. Riskier workloads include user-supplied items that require additional processing. For example, riskier workloads include running complex file converters on user-supplied data or running user-supplied code for products like App Engine or Compute Engine.

For extra security, sensitive services, such as the cluster orchestration service and some key management services, run exclusively on dedicated machines. For even stronger cryptographic isolation of workloads and to protect data in use, Google Cloud supports Confidential Computing services for VMs, Google Kubernetes Engine (GKE) nodes, and more [80].

10.2.7.2 Inter-service access management

A service can use access-management features provided by the infrastructure to specify exactly which other services can communicate with the service. For example, a service can restrict incoming RPCs solely to an allowed list of other services. That service can be configured with the allowed list of the service identities, and the infrastructure automatically enforces this access restriction. Enforcement includes audit logging, justifications, and unilateral access restriction (for engineer requests, for example).

Engineers who operate on services are also issued individual identities. Services can be configured to allow or deny their access based on their identities. All of these identities (machine, service, and employee) are in a global namespace that the infrastructure maintains.

To manage these identities, the infrastructure provides a workflow system that includes approval chains, logging, and notification. For example, the security policy can enforce multi-party authorization. This system uses the two-person rule to ensure that an engineer acting alone cannot perform sensitive operations without first getting approval from another,

authorized engineer. This system allows secure access-management processes to scale to thousands of services running on the infrastructure.

The infrastructure also provides services with the canonical service for user, group, and membership management so that they can implement custom, fine-grained access control where necessary.

10.2.7.3 Encryption of inter-service communication

The infrastructure provides confidentiality and integrity for RPC data on the network. All Google Cloud virtual networking traffic is encrypted. All communication between infrastructure services is authenticated and most inter-service communication is encrypted, which adds an additional layer of security to help protect communication even if the network is tapped or a network device is compromised. Exceptions to the encryption requirement for inter-service communication are granted only for traffic that has low latency requirements, and that also doesn't leave a single networking fabric within the multiple layers of physical security in our data center.

The infrastructure automatically and efficiently (with help of hardware offload) provides end-to-end encryption for the infrastructure RPC traffic that goes over the network between data centers.

10.2.8 Additional reading

Making systems reliable and secure is hard, because it requires attention at every level of the WSC, from the physical campus to the application code written by cloud users. For further reading, we recommend "Site Reliability Engineering" [81], and "Building Secure and Reliable Systems" [3], two books written by Google SRE and security experts.

If there's one common thread to take away from this chapter, it's that scale can facilitate both reliability and security. Scale doesn't make them any easier, but it forces the WSC provider to look for principled, scalable solutions.

References

1. J. R. Hamilton et al., "On designing and deploying internet-scale services.," in *LISA*, vol. 18, 2007, pp. 1–18.
2. B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, *The Site Reliability workbook: practical ways to implement SRE.* " O'Reilly Media, Inc.", 2018.
3. H. Adkins, B. Beyer, P. Blankinship, et al., *Building secure and reliable systems: best practices for designing, implementing, and maintaining systems*. O'Reilly Media, 2020.
4. A. Grattafiori, A. Dubey, A. Jauhri, et al., *The llama 3 herd of models*, 2024. arXiv: 2407.21783 [cs.AI].

5. Red Hat, *What is Linux kernel live patching?* https://www.redhat.com/en/topics/linux/what-is-linux-kernel-live-patching.
6. Amazon Web Services, *Amazon S3 service disruption information (July 20, 2008)*, 2008.
7. B. Chandra, M. Dahlin, L. Gao, et al., "Resource management for scalable disconnected access to web services," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01, New York, NY, USA: Association for Computing Machinery, Apr. 1, 2001, pp. 245–256.
8. Z. S. Bischof, K. Pitcher, E. Carisimo, et al., "Destination unreachable: Characterizing internet outages and shutdowns," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23, New York, NY, USA: Association for Computing Machinery, Sep. 1, 2023, pp. 608–621.
9. E. Brewer, "Lessons from giant-scale services," *IEEE Internet Computing*, vol. 5, no. 4, pp. 46–55, 2001.
10. D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" In *4th Usenix Symposium on Internet Technologies and Systems (USITS 03)*, 2003.
11. J. Gray, "A census of tandem system availability between 1985 and 1990," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 409–418, 1990.
12. W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics," Article 7, vol. 4, New York, NY, USA: Association for Computing Machinery, Nov. 24, 2008, pp. 1–25.
13. D. Ford, F. Labelle, F. I. Popovici, et al., "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10, USA: USENIX Association, Oct. 4, 2010, pp. 61–74.
14. D. Patterson, A. Brown, P. Broadwell, et al., "Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies," Citeseer, Tech. Rep., 2002.
15. M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," in *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1999, pp. 178–187.
16. B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 78, 2007, p. 012 022.
17. B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," 1, New York, NY, USA: Association for Computing Machinery, Jun. 15, 2009, pp. 193–204.
18. A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design," in *Proceedings of the seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII, New York, NY, USA: Association for Computing Machinery, Mar. 3, 2012, pp. 111–122.
19. M. V. Beigi, Y. Cao, S. Gurumurthi, et al., "A systematic study of DDR4 DRAM faults in the field," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 991–1002.
20. Wikipedia contributors, *Chipkill*, Wikipedia https://en.wikipedia.org/wiki/Chipkill, 2025.
21. L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07, New York, NY, USA: Association for Computing Machinery, Jun. 12, 2007, pp. 289–300.
22. B. Schroeder and G. A. Gibson, "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?," 3, vol. 3, New York, NY, USA: Association for Computing Machinery, Oct. 1, 2007, 8–es.

23. E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, ser. FAST '07, San Jose, CA: USENIX Association, 2007.
24. N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: Why some (might) like it hot," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12, New York, NY, USA: Association for Computing Machinery, Jun. 11, 2012, pp. 163–174.
25. Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Heatwatch: Improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 504–517.
26. SNIA, SSD *Endurance*, https://www.snia.org/forums/cmsi/ssd-endurance.
27. S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder, "A study of SSD reliability in large scale enterprise storage deployments," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 137–149.
28. Solidigm, *QLC NAND technology is ready for the data center*, https://www.solidigm.com/content/dam/solidigm/en/site/products/technology/qlc-nand-technology-is-ready/documents/qlc-nand-ready-for-data-center-white-paper.pdf.
29. I. Stoica, D. Song, R. A. Popa, et al., "Cloud programming simplified: A Berkeley view on serverless computing," in *Flash Memory Summit*, 2019.
30. B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016, pp. 67–80.
31. S. Han, P. P. Lee, F. Xu, et al., "An in-depth study of correlated failures in production SSD-based data centers," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 417–429.
32. Y. Kong, M. Zhang, X. Zhan, R. Cao, and J. Chen, "Retention correlated read disturb errors in 3-d charge trap nand flash memory: Observations, analysis, and solutions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4042–4051, 2020.
33. A. Cox, *DDR5: Compelling Total Cost of Ownership (TCO)*, https://www.jedec.org/sites/default/files/Alvin_Cox%20%5BCompatibility%20Mode%5D_0.pdf.
34. Q. Li, M. Ye, T.-W. Kuo, and C. J. Xue, "How the common retention acceleration method of 3D NAND flash memory goes wrong?" In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, ser. HotStorage '21, New York, NY, USA: Association for Computing Machinery, Jul. 27, 2021, pp. 1–7.
35. H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What bugs cause production cloud incidents?" In *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '19, New York, NY, USA: Association for Computing Machinery, May 13, 2019, pp. 155–162.
36. P. H. Hochschild, P. Turner, J. C. Mogul, et al., "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21, New York, NY, USA: Association for Computing Machinery, Jun. 3, 2021, pp. 9–16.
37. Google Cloud, *Seagate and google predict hard disk drive failures with ML*, Google Cloud Blog: https://cloud.google.com/blog/products/ai-machine-learning/seagate-and-google-predict-hard-disk-drive-failureswith-ml.
38. H. S. Gunawi, R. O. Suminto, R. Sears, et al., "Fail-slow at scale: Evidence of hardware performance faults in large production systems," Article 23, vol. 14, New York, NY, USA: Association for Computing Machinery, Oct. 3, 2018, pp. 1–26.

39. S. Lu, B. Luo, T. Patel, et al., "Making disk failure predictions SMARTer," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 151–167.
40. R. Lu, E. Xu, Y. Zhang, et al., "Perseus: A fail-slow detection framework for cloud storage systems," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 49–64.
41. X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07, New York, NY, USA: Association for Computing Machinery, Jun. 9, 2007, pp. 13–23.
42. D. Serenyi, "Cluster-level storage @Google," in *Keynote at the 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Intensive Computing Systems*, 2017.
43. A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2015, pp. 1–17.
44. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: a decade of Clos topologies and centralized control in Google's data center network," 4, vol. 45, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
45. Google Cloud, *Regions and zones*, Compute Engine Documentation: https://cloud.google.com/compute/docs/regions-zones/.
46. Google Cloud, *Deployment architectures - Infrastructure Reliability Guide*, Google Cloud Architecture Center: https://cloud.google.com/architecture/infra-reliability-guide/design#deployment_architectures.
47. H. D. Dixit, S. Pendharkar, M. Beadon, et al., "Silent data corruptions at scale," *arXiv preprint* arXiv:2102.11245, 2021.
48. Google Cloud, *Compute Engine Service Level Agreement (SLA)*, https://cloud.google.com/compute/sla.
49. M. Shamsa and D. Lerner, "Defect mechanisms responsible for silent data errors," in *2024 IEEE International Reliability Physics Symposium (IRPS)*, 2024, pp. 1–5.
50. D. F. Bacon, "Detection and prevention of silent data corruption in an exabyte-scale database system," in *The 18th IEEE Workshop on Silicon Errors in Logic – System Effects*, 2022.
51. D. F. Bacon, "Detection and prevention of silent data corruption in an exabyte-scale database system," in *The 18th IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2022.
52. B. Liu, C. Scott, M. Tariq, et al., "CAPA: An architecture for operating cluster networks with high availability," in *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'24, USA: USENIX Association, Apr. 16, 2024, pp. 1995–2010.
53. M. Al-Fares, V. Beauregard, K. Grant, et al., "Change management in physical network lifecycle automation," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 635–653.
54. Google, *Security - Google Data Centers*, https://www.google.com/about/datacenters/data-security/.
55. dup.
56. Wikipedia contributors, *Stuxnet*, Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Stuxnet.
57. Google Cloud, *Application Layer Transport Security (ALTS)*, Google Cloud Security Documentation: https://cloud.google.com/docs/security/encryption-in-transit/application-layer-transport-security.
58. Google Cloud, *Mutual TLS authentication overview*, Cloud Load Balancing Documentation: https://cloud.google.com/load-balancing/docs/mtls.
59. U. Savagaonkar, N. Porter, N. Taha, B. Serebrin, and N. Mueller, "Titan in depth: Security in plaintext," *Google Cloud Identity and Security Blog*, 2017.

60. OpenTitan, *OpenTitan - Open Source Silicon Root of Trust*, https://opentitan.org/.
61. Chips Alliance, *Caliptra - Hardware Root of Trust*, GitHub Repository: https://github.com/chipsalliance/Caliptra.
62. Google Cloud, *Default encryption at rest*, Google Cloud Security Documentation: https://cloud.google.com/docs/security/encryption/default-encryption.
63. Google Cloud, *Encryption in transit*, Google Cloud Security Documentation: https://cloud.google.com/docs/security/encryption-in-transit.
64. Google Cloud, *MACsec for Cloud Interconnect overview*, https://cloud.google.com/network-connectivity/docs/interconnect/concepts/macsec-overview.
65. Wikipedia contributors, *Trusted execution environment*, Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Trusted_execution_environment.
66. D. Kaplan, J. Powell, and T. Woller, *AMD memory encryption*, https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf, 2016.
67. AMD, *Strengthening VM isolation with integrity protection and more*, White Paper, https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020.
68. Intel, *Intel®Trust Domain Extensions (Intel®TDX) Documentation*, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html.
69. C. Garcia-Tobin and M. Knight, "Elevating security with ARM CCA," 10, vol. 67, New York, NY, USA:Association for Computing Machinery, Sep. 26, 2024, pp. 34–39.
70. Google Cloud, *Cloud Key Management Service*, https://cloud.google.com/security/products/security-keymanagement#manage-encryption-keys-outside-the-cloud.
71. *HotChips keynote: Spectre/Meltdown history*, https://www.youtube.com/watch?v=d5XzVF0sAZo, 2018.
72. P. Kocher, J. Horn, A. Fogh, et al., "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.
73. M. Lipp, M. Schwarz, D. Gruss, et al., "Meltdown," *arXiv preprint* arXiv:1801.01207, 2018.
74. J. V. Bulck, M. Minkin, O. Weisse, et al., "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, 991–1008.
75. Google Cloud, *Sole-tenant nodes*, Compute Engine Documentation: https://cloud.google.com/compute/docs/nodes/sole-tenant-nodes.
76. Google Cloud, *VPC Service Controls Overview*, VPC Service Controls Documentation: https://cloud.google.com/vpc-service-controls/docs/overview.
77. Google Cloud, *Cloud Interconnect overview*, Cloud Interconnect Documentation: https://cloud.google.com/network-connectivity/docs/interconnect/concepts/overview.
78. Google Developers, *Sandboxed API (SAPI)*, https://developers.google.com/code-sandboxing/sandboxed-api.
79. Google, *gVisor - Application Kernel for Containers*, https://gvisor.dev/.
80. Google Cloud, *Confidential Computing*, https://cloud.google.com/security/products/confidential-computing.
81. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google runs production systems*. " O'Reilly Media.", 2016.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

A Twenty Five Year WSC Journey

11

When Google was founded in 1998, it was clear already that successful web search would require enormous amounts of computing power and storage, and that no single computer would be able to handle this task. Consequently, its infrastructure design marked a fundamental shift towards warehouse-scale computing (WSC). Starting as a niche approach optimized for web search, over the past two and a half decades, WSC has evolved dramatically. Today, it's the mainstream approach underpinning all hyperscale companies and cloud platforms, and is poised to be the foundation for the next wave of AI/ML computing in the cloud. In this chapter, we chronicle the evolution of WSC, highlighting pivotal milestones, lessons learned, and the vast opportunities that lie ahead.

11.1 Historical retrospective of twenty-five years of WSC at Google

Warehouse-scale computing, as we have been discussing throughout this book, considers the delivery of computing infrastructure holistically, designing an entire warehouse-scale data center as one computer. Warehouse-scale computers (WSCs) encompass multiple hardware infrastructure components (compute and storage servers, networking and distributed system design, and data center power and cooling) as well as multiple software components (e.g., data center scheduling, management, and hyperscale/cloud workloads). They optimize for multiple metrics: application performance, improved total costs of ownership (TCO), power efficiency, sustainability, reliability, and manageability at scale. Below, we discuss the past 25 years of WSCs, broken down into epochs of five years each (Fig. 11.1).

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_11

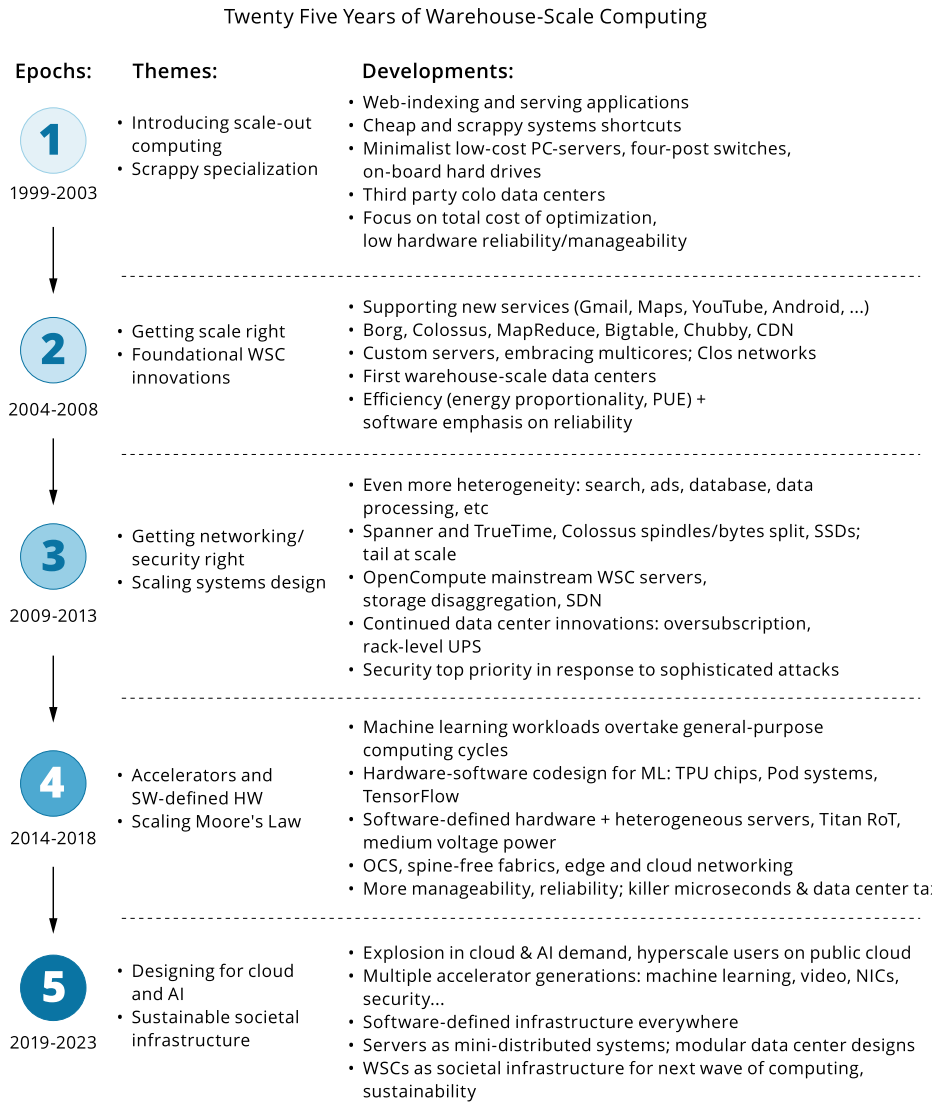


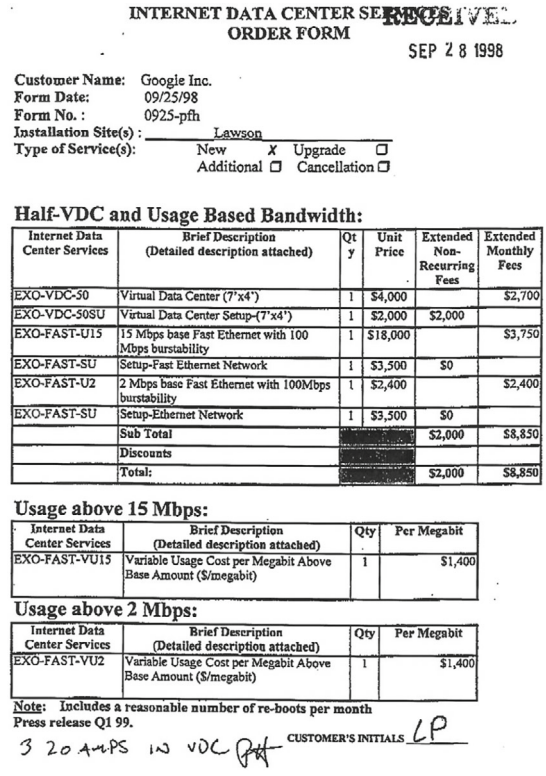
Fig. 11.1 Twenty five years of warehouse-scale computing at Google

11.1.1 Epoch 1: Scale-out computing, scrappy specialization

The first five years of infrastructure design at Google (1999-2003) were about learning to build and deploy large scale-out distributed computing optimized for web search indexing and serving. The core problems to solve were reliability (especially for storage), scalability to larger search indexes and higher query volumes, and reducing cost per query.

Infrastructure design during this period consisted of cheap and scrappy optimizations. There was no time to "do it right," leading to a multitude of pragmatic shortcuts: for example, the initial BigFile storage system used mirrored disks to guard against failures, but required manual work to define a replacement disk and replicate the surviving copy onto the new disk. Similarly, "babysitter" job scheduling scripts, written in python, required manually placing each task on a particular machine, and updating the script when a machine failed. Towards the end of this first period, these ad-hoc systems were replaced by more robust designs including the Google File System (GFS) [1], Global Work Queue (an early version of Borg [2]), and Google front-end (GFE), the reverse proxy in front of all Google services.

All hardware consisted of minimalist low-cost servers built from commodity components normally seen in personal computers [3]. One early server design even used cork for



INTERNET DATA CENTER SERVICES
ORDER FORM

RECEIVED
SEP 2 8 1998

Customer Name: Google Inc.
Form Date: 09/25/98
Form No. : 0925-pfh
Installation Site(s) : Lawson
Type of Service(s): New X Upgrade ☐
Additional ☐ Cancellation ☐

Half-VDC and Usage Based Bandwidth:

Internet Data Center Services	Brief Description (Detailed description attached)	Qty	Unit Price	Extended Non-Recurring Fees	Extended Monthly Fees
EXO-VDC-50	Virtual Data Center (7'x4')	1	$4,000		$2,700
EXO-VDC-50SU	Virtual Data Center Setup-(7'x4')	1	$2,000	$2,000	
EXO-FAST-U15	15 Mbps base Fast Ethernet with 100 Mbps burstability	1	$18,000		$3,750
EXO-FAST-SU	Setup-Fast Ethernet Network	1	$3,500	$0	
EXO-FAST-U2	2 Mbps base Fast Ethernet with 100Mbps burstability	1	$2,400		$2,400
EXO-FAST-SU	Setup-Ethernet Network	1	$3,500	$0	
	Sub Total			$2,000	$8,850
	Discounts				
	Total:			$2,000	$8,850

Usage above 15 Mbps:

Internet Data Center Services	Brief Description (Detailed description attached)	Qty	Per Megabit
EXO-FAST-VU15	Variable Usage Cost per Megabit Above Base Amount ($/megabit)	1	$1,400

Usage above 2 Mbps:

Internet Data Center Services	Brief Description (Detailed description attached)	Qty	Per Megabit
EXO-FAST-VU2	Variable Usage Cost per Megabit Above Base Amount ($/megabit)	1	$1,400

Note: Includes a reasonable number of re-boots per month
Press release Q1 99.
3 20 AMPS IN VDC CUSTOMER'S INITIALS LP

Fig. 11.2 First Google colocation order

insulation, leading to them being called "corkboard servers." Such choices led to unreliable hardware, so an important contribution in this time was to design the web search system to account for and tolerate hardware failures. While hardware has become more reliable over time, daily failures are still inevitable at scale, and this emphasis on hardware co-designed with fault tolerant software continues to be a core principle of WSC systems even today.

CPUs were mainly optimized for speed and cost. Storage consisted of third-party mainstream hard disk drives hosted on each server. "Scale-up" cluster networking was based on four-post 576-port switches with 1 Gbps links and high oversubscription, allowing for 100Mbps of off-rack bandwidth per server. This design required rack-level data placement for large cluster-wide applications like web search, again emphasizing the co-design of infrastructure and applications. There were no Google data centers yet; all the infrastructure was hosted in third party colocation (colo) data centers.

Cost optimization was paramount, with a focus on "total cost of ownership" (TCO). Reliability and manageability in hardware was under-emphasized (or in some cases, ignored) but software was written to work around failures. Colocation data centers charged by space, not power (Fig. 11.2), so initially space efficiency was more important than power efficiency. Towards the end of this epoch, the costs of inefficient server power supplies and on-board DC-DC conversions became clear, leading to the first 12V-only motherboard and highly efficient power supplies.

11.1.2 Epoch 2: Foundational WSC innovations

The second five years of Google (2004-2008) were all about tremendous growth in search (launching more categories, and becoming bigger, faster, cheaper) and introduction of new services (Gmail, Maps, YouTube, Android, Chrome). Increased engineering resources allowed more ambitious solutions. The first production version of Borg [2] doubled cluster utilization with containers. Colossus overcame the scaling limitations of GFS. MapReduce [4], and Bigtable [5] simplified data processing, and many other fundamental services emerged, including the Chubby lock service [6], a Content Distribution Network (CDN) for YouTube, and high-efficiency Google-built data centers. In many ways, this period was about getting foundational infrastructure right.

Infrastructure transitioned to more customized designs. Google started using shipping containers to build modular collections of servers, storage, networking, power, and cooling, and then transitioned to applying the same approach at the warehouse- and data center-scale. Data center efficiency greatly improved with hot-air capture, warm air cooling, and rack-level UPS, standard practices to this day. Two key concepts were developed during this period: (1) energy proportionality [7], reducing power consumption when servers were not fully utilized, and (2) power usage efficiency (PUE), tracking and optimizing the efficiency of power delivery and cooling in the data center.

Servers transitioned to more custom-built designs, adapting to the transition of CPUs from the clock-speed era to the multicore era, and moving away from some more exotic ideas that didn't work (no more corkboard or velcros!). Disks continued to be directly attached to servers, but the introduction of perpendicular media recordings allowed for much higher density.

The very end of this epoch saw the introduction of Clos data center networks based on merchant silicon, greatly improving the off-rack bandwidth available per server. Large applications continued to share clusters, and global caches helped serve traffic closer to users.

Efficiency continued to be paramount, extending beyond traditional server costs to also include power/cooling efficiency. Workload tiers (different classes of service) and Linux containers allowed multiple workloads to share the same server. Hardware reliability improved, but at scale, failures were still assumed to be the default and not exceptional events. Dealing with increased scale continued to be an important concern.

11.1.3 Epoch 3: Getting networking and security right

The next (third) five years of Google (2009-2013) were characterized by even more scale and greater workload heterogeneity, with corresponding innovations in continuing to scale systems design. Three particularly notable highlights of this epoch were innovations in networking, security, and databases.

In terms of WSC workloads, search and ads still continued to be significant, but database and data processing workloads started coming into their own as well. Google Drive was introduced in 2012, and App Engine came out of preview in 2011. A small research group called Google Brain started looking at deep learning. To aid research on clusters and scheduling, Google released a full-month cluster job trace [8], followed by an update in 2019 [9].

Spanner [10] departed from the NoSQL, eventual consistency tradition, and used distributed timestamping (TrueTime) to provide high performance with strong consistency. Colossus introduced erasure coding for better storage efficiency, and allowed separate provisioning of bytes and spindles (seeks) for improved cost amortizations.

Server designs mostly tracked CPU vendors' roadmap. WSC server principles started receiving mainstream acceptance: many enterprise companies created "hyperscale" divisions, and OpenCompute [11] was founded in 2011 to bring WSC computing beyond hyperscalers.

In-house FPGA-controlled NAND flash provided a new storage layer, initially for search and later for fleetwide workloads. The increased cross-section bandwidth of Clos networks allowed storage disaggregation, moving disks from servers to separate disk appliances designed for better performance, maintenance, and more efficient storage provisioning.

Petabit-scale cross-section bandwidths for data center networks allowed almost all applications to ignore rack locality and allowed even SSD storage to be disaggregated from servers

for better utilization and provisioning. Other continued networking innovations underpinned disaggregation and systems advances [12]: notably, cost-effective enterprise-level merchant silicon networking at the campus/WAN levels [13], software-defined networking [14] in the WAN with centralized traffic engineering, and end-to-end bandwidth enforcement across multi-priority traffic. Latency variability forced a continued focus on "the tail at scale" [15].

Security became a big priority, brought to focus by events like sophisticated attacks from overseas actors and revelations from the Snowden leaks. System architectures responded with designs that assumed "zero-trust" and adopted pervasive encryption of data at rest and in transit. WSC power oversubscription increased available data center space by right-sizing deployments to actual low server/cluster utilizations.

11.1.4 Epoch 4: Accelerators, scaling Moore's Law

The next five-year (fourth) epoch (2014-2018) responded to the dramatic slowing of performance-per-cost scaling associated with Moore's law with silicon accelerators, more heterogeneous hardware, and software-defined servers.

Making an inspired bet, Google pivoted WSC innovation to custom silicon for artificial intelligence (AI) and machine learning (ML) workloads (Fig. 11.3). The first such accelerator – Tensor Processing Units (TPUs) [16] – reached production in 2015, demonstrating an order of magnitude better performance and power efficiency relative to CPUs and GPUs. Consistent with core WSC tenets, the solution included numerous innovations in hardware-software co-design (e.g., Tensorflow) and distributed systems design (e.g., TPU "pod" supercomputers). ML computing grew exponentially, soon catching up with the prior huge computing base for non-ML computing: in late 2017, ML cycles surpassed non-ML cycles in the fleet.

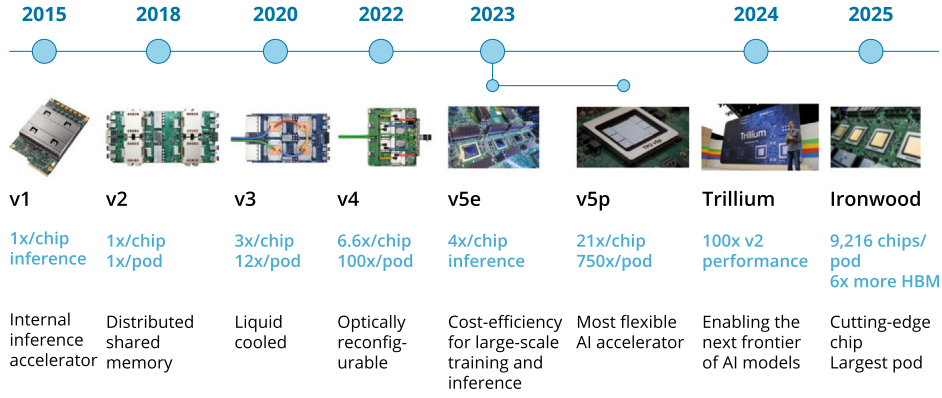


Fig. 11.3 Generations of TPUs

To scale Moore's law, and marking a break from the homogenous server configurations in WSCs till then, systems were optimized for different workloads across heterogeneous CPUs, GPUs, and TPUs. Software-defined servers [17] abstracted hardware complexity, and the compiler/scheduler/management software matched workloads to the computing best suited for it. This also marked a shift in Google's approach to custom designs: in-house specialization focused more on strategic emerging workloads while embracing more industry standardization for mature WSC hardware. Borg evolved considerably during this period to accommodate the greater diversity of both workloads and servers while driving up average utilization significantly [18], [9]. 2014 marked the first release of Kubernetes, an open source container manager inspired by Borg.

Industry trends led to more core counts in CPUs, helium-filled disks for higher densities and lower power, and industry microcontrollers for SSDs and 3D-stacked NAND flash. Driven by Google, server efficiency further improved by switching from 12V to 48V DC for primary power delivery. Networking innovations continued with high speed NICs; optical circuit switching (OCS) [19]; "spine-free" cluster fabrics [20]; software-defined networking management plane optimizations for improved reliability and feature velocity; smarter edge optimizations (Espresso [21]) and cloud virtual networking (Andromeda [22]). A big innovation in data center infrastructure was the successful deployment of medium voltage power delivery [23]; combined with aggressive tiered-workload management, this enabled much higher power oversubscription. Additionally, liquid cooling enabled the much higher power densities of TPU clusters.

The holistic focus on performance, cost, power, security, manageability, and reliability continued. Bare-metal communication including OS-bypass techniques in user-space ("Snap" [24]) allowed low-latency high-IOPS storage, and new latency-mitigating techniques addressed the "killer microsecond" era [25]. Key "data center tax" functions [26] responsible for WSC data management and distributed systems processing were identified, leading to a wave of optimizations. Power management introduced dynamic frequency and sleep state control, and environmental sustainability started becoming a more important consideration. Security improvements emphasized machine integrity and confidentiality and integrity of sensitive data. Google introduced its Titan root-of-trust security chip in 2017, dramatically unveiling it from the tiny stud earring of the keynote speaker to demonstrate its small footprint!

11.1.5 Epoch 5: Cloud & AI, building sustainable societal infrastructure

The most recent five years (2019-2023) saw a remarkable explosion of innovation around AI/ML and public cloud computing. Search and traditional hyperscale workloads continue to grow, but now only represent large private workloads within a larger public cloud computing infrastructure. AI/ML workloads continued to increase in scale, with diverse workloads

(training, tuning, serving) and frameworks (e.g., Jax, Pytorch). WSCs are now foundational societal infrastructure for the next wave of computing.

This period marked yet another step-function increase in the scale of WSC infrastructure, amidst continued headwinds in technology scaling. Multiple generations of new TPU systems introduced new innovations both at the chip level (e.g., support for sparse computation) and the system levels (new interconnects including optics, cooling). Multiple generations of hardware acceleration helped video workloads, with dramatic cost advantages and enabling new video-centric capabilities [27]. (The impact to the media industry from such cloud-scale video acceleration was recognized with a 2024 Technical Emmy award.)

Software-defined servers were additionally optimized for software-defined memory [28], multi-ISA heterogeneity (e.g., ARM servers) and other hardware features (e.g., chiplet-aware scheduling). Software-defined power management expanded to include broader types of oversubscription, at multiple levels of the data center, and across power and cooling [29]. Networking saw the introduction of SmartNICs at WSC scale with accelerated network offloads, and delay-based congestion control achieved 10x tail latency improvements [30]. Software-defined networking continued to evolve (e.g., topology engineering for efficiency, host-based repathing) as also new distributed programming models (e.g., to embrace more declarative programming).

Foundational server design continued to evolve: industry standards and open reference implementations; modular trays to help with form-factor proliferation; supply chain, root-of-trust, confidential computing, and other security optimizations; environmental sustainability and life-cycle carbon footprint management. Beyond traditional monolithic designs, servers evolved to mini-distributed systems managed across multiple "arenas" (daughterboards with their own CPUs, for management, security, networking, storage, etc).

Storage scaling slowed for both density and cost-per-byte. HSMR (hybrid shingled magnetic recording) disks improved data density but required new data placement algorithms. Flash appliances supported independent scaling of compute and SSD storage. data center design reduced cost and construction times with modular prefab infrastructure deployed in smaller increments, and physical server management started using robotics to automate repetitive tasks.

This five year period also saw an increased focus on sustainability as a first-class design criterion. Ambitious targets to achieve net-zero emissions and 24/7 carbon-free energy by 2030 required several new optimizations across hardware (e.g., reduced embodied carbon), software (e.g., carbon-aware scheduling), and operations (e.g., 24/7 matching of supply and demand). New reliability changes stemming from ever smaller transistor sizes manifested in increased faults (silent data corruption [31]), motivating additional focus on testing and reliability. As always, performance and cost-efficiency continued to be paramount, but with the additional challenge of achieving these goals for a heterogeneous multi-tenant cloud infrastructure environment.

Figure 11.4 shows six different server designs across twenty five years of WSC design at Google. The top three pictures are from early generations (note the corkboard and velcro in some of the boards), and the bottom three are from recent generations.

Figure 11.5 shows early Google search server racks hosted at a colocation facility (left), a container-based data center in 2005 (middle), and a modern data center campus in The Netherlands.

Figure 11.6 summarizes key software systems developed as part of this 25-year journey.

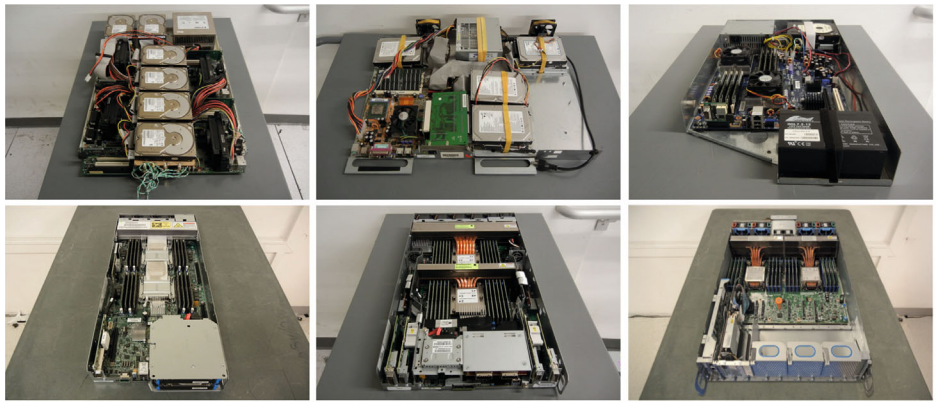


Fig. 11.4 Multiple generations of servers

Fig. 11.5 Housing servers in warehouse-scale computers: An evolution

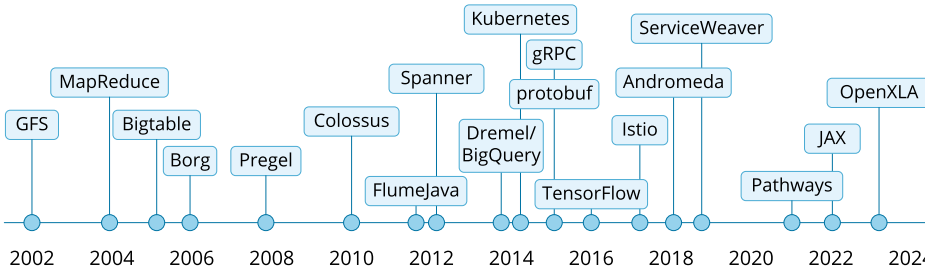


Fig. 11.6 Timeline for key WSC innovations, both internal and open source

11.2 Key lessons from 25 years of WSCs

While our discussions have focused on Google, they also serve as an overview of the broader industry trends around WSCs. Below, we summarize things we learned (sometimes the hard way!) and sketch some important design principles for WSCs.

11.2.1 Scale as a first-class design consideration

WSC infrastructure needs to scale globally to meet the growth of its services. Scale amplifies many traditional design considerations. Small components can add hundreds of millions of dollars of cost. Small power overheads can similarly compound to hundreds of megawatts of power. Failures traditionally considered "rare" (e.g., one in a million) or "impossible" (like cattle causing intermittent failures of a fiber link) can (and do) show up at scale, motivating different approaches to reliability. Maintaining a large WSC fleet can involve handling hundreds of millions of parts per year, so the logistics and supply chain aspects of WSCs can lead to different design choices. Simplicity matters: complex systems are inherently harder to deploy and maintain, more prone to failure, and harder to scale.

WSC design is distributed-systems centric, across compute, storage, and networking. But even with such an approach, every 10X growth in scale requires new thinking because it breaks existing approaches. Architects are never done: important choices need to be revisited with every epoch of growth.

11.2.2 Risk management

You can't innovate if you can't manage risk. Being able to introduce possibly unreliable new hardware or software without threatening the availability of higher level services like search was the key enabler of innovation in the early days of WSC computing. Technical defenses work at multiple levels: systems reschedule around failed servers or disks, block-level checksums detect data corruption, test-driven software deployment detects bugs early, etc.

But beyond technical means, culture is the biggest enabler of innovation, and that's especially true for "operations", the WSC aspect that's perhaps the most underappreciated. Failures will happen, and embracing them as learning opportunities (via blameless post-mortems) allows establishing a rational balance between being safe (but going too slow) and being reckless (going too fast). The site reliability engineering (SRE) discipline has emerged as a key ingredient of WSC design, enabling deep changes even in already large systems. For example, when Google announced their software-defined networking deployment at scale, a telco executive commented "What is amazing is not that you designed the first SDN backbone, it is that you could actually roll it out," pointing to the unusual capability of an operations team to manage risk by co-designing the solution and its rollout.

11.2.3 Managing security deeply

Security needs to go deep. Early generations of WSCs started out like the Internet, relatively open and trust-based, and defended well against external attackers with commercial incentives. But to defend against nation-state actors, much deeper defenses are required. In particular, servers must have a secure, separate silicon root of trust to validate and protect firmware and operating systems; all data must be encrypted, ideally even when being processed; all employees must use phishing-resistant second factor authentications; systems should assume zero trust; important actions or access to user or customer information use multi-party authorization; all production code must be reviewed, and all binaries must have verifiable provenance. Highly skilled Red Teams must regularly test defenses (including physical security); etc.

These past investments have yielded results: today, public cloud and hyperscaler security significantly exceeds typical enterprise security. But security is far from being a solved problem. Supply chain attacks against open source and commercial software threaten every system and need deep defenses at every stage. Systems are too complicated to configure and need built-in, automated audits to prevent mistakes. AI-assisted tools need to detect needle signs of compromise in ever-growing hay stacks of logs. Even better abuse detection needs to prevent bad actors from using public clouds to attack others. Communication systems need to prevent phishing or impersonation attacks while preserving privacy. The ever-growing importance of digital infrastructure will create ever-growing expectations for its security.

Ignore security at your own peril; only the paranoid survive.

11.2.4 Vertically-integrated systems design

WSCs vertically integrate multiple layers of system design: data center infrastructure (power, cooling, building), silicon and hardware (accelerators, servers, switches, storage), systems (software-defined networking, scheduler, data center management), and services/platforms (infrastructure-as-a-service, platforms-as-a-service). Similarly, designs optimize across the entire lifecycle: supply chain, engineering design, deployment, operations, sustainability.

Be intentional about co-design, clearly specifying appropriate division of functionality and separation of concerns across layers with well-defined interfaces and abstractions. A good example is using software to augment hardware fault tolerance when it is too costly to eliminate all risk in hardware. At the same time, co-design without discipline can create systems that are too closely coupled or create "tech islands" (see below).

It is likewise important to pick the right metrics for co-design. While hardware-specific metrics can be helpful, higher-level application-level metrics (and service-level objectives [SLOs]) additionally help identify opportunities to optimize for a global objective function (e.g., service cost per query). Instrumentation and monitoring [32] often are indispensable: while they may come with some overhead for data collection, we have consistently found

that instrumentation-guided optimizations far outweigh these overheads. Machine learning further reinforces the value of carefully curated operational data.

11.2.5 Hardware-software co-design and specialization

WSCs have always emphasized hardware-software co-design, and with decreasing general improvements from Moore's Law, the case for workload specialization has become stronger. TPUs for AI training and inference represent the most prominent example of specialization across multiple layers: (1) systolic arrays for matrix multiply and optical switches at the hardware layer; (2) scale-up and scale-out systems with reconfigurable optical networks at the systems layer; (3) deep fusion, flexible parallelization and communication scheduling in the compiler; (4) bfloat16 and fp8 numerics plus SparseCores at the model/workload layer; (5) optimizations like mixture-of-experts or neural architecture search at the application and algorithmic layer, and (6) energy optimizations including liquid-cooled systems at the data center layer.

Our designs of VCU video accelerators follow a similar approach to co-design across multiple layers [27]. Both examples demonstrate the benefits of thinking beyond classical co-design at the instruction-set architecture level to embrace the full range of options across the systems stack. Interestingly, the key techniques for successful co-design of new accelerators are not that different from those underpinning well-designed software systems: careful decomposition of the problem to minimize complexity; thoughtful definition of interfaces that abstract local information and minimize dependencies, etc. At the same time, dealing with long lead times and costs for hardware development at scale favor modular development/testing and reuse across projects.

11.2.6 Technology islands and industry ecosystems

The initial success of WSCs was driven by being different, out of necessity reinventing many of the then-conventional approaches to system design. However, as WSCs have scaled, and their adoption has increased via public cloud providers, a broad industry ecosystem now supports WSC use cases, allowing "build vs buy" decisions.

Custom designs work best when they target some unique needs of WSC workloads or systems that are currently not satisfied cost-effectively by existing solutions in the market (for example, the design of TPU accelerators for WSC machine learning workloads). For more mature markets, however, volume economics often reduce costs and increase velocity, favoring products built on top of industry standards (example, server and rack form factors).

Focus on building modular, composable, and interoperable architectures built on standardized interfaces. Without this focus on composability and standards, you may end up

on a "tech island" unique to yourself where one custom component forces all others to be custom too. In many ways, this is the hardware equivalent of the monoliths-vs-microservices tradeoff.

11.2.7 A culture of landings versus launches

Over twenty five years of WSC design we have learnt a few important lessons about team culture. One of them is that it is far more important to focus on "what does it mean to land" a new product or technology, instead of focusing on the launch. After all, it was the Apollo 11 landing, not the launch, that mattered. Product launches are well understood by teams, and it's easy to celebrate them. But a launch doesn't by itself create success. Landings aren't always self-evident and require explicit definitions of success -- happier users, delighted customers and partners, more efficient and robust systems – and may take longer to achieve.

While choosing such landing metrics may not be easy, forcing that decision to be made early is essential to success. The landing is the "why" of the project.

11.2.8 A culture of revisiting successful decisions

Across epochs, we have had to revisit past technical decisions, even when the original decision led to exemplary success. For example, for many years we resisted building custom silicon because custom designs can easily be overtaken by the pace of improvements in general CPUs, and they require large volumes to amortize the engineering effort required to build them.

But then Deep Learning became practical, and the speedups of highly specialized processors over CPUs were immense, so we created our first TPU. Almost ten years later, TPUs remain a key differentiator for ML-based products. Similarly, YouTube's large bandwidth costs, combined with the lack of a market for streaming video compression chips, led us to create VCUs, now in their third generation. A similar story has played out in several other design choices over the years.

Do not be afraid to revisit design decisions, especially at every step function of scaling.

11.2.9 Creating moonshots through roofshots

Over the past 25 years, we have achieved orders-of-magnitude improvements (the much publicized 10X improvements, or "moonshots"). Many of them have been the result of the methodical, relentless, sustained pursuit of a series of smaller (1.3-2X) opportunities -- or what we have internally dubbed "roofshots", a term coined by Luiz Barroso.

For example, early WSC infrastructure started with a pile of machines in 3rd party facilities and an intention to do better. We kept improving electrical efficiencies, changing how airflow was provisioned, learned how to pre-fab pieces of the facility for faster construction and higher cost-efficiency, and eventually figured out how to nearly eliminate scheduled downtimes. Roofshot after roofshot and suddenly we had some of the most efficient and reliable data centers in the world, fundamentally different from a traditional design.

Seen from afar, these kinds of achievements could be mistaken as moonshots. They were, in fact, a sequence of roofshots. A sequence of roofshots can produce both quick returns and sustained transformative results. In the words of our late colleague Luiz: "We choose to go to the roof not because it is glamorous but because it is right there. Go out there and have huge dreams, then show up to work the next morning and relentlessly incrementally achieve them."

11.3 Looking ahead

After twenty five years of WSCs, we are at an interesting inflection point. On one hand, computing demand is poised to explode, driven by growth in cloud computing and AI. On the other hand, technology scaling slowdown poses continued challenges to scale costs and energy-efficiency. Combined, these trends mean that some of the most exciting years for WSC design are still ahead of us [33]. Below, we summarize a few grand challenges and opportunities for the community, specifically highlighting key themes around scaling, agility, trust, sustainability, and the disruptive potential of using AI for WSC design.

11.3.1 Efficiency innovation to extend Moore's Law

Responding to the challenges outlined above will require continued innovation across both efficient design (custom silicon), efficient utilization (software-defined hardware), and efficient software (better algorithms).

We will continue to see a plethora of custom silicon accelerators, coarse-grained and fine-grained, and optimized for heterogeneous accelerator-level parallelism. Beyond "doing existing things faster and cheaper", accelerators create opportunities for new functionality at the application level that were not possible before.

Software-defined hardware will require more sophisticated control and automation across multiple levels of software. Automation for data center management, including physical automation in WSC environments, will become widespread. "Room at the top" [34] optimizations need to address inefficiencies in higher-level languages and algorithms, software development practices, and hardware-specific tuning. WSC-scale co-design across hardware and software will continue to be critical to enable these.

11.3.2 Optimizing velocity

The traditional formulation of Moore's law (performance doubles every two years at the same cost) focuses on three variables: performance, cost, and time. As performance and cost improvements slow down, focusing on time — the velocity of hardware development — can be a good way to optimize the "area-under-the curve" for continued improvements. Incremental smaller benefits, when compounded, can still achieve exponential benefits.

To achieve such agile, faster improvements, we need to build more modular hardware platforms with appropriate investment in interfaces, standards, etc. Chiplets in particular allow us to co-design in a multi-die system context [33], allowing cost advantages from die geometries, but also mix-and-match integration across heterogeneous IP blocks and different process technologies.

A particularly exciting development, open source hardware, enables a collaborative and higher-velocity ecosystem that hardware designers can build on — open-source IP blocks (e.g., Caliptra root of trust), verification and testing suites (e.g., CHIPS alliance, OpenCompute), and even open source tools/PDKs (e.g., OpenRoad). Given how profound open source software has been to WSCs, the opportunity for open source hardware is significant.

11.3.3 Trusted computing and sovereignty

New reliability challenges, particularly recent increases in silent data corruption, require new research: from discoverability, localization, and root cause analysis, to serviceability and resilience, to new architectural improvements. WSCs need to be designed to support large-scale in-situ monitoring and testing, with improved software-defined resilience and adaptive service-level optimizations.

Similarly, emerging privacy and security challenges – both new internal and external threats – need new solutions. (We distinguish between security, an infrastructure property, and privacy, an application property.) Confidential computing, homomorphic encryption and even more sophisticated fraud/abuse/threat detection at higher-levels of the stack all require significant investment. Sovereignty will become a first-class WSC design consideration to provide data residency and jurisdictional clarity, and enable more compartmentalized security properties.

11.3.4 Sustainability for a societal infrastructure

Environmental sustainability is both a challenge and an opportunity, touching different areas such as greenhouse gas emissions, water usage, and hazardous materials. Emissions fall into three categories: Scope 1 emissions (e.g., from process gasses with high global warming potential), Scope 2 emissions (e.g., from purchased electricity), and Scope 3 emissions (e.g.,

indirect emissions in the value chain). Opportunities abound in reducing all three of these categories: reduced transportation footprint, improved water utilization and efficiency, minimizing hazardous materials, reduced energy consumption and cleaner sources, adopting circular economies, moving computing to the cloud, etc. The industry needs to adopt consistent metrics to measure and report environmental impact, and broader industry ecosystems to standardize and share best practices.

11.3.5 AI for WSC design

While WSC designs are focused on AI/ML workloads now, we haven't yet seen an equivalent focus on using AI to design WSCs. Future WSCs will feature AI created hardware and software. AI can enable faster iterations with increased coverage and diversity, but will also pose new challenges for testing and validating. AI-assisted tools can potentially enable rewrites of large software systems, opening up new opportunities to quickly evolve out of old ecosystems and evolve interfaces across complex interdependent systems. AI/ML techniques can also optimize control and automation, particularly in the context of software-defined hardware.

AI may be particularly transformative for chip design [35]. The last few years have seen significant advances in using ML across the life cycle: RTL synthesis; placement and routing; verification and validation. More sophisticated models across larger state spaces can optimize across multiple complex objectives spanning power, performance, area, timing rules, congestion, etc. Emerging large language models may dramatically simplify the developer experience. Unlike today where designing a chip can take hundreds of persons working over months, machine learning may enable the dream of designing chips in weeks or even days. We are just getting started in this area.

11.4 Closing remarks

Our work builds on the work of thousands of talented engineers and researchers who have worked on WSCs over the past quarter decade. Their work has helped develop WSC from a set of hand-built computers in a lab running rudimentary web search algorithms to the foundational societal infrastructure it has become today.

We want to particularly dedicate this book to our late colleague, Luiz André Barroso. Luiz joined Google in the early 2000s where he played a key part in enabling a lot of the ideas discussed in this paper. Luiz's work laid the foundation of an entire industry. He didn't just work on how to scale WSCs, he also thought deeply about how to scale WSC innovation. We are grateful for the opportunity this chapter gave us to summarize the history of WSC innovation and hope that this paper inspires others in the community to continue his great legacy.

As we reflect on twenty five years of WSCs and look ahead, the future looks bright. The next era of WSCs will continue to see disruptive innovations: from "mud to cloud," from data center infrastructure to broader cloud computing services; and from "chip to ship," from the design of hardware to their deployment and usage in production. Codesign and collaboration — across the hardware-software stack, across disciplines, and across communities — will be key to this exciting new future.

References

1. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the nineteenth ACM Symposium on Operating systems principles*, ser. SOSP '03, New York, NY, USA: Association for Computing Machinery, Oct. 19, 2003, pp. 29–43.
2. A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2015, pp. 1–17.
3. L. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.
4. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," 1, vol. 51, New York, NY, USA: Association for Computing Machinery, Jan. 1, 2008, pp. 107–113.
5. F. Chang, J. Dean, S. Ghemawat, et al., "Bigtable: A distributed storage system for structured data," Article 4, vol. 26, New York, NY, USA: Association for Computing Machinery, Jun. 1, 2008, pp. 1–26.
6. M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 335–350.
7. L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
8. C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2012, pp. 1–13.
9. M. Tirmazi, A. Barker, N. Deng, et al., "Borg: The next generation," in *Proceedings of the 15th European Conference on Computer Systems*, ser. EuroSys '20, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2020, pp. 1–14.
10. J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Google's globally distributed database," Article 8, vol. 31, New York, NY, USA: Association for Computing Machinery, Aug. 1, 2013, pp. 1–22.
11. Open Compute Project, *Open Compute Project*, https://www.opencompute.org.
12. D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun.* ACM, vol. 55, no. 6, pp. 44–51, Jun. 2012.
13. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: a decade of Clos topologies and centralized control in Google's datacenter network," 4, vol. 45, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
14. S. Jain, A. Kumar, S. Mandal, et al., "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2013, pp. 3–14.

15. J. Dean and L. A. Barroso, "The tail at scale," 2, vol. 56, New York, NY, USA: Association for Computing Machinery, Feb. 1, 2013, pp. 74–80.
16. N. P. Jouppi, C. Young, N. Patil, et al., "In-datacenter performance analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, New York, NY, USA: Association for Computing Machinery, Jun. 24, 2017, pp. 1–12.
17. S. Dev, D. Lo, L. Cheng, and P. Ranganathan, "Autonomous warehouse-scale computers," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20, IEEE Press, Nov. 18, 2020, pp. 1–6.
18. M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13, New York, NY, USA: Association for Computing Machinery, Apr. 15, 2013, pp. 351–364.
19. H. Liu, R. Urata, and A. Vahdat, "Optical interconnects for scale-out data centers," in *Optical Interconnects for Future Data Center Networks*, Springer, 2012, pp. 17–29.
20. L. Poutievski, O. Mashayekhi, J. Ong, et al., "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22, New York, NY, USA: Association for Computing Machinery, Aug. 22, 2022, pp. 66–85.
21. K.-K. Yap, M. Motiwala, J. Rahe, et al., "Taking the edge off with Espresso: scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, Aug. 7, 2017, pp. 432–445.
22. M. Dalton, D. Schultz, J. Adriaens, et al., "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 373–387.
23. V. Sakalkar, V. Kontorinis, D. Landhuis, et al., "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, New York, NY, USA: Association for Computing Machinery, Mar. 13, 2020, pp. 497–511.
24. M. Marty, M. de Kruijf, J. Adriaens, et al., "Snap: A microkernel approach to host networking," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, New York, NY, USA: Association for Computing Machinery, Oct. 27, 2019, pp. 399–413.
25. L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.
26. S. Kanev, J. P. Darago, K. Hazelwood, et al., "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 158–169.
27. P. Ranganathan, D. Stodolsky, J. Calow, et al., "Warehouse-scale video acceleration," *IEEE Micro*, vol. 42, no. 4, pp. 18–26, 2022.
28. A. Lagar-Cavilla, J. Ahn, S. Souhlal, et al., "Software-defined far memory in warehouse-scale computers," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, New York, NY, USA: Association for Computing Machinery, Apr. 4, 2019, pp. 317–330.
29. S. Li, Z. Zhao, K. Chen, and Y. Zhou, "Can we trust run-time profiling for optimizing memory placement decisions?" In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, 2020, pp. 867–882.

30. S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for ultra-low latency," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA: USENIX Association, Apr. 2023, pp. 219–236.
31. P. H. Hochschild, P. Turner, J. C. Mogul, et al., "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21, New York, NY, USA: Association for Computing Machinery, Jun. 3, 2021, pp. 9–16.
32. G. Ren, E. Tune, T. Moseley, et al., "Google-Wide Profiling: A continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, 2010.
33. P. Ranganathan, "A six-word story on the future of VLSI: AI-driven, software-defined, and uncomfortably exciting," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1–4.
34. C. E. Leiserson, N. C. Thompson, J. S. Emer, et al., "There's plenty of room at the top: What will drive computer performance after Moore's law?" *Science*, vol. 368, no. 6495, eaam9744, 2020.
35. J. Dean, *The potential of machine learning for hardware design*, https://www.youtube.com/watch?v=FraDFZ2t__A, 2021.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

A Selection of 25 Top Papers for 25 Years of WSC 12

25 years is a very long time in technology, and warehouse-scale computing has gone from being a niche optimization specialized for Google workloads to being commonplace. This evolution is reflected in a vibrant collection of academic research papers. However, picking the top 25 for this chapter was challenging. To help with our selection, we focused on papers from Google. While this reflects our familiarity with work at Google (and is a nod to the outsized impact Google has had in inventing and scaling warehouse-scale computing), there is a lot of excellent work from outside Google, as demonstrated in the numerous references in this book.

We then asked prominent authors to choose their top papers individually and then combined these scores to identify the top 50 papers. Finally (with great difficulty), we narrowed the list down to the 25 you are going to read about next. Our goal was to give you, the readers, a sense of the top papers in the area, but also provide some breadth across different areas of WSCs.

In this chapter, we also experimented with AI by using Gemini 2.5 to read through our top paper list and provide summaries. We combined that with our own notes and wrote the final summaries you see below. Each paper includes a short paragraph on what the paper is about and then a short (subjective) comment on why we picked it for this top-25 list.

We hope you enjoy reading these excellent papers and have as much fun as we had in compiling this list.

© The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0_12

12.1 The Anatomy of Search: Google's Early Architecture

The Anatomy of a Large-Scale Hypertextual Web Search Engine, Sergey Brin and Lawrence Page, Computer Networks, 30 (1998), pp. 107-117, [1]

This paper marked the beginning of modern web search, demonstrating that the web's hyperlink graph measures page authority (captured by the PageRank algorithm) and greatly improves ranking. The Google prototype also revealed the significant descriptive power of anchor text, successfully utilizing these external labels to improve accuracy. We all take ranking for granted, but this paper showed how to get superior search relevance over content-only methods. Last but not least, their prototype demonstrated one of the key WSC approaches by sharding their web index across inexpensive commodity hardware.

Why we picked this paper: It's **the** classic search engine paper. And it got rejected by the major conference in the field.

12.2 Web Search for a Planet: The Google Cluster Architecture

Web Search for a Planet: The Google Cluster Architecture, Luiz Andre Barroso, Jeffrey Dean, Urs Hölzle, IEEE Micro, 23 (2003), pp. 22-28 [2]

By 2003, Google's architecture had evolved quite a bit from the 1998 prototype that maxed out at two queries per second, and the conceptual ideas of the original prototype had been proven to work in practice. Clusters of 15,000 commodity PCs used sharding to provide horizontal scaling and redundancy. Some of the more frequent hardware failures were now handled automatically. The servers themselves were managed in a relatively manual way because Borg and Linux containers had not been conceived yet. Last but not least, the paper is one of the first ones to discuss power and energy as important system attributes.

Why we picked this paper: It's a short but detailed description of the first real-world WSC stack, optimized for websearch.

12.3 The Google File System

The Google File System, Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, Proceedings of the 19th ACM Symposium on Operating Systems Principles, ACM, Bolton Landing, NY (2003), pp. 20-43 [3]

The Google File System (GFS) was the first real storage system designed at Google, and the first horizontally scalable file system tailored for large-scale, data-intensive applications. Before GFS, the only high-capacity storage systems available required putting expensive

high-performance disks with RAID controllers into a rack. In contrast, GFS assembled a sea of disks, distributed across thousands of machines, into a file system. Files were split into individual chunks, and each chunk was placed on multiple disks to avoid data loss and increase read throughput. A single master node managed metadata. GFS presented a POSIX-like API but relaxed consistency semantics to simplify the system and enhance performance for its target applications. GFS was instrumental in allowing Google to significantly increase its search index size above that of other search engines, and prepared it for storage-heavy future applications like Gmail and YouTube.

Why we picked this paper: How do you write a file system capable of managing tens of thousands of unreliable disks while providing ultrahigh throughput, and how do you implement it in just 18 months? By making judicious choices to simplify key aspects without giving up too much.

12.4 MapReduce: Simplified Data Processing on Large Clusters

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean, Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150 [4]

MapReduce complements GFS: whereas GFS made it easy to store large amounts of data across many devices, MapReduce made it easy to process large datasets across many commodity servers. Users specify "map" functions that process key/value pairs to generate intermediate pairs, and "reduce" functions that merge intermediate values associated with the same key. The runtime system transparently handles parallelization, fault tolerance, data distribution, and load balancing. This abstraction allowed programmers without distributed systems expertise to leverage large clusters effectively. The paper demonstrated MapReduce's utility across diverse tasks like distributed sorting, web access log analysis, and inverted index construction, significantly simplifying large-scale data processing.

Why we picked this paper: Lisp invented the map-reduce pattern, and MapReduce (later entering open source as Hadoop) quickly became the default choice for large-scale batch processing.

12.5 Bigtable: A Distributed Storage System for Structured Data

Bigtable: A Distributed Storage System for Structured Data, Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), {USENIX} (2006), pp. 205-218 [5]

Complementing GFS and MapReduce, Bigtable was the initial solution to hyperscale databases and the start of the NoSQL movement that sacrificed strong transactional consistency in exchange for horizontal scalability. Bigtable stores its data across thousands of commodity servers and exposes a sparse, distributed, persistent multi-dimensional sorted map data model. Data is indexed by row key, column key, and timestamp. Bigtable uses GFS for underlying storage and provides high scalability, availability, and performance. Its strength in providing low-latency reads/writes and efficient scans over massive datasets strongly influenced subsequent NoSQL designs.

Why we picked this paper: It started the NoSQL movement, which is still going strong today (e.g., MongoDB, Redis, Firestore, DynamoDB). Over the past decade, however, Spanner has replaced BigTable for many applications at Google because it offers fully transactional storage with comparable performance.

12.6 The Chubby Lock Service for Loosely-coupled Distributed Systems

The Chubby lock service for loosely-coupled distributed systems, Mike Burrows, 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), {USENIX} (2006) [6]

WSC services need a way to bootstrap their configuration data, and multiple replicas of a single service need to agree who's the leader. Chubby provides this functionality in a single, simple API. It uses Paxos for fault-tolerant consensus among its own replica servers. Its client-side caching mechanism, coupled with keep-alive sessions and event notification, minimizes server load while maintaining consistency.

Why we picked this paper: Chubby is a classic example of a WSC service with a simple but indispensable API that solves common problems. It inspired today's popular OSS equivalents, Apache ZooKeeper and Kubernetes' etc.

12.7 The Case for Energy-Proportional Computing

The Case for Energy-Proportional Computing, Luiz André Barroso, Urs Hölzle, IEEE Computer, 40 (2007) [7]

Computer systems, particularly servers in data centers, should consume power in proportion to the amount of work performed. Most servers aren't energy proportional: they consume a significant fraction of their peak power even when idle or operating at low utilization. This short paper defined the problem and argued that better energy proportionality would significantly reduce data center operational costs and environmental impact.

Why we picked this paper: Sometimes a simple observation, phrased in a way everyone can understand, can influence the direction of an industry.

12.8 Dremel: Interactive Analysis of Web-Scale Datasets

Dremel: Interactive Analysis of Web-Scale Datasets, Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Proc. of the 36th Int'l Conf on Very Large Data Bases (2010), pp. 330-339 [8]

Dremel implements interactive, ad-hoc querying of extremely large, read-only, nested datasets. Its high-throughput parallel query execution engine may use thousands of servers for a single query. Columnar storage allows better compression and significantly reduces I/O load because queries read only partial rows, skipping unused record fields. Its query engine uses a massively parallel tree architecture to execute SQL-like queries over petabytes of data in seconds. For many analytics tasks, Dremel reduced the programming effort and the execution time by several orders of magnitude compared to MapReduce.

Why we picked this paper: Dremel was the precursor to today's leading data lakehouse product, BigQuery, and rapidly displaced MapReduce for many use cases. If you'd like to understand horizontal scalability, read this paper.

12.9 Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag, Google Technical Report dapper-2010-1, Apr. 2010 (2010) [9]

The Dapper tracing infrastructure observes cluster-scale applications with low overhead, application-level transparency, and scalability. It provides developers visibility into requests as they propagate through large microservice-based architectures. By assigning globally unique IDs to requests and propagating context, Dapper reconstructs causal paths and timings across different services. The system employs adaptive sampling to manage data volume while capturing representative traces. Without Dapper (which has evolved into OSS as OpenTelemetry) it would be difficult to understand why a service is slow today, or why some requests are much slower than others.

Why we picked this paper: Dapper is an elegant solution to a problem that is much harder to solve than it looks at first sight.

12.10 Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers

Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers, Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, Robert Hundt, IEEE Micro (2010), pp. 65-79 [10]

Google-Wide Profiling (GWP) provides continuous, low-overhead profiling for WSC applications. It collects performance data (CPU usage, memory allocation, synchronization contention) from running jobs with minimal performance impact, using sampling. Its ubiquitous data collection identifies performance bottlenecks, attributes resource consumption accurately across and inside services, and tracks performance regressions over time. GWP provides the fleet-wide visibility essential for improving the efficiency of warehouse-scale applications without requiring application-specific instrumentation. GWP's in-depth, accurate profiling data supports many use cases, including understanding hardware differences, enabling feedback-directed compiler optimization, and discovering frequently used library code across all binaries.

Why we picked this paper: You'll be surprised at how essential this information is, and that it's possible to collect it with just 0.01% overhead.

12.11 Spanner: Google's Globally-Distributed Database

Spanner: Google's Globally-Distributed Database, James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes et al, Symposium on Operating Systems Design and Implementation (OSDI) (2012), [11]

Spanner is the first database that uniquely combines transactional consistency (external consistency, the highest consistency standard) with high availability and horizontal scalability across geographically distributed data centers. It achieves this through features like automatic sharding, replication using Paxos, and synchronous replication. A key innovation is the TrueTime API, which uses GPS and atomic clocks to generate tightly synchronized timestamps with bounded uncertainty, enabling consistent distributed transactions, consistent snapshot-reads across the database, and lock-free reads. Spanner supports a relational data model and SQL-like query language, providing familiar database semantics at a global scale.

Why we picked this paper: At first glance, Spanner seems to violate the CAP theorem which originally motivated NoSQL systems to forgo full consistency to get availability and scalability. Many see this paper as the #1 database paper of the past 25 years.

12.12 B4: Experience with a Globally Deployed Software Defined WAN

B4: Experience with a Globally Deployed Software Defined WAN, Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong et al, Proceedings of the ACM SIGCOMM Conference, Hong Kong, China (2013), pp. 3–14 [12]

The B4 paper details Google's experience designing, deploying, and operating a private Software-Defined Wide Area Network (SDN WAN) to connect its global data centers. The system separates the control plane from the data plane, using centralized traffic engineering controllers to manage bandwidth allocation and routing across the physical network. B4 employs custom switch hardware and OpenFlow protocols, demonstrating SDN's effectiveness in managing large, complex WANs for improved efficiency, flexibility, and cost-effectiveness compared to traditional WAN architectures.

Why we picked this paper: In many ways, B4 is to large networks what Borg is to large server pools: the only way to manage tens of thousands of devices without losing your mind.

12.13 The Tail at Scale

The Tail at Scale, Jeffrey Dean, Luiz André Barroso, Communications of the ACM, 56 (2013), pp. 74-80 [13]

Tail latency—high-percentile latency—in large-scale distributed systems has a disproportionate impact on user experience. In systems involving thousands of servers for a single request, even rare, small delays on individual components often dominate overall response time due to fan-out. Many factors cause latency variability, including resource sharing, background activities, and queueing delays. The paper outlines several techniques to mitigate tail latency, such as hedging requests (sending the same request to multiple replicas), latency-induced probation, and micro-partitioned data.

Why we picked this paper: Horizontal scaling is awesome until it isn't, and tail latency can spoil the party.

12.14 Profiling a Warehouse-scale Computer

Profiling a warehouse-scale computer, Svilen Kanev, Juan Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, David Brooks, ISCA '15 Proceedings of the 42nd Annual International Symposium on Computer Architecture, ACM (2014), pp. 158-169 [14]

This study complemented Google-Wide Profiling by collecting significantly more data, mostly at the instruction level. The authors analyze instruction mix, CPI (Cycles Per Instruc-

tion), cache misses, branch mispredictions, and front-end stalls across diverse workloads. They demonstrate how microarchitectural features interact with software at scale, revealing bottlenecks like front-end stalls that are often masked in smaller studies.

Why we picked this paper: Real-world WSC loads don't resemble common benchmarks at all. While hardware has changed dramatically over the past decade, most of the observations first made in this paper still hold today.

12.15 Large-scale Cluster Management at Google with Borg

Large-scale cluster management at Google with Borg, Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, John Wilkes, Proceedings of the European Conference on Computer Systems (EuroSys), ACM, Bordeaux, France (2015) pp. 18:1–18:17 [15]

Borg is a large-scale cluster manager responsible for scheduling, deploying, monitoring, and managing applications across tens of thousands of machines. It handles both long-running services and batch jobs, focusing on high reliability, scalability, and efficient resource utilization through techniques like resource reclamation and fine-grained sharing. Borg uses a logically centralized primary scheduler that is replicated for fault tolerance. Distributed agents (Borglets) provide fine-grained resource management on each machine. Borg allocates resources based on application priorities and quotas, supports declarative job specifications, and provides features like rolling updates and health checking.

Why we picked this paper: Borg laid the groundwork for Kubernetes. Its message to manually managed servers was simple: you will be assimilated!

12.16 Site Reliability Engineering: How Google Runs Production Systems

Site Reliability Engineering: How Google Runs Production Systems, Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy, O'Reilly (2016) [16]

Managing thousands of services across myriads of servers shouldn't be an art, it should be a science. Site Reliability Engineering (SRE) embeds software engineering principles into infrastructure and operations tasks, focusing on automation, reliability, and scalability. It provides a blueprint for organizations aiming to build and maintain highly reliable, scalable software systems. Crucially, absolute reliability is a non-goal; rather, it's balancing reliability and velocity, and learning from failures via blameless postmortems.

Why we picked this paper: While we don't expect everyone to read the whole book, you should absolutely read the introduction [17] by Ben Treynor Sloss: "Hope is not a strategy."

12.17 In-Data center Performance Analysis of a Tensor Processing Unit

In-Data center Performance Analysis of a Tensor Processing Unit, Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al, ISCA (2017), pp. 1–12 [18]

This paper disclosed Google's first-generation Tensor Processing Unit (TPU), the first custom ASIC deployed at scale to accelerate AI inference workloads. The authors describe the TPU's architecture, especially its large matrix multiplier unit and on-chip memory optimized for deep learning computations. TPUs demonstrate significantly higher performance and energy efficiency on production AI workloads relative to contemporary CPUs and GPUs. The analysis highlights the benefits of domain-specific hardware acceleration for critical large-scale workloads, and summarizes key learnings underpinning the substantial improvements in throughput and energy efficiency for machine learning inference tasks.

Why we picked this paper: The original TPU paper concluded: "Order-of-magnitude differences between products are rare in computer architecture, which may lead to the TPU becoming an archetype for domain-specific architectures."

12.18 Attack of the Killer Microseconds

Attack of the killer microseconds, Luiz André Barroso, Mike Marty, David Patterson, Parthasarathy Ranganathan, Communications of the ACM, 60(4) (2017), pp. 48-54 [19]

WSCs need to optimize for microsecond-level latencies, driven by faster data center networks, new memory hierarchies, and accelerators. Using two case studies (how to waste a fast data center network, and how to waste a fast data center processor), the paper points out how system optimizations targeting nanosecond- and millisecond-scale events are inadequate for events in the microsecond range. New techniques can achieve high performance at microsecond latencies but need co-design across hardware, kernel, and application layers.

Why we picked this paper: Much like energy proportionality paper discussed earlier, this paper highlights a simple, yet important, challenge for WSCs, and led to a lot of follow-on optimizations in the WSC systems stack.

12.19 Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization

Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization, Mike Dalton, David Schultz, Ahsan Arefin, Alex Docauer et al, 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, pp. 373–387. [20]

Andromeda describes Google Cloud Platform's network virtualization stack. Its SDN (software-defined network) design provides high-performance, secure networking for virtual machines. Andromeda decouples the control plane from a distributed data plane implemented partially on-host and partially offloaded to hardware. This architecture enables rapid provisioning of network services (VPCs, firewalls, load balancers), offers performance approaching bare-metal networks, and ensures strong security isolation between tenants. The design prioritizes product velocity and operational agility, simplifying new network features and scaling infrastructure efficiently.

Why we picked this paper: Network virtualization wants to be implemented in hardware, since it touches every single packet in high-speed cluster networks. By carefully splitting functionality across hardware and software, Andromeda combines the benefits of both. That's easier said than done.

12.20 Snap: a Microkernel Approach to Host Networking

Snap: a Microkernel Approach to Host Networking, Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld et al, In ACM SIGOPS 27th Symposium on Operating Systems Principles, ACM, New York, NY, USA (2019), pp. 416–431 [21]

Network hardware provides hundreds of Gbps, but traditional kernel APIs impose too high an overhead to actually use this capacity. Snap implements a high-performance host networking stack outside the kernel using a microkernel-inspired architecture. This modularity enhances security by isolating components and allows deploying specialized network functions (e.g., custom protocols, hardware offloads) without modifying the core kernel.

Why we picked this paper: An excellent overview of modern host networking, and how user-space packet handling can be used at scale.

12.21 Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale

Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale, Shaohong Li, Xi Wang, Xiao Zhang, Vasileios Kontorinis, Sreekumar Kodakara, David Lo, Parthasarathy Ranganathan, 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), {USENIX} Association (2020), pp. 1241-1255 [22]

Thunderbolt enforces power caps in heavily oversubscribed large data centers. It uses workload classification and QoS feedback to apply power caps intelligently, prioritizing latency-sensitive tasks and selectively throttling background or batch jobs. This approach maximizes useful work and hides the existence of power capping from the most important users, improving data center efficiency by safely running power infrastructure near its limits

when necessary. (If you have time, you should pair your reading of this paper with the original "Power provisioning for a warehouse-sized computer" paper [23] that introduced power oversubscription at Google.)

Why we picked this paper: Oversubscription lets you have your cake and eat it too, but that requires a very carefully designed control plane. The effort is worthwhile since it leads to dramatic power utilization improvements.

12.22 Swift: Delay is Simple and Effective for Congestion Control in the Data center

Swift: Delay is Simple and Effective for Congestion Control in the Data center, Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan Wassel, et al, SIGCOMM 2020 (2020), pp. 583–598 [24]

TCP has dealt with internet congestion effectively by using packet loss to detect congestion. However, it doesn't work well in data center networks which exhibit very high speeds combined with very low latencies and small-buffer switches. Swift shows that a very simple protocol based on accurate measurements of packet delays outperforms all others. Its simple, end-host-based approach achieves extremely low network queues, high link utilization, and fast convergence.

Why we picked this paper: Swift is so simple, and its performance so good, that it completely changed people's expectations of what a WSC network can do. In fact, initially it caused some confusion: twice, bugs were incorrectly filed for monitoring failures because highly-utilized links reported zero loss.

12.23 Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild

Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild, Parthasarathy Ranganathan, Danner Stodolsky, Jeff Calow, Jeremy Dorfman et al, Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Association for Computing Machinery, New York, NY, USA (2021), pp. 600-615 [25]

Video Coding Units (VCUs) dramatically accelerate video processing, and represent the most popular WSC accelerator after AI accelerators. The authors describe the VCU hardware, software drivers, server integration, and job management systems, as well as the challenges of integrating specialized hardware into a massive, general-purpose computing environment, including fault tolerance, monitoring, and programming models.

Why we picked this paper: VCUs are a poster child for hardware-software codesign for warehouse-scale accelerators. And they won a technical Emmy award!

12.24 Jupiter Evolving: Transforming Google's Data center Network via Optical Circuit Switches and Software-Defined Networking

Jupiter Evolving: Transforming Google's Data center Network via Optical Circuit Switches and Software-Defined Networking, Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, et al, Proceedings of ACM SIGCOMM 2022, pp. 1–17 [26]

Starting from a traditional Clos topology, the Jupiter network evolved to a more flexible direct-connect architecture, with MEMS-based Optical Circuit Switches (OCS) that provide dedicated optical paths to dynamically reconfigure the network topology. WSC networks are continually growing and changing, and thus many design aspects of Jupiter support flexibility, incremental updates, sophisticated traffic engineering and dynamically reconfigured traffic flows Automated network operations support incremental capacity delivery and topology engineering without disrupting live services.

Why we picked this paper: If we had to pick one paper about Google's data center network, this is that paper. Following up from the original "Jupiter Rising" paper [27], this paper is excellent both in explaining technical details as well as retrospectives explaining how the design evolved to where it is today.

12.25 Pathways: Asynchronous Distributed Dataflow for ML

Pathways: Asynchronous Distributed Dataflow for ML, Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, et al, MLSys 2022 (2022) [28]

Traditional ML training runs a large cluster's accelerators in a highly synchronous, lockstep way that requires all accelerators to be the same. However, models have become too large to fit into a single cluster. Pathways is a new orchestration layer for accelerator systems to train and serve even larger-scale AI models. Pathways' architecture utilizes asynchronous distributed dataflow and routes computation across heterogeneous hardware accelerators (CPUs, GPUs, TPUs), in combination with a single-controller model and centralized scheduling. This approach allows moving beyond current dense, single-task models towards sparse, multi-task models that can learn many tasks simultaneously and leverage sparsity for efficiency.

Why we picked this paper: This paper is an excellent example of model-systems codesign and distributed systems innovation needed to scale deep learning.

References

1. S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, 1998.
2. L. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.
3. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the nineteenth ACM Symposium on Operating systems principles*, ser. SOSP '03, New York, NY, USA: Association for Computing Machinery, Oct. 19, 2003, pp. 29–43.
4. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," 1, vol. 51, New York, NY, USA: Association for Computing Machinery, Jan. 1, 2008, pp. 107–113.
5. F. Chang, J. Dean, S. Ghemawat, et al., "Bigtable: A distributed storage system for structured data," Article 4, vol. 26, New York, NY, USA: Association for Computing Machinery, Jun. 1, 2008, pp. 1–26.
6. M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, USA: USENIX Association, Nov. 6, 2006, pp. 335–350.
7. L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
8. S. Melnik, A. Gubarev, J. J. Long, et al., "Dremel: Interactive analysis of web-scale datasets," 1-2, vol. 3, VLDB Endowment, Sep. 1, 2010, pp. 330–339.
9. B. H. Sigelman, L. A. Barroso, M. Burrows, et al., "Dapper, a large-scale distributed systems tracing infrastructure," 2010.
10. G. Ren, E. Tune, T. Moseley, et al., "Google-Wide Profiling: A continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, 2010.
11. J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Google's globally distributed database," Article 8, vol. 31, New York, NY, USA: Association for Computing Machinery, Aug. 1, 2013, pp. 1–22.
12. S. Jain, A. Kumar, S. Mandal, et al., "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, New York, NY, USA: Association for Computing Machinery, Aug. 27, 2013, pp. 3–14.
13. J. Dean and L. A. Barroso, "The tail at scale," 2, vol. 56, New York, NY, USA: Association for Computing Machinery, Feb. 1, 2013, pp. 74–80.
14. S. Kanev, J. P. Darago, K. Hazelwood, et al., "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, New York, NY, USA: Association for Computing Machinery, Jun. 13, 2015, pp. 158–169.
15. A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2015, pp. 1–17.
16. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google runs production systems*. " O'Reilly Media.", 2016.
17. B. T. Sloss, *Introduction to Site Reliability Engineering*. " O'Reilly Media.", 2016.
18. N. P. Jouppi, C. Young, N. Patil, et al., "In-data center performance analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, New York, NY, USA: Association for Computing Machinery, Jun. 24, 2017, pp. 1–12.
19. L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.

20. M. Dalton, D. Schultz, J. Adriaens, et al., "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 373–387.
21. M. Marty, M. de Kruijf, J. Adriaens, et al., "Snap: A microkernel approach to host networking," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, New York, NY, USA: Association for Computing Machinery, Oct. 27, 2019, pp. 399–413.
22. S. Li, X. Wang, F. Kalim, et al., "Thunderbolt: Throughput-optimized, quality-of-service-aware power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI20)*, 2020, pp. 1241–1255.
23. X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07, New York, NY, USA: Association for Computing Machinery, Jun. 9, 2007, pp. 13–23.
24. G. Kumar, N. Dukkipati, K. Jang, et al., "Swift: Delay is simple and effective for congestion control in the data center," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '20, New York, NY, USA: Association for Computing Machinery, Jul. 30, 2020, pp. 514–528.
25. P. Ranganathan, D. Stodolsky, J. Calow, et al., "Warehouse-scale video acceleration: Co-design and deployment in the wild," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, New York, NY, USA: Association for Computing Machinery, Apr. 17, 2021, pp. 600–615.
26. L. Poutievski, O. Mashayekhi, J. Ong, et al., "Jupiter evolving: Transforming google's data center network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22, New York, NY, USA: Association for Computing Machinery, Aug. 22, 2022, pp. 66–85.
27. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: a decade of Clos topologies and centralized control in Google's data center network," 4, vol. 45, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
28. P. Barham, A. Chowdhery, J. Dean, et al., "Pathways: Asynchronous distributed dataflow for ML," in *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1*, 2022, D. Marculescu, Y. Chi, and C. Wu, Eds., mlsys.org, 2022.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Glossary

Annualized Failure Rate (AFR) percentage of components expected to fail per year
Air Economization cooling system using cold outside air for cooling
ASIC Application-Specific Integrated Circuit
Availability fraction of time a system is available for use
Availability Zone (Zone) location within a region, providing fault tolerance
Best-Effort (BE) workload tier with no service-level objectives
Baseboard Management Controller (BMC) server management and monitoring board
Borg Google's cluster management system and scheduler
Brownout state where a device doesn't fail but performance drops unacceptably
Busbar conductive metal bar used for power distribution within a data center
Capital Expenditure (Capex) spend on assets like data centers and servers
Cooling Distribution Unit (CDU) provides liquid cooling to racks
Clos Network multi-stage switching fabric used in data center networks
Colocation (Colo) data center facility hosting servers from multiple companies
Computer Room Air Conditioner (CRAC) a conventional cooling unit
Critical Power total power that can be continuously supplied to IT equipment
Distributed Denial of Service (DDoS) cyberattack sending simultaneous requests from many sources to exhaust service capacity
Dual In-line Memory Module (DIMM) a DRAM memory module
Disaggregation separating compute and storage for better utilization
Drive Writes Per Day (DWPD) a measure of SSD endurance
Egress outgoing network traffic
Endurance number of program/erase cycles an SSD can withstand
Energy Proportionality the principle that a system's energy consumption should scale proportionally with its utilization
Failures In Time (FIT) the number of failures in 10^9 hours
Four Nines, Five Nines 99.99% and 99.999% uptime

© The Editor(s) (if applicable) and The Author(s) 2026

L. A. Barroso et al., *The Data Center as a Computer*, Synthesis Lectures on Computer Architecture, https://doi.org/10.1007/978-3-031-99489-0

Hardware Root of Trust (RoT) secure hardware component that provides a trustworthy starting point for a system
High-Bandwidth Memory (HBM) on-chip memory used in accelerators
Ingress incoming network traffic
Infrastructure Processing Unit (IPU) smart NIC for offloading infrastructure tasks
Mean Time Between Failures (MTBF) average time elapsed between successive failures of a system
Mean Time To Repair (MTTR) average time required to restore a system after a failure
Microelectromechanical System (MEMS) tiny, micrometer-sized devices that combine both mechanical and electrical components
Optical Circuit Switch (OCS) device dynamically connecting optical fibers
Operational Expenditure (OpEx) recurring costs of running equipment
Oversubscription provisioning less capacity than the theoretical maximum demand, leveraging statistical multiplexing
Power Distribution Unit (PDU) distributes power to server racks
Power Capping limiting the maximum power consumption of a device or data center
Power Usage Effectiveness (PUE) ratio of total power to a data center divided by the power actually delivered to IT equipment
Quality of Service (QoS) prioritizing network traffic or resource allocation
Region geographically localized area containing multiple availability zones
Reliability the probability of failure-free operation over a period
Root of Trust (RoT) [See Hardware Root of Trust (RoT)]
Scale-Up Building larger, shared-memory systems
Software-Defined Networking (SDN) managing network control plane in software
Server PUE (SPUE) ratio of total server input power to its useful power
Service Level Agreement (SLA) formal agreement specifying service-level objectives and penalties for not meeting them
Service Level Indicator (SLI) metric measuring a particular aspect of service performance
Service Level Objective (SLO) target value for a service level indicator
Sharding splitting a dataset into smaller fragments (shards) distributed across machines
Single Point of Failure a component whose failure would cause the entire system to fail
Site Reliability Engineering (SRE) the practice of applying software engineering principles to IT operations
System on a Chip (SoC) an integrated circuit containing multiple components
Solid State Drive (SSD) non-volatile storage using NAND Flash memory
Tail Latency latency of the slowest requests, representing the tail of the latency distribution
Total Cost of Ownership (TCO) the total cost of an asset over its lifetime
Top of Rack (ToR) Switch network switch located at the top of a server rack
Tensor Processing Unit (TPU) Google's custom accelerator for AI/ML
VCU Video Coding Unit hardware accelerator for video processing
Verified Boot cryptographically validating firmware and software during the boot process

Vertical Scaling increasing the resources (CPU, memory) of a single machine
Voltage Regulator Module (VRM) DC-DC converter on the motherboard
Water Usage Effectiveness (WUE) metric for data center water efficiency
Zone [See Availability Zone (Zone)]

References

1. Google Cloud, *Cloud Run*, https://cloud.google.com/run
2. D. DeWitt, and J. Gray, "Parallel Database Systems: The Future of High PerformanceDatabase Systems," 6, vol. 35, New York, NY, USA: Association for Computing Machinery, Jun. 1, 1992, pp. 85–98.
3. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," 4, vol. 45, New York, NY, USA: Association for Computing Machinery, Aug. 17, 2015, pp. 183–197.
4. A. Singh, J. Ong, A. Agarwal, et al., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," 9, vol. 59, New York, NY, USA: Association for Computing Machinery, Aug. 24, 2016, pp. 88–97.
5. https://drive.google.com/file/d/186hJyFq4Seld5ys5taoxSWsztfKvEoMV/view.
6. P4.org, *P4 Language Consortium*, https://www.p4.org.
7. Net Zero data centers, *Projects*, https://www.netzerodatacenters.com/projects.
8. Uptime Institute, *2014 Data Center Industry Survey*, Uptime Institute Journal: https://journal.uptimeinstitute.com/2014-data-center-industry-survey/, 2014.
9. Google Cloud, *Google Data Center Security: 6 Layers Deep*, https://www.youtube.com/watch?v=kd33UVZhnAA, 2020.
10. Google Cloud, *Google Cloud Observability*, https://cloud.google.com/products/observability.
11. V. Kontorinis, L.E. Zhang, B. Aksanli, et al., "Managing distributed UPS energy for effective power capping in data centers," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*,2012, pp. 488–499.
12. Intel, *Intel and Micron Introduce New 3D XPoint*™ *Technology*, Webcast: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html.
13. Amazon Press Center, *AWS Makes Water Positive Commitment to Return More Water to Communities Than It Uses by 2030*, Press Release: https://press.aboutamazon.com/2022/11/aws-makes-water-positive-commitment-to-return-more-water-to-communities-than-it-uses-by-2030, 2022.
14. D. Lo, L. Cheng, R. Govindaraju, et al., "Heracles: Improving resource efficiency at scale," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 450–462.
15. Wikimedia Commons contributor, *File:Hut 8 Medicine Hat box 1.jpg*, Wikimedia Commons: https://upload.wikimedia.org/wikipedia/commons/b/bb/Hut_8_Medicine_Hat_box_1.jpg.
16. Google, *Site Reliability Engineering Books*, https://sre.google/books/.
17. https://research.google/pubs/web-search-for-a-planet-the-google-cluster-architecture/.
18. V. Sakalkar, V. Kontorinis, D. Landhuis, et al., "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 497–511.
19. A. Shehabi, S. Smith, D. Sartor, et al., *United States data center energy usage report*, https://eta.lbl.gov/publications/united-states-data-center-energy, 2016.
20. G. Team, R. Anil, S. Borgeaud, et al., *Gemini: a family of highly capable multimodal models*, https://arxiv.org/abs/2312.11805, 2023.
21. B.H. Sigelman, L.A. Barroso, M. Burrows, et al., *Dapper, a large-scale distributed systems tracing infrastructure*, https://research.google/pubs/pub36356/, 2010.
22. A. Radovanović, R. Koningstein, I. Schneider, et al., *Carbon-Aware Computing for Datacenters*, https://arxiv.org/abs/2106.11750, 2021.

23. H. Touvron, T. Lavril, G. Izacard, et al., *LLaMA: Open and efficient foundation language models*, https://arxiv.org/abs/2302.13971, 2023.
24. Open Compute Project, *Open Compute Project*, https://www.opencompute.org.
25. Samsung Semiconductor, *Ultra-Low Latency with Samsung Z-NAND SSD*, https://download.semiconductor.samsung.com/resources/brochure/Ultra-Low%20Latency%20with%20Samsung%20Z-NAND%20SSD.pdf.
26. Google Cloud, *TPUs improved carbon efficiency of AI workloads by 3x*, Google Cloud Blog: https://cloud.google.com/blog/topics/systems/100-million-li-ion-cells-in-google-data-centers, 2025.
27. P.W. Huber, and M.P. Mills, "The internet's dirty little secret," *Forbes*, 1999.
28. A.S.G. Andrae, and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.
29. P. Barham, A. Chowdhery, J. Dean, "Pathways: Asynchronous distributed dataflow for ML", in *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*, 2022.
30. A.B. Lovins, *Artificial Intelligence meets natural stupidity: Managing the risks*, https://integrative-design-for-radical-energy-efficiency.stanford.edu/sites/extreme_energy_efficiency/files/media/file/data-centersaiel-dr-16-10-may-2025.pdf, 2025.
31. I. Schneider, H. Xu, S. Benecke, et al., "An Introduction to Life-Cycle Emissions of AI Hardware," *IEEE Micro*, https://doi.ieeecomputersociety.org/10.1109/MM.2025.3592568, 2025.
32. C. Elsworth, K. Huang, D. Patterson, et al., *Measuring the environmental impact of delivering AI at Google Scale*, https://arxiv.org/abs/2508.15734, 2025.