future internet

MDPI

Article

Capacity Planning for Software Applications in Natural Disaster Scenarios

Juan Ovando-Leon [1,†], Luis Veas-Castillo [2,†], Veronica Gil-Costa [3,*,†] and Mauricio Marin [1,†]

[1] Departamento de Ingeniería Informática, Universidad de Santiago de Chile, Santiago 9170125, Chile; juan.ovando@usach.cl (J.O.-L.); mauricio.marin@usach.cl (M.M.)
[2] Instituto de Informática, Universidad Austral de Chile, Valdivia 5110566, Chile; luis.veasc@inf.uach.cl
[3] Facultad de Cs. Fisico Matematicas y Naturales, Universidad Nacional de San Luis, San Luis 5700, Argentina
* Correspondence: gvcosta@unsl.edu.ar
† These authors contributed equally to this work.

Abstract

In recent years, there has been a notable increase in the development of social media and humanitarian applications based on bots, designed to provide assistance during large-scale natural disasters. These applications play a crucial role in managing the chaos and useful to satisfy the urgent needs for rescue and relief that arise when catastrophes disrupt daily routines. However, they often encounter challenges during emergencies, such as dynamic and unpredictable variations in user workload, which can affect service quality and application stability. To tackle these challenges, we propose a capacity planning methodology to determine the optimal number of replicas and partitions for each component of an application and distributes them across virtual machines in server clusters. By bridging the gap between the algorithms executed in the applications and the performance characteristics of their implementations, this methodology enables applications to scale efficiently. It helps maintain response times and average utilization within user-defined ranges while providing fault tolerance to prevent component saturation. We validate the proposed methodology with tree bot-based applications devised to be use after a natural disaster strikes. Our experimental results show the effectiveness of the methodology, with estimation errors ranging from 1% to 15% for utilization and average response times. Furthermore, the methodology serves as an effective elasticity tool, allowing for the adjustment of component replicas based on user's requests.

Keywords: capacity planning; bot-based applications; natural disasters

Check for updates

Academic Editors: Dimitrios Dechouniotis and Ioannis Dimolitsas

Received: 8 November 2025
Revised: 19 December 2025
Accepted: 22 December 2025
Published: 1 January 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

In recent years, there has been a notable increase in the development of social media and humanitarian applications based on bots, designed to provide assistance during large-scale natural disasters. These applications play a crucial role in managing chaos and supporting urgent rescue and relief needs when catastrophes disrupt daily routines. However, despite their importance, these systems face significant challenges precisely during emergency situations, when high reliability and responsiveness are essential.

A major difficulty arises from the dynamic and unpredictable variations in user workload that occur during disasters. Sudden bursts of simultaneous requests can rapidly saturate application components, affecting service quality and system stability. These challenges are intensified by the heterogeneous and geographically distributed nature of the underlying computing resources on which these applications are deployed [1].

During emergency scenarios, it is common for different organizations—such as universities or public institutions—to voluntarily provide computing infrastructure to support these applications [2]. Unlike standard cloud environments, these resources vary widely in performance, reliability, and connectivity, making traditional capacity-planning approaches difficult to apply effectively [3,4].

Capacity planning ensures that a sufficient number of computational resources is supplied in a timely manner to handle the constantly changing flows of applications and user requirements efficiently [5–7]. However, it becomes complex when services run on container-based architectures deployed over clusters of physical and virtual machines (VMs) using technologies such as Docker and Kubernetes [8–10]. These technologies enable elasticity and fault tolerance, yet determining how many replicas or partitions of each component are needed—and how to deploy them efficiently—remains a nontrivial task. Ensuring stable response times and keeping average utilization below predefined thresholds is particularly challenging under the extreme workload fluctuations typical of disaster scenarios. Existing studies address capacity planning in various domains [11–15], but to the best of our knowledge no methodology specifically targets the heterogeneous, ad hoc, multi-cluster environments used for emergency applications [16–18]. This context involves unique constraints, including the urgent need to deploy applications quickly on diverse infrastructure assembled from multiple collaborating organizations.

These challenges highlight the need for a practical, robust, and easily applicable capacity-planning methodology that helps engineers determine the number of replicas and partitions required for each component while considering heterogeneous hardware, network latencies, resource availability, and fault-tolerance requirements. Furthermore, such a methodology should bridge the gap between application-level behavior and the performance characteristics of the underlying infrastructure, ensuring that applications can scale efficiently during disaster response.

In this study, we propose a novel capacity-planning methodology specifically designed for heterogeneous applications deployed on clusters of commodity hardware, typically available from universities and public institutions (https://www.reuna.cl/infraestructura-digital/#red-nacional, accessed on 21 December 2025). Unlike previous approaches, which focus on homogeneous cloud environments or assume stable infrastructures, our methodology introduces several distinct contributions. First, we provide a two-stage algorithm that integrates operational analysis for open systems (OAOS) and queuing theory to estimate the required number of replicas and partitions for each application component under highly variable workloads. Second, we incorporate resilience-based and multi-zone deployment policies, explicitly addressing the challenges of geographically distributed and reliability-diverse clusters—an aspect not considered in existing work. Third, our methodology produces deployment decisions that account for network latencies, heterogeneous VM-PM configurations, and historical failure behavior, enabling more accurate and robust planning in disaster-oriented environments.

We evaluate the methodology using three bot-based applications intended for use after natural disasters. Experimental results show estimation errors between 1% and 15% for utilization and average response times, demonstrating strong correlation with real execution and validating the accuracy of the approach. Additionally, we show that the methodology acts as an effective elasticity mechanism, enabling dynamic adjustment of replicas as workload intensity changes.

Beyond the specific context of federated academic clusters, the challenges addressed in this work are closely aligned with the broader cloud–edge–IoT continuum. Modern distributed infrastructures—including public clouds, edge platforms, and IoT gateways—face similar constraints related to heterogeneous hardware capabilities, multi-zone deployment

requirements, resilience strategies, and workload volatility. The analytical performance modeling and replica estimation techniques used in our methodology remain applicable across these environments, while the deployment step parallels common practices in public cloud availability zones and edge-aware orchestration. By addressing heterogeneity, resilience, and distributed resource allocation, this work contributes concepts that extend naturally to cloud–edge–IoT orchestration scenarios.

The remainder of this paper is organized as follows. In Section 2 we present related work. In Section 3 we present the proposed methodology. In Section 5 we discuss the limits of our our proposal. Section 4 presents the experimental results, and Section 6 concludes.

2. Related Work

Capacity planning is a widely utilized practice across various fields [19–23]. Some studies [24–27] evaluate cloud performance by employing a queuing model fed with CPU utilization data, representing task processing in virtual machines. Other works, such as [28,29], use convolutional neural network (CNN) and long short-term memory (LSTM) approaches. These approaches have struggled to effectively model different loading patterns. To address this challenge, the work in [29] introduces a hybrid model that combines CNN and LSTM. The authors utilize 1D-CNN for extracting local patterns and LSTM for learning temporal dependencies within these patterns and the input. They leverage the 1D-CNN feature extraction ability for learning an input-to-output mapping, which results in improved performance compared to LSTM and bidirectional LSTM (BLSTM). The authors emphasize that solely learning temporal relationships from local patterns or predicting solely from CNN-extracted features does not generalize well to new data, leading to larger prediction errors. They conclude that incorporating temporal patterns within the server load and expected loading patterns' specific characteristics enhances model learning and performance.

The work in [28] extends the work presented in [29] by proposing a model based on multi-step CPU usage, aiming to model random fluctuations and continuous and periodic new patterns from contiguous and non-contiguous CPU load values, augmented with daily and weekly time patterns. Experiments with traces from Google and Alibaba show that the proposed model can predict with low error.

The authors [30] propose a capacity planning approach for applications based on the profile of application components, application scenarios, and the desired SLA and maximum workload. These parameters are used to apply queuing theory and Mean Value Analysis (MVA), determining the number of replicas needed to support the workload. While the focus is on proposing a deployment that maintains performance and availability on a homogeneous platform, the specific implementation of availability is not specified, but rather it is indicated that availability is achieved through component distribution.

The work in [31] extends the model presented in [32] so that data engineers can calculate the impact of sustainability and cost issues related to energy and cooling systems. The authors propose availability models based on Reliability block diagram (RBD) and stochastic Petri nets to represent the data center combined with power flow model to increase energy savings. The authors show results that allow improving the availability of resources with a slight increase in cost and sustainability.

The authors in [33] propose a hybrid model to predict the short-term utilization of a server hosted in a cloud. The model decomposes the server utilization sequence into relatively stable intrinsic mode function (IMF) components and a residual component to improve prediction accuracy. Then, the efficient IMF components are selected and then reconstructed into three new components to reduce the prediction time and error accumulation due to too many IMF components.

The work in [34] proposes a model to evaluate the capacity-oriented availability (COA) of resources available in a private cloud. The model is based on RBD to represent the operational infrastructure, and also uses stochastic Petri nets for capacity-oriented availability assessment. The authors in [35] propose that data center administrators, particularly infrastructure-as-a-service (IaaS) cloud administrators, use a quota-based admission control mechanism that can reject requests for virtual machines when the infrastructure is in high demand. To avoid rejecting requests in high-demand situations, the authors propose that providers can define admission rate targets for different classes and use a method based on queuing theory models. The method limits the number of requests that can be queued to the number of CPUs available in the cloud and estimates the behavior of a quota-based admission control mechanism to find the minimum capacity required to meet service level objectives (SLOs) for the availability and admissibility of virtual machines.

The authors in [36] propose a method based on the history of requirements to estimate the management and estimation of capacities in the cloud. In particular, it uses time series to predict resource allocation demands. On the other hand, it uses the distribution of the useful life of virtual machines to estimate requests for releasing or deprovisioning resources. The experiments are performed using a virtual machine trace log obtained from IBM Smart Cloud Enterprise. This work is extended by the authors [37], to propose a measure that allows quantifying the prediction error. The proposal of this second article not only addresses the problem of capacity planning, but also analyzes the difficulties of assigning virtual machines.

The authors in [38] focus on the Software as a Service (SaaS) model of the cloud. The authors propose a capacity planning algorithm based on closed queuing networks and Mean Value Analysis (MVA). The algorithm is divided into two parts. The first part calculates a preliminary configuration that guarantees the SLA (Service Level Agreement) restrictions, and a second part is used to minimize the cost of the service. The authors in [30] present capacity planning for applications based on the profile of the application components, the application scenario, and the SLA, and the desired maximum workload. These parameters are used to apply queuing theory and Mean Value Analysis (MVA), thus determining the number of replicas necessary to support the workload. Although the focus is on proposing a deployment that maintains performance and availability, it does not specifically specify how to carry out availability but instead indicates that availability occurs through the distribution of components.

More recently, the work in [39] provides an extensive review of generation capacity planning from a reliability perspective, emphasizing the growing impact of renewable energy sources and energy storage systems. The inherent uncertainty of intermittent resources raises significant reliability concerns in power systems, motivating the development of new assessment and optimization frameworks for generation expansion planning. The paper also identifies energy storage systems as key enablers to mitigate these reliability issues.

The work in [40] presents the design of a microservices-based image classification workflow that incorporates an ensemble learning algorithm to improve prediction robustness and accuracy. The workflow is modeled using Extended Queuing Networks (EQNs) to capture detailed interactions between function stations and queues. The performance analysis identifies bottlenecks and supports a capacity planning strategy that dynamically adjusts the number of function instances to prevent overutilization and maintain stable resource utilization.

While the studies discussed above provide valuable foundations in capacity planning, performance modeling, and deployment strategies for distributed systems, most of them focus primarily on academic clusters or traditional cloud environments. However, recent advances in public and commercial cloud infrastructures—particularly those of-

fering multi-zone high availability, managed disaster-recovery services, and integrated cloud–edge–IoT orchestration—introduce additional perspectives that are highly relevant to modern emergency-response scenarios. Cloud-based Disaster Recovery (CBDR) solutions have become essential alternatives for business continuity, contrasting with the complexity and high cost of traditional methods. CBDR not only offers advantages in cost efficiency and scalability but also establishes key components such as data replication and failover mechanisms.

The authors in [41] describe the difficulties and techniques for managing capacities in cloud environments, particularly in private clouds, in order to deploy new services and maintain good performance of running services. It also shows best practices that can be taken into consideration, and some techniques and tools that can be used for this purpose. The work in [42] analyzes the critical integration of Site Reliability Engineering (SRE) strategies into cloud-based disaster recovery (DR) frameworks to enhance operational resilience and business continuity. DR in the cloud is defined as leveraging cloud services to restore data, applications, and operations quickly with minimal downtime. SRE provides a structured methodology for managing DR by embedding reliability principles, focusing on high service availability through a combination of engineering practices, automation, and proactive management. Key SRE strategies highlighted include implementing robust redundancy mechanisms, such as multi-region deployments and automated failover processes, and leveraging proactive monitoring and alerting to detect issues early. Furthermore, the paper emphasizes the use of chaos engineering principles to simulate failures and validate DR plans, ensuring systems are resilient and can recover gracefully.

Specifically within the AWS Cloud ecosystem, recent work has detailed the architecture and implementation of effective DR and Business Continuity (BC) solutions. The work in [43] explores cloud-based disaster recovery (CBDR) as an essential solution for modern business continuity, contrasting it with costly traditional methods. It details key CBDR components, such as data replication, failover mechanisms, and discusses its advantages, including cost efficiency and scalability. Furthermore, the article examines CBDR implementation using services from leading cloud providers, namely Amazon Web Services (AWS), Google Cloud, and Microsoft Azure. Finally, it addresses critical challenges such as data security and network latency, while suggesting best practices for the successful implementation of cloud-based disaster recovery strategies.

The work in [44] explores the fundamental concepts and real-world applications of implementing effective Disaster Recovery (DR) and Business Continuity (BC) solutions within the AWS Cloud ecosystem, aiming to ensure operational resilience and data integrity in an increasingly digital world. The paper analyzes AWS services, such as Amazon S3 and AWS Elastic Disaster Recovery, detailing how organizations leverage these tools to achieve specific Recovery Time Objectives (RTOs) in minutes and Recovery Point Objectives (RPOs) in seconds. The author shows successful DR/BC implementation across diverse industry sectors requires a careful balance of technical architecture, security considerations, and business requirements. Similarly, the work in [45] provides a comprehensive assessment of the costs and benefits of various AWS DR solutions to facilitate decision-making across enterprises of different sizes. The work establishes the critical trade-off among cost, recovery speed (RTO/RPO), complexity, and service availability by analyzing four primary AWS strategies: Backup and Restore, Pilot Light, Warm Standby, and Multi-Site Active/Active.

Regarding Microsoft solutions, the evaluation of Azure Site Recovery (ASR) shows its effectiveness as a cloud-based disaster recovery service. The work in [46] evaluates Azure Site Recovery (ASR), a cloud-based disaster recovery service that automates data replication, failover, and failback. Through real-world tests using Linux virtual machines

on Azure, the authors assess the ability of ASR to meet the Recovery Time Objectives (RTOs) and Recovery Point Objectives (RPOs). The results show strong performance, with RTOs under 30 s and RPOs within five minutes, and high reliability across key operations such as Finalize Protection and Unplanned Failover (over 95% success). The paper also identifies optimization opportunities in large-scale replication and Azure-native failover processes. Overall, the evaluation offers practical guidance for organizations adopting scalable, cloud-based disaster recovery solutions.

Unlike those previous studies, our work addresses a scenario that is not covered in existing capacity-planning frameworks: emergency response applications deployed on heterogeneous, geographically distributed, ad hoc clusters composed of commodity servers voluntarily provided by universities and public institutions during natural disasters. This environment is fundamentally different from commercial clouds or homogeneous academic clusters, as it exhibits (1) high heterogeneity in hardware, virtualization layers, and network latencies, (2) variable and uncertain resilience characteristics, including frequent and correlated failures, (3) multi-zone constraints determined by geography rather than cloud availability zones, and (4) extreme workload volatility caused by sudden surges in demand during emergencies. The key contribution of our work is the introduction of a capacity-planning methodology explicitly designed for this heterogeneous, multi-cluster, failure-prone context.

3. Capacity Planning Methodology

3.1. Conceptual Overview

The proposed capacity planning methodology relies on a set of interrelated models that allow emulating the environment where the applications are used. The applications are deployed on physical machines (PMs) and virtual machines (VMs).

Figure 1 shows the conceptual overview of our proposal. The left side of the figure shows the models—statistical, application, architectural and infrastructure—which constitute the conceptual backbone of the proposed methodology. The right side shows the implemented components. The bottom of the figure shows the environment component, which encompasses the entire environment used to deploy the applications and the JSON files containing the data that feeds the different models.

The statistical model captures and stores the service times associated with each task within the applications. Given the heterogeneity of the underlying infrastructure—comprising various types of physical (PM) and virtual machines (VMs)—it is necessary to maintain performance statistics for each possible PM-VM combination. The model is composed of two main components. The PerformancePerVm component stores the processing time. The TimeFunDistribution component enables access to the corresponding statistical profile based on VM and PM types. This probabilistic approach allows modeling the inherent uncertainty of computational environments and supports the accurate simulation of heterogeneous infrastructures, accounting for different resource usages such as CPU, memory, and disk throughout the execution of the methodology.

The application model provides a formal representation of each application, including its components, executed tasks, and performance parameters derived from benchmarking processes. It consists of several interdependent elements: App, which represents an application; Component, which defines its logical modules; Activity, which models each functional use case; Work, an atomic unit of computation or communication; and Data, which encapsulates both timing distributions and routing information for subsequent tasks or network requests.

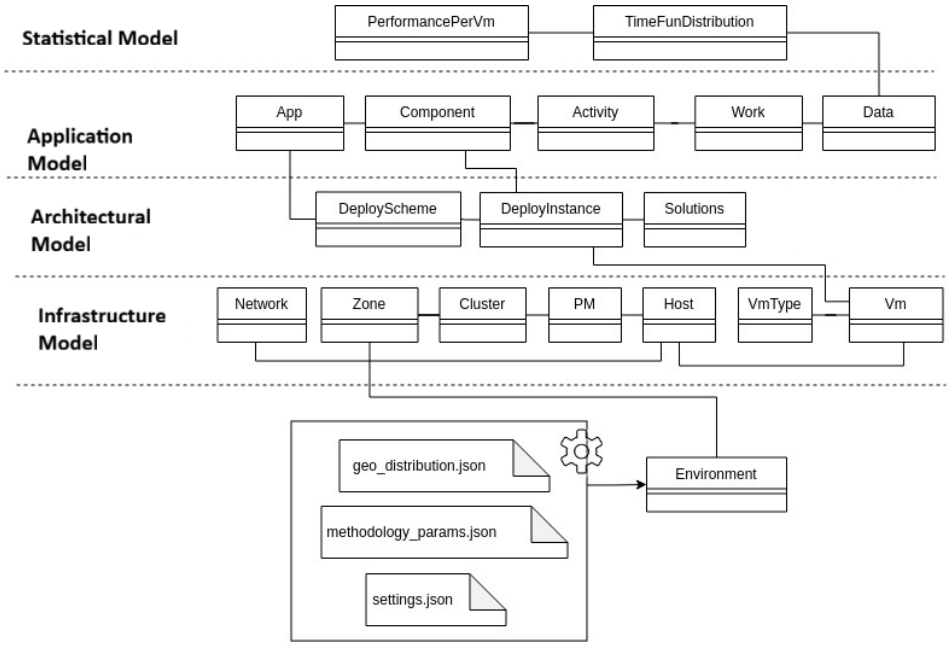


Figure 1. Overview of the proposed two-stage capacity-planning algorithm. On the left we show the conceptual models used in our proposal. For each model we show the implemented components for each model.

The architecture model defines how each application is deployed across the computational environment. Each application is associated with a DeployScheme, which maintains the list of instantiated components and the metadata of their deployment. The DeployInstance component represents each deployed instance, encapsulating information such as the component hosted, its assigned VM, and the corresponding network addresses. Additionally, the Solution component manages the different deployment configurations, storing their respective performance metrics and maintaining the currently active configuration.

The infrastructure model describes the underlying computational resources that host and interconnect the applications. It comprises several hierarchical elements: Zone, representing a geographic area containing multiple clusters; Cluster, a collection of physical machines with associated resilience parameters (e.g., Mean Time To Failure and Mean Time To Repair); and Host, which represents a server—either physical (PM) or virtual (VM). The VmType defines the specifications of each virtual machine type, including CPU cores, memory capacity, and storage size. This model supports the simulation of geographically distributed infrastructures, and enables the evaluation of performance and reliability across distinct clusters and zones, reflecting realistic deployment scenarios.

The models presented in this section provide the fundamental performance indicators—such as service demands, utilization levels, and estimated response times—that characterize the behavior of each application component under different workload conditions. However, these analytical results alone are not sufficient to produce a complete deployment plan. To transform these performance estimates into concrete decisions regarding the number of replicas, partitions, and their placement across heterogeneous clusters, the methodology employs a structured decision-making process. The next section introduces this process through a Two-Step Algorithm that builds upon the analytical outputs derived here, guiding the generation and evaluation of feasible deployment solutions aligned with resilience and multi-zone requirements.

The methodology relies on the following key assumptions to ensure reproducibility and practical applicability: (i) request arrivals to application components follow statistically characterizable patterns that can be modeled using arrival rates; (ii) service times for each

task are obtained through systematic benchmarking and represented as probability distributions specific to each VM-PM combination, capturing performance variability across heterogeneous infrastructure; (iii) component replicas operate independently, allowing horizontal scaling without inter-replica coordination overhead; (iv) network latencies between clusters are modeled based on measured bandwidth characteristics and geographical proximity, enabling accurate estimation of inter-component communication costs; and (v) fault tolerance is achieved through k-replica redundancy strategically distributed across multiple geographical zones, ensuring availability even under partial infrastructure failures. These assumptions form the basis for the analytical modeling approach and deployment policies presented in the following sections.

3.2. Capacity Planning: Two Step Algorithm

This section presents the Two-Step Algorithm that operationalizes those results into concrete capacity-planning decisions. The purpose of the algorithm is to determine the number of replicas and partitions required for each application component and to generate feasible deployment configurations across heterogeneous clusters. To achieve this, the algorithm first computes resource requirements using the analytical model, and then evaluates a set of candidate deployment solutions according to resilience constraints, multi-zone placement policies, and network-latency considerations. By integrating both analytical modeling and deployment constraints, the Two-Step Algorithm forms the core of the proposed methodology. In essence, this methodology allows us to address the following question: given a workload and a desired average utilization, what is the required number of resources to operate a set of applications without surpassing a predetermined percentage of resource utilization?

The capacity planning methodology uses a two-stage algorithm. Figure 2 shows the general outline of the algorithm which is formed by stages and steps. The name of each stage is represented by a green box, while the input parameters are represented by a yellow box.

The first stage, called the (1) Capacity Planning by Application Component, has two steps: (1) operational analysis for open systems—OAOS—and queuing theory, and (2) multizone policy. It receives the following as input:

- The infrastructure data, which consists of information about computer clusters, communication networks and information from servers or physical machines (PM), and virtual machines (VMs) with their respective characteristics.
- Geographic areas where the clusters are located.
- Application models, which represent the characteristics of the applications, their tasks, and the time distributions of each task.
- The user parameters include the level of fault tolerance, the maximum response time, the desired utilization of the available resources when executing the applications, and the multi-zone level that indicates how many different zones the application should be deployed in.

The second stage, called (2) capacity planning considering networks, receives as input the same parameters as the first stage along with the deployment solutions for each component of each application computed in the previous stage. It has four steps: 2.1 cluster ordering, 2.2 analysis of combinations of component solutions with OAOS including networks, 2.3 sorting solutions according to policies and 2.4 solution selection.

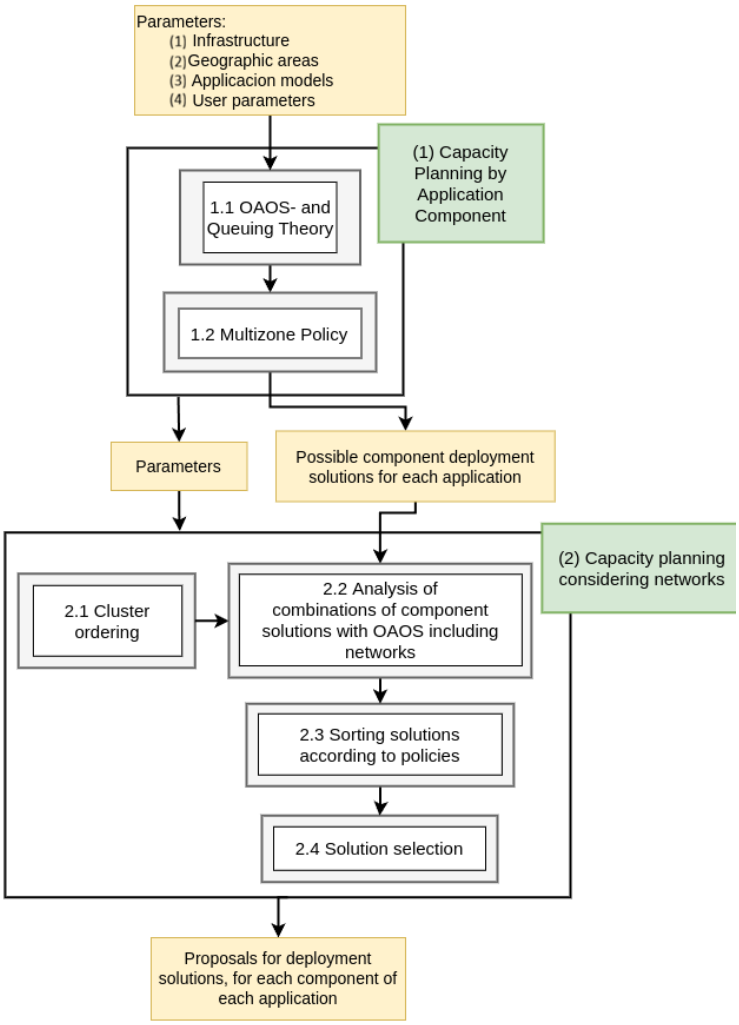


Figure 2. Proposed methodology: stages, steps, input and outputs.

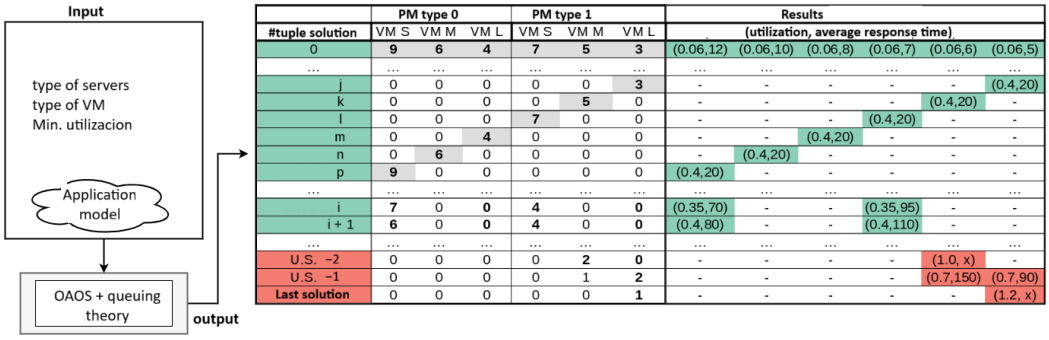
3.3. Step 1: Capacity Planning by Application Component

3.3.1. Step 1.1—OAOS and Queuing Theory

In the first step of this stage 1.1 OAOS and queuing theory, we use operational analysis for open systems (OAOS) formulae as well as queuing theory to estimate the number of virtual machines (VMs) needed to support a given workload. The solution space consists of tuples of VMs. Each tuple contains the estimated utilization and average response time for a particular VM configuration and application.

The process illustrated in Figure 3 begins with the input of the available server types—physical machines (PMs) and virtual machines (VMs)—along with their characteristics such as RAM capacity, number of cores, and processing speed. In the example, two types of physical machines (PM Type 0 and PM Type 1) and three types of virtual machines—large (VM L), medium (VM M), and small (VM S)—are considered. Additionally, this step receives the application models, which describe the number of components, tasks, and time distributions, as well as user-defined parameters such as the desired CPU utilization range and the fault-tolerance level. The output of this step is the solution space, represented as vectors combining physical and virtual machine configurations. Each vector element specifies the number of VMs to deploy, along with the estimated utilization and mean response time for each VM subset. For instance, solution tuple #0 includes different combinations of small, medium, and large VMs deployed on PM 0 and PM 1, with corresponding utilization and response time estimates (right side of the image). Gray numbers

in the table indicate the maximum number of VMs required to achieve the minimum utilization specified by the user. The algorithm iterates over all possible combinations of VMs—from the last tuple, which contains only one VM, up to tuple 0—to find the most efficient deployment configuration. Color coding is used to distinguish solution types: green tuples represent safe configurations (utilization below a user-defined threshold, e.g., 40%), while red tuples correspond to unsafe configurations. Unsafe solutions are not immediately discarded, as they may serve as alternative options if no safe deployment is feasible. As illustrated in Table 1, each solution is represented as a tuple describing the number and type of VMs to be deployed for each PM type. For every VM type, the tuple also includes a pair (utilization and mean response time), which are values estimated by the proposed methodology.



#tuple solution	PM type 0 VM S	VM M	VM L	PM type 1 VM S	VM M	VM L	Results (utilization, average response time)					
0	**9**	**6**	**4**	**7**	**5**	**3**	(0.06,12)	(0.06,10)	(0.06,8)	(0.06,7)	(0.06,6)	(0.06,5)
…	…	…	…	…	…	…	…	…	…	…	…	…
j	0	0	0	0	0	**3**	-	-	-	-	-	(0.4,20)
k	0	0	0	0	**5**	0	-	-	-	-	(0.4,20)	-
l	0	0	0	**7**	0	0	-	-	-	(0.4,20)	-	-
m	0	0	**4**	0	0	0	-	-	(0.4,20)	-	-	-
n	0	**6**	0	0	0	0	-	(0.4,20)	-	-	-	-
p	**9**	0	0	0	0	0	(0.4,20)	-	-	-	-	-
…	…	…	…	…	…	…	…	…	…	…	…	…
i	**7**	0	**0**	**4**	0	**0**	(0.35,70)	-	-	(0.35,95)	-	-
i + 1	**6**	0	**0**	**4**	0	**0**	(0.4,80)	-	-	(0.4,110)	-	-
…	…	…	…	…	…	…	…	…	…	…	…	…
U.S. −2	0	0	0	0	**2**	**0**	-	-	-	-	(1.0, x)	-
U.S. −1	0	0	0	0	1	**2**	-	-	-	-	(0.7,150)	(0.7,90)
Last solution	0	0	0	0	0	**1**	-	-	-	-	-	(1.2, x)

Figure 3. The solution space consists of physical machines (PMs) and virtual machines (VMs). Utilization and average response time are estimated for each possible solution.

Table 1. Representation of a deployment solution showing the number and types of virtual machines assigned to each physical machine, together with the estimated utilization and average response time calculated by the methodology.

Type of PM	VM	Number	Utilization	Avg. Response Time
PM1	VM S	9	0.06	12
	VM M	6	0.06	20
	VM L	4	0.06	8
PM2	VM S	7	0.06	7
	VM M	7	0.06	7
	VM L	5	0.06	5

To this end, our methodology executes four steps and uses mathematical formulas based on operational analysis and queuing theory. First, we calculate the number of replicas c necessary to obtain a given utilization U, where ST is the service time in milliseconds and λ is the arrival rate of requests to the component with using Equation (1):

$$c = ST \times \lambda / U \tag{1}$$

Second, we generate a tuple with the maximum number of heterogeneous resources that satisfy the minimum utilization. This tuple corresponds to the aggregation of the maximum values calculated for the homogeneous tuples in the first step.

In the third step, we use hyperplanes to reduce the solution space. These hyperplanes are calculated based on the minimum and maximum utilizations required by the user and eliminates solutions that are outside these limits.

In the fourth step, Equation (2) allows us to obtain the utilization of a system, given the service time ST and the workload (λ) expressed as incoming requests per millisecond to the system. c corresponds to the number of processing servers or replicas of a component, i

is the server identifier, and r is the type of request. Equation (3) corresponds to the total utilization of the system.

Equation (4) calculates the average response time (TMR) W_i for requests of type i for queues of type G/G/c. C_a is the coefficient of variation of the inter-arrival time of requests and C_s is the coefficient of variation of the service time. $C(p, c)$ corresponds to Equation (6) which expresses the Erlang-C formula [15]. The variables p and c are utilization and number of replicas (servers) respectively. $E[ST]$ is the service time value of the activity. Equation (5) is the average response time of the system, and corresponds to the sum of the W_i:

$$U_i = \frac{\lambda_i \times ST_i}{c} \tag{2}$$

$$U_{total} = \Sigma U_i \tag{3}$$

$$W_r \approx \frac{C(p,c)}{c(1-p)/E(ST)} \times \frac{{C_a}^2 + {C_s}^2}{2} \tag{4}$$

$$W = \Sigma W_r \tag{5}$$

$$C(p,c) = \frac{(c\ p)^c/c!}{(1-p)\Sigma[(c\ p)^n/n!] + (c\ p)^c/c!} \tag{6}$$

In this way, at the end of the first step of stage 1, we obtain the solution space of possible deployments for each application component. Each solution is a combination of <VMs,PM> pairs where the component can be deployed. Additionally, this step calculates the utilization and average response time for each deployment solution.

3.3.2. Step 1.2—Multizone Policy

The second step of the first stage is responsible for limiting the solution space, removing instantly those solutions that do not have a number of VMs such that it can satisfy the multizone criteria given by the user. So, for example, if the multizone criterion is 2, all those solutions that have only one VM, will be eliminated from the solution space. The solutions remaining go to the next stage.

3.4. Step 2: Capacity Planning Considering Networks

3.4.1. Step 2.1—Cluster Ordering

In the step 2.1 Cluster ordering, we use heuristic algorithms and analytical formulas to order the clusters. We apply two different ordering methods. The first one is resilience sorting which uses historical data like Mean Time To Failure (MTTF), Mean Time To Repair (MTTR), and the Mean Time Between Failures (MTBF) to deploy the application in the servers with the best resilience according to their historical data. The second ordering algorithm corresponds to an affinity ordering which selects the clusters which contain the greatest number of servers of the type required by the solution, so that allows the components of the applications to be deployed in the same cluster.

Cluster Ordering by Resilience

The purpose of this algorithm is to rank, within each geographical zone, the available computing clusters so that deployment proposals preferentially select those with better historical resilience. The inputs to this process are (i) the geographical zones, (ii) the clusters and physical machines (PMs) associated with each zone, and (iii) historical resilience records for each cluster and/or PM.

Resilience metrics can be defined at the cluster or server level and are typically expressed in hours. The methodology considers the following metrics:

- Mean Time to Failure (MTTF): average number of hours before a failure occurs.

- Mean Time to Repair (MTTR): average number of hours required to restore operation after a failure.
- Mean Time Between Failures (MTBF): computed as

$$MTBF = MTTF + MTTR.$$

- Availability: fraction of time during which the system remains operational, given by

$$Availability = \frac{MTBF}{MTBF + MTTR}.$$

To prioritize clusters, the algorithm first segments them based on the resilience data available: clusters with available metrics (MTBF, MTTR, and MTTF), and then clusters without metrics. Within each segment, clusters are ordered according to their metric values. Because clusters and servers can have different values for the same metric, a weighted value is obtained as shown in Equation (7), where the weights (w) associated with clusters and servers are given by the user:

$$\text{metric}_{\text{weighted}} = \text{metric}_{\text{cluster}} \times w_{\text{cluster}} + \text{metric}_{\text{servers}} \times w_{\text{servers}} \quad (7)$$

where the weights w_{cluster} and w_{servers} are selected by the user. Metric values represent averages; for example, MTTF corresponds to the mean MTTF of all servers in the cluster.

Finally, clusters in each zone are sorted according to the weighted metric using the following criteria:

- Availability: descending order.
- MTBF: descending order.
- MTTR: ascending order.
- MTTF: descending order.

Table 2 presents an example for a cluster and its servers, assuming equal weights (0.5). The final column reports the resulting metric used to position the cluster within the segment, in this example corresponding to the segment with available metrics.

Table 2. Example of the computation of the four resilience metrics for a server belonging to cluster X.

Metric	$ClusterX$	$Server_{Cluster\ x}$	w	Operation	Result
Availability	0.972	0.9861	(0.5, 0.5)	$0.972 \cdot 0.5 + 0.9861 \cdot 0.5$	0.97905
MTBF	720	720		$720 \cdot 0.5 + 720 \cdot 0.5$	720
MTTR	20	10		$20 \cdot 0.5 + 10 \cdot 0.5$	15
MTTF	700	710		$700 \cdot 0.5 + 710 \cdot 0.5$	705

Cluster Ordering by Affinity

The purpose of this algorithm is to rank the clusters within each geographical zone so that deployment solutions preferentially select those containing the largest number of servers of the types required by the proposed configuration. This ensures that, when choosing the first cluster to deploy the application components, at least one VM can be instantiated within the same cluster in each zone, thereby reducing the communication latency between dependent components.

The ordering depends on the server types required by each possible deployment solution. Therefore, a separate ordering is performed for every combination of deployment options associated with the applications.

The procedure first identifies the PM types demanded by a deployment solution. Then, for each cluster in a zone, it counts how many PMs of the required types are available.

Clusters are subsequently ordered by prioritizing those offering the highest number of required PMs. If two clusters provide the same number of required PMs, the one with a larger total number of PMs is selected.

For example, consider a deployment solution that requires PM types 0 and 1, and three clusters with the following compositions:

- Cluster_0: 5 PMs of type 0, 0 PMs of type 1.
- Cluster_1: 1 PM of type 0, 1 PM of type 1.
- Cluster_2: 2 PMs of type 0, 1 PM of type 1.

In this case, Cluster_2 would be ranked first because it contains both required PM types and offers a total of three PMs of these types. Cluster_1 also satisfies the type requirements but provides only two such PMs. Cluster_0 is ranked last since it includes only one of the required types.

3.4.2. Step 2.2—Analysis of Combinations of Component Solutions with OAOS Including Networks

The step 2.2 Analysis of combinations of component solutions with Operational Analysis for Open Systems (OAOS) including networks analyzes the performance of the applications including communication network costs and the impact of that communication in all possible deployment solutions.

With the input parameters and the solutions provided by the first stage of our proposal, we create 9 lists with deployment solutions. The first list meets all the requirements given by the user, meanwhile the ninth list does not meet any of them. The assignment criteria for to each list are the following:

- List 1: The solutions keep utilization and response time within the range given by the user.
- List 2: The solutions keep the utilization within the range given by the user. However, the response time is above the limit given by the user.
- List 3: The solutions present an utilization less than U_{min} (under-utilization). The response time is under the limit given by the user.
- List 4: The solutions present an utilization less than U_{min} (under-utilization). However, the response time is above the limit given by the user.
- List 5: The solutions present an utilization greater than U_{max}, but less than 100%. The response time is under the limit given by the user.
- List 6: The utilization is greater than U_{max}, but less than 100%. The response time is above the limit given by the user.
- List 7: The utilization is greater than 100%. The response time is under the limit given by the user.
- List 8: The utilization is greater than 100%. The response time is above the limit given by the user.
- List 9 to 16: They have the same characteristics as lists 1 to 8, but the solutions do not have enough VMs in a scenario of k failures. In other words, there are no servers available to deploy any of the solutions from the first eight lists. However, during a real scenario it is desirable to avoid this situation because it would seriously compromise the stability of the application.

Table 3 shows the summary of the lists and their fault tolerance criteria (T.F.), utilization, and average response time (A.R.T.). The symbol (*) indicates that the criterion is not a restriction and can use any value.

Table 3. Summary of criteria used to generate the nine lists that segment the solution space.

# List	T.F.	Utilization	A.R.T.
1	yes	MIN_U ≤ U ≤ MAX_U	yes
2	yes	MIN_U ≤ U ≤ MAX_U	No
3	yes	U < MIN_U	yes
4	yes	U < MIN_U	No
5	yes	MAX_U < U < 1	yes
6	yes	MAX_U < U < 1	No
7	yes	1 < U	yes
8	yes	1 < U	No
9	No	(*)	(*)

3.4.3. Step 2.3—Sorting Solutions According to Policies

A metric called the Efficiency and Performance Metric (EPM) is introduced to rank deployment solutions according to two main criteria: (1) the number of virtual machines (VMs) used, and (2) the computational performance of those VMs. The goal is to prioritize configurations that minimize the number of VMs while maintaining high computational efficiency.

The EPM value for a given solution is computed using Equation (8):

$$EPM = \Sigma[\#MV_i \times \frac{1}{\#CPU_{MV_i}} \times UU_i],\ i : tipo\ de\ < MV, PM > \tag{8}$$

where we have the following:

1. $\#MV_i$ is the number of VMs of type i;
2. $\frac{1}{\#CPU_{MV_i}}$ is the inverse of the number of CPUs in each VM of type i;
3. UU_i represents the unit utilization, i.e., the utilization of a single VM of type i under a given workload λ.

Our goal is to minimize EPM. The metric increases with the number of VMs (penalizing excessive use of resources), decreases when a VM type has more CPUs, and improves (lower EPM) when computational performance is higher (lower UU_i).

The next three examples illustrate how to use it:

$$< MV_0, PM_0 >:\ 6\ MV \times \frac{1}{2\ CPU} \times 0.5 = 1.5 \tag{9}$$

$$< MV_1, PM_0 >:\ 3\ MV \times \frac{1}{4\ CPU} \times 0.5 = 0.375 \tag{10}$$

$$< MV_1, PM_1 >:\ 3\ MV \times \frac{1}{4\ CPU} \times 0.2 = 0.15 \tag{11}$$

The comparison between Equations (9) and (10) shows that even when total CPU resources are equal (12 CPU), the configuration with fewer VMs (Equation (10)) is preferred due to the reduced overhead. Between the second and third cases, the difference arises from deploying on a different physical machine (PM type 1), which achieves a lower unit utilization ($UU = 0.2$) and thus a lower EPM. This indicates better computational performance on PM type 1.

Each EPM value is stored as metadata for the corresponding solution. Once all EPM values are computed, the deployment solutions are ranked within their respective lists, resulting in an ordered set of potential configurations classified by efficiency and performance.

3.4.4. Step 2.4—Solution Selection

Finally, in the fourth step, we evaluate the solutions in the lists. To this end, we select the first solution from the first list. We evaluate if feasible to deploy it. If so, we deploy the solution in the infrastructure model which represents the physical computing resources which include the zone, the network, the clusters among others. Otherwise, the next possible solution is selected.

Once the solution is deployed in the infrastructure model, we analyze whether the deployment is multi-zone. If so, the solution is chosen to deploy the application components. Then we continue analyzing the next application. Otherwise, the deployment solution is discarded and the next combination of solutions is attempted to be deployed.

The Two-Step Algorithm thus completes the methodological framework by combining analytical performance estimation with deployment decision policies tailored to heterogeneous, multi-zone environments. Having defined how replicas, partitions, and feasible deployment solutions are generated, the next sections evaluate the effectiveness of this methodology through a series of experiments involving real emergency-response applications. These evaluations are followed by a discussion of practical deployment considerations and limitations, providing a broader perspective on the applicability of the proposed approach before presenting the final conclusions.

4. Experiments

We evaluate our proposed methodology with three bot-based applications [47] devised to be used after a natural disaster strikes. The first application, Jayma (https://citiaps.usach.cl/portafolio/jayma, accessed on 21 December 2025), disseminates information regarding individuals affected during a natural disaster to their corresponding network of contacts. The second one, Ayni (https://citiaps.usach.cl/portafolio/ayni, accessed on 21 December 2025), coordinates volunteer activities by managing and assigning operational tasks. The third application, Rimay (https://citiaps.usach.cl/portafolio/rimay, accessed on 21 December 2025), is responsible for registering volunteers and organizing campaigns and emergency-related actions. These developed apps are executed on hardware typically available in public institutions—such as universities—during emergency situations. Most architectural components rely on open-source technologies, while certain functionalities depend on external services, including platforms such as Messenger or Telegram. In operational terms, the applications are deployed on a computing platform built upon container technologies, orchestration mechanisms, and virtual machines. Containers encapsulate each application and its dependencies into portable units, enabling execution independently of the underlying operating system. The orchestration layer supports transparent mobility and provides fault tolerance through the replication of stateless services and data components, ensuring continuity of operation without requiring recovery procedures after failures.

To provide a comprehensive evaluation of the proposed methodology, this section presents three complementary assessments addressing varied scenarios, quantitative validation metrics, and sensitivity analyses. Section 4.1 evaluates prediction accuracy across three bot-based applications (Jayma, Rimay, and Ayni) under four different workload intensities (600, 900, 1200, and 1500 requests per 30 s), employing root mean square error (RMSE) and Pearson correlation coefficients as quantitative metrics to measure the methodology's effectiveness at the component level. Section 4.2 conducts a sensitivity analysis by evaluating the methodology's behavior under dynamic workload variations using three arrival rates (λ_0, λ_1, λ_2) that simulate realistic disaster scenarios with unpredictable demand fluctuations, demonstrating the methodology's robustness and elasticity capabilities. Together, these

evaluations validate the methodology's accuracy, adaptability, and practical applicability across diverse operational conditions.

Figure 4 shows the general outline of the architecture of the applications, where the proxy is the access point, and we use the software Nginx [48], which is the application server. The container orchestrator is responsible for managing the execution flow and routing incoming requests to the appropriate containerized services (including data repositories such as MongoDB, Cassandra, and MariaDB). In addition, the orchestrator launches new application instances as needed—either in response to component failures or to accommodate increases in workload. For this purpose, the ImageRepo stores the container images required to instantiate any component on demand.

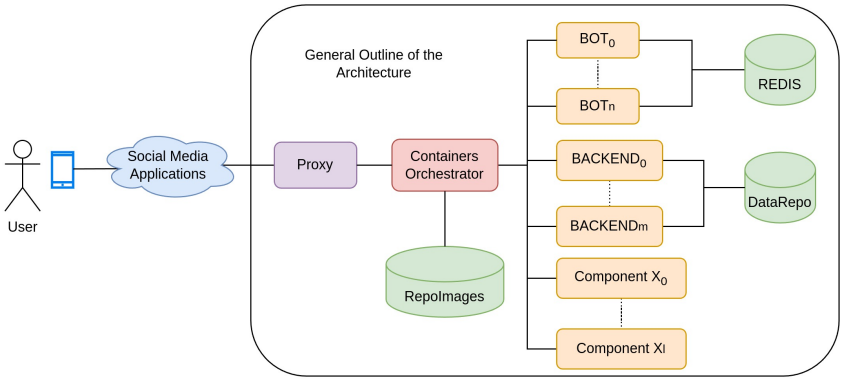


Figure 4. General design and components of the bot-based applications.

We run the experiments on two processors with 32 cores 1298 MHz, 32 GB RAM, and 1.8 TB HDD. The application components are deployed isolated on virtual machines using Dockers. A virtual machine is used to run Nginx, one for each Backend component and another for the data repository. In particular, MongoDB is used as the data repository component. Each virtual machine has 2 cores, 4 GB of RAM and 100 GB of hard disk. To detect bottlenecks and measure resource utilization, we use Jmeter 5.3, which is free software implemented in Java. The bandwidth between the processors reported by the iperf tool (https://iperf.fr/, accessed on 21 December 2025) is 17 Gbits/s.

4.1. IoT Metrics Used in Our Capacity Planning Methodology

IoT-based software systems deployed in natural disaster scenarios must be evaluated using metrics that capture both performance efficiency and service continuity under disruptive conditions. Although a wide range of IoT metrics has been proposed in the literature [49], this work focuses on performance and dependability-oriented metrics that are directly actionable for capacity planning and deployment decision-making.

Table 4 shows the metrics used in this work and the correspondence to IoT metrics. The proposed methodology relies on two primary performance metrics: CPU utilization and average response time. CPU utilization represents a standard resource consumption metric widely used in cloud and IoT-enabled infrastructures to assess saturation levels and capacity adequacy. While utilization is not always explicitly listed among high-level IoT security metrics, it is closely related to technical resource consumption indicators such as processing load, energy usage, and memory consumption commonly considered in IoT performance studies. Average response time, on the other hand, captures the quality of service perceived by end users and is conceptually aligned with latency- and round-trip-time (RTT)-based metrics frequently adopted in IoT monitoring and assessment frameworks.

Table 4. Correspondence of metrics.

Metric	Correspondence in IoT Metrics	Role in Our Methodology
CPU Utilization	Resource consumption metrics	Indicator of saturation
Average Response Time	RTT/latency metrics	Perceived quality of service
Availability	Availability (IoT/SRE)	Continuity of service
MTTF, MTTR, MTBF	Reliability and maintainability metrics	Input for resilient ranking

In addition to performance metrics, the methodology incorporates availability, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF), and Mean Time To Repair (MTTR) as historical indicators used to characterize infrastructure behavior. These metrics are well-established in both IoT and cloud engineering domains and are commonly associated with system reliability and maintainability analysis. In the proposed approach, they are used as inputs for a resilience-based cluster ranking policy that prioritizes deployment options capable of sustaining acceptable service levels during disaster conditions.

It is important to distinguish between reliability and resilience, as these terms are not interchangeable in system engineering. Reliability refers to the probability that a system operates without failure for a specified period of time under given conditions, and it is quantified using metrics such as MTTF and MTBF. Resilience, in contrast, refers to the ability of a system to absorb disturbances, adapt to adverse conditions, and recover its functionality within acceptable time and performance bounds. In this work, reliability metrics are used to support the quantification of resilience, but resilience is treated as a higher-level property that explicitly accounts for disruption and recovery, which are central aspects of natural disaster scenarios.

Finally, the accuracy of the proposed capacity planning methodology is evaluated using statistical validation metrics, namely root mean square error (RMSE) and Pearson correlation coefficient. These metrics are not intended to characterize IoT system behavior but rather to assess the fidelity of the proposed models in estimating utilization and response time under different deployment scenarios.

4.2. Effectiveness

In this section we evaluate the effectiveness of our proposed methodology. To this end, we compare the utilization level and the execution time estimated by our methodology and the ones reported by a real execution of the applications. In particular, we compute the root mean square error which is a measure of the difference between the values obtained with the proposed methodology and the values obtained in real executions for the average response time and CPU utilization metrics. It is defined as $\epsilon_m = \sqrt{(\sum(x_i - \overline{x})^2/(n \cdot (n-1))}$, where n is the sample size or amount of data analyzed. We also calculate the Pearson correlation.

In the following experiments, we show the results reported by the components of the three bot-based applications Jayma, Rimay, and Ayni. Each application uses different components like the Bot, the Database (DB), a cache and a backend. The Jayma application executes five tasks such as register, get information, report status, and alerts. The Ayni application executes 12 tasks including create emergency, create volunteer, get information, task per volunteer, etc. The Rimay application executes four tasks such as send report, create emergency, and get help. We send 600, 900, 1200, and 1500 requests every 30 s to each application. Therefore, we increase the workload of each application every 30 s.

Table 5 shows the root mean square error between actual and estimated utilization using the proposed methodology for Jayma's tasks. The error is shown for each component, both BOT and database (DB). Each column in the table shows the results obtained for different request arrival rates. Section 0 indicates a low arrival rate of 600 requests every

30 s, and Section 3 indicates a high arrival rate of 1500 requests every 30 s. The results show that in most cases, the reported error is less than 10%, with the highest error being 15.20%.

Table 5. Mean square error of utilization obtained for Jayma's tasks.

Task	Component	ϵ_m (Utilization)			
		Section 0	Section 1	Section 2	Section 3
1	BOT	9.99	6.48	4.52	3.11
	BD	2.55	2.43	2.71	4.19
2	BOT	9.74	10.31	3.32	5.47
	BD	3.27	1.32	3.53	5.60
3	BOT	3.05	3.07	8.40	13.17
	BD	2.95	2.51	3.69	9.79
4	BOT	4.83	3.98	5.46	5.14
	BD	1.78	3.12	3.88	6.31
5	BOT	2.82	5.14	8.16	15.20
	BD	1.15	3.82	5.07	10.86

Table 6 shows the root mean square error of the utilization obtained for the tasks executed in Rimay for each component. In all cases, the error is less than 8%. The largest error is reported for the DB component.

Table 6. Mean squared error of utilization for Rimay tasks.

Id Task	Component	ϵ_m (Utilization)
1	BOT	2.19
	BACKEND	0.88
	BD	5.10
2	BOT	2.09
	BACKEND	2.84
	CACHE	0.52
	BD	4.33
3	BOT	3.65
	BACKEND	3.27
	CACHE	0.26
	BD	1.92
4	BOT	1.70
	BACKEND	2.89
	CACHE	0.33
	BD	0.85
5	BOT	4.12
	BACKEND	4.65
	CACHE	1.07
	BD	8.00

Table 7 shows the root mean square error obtained for the usage reported by the proposed methodology and the data obtained by running the application on a real infrastructure. We present the results for the most relevant tasks. Notice that not all tasks access the same components. The maximum error reported is 8.4% for the BOT component.

Table 7. Mean squared error of utilization for Ayni activities.

Task	Component	ϵ_m (Utilization)
1	BACKEND	2.16
	BD	1.12
2	BACKEND	0.44
	BD	1.63
3	BACKEND	2.25
	BD	0.77
4	BOT	0.96
	BACKEND	1.89
	CACHE	0.12
	BD	1.59
5	BOT	8.40
	BACKEND	4.18
	CACHE	2.18
	BD	1.79

Table 8 summarizes the errors reported by the applications and shows the maximum and minimum root mean square error and the minimum Pearson correlation for the utilization reported by the all the components of the applications when executing the different tasks. Results show that the Pearson correlation is always higher than 90% meaning that the results reported by the methodology and the ones obtained with the real deployment of the applications are correlated (the values evolve in the same direction, that is, if one goes up, the other also goes up and vice versa). We also show that the maximum root mean square error is 15.2. This value is reported by the BOT component of the Jayma application.

Table 8. Pearson correlation, maximum and minimum root mean square error computed for the utilization of different components of the applications.

App	Component	ϵ_m (Utilization)		Pearson
		Highest	Lowest	
Jayma	BOT	15.2	2.82	96.5
	BD	10.86	1.15	96.2
Rimay	BOT	4.12	1.7	95.48
	BD	8.00	0.85	95.06
	CACHE	1.07	0.26	93.68
	BACKEND	4.65	0.88	97.11
Ayni	BOT	8.4	0.96	91.44
	BD	1.79	0.77	91.25
	CACHE	2.18	0.12	92.6
	BACKEND	4.18	0.44	92.34

Table 9 shows the maximum and minimum root mean square error and the minimum Pearson correlation for the execution reported by the all the tasks of the applications. Results also show that the Pearson correlation is high. The lowest value of 83.92% is reported by the Jayma application. However, this value is higher than 50%, meaning that the results reported by the methodology and the ones obtained with the real deployment of the applications are correlated. The maximum root mean square error of 15.94 is reported by the Jayma application.

Table 9. Pearson correlation, and maximum and minimum root mean square errors computed for the execution time of different tasks of the applications.

App	ϵ_m (Time)		Pearson
	Highest	Lowest	
Jayma	15.94	3.39	83.92
Rimay	12.17	2.51	93.92
Ayni	12.96	3.22	98.82

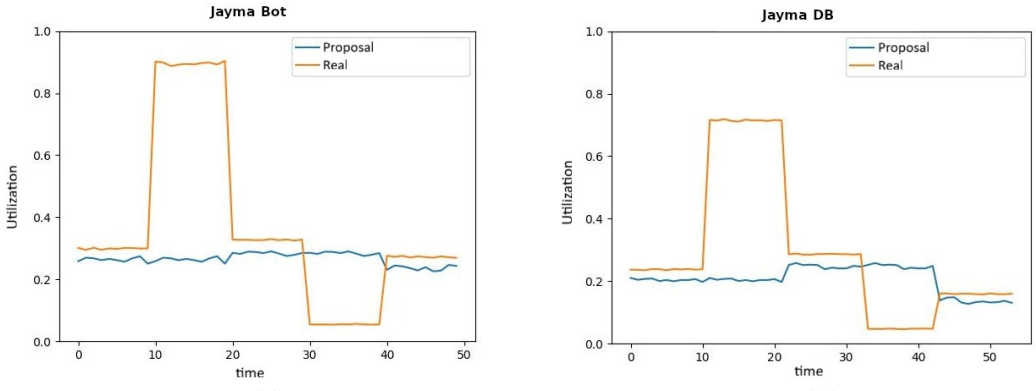
4.3. Sensitivity Analysis and Elasticity Evaluation

During large-scale natural disasters, applications need to dynamically adjust their capacity by adding or removing replicas and partitions. This ensures the application remains stable and performs well despite fluctuating request rates. To assess the elasticity capabilities of our proposal, we conduct experiments using three arrival rates: a medium rate (λ_0), a high rate (λ_0), and a low rate (λ_2). The initial application deployment is determined by our methodology to keep resource utilization below 30%.

Figures 5 and 6 illustrate the utilization level reported by the applications (orange line) deployed with the solution given by the methodology when it is feed with a medium arrival rate. In this case, the number of replicas and partitions remains fixed during the experiment. We also show the results reported by the applications when the proposed methodology is used upon arrival rate changes (blue line). That is, the number of replicas and partitions is modified according to the solutions presented by our methodology. During the first 10 s we use a low incoming rate of requests λ_0. Subsequently, in the range from $x = 10$ to $x = 20$ we increase the number of incoming requests to λ_1. From $x = 20$ onwards, the request rate drops to λ_2, causing a drastic decrease in utilization.

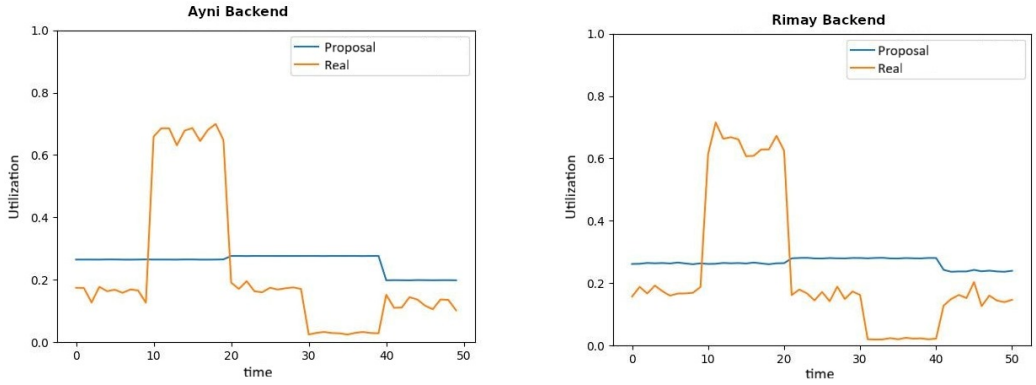
Results show that the utilization reported by the applications without the methodology is drastically impacted by the arrival rate. Under a high arrival rateλ_1, their utilization levels fluctuate between 60% and 90%. In contrast, when the proposed methodology is applied, the utilization level remains around 30%. These findings indicate that our proposal serves as an effective tool to enhance application elasticity and maintain consistent utilization levels.

While these results show the accuracy and usefulness of the proposed methodology, practical deployment introduces additional considerations that must be taken into account. These aspects are analyzed in the following section.



(**a**) (**b**)

Figure 5. (**a**) Utilization reported by the Jayma BOT with and without our proposal, under different incoming rates λ. (**b**) Utilization reported by the Database.



(**a**) (**b**)

Figure 6. (**a**) Utilization reported by the Rimay Backend with and without the proposed methodology, under different incoming rates λ. (**b**) Utilization reported by the Ayni Backend.

5. Discussion

The proposed capacity-planning methodology is designed to support the deployment of emergency-oriented applications on heterogeneous and geographically distributed infrastructures. While the experimental results demonstrate its accuracy and adaptability, several practical considerations must be acknowledged to better understand its applicability in real-world environments.

A first consideration concerns the availability and reliability of the underlying infrastructure. During natural disasters, the computing resources provided by universities or public institutions may experience intermittent connectivity, partial failures, or degraded performance. Although our methodology incorporates resilience-based policies and multi-zone deployment to mitigate these issues, its effectiveness ultimately depends on the availability of sufficient operational nodes. In environments with extremely high failure rates or severe network disruptions, the space of feasible deployment solutions may be significantly reduced.

Second, the methodology relies on accurate statistical profiles of component service times and network latencies. These parameters are obtained through benchmarking under controlled conditions, but real emergency scenarios may introduce workload patterns or communication bottlenecks that were not fully captured during data collection. Such discrepancies may affect the precision of utilization and response-time estimates. Integrating online profiling is an important direction for future work to enhance robustness during evolving disaster conditions.

Third, the approach assumes the presence of container orchestration platforms, such as Kubernetes or Docker, which enable replica management, fault tolerance, and resource isolation. While these technologies are widely adopted, not all organizations participating in emergency response may have them preconfigured or properly maintained. Practical deployment therefore requires coordination agreements, predefined configuration templates, and training for local system administrators to ensure consistent execution across institutions.

Another important practical aspect relates to the computational cost of exploring the solution space. Although hyperplanes and filtering strategies help reduce the number of candidate deployments, the complexity may increase for large-scale applications with many components or in scenarios with a vast diversity of VM-PM types. Future work may incorporate heuristic search, machine learning-based prediction, or incremental optimization to accelerate solution discovery while preserving accuracy.

In addition, the methodology currently focuses on CPU utilization and average response time as primary performance indicators. However, other resource constraints,

such as memory usage, disk I/O, or energy consumption, may also play critical roles in real deployments. Extending the model to incorporate multi-resource constraints and sustainability metrics would improve its applicability in long-lasting emergency operations.

Finally, although the methodology was validated using three bot-based applications, broader evaluation involving different classes of emergency systems—such as geospatial analysis tools, streaming-based monitoring services, or multi-agent coordination platforms—would provide deeper insights into its generality and scalability.

Generalization to Cloud, Edge, and IoT Environments

Although the proposed methodology was validated using heterogeneous clusters provided by universities through REUNA, its principles can be generalized to a broader cloud–edge–IoT network. Modern public cloud infrastructures offer features—such as availability zones, heterogeneous VM families, fault domains, autoscaling groups, and resilience policies—that align closely with the concepts modeled in our Two-Step Algorithm.

In these environments, the analytical performance estimates can be applied to different instance types, while the deployment phase can leverage cloud-native mechanisms for replica placement, multi-zone resilience, and network-aware scheduling. Similarly, edge and IoT infrastructures introduce additional constraints such as limited compute capacity, energy restrictions, and increased latency variability. These can be naturally incorporated into the methodology by extending the resource constraints and performance parameters used during solution generation.

Furthermore, next-generation 5G environments and cloud-based disaster recovery services provide opportunities to explore dynamic provisioning, mobility-aware workloads, and low-latency multi-access edge computing (MEC) scenarios. Expanding the methodology to explicitly model these environments represents a promising direction for future work, and would further enhance its applicability across modern distributed architectures used in emergency response.

6. Conclusions

This work presented a capacity planning methodology tailored for data center engineers and application designers, particularly useful in scenarios involving natural disasters. It enables the calculation of the number of replicas and partitions required for each application component, ensuring stability. Additionally, it determines the appropriate server and virtual machine for deployment. The methodology is designed for use with heterogeneous servers provided voluntarily by institutions such as universities. During peaceful periods, these servers are allocated to various tasks within each institution. However, in the event of a natural disaster, they are reassigned to support applications designed to coordinate volunteers. Furthermore, the methodology accounts for network latencies, which vary within each server cluster, as well as the interconnection network of these clusters.

The methodology comprises two stages, each with a varying number of steps. In general, we utilize a performance model based on mathematical equations, queuing theory, and operational analysis of open systems to determine the number of replicas and partitions needed to ensure that an application does not exceed a user-defined level of utilization. Subsequently, we apply resilience policies (multi-zone, cluster ordering, and solution ordering) to prioritize solutions that enable geographically distributed deployment of the application, ensuring affinity between its different components, and selecting clusters with the best historical resilience. The methodology generates a deployment plan and analyzes utilization and average response time, taking into account network latencies.

We evaluated our methodology with three applications. The results showed that the proposed approach can accurately estimate both the resource utilization and response

time of each application tasks with minimal error. Additionally, we illustrated that the methodology can dynamically adjust the number of replicas for each component of each application based on the request arrival rate.

As future work, we plan to extend our methodology with online profiling capabilities that enable continuous model updates under evolving workloads, and to incorporate additional resource constraints—such as cases where memory, storage, or network bandwidth become bottlenecks before CPU. We also intend to validate the approach with a broader set of emergency-response applications operating under diverse architectural conditions. Furthermore, we will explore its applicability within next-generation IoT and 5G environments, where mobility, edge computing, and ultra-low latency introduce new challenges and opportunities for resilient capacity planning.

Author Contributions: Software, J.O.-L. and L.V.-C.; Investigation, J.O.-L., L.V.-C., V.G.-C. and M.M.; Writing—original draft, V.G.-C. and M.M.; Writing—review and editing, V.G.-C. and M.M.; Supervision, V.G.-C. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Agencia Nacional de Investigación y Desarrollo de Chile (ANID) under grant Basal Centre CeBiB code AFB240001.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pratama, W.A.; Prasetyo, K.W. Leveraging Cloud Computing for Network Infrastructure Disaster Mitigation and Recovery (A Case Study at STIKI Malang). *J. Komputasi* **2025**, *1*, 96–105. [CrossRef]
2. Moreno, J.; Shaw, D. *Guia de Orientaciones para la Gestión de Voluntarios Espontáneos en Situaciones de Emergencia y Desastre*; Mesa Intersectorial para la Gestión de Voluntarios Espontáneos en Desastres de la Provincia de Concepción: Concepción, Chile, 2018. Available online: https://biblioteca.plataformavoluntariado.org/wp-content/uploads/2023/09/guia-voluntarios-situaciones-de-emergencia.pdf (accessed on 21 December 2025).
3. Rahmani, T.A.; Belalem, G.; Mahmoudi, S.A.; Merad-Boudia, O.R. Machine learning-driven energy-efficient load balancing for real-time heterogeneous systems. *Clust. Comput.* **2024**, *27*, 4883–4908. [CrossRef]
4. Gai, K.; Du, Z.; Qiu, M.; Zhao, H. Efficiency-aware workload optimizations of heterogeneous cloud computing for capacity planning in financial industry. In Proceedings of the 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 3–5 November 2015; pp. 1–6.
5. Zhang, J. Analysis of Dynamic Capacity Management Technology in Cloud Computing Infrastructure. *J. Comput. Signal Syst. Res.* **2025**, *2*, 76–82.
6. Feng, X.; Zhang, S.; Jiao, T.; Guo, C.; Song, J. Adaptive container auto-scaling for fluctuating workloads in cloud. *Future Gener. Comput. Syst.* **2025**, *172*, 107872. [CrossRef]
7. Pereira, C.D.C. A functional paradigm for capacity planning of cloud computing workloads. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Madrid, Spain, 25–28 May 2011; pp. 281–283.
8. Soppelsa, F.; Kaewkasi, C. *Native Docker Clustering with Swarm*; Packt Publishing Ltd.: Birmingham, UK, 2016.
9. Burns, B.; Beda, J.; Hightower, K. *Kubernetes*; Dpunkt: Heidelberg, Germany, 2018.
10. Farcic, V. *The DevOps 2.1 Toolkit: Docker Swarm*; Packt Publishing Ltd.: Birmingham, UK, 2017.
11. Arlitt, M.; Krishnamurthy, D.; Rolia, J. Characterizing the Scalability of a Large Web-Based Shopping System. *ACM Trans. Internet Technol.* **2001**, *1*, 44–69. [CrossRef]
12. Wang, H.; Sevcik, K.C. Experiments with Improved Approximate Mean Value Analysis Algorithms. In *Computer Performance Evaluation*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 280–291.
13. Aslam, A.; Anuarovna, Z.D. Real-time workload prediction and resource optimization for parallel heterogeneous high-performance computing systems architectures. *Ubiquitous Technol. J.* **2025**, *1*, 40–47. [CrossRef]
14. Kejariwal, A.; Allspaw, J. *The Art of Capacity Planning: Scaling Web Resources in the Cloud*; O'Reilly Media: Sebastopol, CA, USA, 2017.

15. Menasce, D.A.; Almeida, V.A.; Dowdy, L.W.; Dowdy, L. *Performance by Design: Computer Capacity Planning by Example*; Prentice Hall Professional: Upper Saddle River, NJ, USA, 2004.
16. Khankeh, H.; Farrokhi, M.; Saatchi, M.; Pourebrahimi, M.; Roudini, J.; Rahmatali Khazaee, A.; Jenabi Ghods, M.; Sepahvand, E.; Ranjbar, M.; Hosseinabadi-Farahani, M. Influencing factors on building and improving social trust in emergency and disaster relief efforts: A systematic review. *Disaster Resil. Built Environ.* **2025**, *16*, 129–142. [CrossRef]
17. Simmons, C. Crisis Management & Organizational Learning: How Organizations Learn from Natural Disasters. Available at SSRN 1351069. 2009. Available online: http://ssrn.com/abstract=1351069 (accessed on 21 December 2025).
18. Lin Moe, T.; Pathranarakul, P. An integrated approach to natural disaster management: Public project management and its critical success factors. *Disaster Prev. Manag. Int. J.* **2006**, *15*, 396–413. [CrossRef]
19. Simmons, C. Optimal Energy-efficient Resource Allocation and Fault Tolerance scheme for task offloading in IoT-FoG. *Comput. Netw.* **2024**, *238*, 110080.
20. Balci, M.; Kumral, M. Capacity planning in open-pit mines under economies of scale and block sequence considerations. *Miner. Econ.* **2025**, *38*, 425–443. [CrossRef]
21. Zhou, F.; Chen, L.; Zhao, J.; Wang, W. Capacity planning for integrated energy system based on reinforcement learning and multi-criteria evaluation. *Energy Syst.* **2025**, *16*, 441–470. [CrossRef]
22. Hu, B.; Yang, X.; Zhao, M. Online energy-efficient scheduling of DAG tasks on heterogeneous embedded platforms. *Syst. Archit.* **2023**, *140*, 102894. [CrossRef]
23. Jurincic, I. *Carrying Capacity Assessment of Slovene Istria for Tourism*; WIT Press: Southampton, UK, 2025.
24. Li, C.; Zheng, J.; Okamura, H.; Dohi, T. Performance Evaluation of a Cloud Datacenter Using CPU Utilization Data. *Mathematics* **2023**, *11*, 513. [CrossRef]
25. Kirsal, Y.; Ever, Y.K.; Mostarda, L.; Gemikonakli, O. Analytical modelling and performability analysis for cloud computing using queuing system. In Proceedings of the IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, 7–10 December 2015; pp. 643–647.
26. Varma, P.S.; Satyanarayana, A.; Sundari, M.V.R. Peformance analysis of cloud computing using queuing models. In Proceedings of the Cloud Computing Technologies, Applications and Management (ICCCTAM), Dubai, United Arab Emirates, 8–10 December 2012; pp. 12–15.
27. Kumar, R.; Soodan, B.S.; Kuaban, G.S.; Czekalski, P.; Sharma, S. Performance analysis of a cloud computing system using queuing model with correlated task reneging. *J. Phys. Conf. Ser.* **2021**, *2091*, 012003. [CrossRef]
28. Patel, E.; Kushwaha, D.S. An Integrated Deep Learning Prediction Approach for Efficient Modelling of Host Load Patterns in Cloud Computing. *J. Grid Comput.* **2022**, *21*, 5. [CrossRef]
29. Patel, E.; Kushwaha, D.S. A hybrid CNN-LSTM model for predicting server load in cloud computing. *J. Supercomput.* **2022**, *78*, 1–30. [CrossRef]
30. Roy, N.; Dubey, A.; Gokhale, A.; Dowdy, L. A Capacity Planning Process for Performance Assurance of Component-Based Distributed Systems. In Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering, Karlsruhe, Germany, 14–16 March 2011; pp. 259–270.
31. Callou, G.; Andrade, E.; Ferreira, J. Modeling and Analyzing Availability, Cost and Sustainability of IT Data Center Systems. In Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 6–9 October 2019; pp. 2127–2132.
32. Callou, G.; Maciel, P.; Tutsch, D.; Ferreira, J.; Araújo, J.; Souza, R. Estimating sustainability impact of high dependable data centers: A comparative study between brazilian and us energy mixes. *Computing* **2013**, *95*, 1137–1170. [CrossRef]
33. Chen, J.; Wang, Y. A hybrid method for short-term host utilization prediction in cloud computing. *J. Electr. Comput. Eng.* **2019**, *2019*, 2782349. [CrossRef]
34. Melo, C.; Matos, R.; Dantas, J.; Maciel, P. Capacity-Oriented Availability Model for Resources Estimation on Private Cloud Infrastructure. In Proceedings of the 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), Christchurch, New Zealand, 22–25 January 2017; pp. 255–260.
35. Carvalho, M.; Menascé, D.A.; Brasileiro, F. Capacity planning for IaaS cloud providers offering multiple service classes. *Future Gener. Comput. Syst.* **2017**, *77*, 97–111. [CrossRef]
36. Jiang, Y.; Perng, C.S.; Li, T.; Chang, R. Self-Adaptive Cloud Capacity Planning. In Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 73–80.
37. Jiang, Y.; Perng, C.S.; Li, T.; Chang, R.N. Cloud Analytics for Capacity Planning and Instant VM Provisioning. *IEEE Trans. Netw. Serv. Manag.* **2013**, *10*, 312–325. [CrossRef]
38. Kouki, Y.; Ledoux, T. SLA-driven capacity planning for Cloud applications. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, Taiwan, 3–6 December 2012; pp. 135–140.
39. Ghanbarzadeh, T.; Habibi, D.; Aziz, A. Addressing reliability challenges in generation capacity planning under high penetration of renewable energy resources and storage solutions: A review. *Renew. Sustain. Energy Rev.* **2025**, *212*, 115–461. [CrossRef]

40. Zafarzade, Z.; Ataie, E. Capacity planning of a microservices-based image classification application using analytic modeling. *Computing* **2025**, *107*, 123. [CrossRef]
41. Sabharwal, N.; Wali, P. *Cloud Capacity Management: Cloud Capacity Management*; Apress: Totowa, NJ, Canada, 2013.
42. Alozie, C.E.; Akerele, J.I.; Kamau, E.; Myllynen, T. Disaster recovery in cloud computing: Site reliability engineering strategies for resilience and business continuity. *Int. J. Manag. Organ. Res.* **2024**, *3*, 36–48. [CrossRef]
43. Ganesan, P. Cloud-Based Disaster Recovery: Reducing Risk and Improving Continuity. *J. Artif. Intell. Cloud Comput.* **2024**, *3*, 2–4. [CrossRef]
44. Kopparthi, V.J.R. Architecture and Implementation of Cloud-Based Disaster Recovery. *Int. J. Multidiscip. Res.* **2024**, *6*, 1–10.
45. Somi, V. Cost-Benefit Analysis of AWS Disaster Recovery Options for Enterprise Environments. *IJSAT-Int. J. Sci. Technol.* **2025**, *16*, 1–17. [CrossRef]
46. Vironica, Y.; Hertiana, S.; Karna, N. Evaluating Azure Site Recovery for Disaster Recovery. In Proceedings of the International Seminar on Intelligent Technology and Its Applications (ISITIA), Surabaya, Indonesia, 23–25 July 2025; pp. 196–200.
47. Ovando-Leon, G.; Veas-Castillo, L.; Gil-Costa, V.; Marin, M. Bot-Based Emergency Software Applications for Natural Disaster Situations. *Future Internet* **2022**, *14*, 81. [CrossRef]
48. DeJonghe, D. *Nginx CookBook*; O'Reilly Media: Sebastopol, CA, USA, 2020.
49. Ebad, S.A. Quantifying IoT security parameters: An assessment framework. *IEEE Access* **2023**, *11*, 101087–101097. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.