UNICO I+D Project
Grant No.

SORUS-RAN-A2.3-E2 (E12)

Algorithms for network optimization: final design and implementation

Abstract

This deliverable presents the design and evaluation of a set of original algorithms for cost-aware and reliable autoscaling in vRAN infrastructures. Based on queueing models and a detailed characterization of both operational and capital expenditures, an optimization framework is developed to meet strict reliability targets while minimizing total cost. Simulations using real server profiles show that the proposed solution achieves up to 22% cost savings compared to classical methods, reaching near-optimal results with low computational complexity. This work provides a solid foundation for the development of self-optimizing and energy-efficient B5G/6G networks.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia

Document properties

Document number	SORUS-RAN-A2.3-E2 (E12)
Document title	Algorithms for network optimization: final design and implementation
Document responsible	Marco Gramaglia
Document editor	Juan Manuel Montes
Editorial team	Marco Gramaglia, Pablo Serrano, José Gallego
Target dissemination level	Public
Status of the document	Final
Version	1.0
Delivery date	30-06-2025
Actual delivery date	30-06-2025

Production properties

Reviewers	María Molina

Disclaimer

This document has been produced in the context of the SORUS-RAN Project. The research leading to these results has received funding from the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D programme.

Contents

List of Figures

List of Tables

List of Acronyms

Beyond 5G: B5G

Virtualized Radio Access Networks: vRAN

Reconfigurable Intelligent Surfaces: RIS

Unmanned Aerial Vehicles: UAVs

Central Units: CUs

Distributed Units: DUs

Radio Units: RUs

Radio Network Controller: RNC

New Radio: NR

Cloud RAN: CRAN

Automated Neighbor Relation: ANR

Machine Learning: ML

Artificial Intelligence: AI

Real-Time RAN: rtran

RAN Intelligent Controller: RIC

Network Data Analytics Function: NWDAF

Network Function Virtualization: NFV

Virtual Network Functions: VNFs

Ultra Reliable Low Latency Communications: URLLC

Capital Expenditure: CapEx

Operating Expenditure: OpEx

Resumen Ejecutivo

Este entregable se basa en nuestro análisis previo del estado del arte en la asignación de recursos y el diseño de vRAN con conciencia energética, donde identificamos limitaciones clave en los enfoques actuales para lograr garantías de fiabilidad, eficiencia de costes y escalado automático dinámico en infraestructuras virtualizadas. En esta nueva fase del trabajo, pasamos del análisis al desarrollo original de algoritmos, presentando un marco de optimización específicamente diseñado para el escalado automático fiable y consciente de costes de granjas de servidores vRAN.

En el núcleo de este entregable se encuentra un conjunto de algoritmos que hemos diseñado desde cero, combinando modelos teóricos de colas con una abstracción detallada de costes que integra tanto los gastos de capital (CapEx) como los gastos operativos (OpEx). Estos algoritmos están diseñados para cumplir con exigentes objetivos de fiabilidad —como los impuestos por acuerdos de nivel de servicio— al mismo tiempo que minimizan el coste total de propiedad. Nuestro enfoque va más allá de los métodos heurísticos convencionales o las adaptaciones de técnicas clásicas, proponiendo nuevas formulaciones y estrategias de resolución específicamente orientadas a las particularidades de los entornos virtualizados de acceso radio.

Para validar nuestros algoritmos, llevamos a cabo extensas campañas de simulación utilizando perfiles reales de servidores y patrones de demanda realistas. Los resultados muestran que nuestra solución logra hasta un 22 % de reducción de costes en comparación con técnicas clásicas de escalado automático basadas en teoría de colas. Además, nuestros algoritmos alcanzan un rendimiento a menos del 3 % de las soluciones obtenidas por búsqueda exhaustiva, pero con una complejidad computacional muy inferior, demostrando así un sólido equilibrio entre optimalidad y escalabilidad.

Mediante la introducción de estos nuevos algoritmos y su validación bajo condiciones realistas, este entregable establece una metodología sistemática para el diseño de infraestructuras vRAN energéticamente eficientes, fiables y rentables. Sienta las bases para futuras líneas de investigación, incluyendo la integración de planificación de recursos basada en movilidad y predicción de tráfico, así como mecanismos de reconfiguración dinámica de red. En última instancia, nuestro trabajo contribuye al objetivo más amplio de habilitar arquitecturas de red B5G/6G inteligentes y autooptimizables, capaces de adaptarse dinámicamente a la evolución de la demanda de los usuarios y las condiciones operativas.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Executive Summary

This deliverable builds upon our previous in-depth analysis of the state of the art in resource allocation and energy-aware vRAN design, where we identified key limitations in current approaches to achieving reliability guarantees, cost efficiency, and dynamic autoscaling within virtualized infrastructures. In this new phase of the work, we shift from analysis to original algorithmic development, presenting a novel optimization framework specifically tailored for cost-aware and reliable autoscaling of vRAN server farms.

At the core of this deliverable is a set of algorithms we have designed from the ground up, combining queuing-theoretic modeling with a fine-grained cost abstraction that integrates both capital expenditures (CapEx) and operational expenditures (OpEx). These algorithms are engineered to meet stringent reliability targets—such as those imposed by service-level agreements—while minimizing the total cost of ownership. Our approach goes beyond conventional heuristics or adaptations of classical methods by proposing new formulations and solution strategies that are purpose-built for the unique characteristics of virtualized radio access environments.

To evaluate our algorithms, we carry out extensive simulation campaigns using real-world server profiles and demand patterns. Results show that our solution achieves up to 22% cost reductions compared to baseline queueing-based autoscaling techniques. Importantly, our algorithms yield performance within 3% of exhaustive search solutions, while operating at a fraction of their computational complexity—demonstrating a strong balance between optimality and scalability.

By introducing these new algorithms and validating them under realistic conditions, this deliverable establishes a systematic methodology for the design of energy-efficient, cost-effective, and reliable vRAN server infrastructures. It lays the groundwork for future research directions, including the integration of mobility-aware and traffic-predictive resource planning, as well as real-time network reconfiguration mechanisms. Ultimately, our work contributes to the broader goal of enabling intelligent, self-optimizing B5G/6G network architectures that can adapt dynamically to evolving user demands and operational conditions.

1. Introduction

With the arrival of Network Function Virtualization (NFV) [1], mobile services will be implemented as interconnected virtual network functions (VNFs) hosted by cloud servers [2]. To make efficient use of resources, these VNFs need to be scaled up and down based on their load [3]. This scaling does not impose significant challenges when dealing with traditional best-effort services, and therefore the impact of non-zero activation times or the fallibility of servers is negligible. However, when dealing with services with stringent reliability requirements—such as Ultra Reliable Low Latency Communications (URLLC), which demand reliability levels of up to 5 or 6 nines [4]—these factors become critical and cannot be ignored.

This deliverable builds upon our previous work, where we analyzed the state of the art in resource allocation and energy-aware vRAN operation, identifying key gaps in the integration of reliability-aware autoscaling mechanisms and cost-efficient infrastructure design within virtualized environments. In that study, we explored how intelligent resource management strategies, grounded in queuing theory and energy-efficiency considerations, could be leveraged to meet the demanding requirements of B5G networks. This foundational analysis established the need for algorithmic approaches capable of jointly optimizing server activation policies, energy consumption, and cost models while maintaining strict service reliability guarantees.

In our earlier contributions [5], [6], we specifically examined the non-negligible impact of non-zero server start-up times and finite server lifetimes on the reliability of services provided by auto-scaling server farms. Initially, we considered a fixed scenario with a given server farm and developed an analytical model to optimize its activation and deactivation thresholds [5]. Subsequently, we proposed a configuration mechanism that dynamically adjusts these thresholds to meet specific reliability targets under varying load conditions [6]. These works demonstrated that, depending on traffic patterns and service requirements, deploying a few carrier-grade servers (powerful, highly reliable, but energy-intensive) may be less efficient than operating many consumer-grade servers (less powerful, but more energy-efficient).

Motivated by these findings and grounded in the insights from the previous deliverable, this work advances from analysis to design optimization. Here, we tackle the problem of determining the optimal deployment configuration for a server farm within a vRAN context. Given (i) a specific service characterized by a target reliability guarantee and known arrival and service rates, and (ii) a set of candidate server types with defined characteristics, we aim to compute the optimal mix (number and type of servers) that minimizes total cost while meeting reliability requirements. To evaluate cost, we adopt a general model incorporating both capital expenditure (CapEx) and operating expenditure (OpEx), though the framework can also be extended to cloud-based service provisioning scenarios requiring resource pre-booking with diverse cost-performance trade-offs.

We validate our proposal through extensive simulations, modeling five different server types representative of real-world equipment. Our results show that the proposed method reduces costs by 22% compared to classical queueing-based approaches, achieving solutions within 3% of

exhaustive numerical searches at only 10% of their computational complexity. Compared to prior works on NFV scaling under reliability constraints (reviewed in Section 2), this deliverable introduces the following key contributions:

- Development of an analytical model to estimate the resource requirements of a server farm needed to support a given service with defined reliability guarantees.
- Introduction of a cost model combining CapEx and OpEx terms, linked with the analytical model to estimate server utilization and associated costs.
- Formalization of the optimal server farm design problem, selecting both the type and quantity of servers that minimize cost while meeting reliability requirements.
- Extension of the design methodology to heterogeneous multi-service scenarios, accommodating services with diverse reliability needs.
- Validation of both the analytical model and the design algorithm via extensive simulations based on real-world server profiles and operational conditions.

This deliverable therefore transitions from the theoretical SOTA analysis performed earlier to a concrete algorithmic approach for cost-aware, reliability-driven autoscaling of vRAN server farms, setting the stage for subsequent integration with mobility-driven resource planning and dynamic network reconfiguration strategies.

2. Related work

2.1. Resource Allocation for NFV

The research community has shown significant interest in dynamically managing cloud resources to minimize consumption. For example, in [26], the authors analyze the impact of various static algorithms for activating and deactivating resources, as well as reallocating tasks within a data center, with a focus on reducing energy consumption and minimizing service violations. In a subsequent study [27], they suggest adapting thresholds based on estimated conditions. Finally, control theory has historically been leveraged for energy-efficient resource allocation in cloud computing systems, as outlined in [28]. For example, [29] applies control theory principles for load balancing and CPU frequency selection, which differ from our previous work [6] where control theory is applied to drive the system to the desired reliability levels while minimizing energy consumption. However, the techniques proposed therein tackle distinct challenges compared to those addressed in this deliverable, and none specifically account for both the waiting queue time of the tasks and the fact that servers has non-zero boot up times.

2.2. Analysis of reliability in NFV

The study by [30] meticulously examines the reliability of a carrier-grade server system, employing a fault tree model at a high level that intricately links various lowerlevel Markov models. These models account for the inherent fallibility of hardware components such as CPUs and memory modules. A similar methodology is pursued in [31], where the authors examine the reliability of both virtualized and non-virtualized systems comprising two hosts. Furthermore, [32] delves into a related system, conducting a sensitivity analysis to pinpoint parameters that significantly impact reliability. A closer examination akin to our research is presented in [33], where a Markov chain is used to model a server farm, factoring in setup delays concerning response time and power consumption. Similarly, [3] explores the analysis within the realm of 5G/6G networks, using thresholds to manage instance power and performance evaluation in terms of power consumption and waiting time. However, none of these works have proposed a theoretical model to asses the design of an optimal server farm, targeting a desired level of reliability while minimizing infrastructure costs.

2.3. NFV and Reliability

In prior works [5], [6], [24], we have addressed a system similar to the one analyzed in this deliverable. In [5], we characterized service reliability and derived an optimal configuration of the server farm to support a required reliability while minimizing the resource consumption. In [24], we studied the trade-offs of a server farms in terms of reliability and power consumption based on a static configuration. Our analyses of [5], [24] are static and require knowledge about the system load, while other proposal rely on stochastic optimization [11] to find the best trade-off between resource consumption and average waiting time. In [6], we introduced a control theory algorithm that

dynamically adapted the configuration to reach the desired point of operation. However, all these papers assumed a fixed server farm where machines are characterized in terms of a number of parameters (e.g. energy consumption, capacity, lifetime). In contrast to this analysis and configuration problems, in this deliverable we address the design problem: given a given a set of parameters defining a service and a list of candidate server types, that could be used to deploy the server farm, select the most adequate server type to support the service.

2.4. Motivation

The growing reliance on virtualized infrastructures to support critical services in next-generation mobile networks introduces significant challenges, particularly in ensuring the reliability of auto-scaling server farms [7], [8]. Unlike traditional best-effort services, where minor delays in resource activation or occasional hardware failures have little impact, mission-critical applications demand stringent reliability guarantees. Services such as industrial automation, autonomous driving, and remote healthcare require failure probabilities as low as one in a million (99.9999%), making even brief disruptions unacceptable. These requirements impose not only an operational challenge but also a design challenge: the number of active resources must be dynamically adapted to the observed traffic, and the type and number of servers must be selected to meet reliability goals without incurring excessive cost.

Regarding the operational challenges, since traffic loads fluctuate dynamically, adaptive resource allocation is needed to prevent service interruptions while minimizing Operational Expenditures (OpEx). Some solutions rely on auto-scaling techniques to adapt to the traffic load [9], [10] but often assume instantaneous activation of pshysical machines (PMs), which is unrealistic in practical deployments, while other solutions rely on analytical tools while taking into account the non-zero boot up times and fallability of servers [5], [11] (we review the related work in Section 2). Since these works do not tackle the design challenge, one possible strategy would be over-provisioning, where additional servers are deployed to ensure redundancy. While this approach improves reliability, it may result in significant Capital Expenditures (CapEx).

In this deliverable, we develop a cost-aware optimization framework that selects the optimal type and number of servers, minimizing both CapEx and OpEx while meeting stringent reliability constraints. We focus on the worst-case scenario, assuming that the peak traffic load corresponds to the average traffic load as this imposes the most demanding operational constraints, ensuring that our methodology remains applicable to other scenarios.

3. System Model

We assume the same system as in [5], [6], i.e., an autoscaling server farm for NFV, comprising a centralized infrastructure manager (IM) and several physical machines (PMs). Initially, we assume a homogeneous server deployment (i.e., server homogeneity), meaning that all PMs are modeled after the same type of server. This assumption simplifies the operational procedures, as maintaining a uniform hardware profile reduces management complexity and operational costs, making heterogeneous deployments more costly [12]. We describe how our framework can be adapted to heterogeneous scenarios in Section 5.5.

In the current deliverable, we refer with "task" to a traffic session, which refers to an instance of a stringent URLLC service such as remote-assisted driving, an industrial slice in a factory environment, or tele-operated driving. Assuming independence among traffic sessions, following the Palm–Khintchine theorem the aggregate arrival process asymptotically behaves like a Poisson process.

Following the above, a deployment can be modeled with a set of parameters that characterizes its performance: each PM can support a maximum of M tasks, which we refer to as server capacity, and has an exponential lifetime[1] and boot up times, with average $1/\beta$ and $1/\alpha$, respectively (the key variables in this deliverbale are summarized in Table 1). Following [14], [15], we assume a load-proportional power consumption model for the PMs, characterized by a fixed term P_{idle}, and the proportional term P_{load}, which can be computed as the difference in tasks between the so-called peak power consumption P_{peak} and the idle term, i.e.,

$$P_{load} = \frac{P_{peak} - P_{idle}}{M}$$

Each server has a monetary cost K and a lifespan R. We denote with τ_i the set of parameters that characterizes a given server type i, i.e., $\tau_i = \{M_i, \beta_i, \alpha_i, P_{idle,\,i}, P_{load,\,i}, K, R\}$, and with $T = \{\tau_1, \tau_2, \dots\}$ the set of candidate server types.

Tasks arrive to the system following a Poisson process (note that this assumption is relaxed in our performance evaluation) at a rate λ and require an exponential service time of average $1/\mu$. During the initial planning phase, we rely on parameter estimates to dimension the system as optimally as possible, while at runtime, we employ adaptive mechanisms (such as stochastic optimization [11]) to guide operational decisions, including determining the number of active servers. Moreover, system measurements collected during operation can be used to iteratively refine parameter estimates, thereby improving system dimensioning over time.

The IM balances the load across PMs, seamlessly migrates tasks whenever needed (e.g., a machine is about to fail), and powers PMs on and off as needed. Each PM can be in one of three states: active (serving tasks), booting up (initiating due to a need for more resources), or stopped (either due to a

[1] Additional experiments (not reported here due to space constraints) indicate that our framework also applies to other scenarios. These include: (i) correlated failures, modeled as a twostate Markov chain with distinct average lifetimes in each state [13]; and (ii) non-exponential distributions for boot-up times and lifetimes. Specifically, we tested constant and Weibull-distributed boot-up times, and Weibull-distributed lifetimes across multiple server classes, using parameters chosen to match the same mean values as in the exponential case.

crash or because they are not needed to handle the current traffic load). To power on/off the PMs, the IM implements the following policy. At least one server is kept active at all times to avoid any delay to serve in arriving tasks. For the rest of the servers, a threshold-based policy is followed, where the thresholds to power on or off a server depend on the number of active servers, denoted by m: a new server is activated when the number of tasks in the system reaches s_m, and deactivated when the number of tasks reaches $s_m - 1$. The motivation behind this lack of hysteresis is motivated by the objective of achieving the largest possible energy savings, which is accomplished by keeping the absolute minimum number of servers active at any given moment to meet performance guarantees [5]. These energy savings are gained at the expense of a higher frequency of server state transitions, which might lead to some hardware wear-and-tear.

The farm serves tasks with high reliability requirements, e.g., an industry 4.0 service [4] or autonomous driving services [16]. When a task arrives, the IM selects an active PM with sufficient resources to handle it. If no PMs are available to handle the task, the IM queues it until either a task finishes service or a PM boots up. In any case, this results in a service disruption, which negatively affects the committed reliability of the task. Since PMs are prone to failure, an active PM may crash at any time. If it was processing tasks, we assume that these can be seamlessly (i.e., with negligible latency and energy overheads) migrated to another active PM, where sufficient resources are available –note that this is already supported in Linux environments [17] with open-source technologies such as ACHO [18].[2] In case there are not enough active PMs, the affected tasks are placed on hold until sufficient PMs are activated again, which again is considered a service disruption.

We assume that the successful provisioning of the service requires a minimum reliability level. This reliability level is determined by the probability of a task being disrupted, which is denoted as P_f, being below a maximum threshold, denoted as T_f. Although the terms "reliability" and "failure probability" refer to complementary terms, e.g., a reliability of 3 nines (99,9%) corresponds to a failure probability of 10^{-3}, for readability reasons we will use them interchangeably throughout the deliverable.

Variable	Description
N	Maximum number of physical servers
M	Capacity of one server
λ	Task arrival rate
$1/\mu$	Average service time of a task
$1/\beta$	Average lifetime of a server
$1/\alpha$	Average boot time of a server

[2] For instance, assuming a failover mechanism over a reliable wired network, as in [18], and a memory transfer of 64 MB over a 1 Gbps link, each migration requires approx. 500 ms, which is significantly shorter than typical session durations. Based on our simulation results and the power model in [19], this results in an additional power consumption of approx. 0.18 W, which is negligible compared to the idle consumption of a server.

s_m	Activation threshold with $_m$ active servers
C	Total cost
C_{Cap}	Cost associated to CapEx
C_{Op}	Cost associated to OpEx
K	Equipment cost
R	Equipment lifespan
N_u	Average number of users in the system
N_a	Average number of active servers
P_{load}	Energy consumption due to resource usage
P_{idle}	Energy consumption due to active server count
P_f	Failure probability of the server farm
T_f	Target failure probability

TABLE 1 KEY VARIABLES USED THROUGHOUT THE DELIVERABLE

To guarantee the required reliability, we assume that the server farm executes the algorithm presented in [6], which automatically drives the (de)activation thresholds $\{s_m\}$ to an adequate point of operation (note that in [6] we illustrate that the algorithm provides the most energy saving operation, but we did not provide the actual values of these parameters). In this Deliverable, we address the optimal design of the server farm, i.e., given the set of available candidate server types T , determine the most appropriate server type, which is denoted by τ^*, and the required number of servers, denoted by N, that guarantees the required performance while minimizing the cost.

4. Analysis and design of the server farm

In this section, we formalize the optimization problem that we address in this Deliverable, which is the choice of the optimal server type and the number of servers to support a given service while minimizing the cost. To this aim, we first introduce our cost model, and then an analytical model to characterize the operation of the server farm, which is used by the algorithm to select the best server type.

4.1. Cost model

Following the usual assumptions in the literature (e.g., [20]), we assume that the total cost of the server farm composed of PMs of type τ is given by the sum of the Capital Expenditure (CapEx) and Operating Expenditure (OpEx).

$$C(\tau) = C_{Cap}(\tau) + C_{Op}(\tau), \quad (2)$$

where the CapEx term C_{Cap} is determined by initial investments in infrastructure, while the OpEx term C_{Op} is determined by the cost of running and maintaining the service. More specifically, C_{Cap} is determined by the number of servers required to provide a service N, their cost K, and the equipment lifespan in hours R, according to:

$$C_{Cap} = N \times \frac{K}{R}, \quad (3)$$

where R is the average lifespan for the type of server considered, considering the expected load and wear-and-tear effects; the OpEx term is determined by the resource consumption due to the service provisioning. Following our previous work [6], this is given by the number of resources required to process the tasks, and therefore the OpEx term can be expressed as

$$C_{Op} = (P_{load} N_u + P_{idle} N_a) \times kWh, \quad (4)$$

the average number of users in the server farm, N_a denotes the average number of active servers, and kWh represents the monetary cost of energy per hour.[3] Note that we consider that the server farm is working continuously in a time span of a year.

Both (3) and (4) depend on several parameters: a subset of them corresponds to numerical figures that are determined for a given server type τ_i, i.e., K, R, P_{load}, and P_{idle}, while the rest of them depend on the operation conditions, i.e., the average number of users N_u, the total number of servers N, and the average number of active servers N_a. We next present an analytical model to compute these.

[3] For simplicity, our cost model assumes a fixed average energy price. This could be extended to incorporate dynamic pricing (e.g., time-of-day tariffs) by adjusting for load and cost variations across time periods.

4.2. Average number of users N_u and user distribution

To compute the average number of tasks and their distribution, we make the approximation that incoming tasks never wait to be attended. This approximation is motivated by the fact that the number of tasks that will have to wait is very small and hence this approximation will have a very small impact on the resulting task distribution. Assuming that the impact of server failures is negligible, the system behaves as an *M/M/∞* system [21], and therefore the probability of having *n* users in the system, denoted by p_n, follows a Poisson distribution given by the following expression

$$p_n = \left(\frac{\lambda}{\mu}\right)^n \frac{e^{-\lambda/\mu}}{n!}, \quad (5)$$

Where the average number of users is

$$N_u = \frac{\lambda}{\mu}, \quad (6)$$

4.3. Total number of servers N

To compute the required number of servers, we look at the number of servers that are needed to be able to serve all tasks with a very high probability (provided that all servers are active). In particular, we enforce that the probability that an arriving task does not have to wait, when all servers are active, is well above 1–T_f, i.e., the failure due to all servers being busy is much smaller than the failure due to other reasons, which can be as large as T_f. More specifically, we enforce that the probability that a task does not have to wait is equal to $s = 1 - T_f/X$, where X is a sufficiently large value (unless otherwise stated, in the rest of the Deliverable we take $X = 10$).

Note that following (5) the *s*-percentile of the total number of users in the system, denoted as $N_u(s)$, is given by

$$N_u(s) = \min\left\{Q \in \mathbb{N} \middle| \sum_{n=0}^{Q} \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n e^{-\frac{\lambda}{\mu}} \geq s\right\}, \quad (7)$$

To meet the requirements stated above, we need to ensure that the system can host up to $N_u(s)$. This means that, if the size of a server in tasks is given by M, the number of servers in the system needs to be dimensioned as follows:

$$N = \frac{N_u(s)}{M}, \quad (8)$$

4.4. Average Number of active servers N_a

Given the user distribution probability $\{p_n\}$ provided by (5), and the set of activation thresholds $\{s_m\}$, which is computed below, the average number of active servers N_a is given by

$$N_a = 1 + \sum_{m=2}^{N} m \sum_{n=s_m}^{s_{m+1}-1} p_n, \quad (9)$$

where the first term accounts for the fact that there is always at least one active server active, and the second term computes the weighted average of a number of servers m and the probability of having that number of servers active (there are m active servers if the number of users is between s_m and $s_{m+1} - 1$).

4.5. Computation of the activation thresholds

As described in Section 3, the activation thresholds $\{s_m\}$ are automatically configured using a control theoretical mechanism [6] that guarantees that the failure probability P_f is below the target T_f while minimizing the energy consumption. In this section, we provide a theoretical analysis to estimate the value of these thresholds.

To perform the analysis, again we neglect the impact of the finite server lifetime on failures, i.e., we assume that all service failures are due users arriving to the system with not enough resources to immediately start service. This assumption reflects the server farm's operation, where the auto-scaling scheme enables re-routing tasks from a failing server to other available resources; our simulations confirm such direct failures have a negligible impact on overall system reliability. We consider a scenario with m active servers, and assume that the activation threshold for an additional server is set to k. Following the above, a failure will occur right after the system has k users (and triggers the activation of a new server) if the total number of users exceeds the current total capacity of the farm ($m \times M$) before another server has been fully activated, which requires a time T_{on}. We denote this conditional probability upon arrival as $P_f(k, T_{on})$, which corresponds to the probability of reaching a state with more than $m \times M$ users in less than T_{on}, starting from a state with k users. By denoting with $P_{i,j}(t)$ the probability of reaching state j from i in less than t, the probability $P_f(k, T_{on})$ can be computed as

$$P_f(k, T_{on}) = \sum_{j=mM+1}^{NM} P_{k,j}(T_{on}), \quad (10)$$

We exemplify the above formulation with the toy example depicted in Fig. 1. The figure illustrates a scenario with $m = 2$ active servers, where each server has a capacity $M = 5$. Assuming that the activation threshold for the third server is $k = 9$ and a total of $N = 10$ servers (i.e., a maximum of 50 simultaneous tasks), the conditional probability of a failure when the system is in state $k = 9$ is given by

$$P_f(9, T_{on}) = \sum_{j=11}^{50} P_{9,j}(T_{on}), \quad (11)$$

To compute $P_{i,j}(t)$, we assume that the transition matrix of the system $\boldsymbol{Q}$ is the same as the one from a classical *M/M/c* queue [22] with $c = N$.

Based on this, assuming an initial probability distribution vector $\boldsymbol{P}(0)$, the probability distribution vector after t is given by [23]

$$\boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{P}(\boldsymbol{0})e^{tQ}, \quad (12)$$

and therefore $P_{i,j}(t)$ can be computed by substituting $\boldsymbol{P}(0)$ with a distribution vector with 1 in the i-th position.

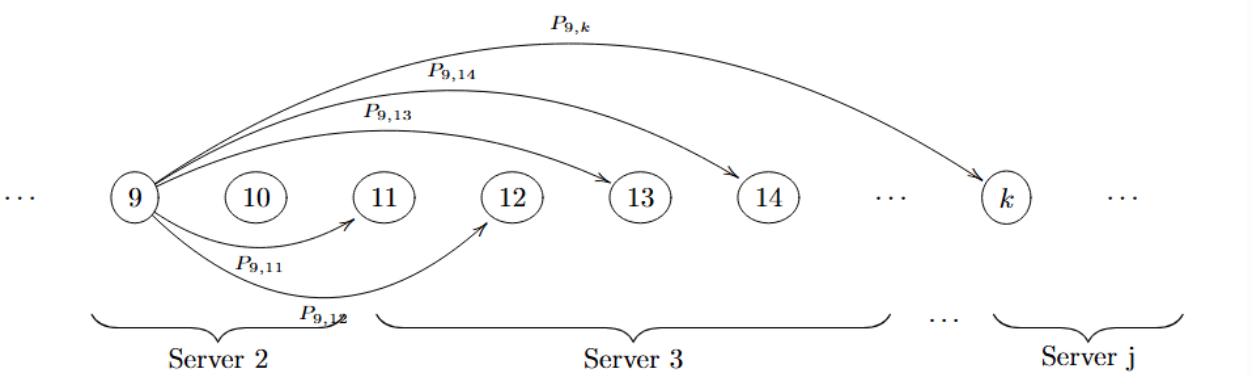

FIGURE 1FAILURE PROBABILITY WITH ACTIVATION THRESHOLD K = 9 FOR M = 2 SERVERS WITH CAPACITY M = 5.

To determine the activation thresholds, we assume that the impact of each threshold is independent of the others. We also assume that all tasks that arrive while server *m* is booting are assumed to have arrived when the number of users in the system was exactly equal to its activation threshold k (rather than during states with fewer users). Under these conditions, Eq. (10) can be used to compute the failure probability associated with server m when threshold k is applied. Under these assumptions, the actual failure probability P_f is upper-bounded by the conditional failure probabilities associated with the activation of each server. As a result, if we ensure that each of these conditional probabilities remains below T_f, so will be the actual failure probability. Following this reasoning, we estimate the optimal activation thresholds s_m^* as

$$s_m^* = \max k \in [(m-2)M + 1, \dots, (m-1)M], \quad (13a)$$

$$s.t. P_f(k, T_{on}) < T_f, \quad (13b)$$

where we select the largest activation threshold out of those fulfilling the reliability requirement to minimize the activation of servers and therefore the energy consumption.

4.6. Optimal design of the server form

Following the above, the optimization problem can be formalized as follows. Let $\mathcal{T}$ denote the set of all possible server types, $C(\tau)$ denote the cost of using server type τ to support the service, and $P_f(\tau)$ the corresponding failure probability. The optimization problem is to find the optimal server type τ^* defined as follows:

$$\tau^* = \min_{\tau \in \mathbb{T}} C(\tau), \quad (14a)$$

$$s.t. \quad P_f(\tau) \leq T_f, \quad (14b)$$

Since server types have no relation with each other, nor between the parameters that characterize their performance, there is no alternative to performing an exhaustive search on all server types. Based on this, to compute the optimal server design, we follow these steps:

1) We compute the average number of users using (6).
2) Then, for each server type $\tau \in \mathbb{T}$:
    a) We compute the total number of servers using (8).
    b) We estimate the activation thresholds with (13a).
    c) We compute the average number of active servers using (9).
    d) We compute the total cost with (2).
3) Finally, we select the deployment τ^* with the mini-mum cost.

5. Performance Evaluation

In this section, we first assess the accuracy of the analytical model and then validate the proposed algorithm to design the server farm. Results from the analytical model are obtained using MATLAB Release 2023a, while simulation results are obtained using a discrete event simulation written in C++. This simulator was also used in our previous works [5], [6], [24]. We perform as many replications as required until the confidence intervals are below 1% of the average (not shown for clarity). Note that the simulation does not relax certain assumptions of the analytical model such as the server infallibility. All computations are performed on a server equipped with an Intel Core i7 CPU with 4 cores, 8 threads, operating at a base frequency of 1.30GHz, and supported by 16 GB of RAM.

	Carrier	Enterprise	Consumer	Rack	Blade
Capacity M $(tasks)$	16	4	2	12	32
Boot-up $1/\alpha$ (min)	3	8	18	2	1
Lifetime $1/\beta$ $(days)$	32	16	8	7	4
P_{peak} (W)	270	78	7.6	70	241
P_{load} (W)	7.5	18.22	1.5	3.33	22.84
P_{idle} (W)	150	5.1	4.6	30	20
$Cost$ K (€)	2000	500	100	1000	300
Lifespan R $(years)$	10	6	2	8	4

TABLE 2 DEPLOYMENTS CONSIDERED IN THE PERFORMANCE EVALUATION.

We assume that boot up times and lifetimes are exponential random variables too, with an average that depends on the server type τ. We focus on the following set of target failure probabilities $T_f = \{10^{-3}, 10^{-4}, 10^{-5}\}$, which correspond to a reliability between three nines (99.9%) and five nines (99.999%).

5.1. Server types

In our experiments, we consider five different server types, ranging from cost-effective consumer-grade machines to high-performance blade servers. These configurations were previously defined in [6], [24], and for completeness, we summarize their performance parameters in Table 2. The selected server types cover a broad range of characteristics, ensuring that our analysis and design are validated under diverse conditions. However, our analysis is not tied to these specific configurations, i.e., our model can be applied to assess the performance of alternative parameterizations and support the design of other server farm architectures. Lastly, we emphasize that this list is not an exhaustive catalog of possible server types.

5.2. Model validation

We first confirm the validity of the analytical model to estimate the required performance figures to compute the cost for a given configuration τ, namely, the average number of users N_u, the number of servers N, and the average number of active servers N_a, which in turn depends on the activation thresholds $\{s_m\}$. To this aim, we next compare the results obtained using the analytical model with those obtained using simulations. To obtain these, we use the following methodology. For a given value of λ and μ, we initially set the number of servers to $N = \lambda/\mu$ (i.e., the load in Erlangs), and run the simulations using the configuration algorithm in [6] that aims at minimizing resource consumption while ensuring that the failure probability P_f is below the target value T_f. If P_f is above T_f, we increase the number of servers N by one and repeat the process, until P_f is below T_f. Throughout or model validation, we consider three different inter-arrival distributions, each represented with a distinct symbol in the figures:

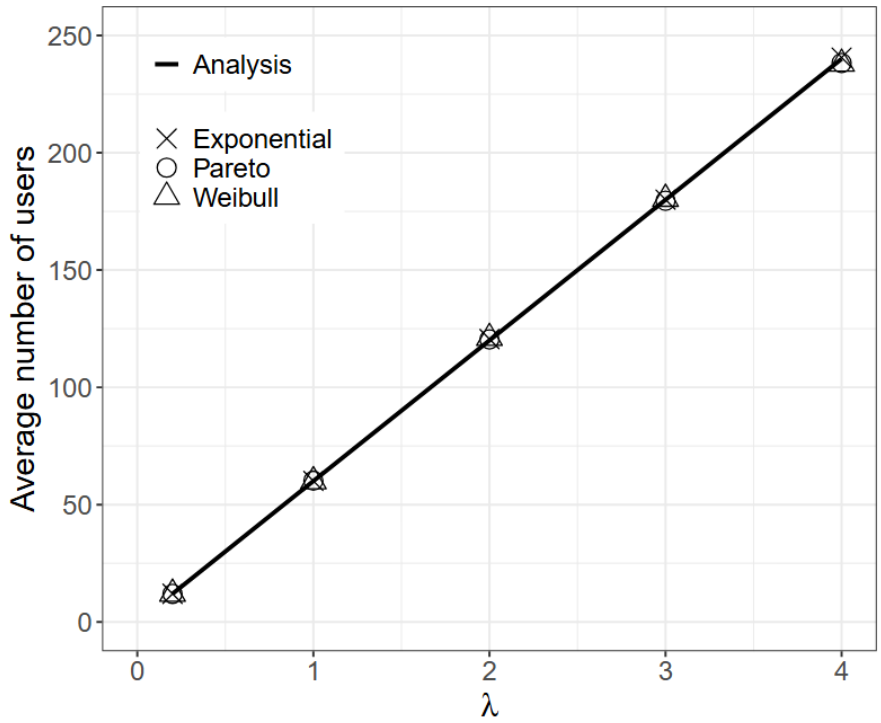

FIGURE 2 AVERAGE NUMBER OF USERS VS. λ USING ANALYSIS (LINES) AND SIMULATIONS (SYMBOLS).

- Exponential distribution, with rate $\lambda = \{0.2, 0.4, \ldots, 4.0\}$ tasks/min.
- Pareto distribution, with shape parameter $\alpha = 2$ and scale parameter $x_m = \{2.5, 1.25, \ldots, 0.125\}$ min/tasks.
- Weibull distribution, with shape parameter $k = 2$ and scale parameter $\theta = \{5.6, 1.12, \ldots, 0.28\}$ min/tasks.

5.2.1. Average Number of users and distribution of users

We start our model validation by comparing the analytical results of Section 4.2 with those obtained via simulations. We first compare the average number of users obtained using (6) with those computed using simulations, for all considered server types and the different arrival rates considered.

We present the results in Fig. 2, using points for the simulation results and lines for the analytical values. The figure confirms the good accuracy of the analytical model, since the results practically overlap.

We also compare the distribution of users using (5) with those computed using simulations, for all the server types and three selected values of λ. We depict the probability mass function of the user distribution in Fig. 3, using bins for simulation results and black lines for the analytical values obtained using (5). These results also confirm the accuracy of the distribution of users of the analytical model, as the differences between the theoretical and experimental distributions are very small.

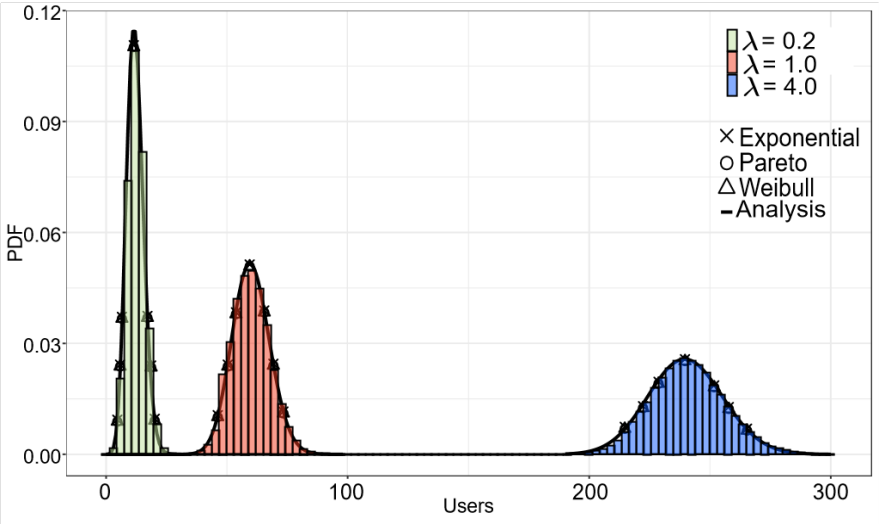

FIGURE 3USER DISTRIBUTION FOR DIFFERENT λ VALUES: ANALYSIS (LINE) AND SIMULATION (BINS AND SYMBOLS).

5.2.2. Total number of servers

Here we assess the validity of our analysis to dimension the server farm. To this aim, we compare the total number of servers N required to guarantee $P_f < T_f$ using the methodology described for the simulations with the values obtained via (8). We perform the comparison for the same values of λ as before and all server types using $T_f = 10^{-4}$, and depict the corresponding results in Fig. 4, using lines for the analytical results and points for those obtained using simulations.

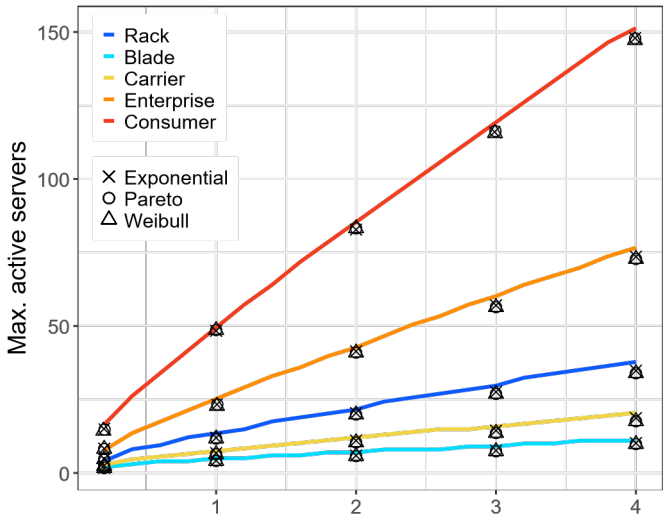

FIGURE 4 MAXIMUM NUMBER OF SERVERS VS. λ USING ANALYSIS (LINES) AND SIMULATION (SYMBOLS).

Like in the previous case, the results confirm the accuracy of the model, as the results practically overlap for all considered values of λ and server type, with an average error of 4.6% and the maximum error being below 10%. As expected, the number of required servers grows with the inverse of the server capacity M, with the consumer-grade server type requiring the maximum number of servers, and the blade type the minimum. We find that the analytical figures are always above the ones obtained using simulations, and therefore the total number of servers according to our design never falls below the required number of servers. Finally, we conducted the same experiment for the rest of server types and T_f values, obtaining an average error of 4.8% and a maximum error below 12%.

λ (tasks/min)		$s2$	$s3$	$s4$	$s5$	$s6$	$s7$	$s8$	$s9$
0.2	Sim.	22	34	46	58	70	82	94	106
	Ana.	21	33	45	57	70	82	94	106
	Δs	1	1	1	1	0	0	0	0
0.4	Sim.	21	33	45	57	69	82	94	106
	Ana.	20	32	44	56	68	80	93	105
	Δs	1	1	1	1	1	2	1	1
1.0	Sim.	16	28	40	52	64	76	88	100

	Ana.	16	28	41	53	65	77	90	102
	Δs	0	0	-1	-1	-1	-1	-2	-2
2.0	Sim.	14	26	38	50	62	74	86	98
	Ana.	13	25	37	49	61	73	85	98
	Δs	1	1	1	1	1	1	1	0
3.0	Sim.	13	25	37	49	61	73	85	97
	Ana.	13	25	37	49	61	73	85	97
	Δs	0	0	0	0	0	0	0	0
4.0	Sim.	12	24	36	48	60	72	84	96
	Ana.	12	24	36	48	60	72	83	96
	Δs	0	0	0	0	0	0	1	0

TABLE 3 ACTIVATION THRESHOLDS FOR THE RACK SERVER DEPLOYMENT AND DIFFERENT VALUES OF λ.

5.2.3. Activation Thresholds

Here we validate the analysis presented in Section E to estimate the values of the activation thresholds $\{s_m\}$ configured by the algorithm. To this aim, we first focus on the configurations using the rack server type and set $T_f = 10^{-3}$ as the maximum failure probability and compare the first eight values of the vector $\{s_m\}$ for different values of the arrival rate λ as in the previous sections. The simulation (Sim.) and analytical (Ana.) results are presented in Table 3, as well as the difference (Δs).

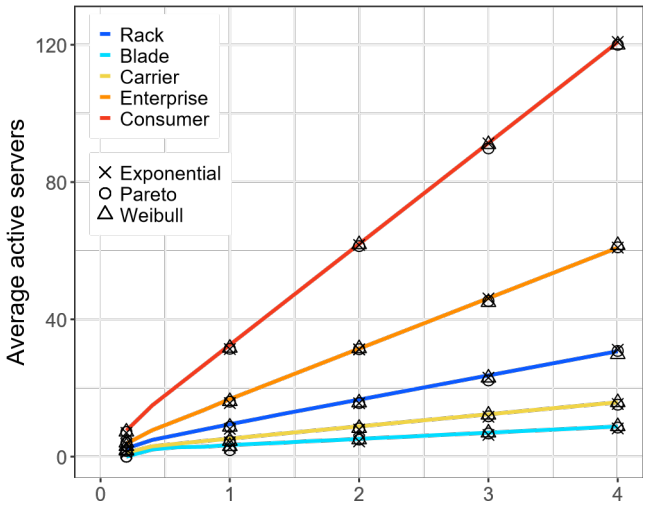

FIGURE 5 AVERAGE NUMBER OF ACTIVE SERVERS VS. λ USING ANALYSIS (LINES) AND SIMULATION (SYMBOLS).

The table illustrates that as the load grows, the activation thresholds decrease, since servers need to be activated with more anticipation to accommodate the incoming tasks. It also shows a good match between the results predicted by the model and those obtained using simulations, with a mean absolute difference of 0.6 tasks and a maximum difference of 2 tasks. We repeated the same experiment for the other types of servers and the considered T_f values, with the mean absolute difference being 0.7 tasks and the maximum absolute difference being 2 tasks. These results again confirm the accuracy of the analytical model to estimate the configuration of the server farm.

5.2.4. Average number of active servers

Finally, we assess the accuracy of (9) to estimate the average number of active servers, using the same methodology as before. As in the previous cases, we first assume $T_f = 10^{-4}$ and different values of the traffic load. We represent in Fig. 5 with lines the results from the analytical model and with points those using simulations. The figure confirms the accuracy of the model, with an average absolute error of 0.45 servers and a maximum absolute error of 0.62 servers. Like in the case of Fig. 4, the larger the capacity M of the server type, the smaller the average number of active servers.

Based on the above results, we confirm the accuracy of the analytical model to predict the performance of a server farm for different traffic loads, reliability requirements, and server types. We next assess the performance of the algorithm proposed to design the server farm.

5.3. Design of the server farm

Following the validation of the analytical model, we next assess the performance of the algorithm presented in Section 4.6 to design a server farm. To this aim, we assume the same set of traffic rate values λ and target failure probability levels $T_f = \{10^{-3}, 10^{-4}, 10^{-5}\}$ considered before. To provide an adequate context, we compare the results from our algorithm against two benchmarks:

- An exhaustive search in the configuration space.

- A benchmark based on the Erlang-C [25], which determines for each type of server the number of resources required to ensure that the probability of blocking meets a specific reliability target (i.e., $P_B = 1 - R$), and selects the minimum.

For each considered scenario, we compute:

- For each of the five server types considered, the minimum cost (C_s) obtained via simulations using a search on the total number of servers.
- The optimal server type τ^* and corresponding cost (C_a) according to our algorithm.
- The difference between the minimum cost obtained with the numerical search and the one obtained with our algorithm (ΔC).
- The cost of the benchmark design C_b based on the Erlang-C, and the corresponding difference vs. C_a, denoted as ΔB.

We provide the resulting figures for the λ and T_f values considered in Table 4. For each scenario, we highlight in gray the minimum cost obtained using simulations, and in bold font the resulting τ^* whenever the best type of server according to our method corresponds to the one that minimizes costs using simulations.

There are several results that can be derived from Table 4. First, both for simulations and analysis, the minimum cost increases as the load increases, since more resources are needed to accommodate the incoming traffic. Second, there is no optimal server configuration for all scenarios, as the best server type alternates between the five considered types depending on the load and reliability considered. Third, we note that our configuration algorithm provides the optimal server type in 16 out of the 18 scenarios considered (i.e., 88.8% of the scenarios), and that for those two scenarios the relative error in terms of cost is smaller than 2%. The average cost difference between the configuration provided by the algorithm and the simulations is 3%, which confirms the effectiveness of our proposal to design a server farm. Fourth, the comparison of the cost between the analysis (C_a) and benchmark (C_b) highlights the advantages of using our proposed method over a predefined benchmark configuration. We note that the absolute cost for the benchmark is notably higher than the one by the analysis for all combinations. The benchmark consistently results in a higher number of servers due to its more pessimistic assumptions. The relative cost difference (ΔB) demonstrates that relying on a fixed server configuration can lead to significant cost inefficiencies, with the benchmark costing on average 22% more than the optimized analysis-based approach. In some cases, such as λ = 0.2 and $T_f = 10^{-4}$, the cost increase reaches 54%, reinforcing the importance of dynamically selecting the optimal server type rather than adhering to a static deployment strategy. Finally, the table also highlights the importance of an adequate selection of the best server type, in addition to its optimal configuration, since there are substantial differences in terms of cost between optimal server deployments with different types. For instance, for λ = 0.2 and T_f = 10−3 (first row of the table), there is a factor of 3.33× between the minimum cost using the Enterprise type of server and the one using the Rack type of server, while the average difference across the table is 1.64×.

5.4. Computational Time

Finally, we compare the time to determine the optimal design using the method presented in Section 4 with an exhaustive search using simulations. To this aim, we compute the total execution time required for each approach. We assume the same scenarios as before with different values of λ and the reliability levels $T_f = \{10^{-3}, 10^{-4}, 10^{-5}\}$. We illustrate the results in Fig. 6, using a logarithmic scale on the y axis.

According to the figure, the proposed method results in significantly shorter execution times for all values of λ and T_f. Furthermore, these times are practically constant, while simulation times increase with the traffic load, and with the inverse of T_f. Based on these results, we conclude that the analysis developed in Section 4 offers a cost-effective solution, particularly well suited for scenarios demanding robust, scalable, and resource efficient methodologies.

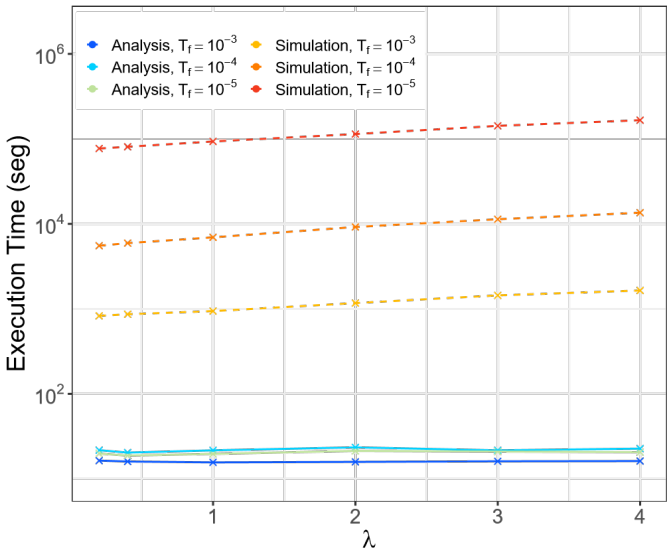

FIGURE 6 EXECUTION TIME VS. LAMBDA REQUIRED BY SIMULATION (DASHED LINES) AND ANALYSIS (CONTINUOUS LINES) FOR DIFFERENT VALUES OF T_f.

		Simulation Cost (Cs)					Analysis			Benchmark	
λ	Tf	Carrier	Enterprise	Consumer	Rack	Blade	τ*	Ca	ΔC	Cb	ΔB
0.2	10^-3	495.98	256.26	665.26	855.4	260.03	Blade	260.6	1.70%	388	49.01%
	10^-4	498.17	268.04	668.52	858.25	261.73	Enterprise	265.2	1.30%	408	54.00%
	10^-5	502.53	275.78	671.83	861.07	265.21	**Blade**	269.7	1.66%	474	75.76%
0.4	10^-3	824.32	712.03	1006.47	1569.52	706.9	**Blade**	716.1	1.28%	777	8.47%
	10^-4	828.42	763.08	1013.58	1573.58	710.14	**Enterprise**	766.9	0.50%	840	9.64%
	10^-5	835.07	770.93	1125.36	1580.05	770.55	**Blade**	787.8	2.24%	1084	37.63%
1	10^-3	1634.8	1557.3	1360.3	1860.12	1631.93	**Consumer**	1365	0.36%	1516	11.06%
	10^-4	1681.6	1729	1747.32	1865.57	1747.15	**Enterprise**	1747	1.05%	1860	6.47%
	10^-5	2055.1	2028.34	2022.25	2056.09	2031.67	**Consumer**	2190	6.50%	2431	11.00%
2	10^-3	2995.7	3131.26	2804.93	2812.14	3121.38	**Consumer**	3009	7.26%	3119	6.97%
	10^-4	3010.1	3150.12	2899.54	2892.2	3135.09	**Consumer**	3094	6.70%	3191	10.05%
	10^-5	3140.2	3175.16	3126.13	3198.45	3150.67	**Consumer**	3205	2.52%	3365	6.76%
3	10^-3	3861.4	3891.09	3917.21	3901.75	461.36	**Carrier**	4101	6.20%	4838	17.96%
	10^-4	3975.1	4005.11	3990.17	3947.93	4625.2	**Rack**	4148	4.40%	5199	31.36%
	10^-5	4102.2	4188.63	4190.37	4321.54	4660.33	**Consumer**	4334	4.49%	5867	35.41%
4	10^-3	5321.7	5561.98	5432.71	5412.82	5871.76	**Carrier**	5588	5.01%	6007	7.50%
	10^-4	5640.2	5580.29	5521.96	5512.15	6189.34	**Rack**	5644	1.36%	6345	12.43%
	10^-5	5980.3	5821.87	5820.19	5847.63	5930.54	**Consumer**	5893	1.36%	6641	12.67%

TABLE 4 MINIMUM COST DESIGN USING SIMULATIONS (C_s), OUR ANALYSIS (C_a), AND THE BENCHMARK (C_b).

5.5. Heterogeneous scenarios

As discussed in Section 3, we initially consider homogeneous scenarios, where a single type of service is provided by a single type of server. In this section, we relax this assumption to demonstrate how the proposed framework can be extended to design heterogeneous scenarios. Specifically, we consider a case with two types of services, labeled 1 and 2. Both have the same service rate μ, but differ in their reliability requirements, T_1 and T_2, as well as their arrival rates, λ_1 and λ_2, respectively.

One way to design a heterogeneous server farm using our framework is to independently determine the optimal server type for each type of traffic. We denote these as τ_i^* for $i \in 1,2$. Assuming that the Infrastructure Manager (IM) redirects each type of task to the corresponding server type, the total cost of this heterogeneous design is given by:

$$C_{het} = C(\tau_1^*) + C(\tau_2^*), \quad (15)$$

As a benchmark, we assume a homogeneous design to support the total traffic $\lambda_1 + \lambda_2$ and the most stringent reliability requirement, $min(T_1, T_2)$. The resulting cost of this design is denoted as C_{hom}. Table 5 presents the resulting values of C_{het} and C_{hom} for different values of $\lambda_1, \lambda_2, T_1$, and T_2. It also reports the relative difference between the homogeneous and the heterogeneous design, ΔB. Note that for some scenarios, the cost $C_{hom}$is reused from Table 4.

λ_1	T_1	λ_2	T_2	C_{het}	C_{hom}	ΔB
0.3	10^{-3}	0.7	10^{-4}	1831.92	1729.00	-5.95%
0.3	10^{-4}	0.7	10^{-3}	1714.34	1729.00	0.85%
0.3	10^{-4}	0.7	10^{-5}	2032.31	2022.25	-0.50%
1.5	10^{-3}	1.5	10^{-4}	3892.55	3947.93	1.40%
1.5	10^{-4}	1.5	10^{-3}	3892.55	3947.93	1.40%
1.5	10^{-4}	1.5	10^{-5}	4422.17	4147.31	-6.63%
1.5	10^{-3}	2.5	10^{-4}	5218.46	5512.15	5.33%
1.5	10^{-4}	2.5	10^{-3}	5159.25	5512.15	6.40%
1.5	10^{-4}	2.5	10^{-5}	5509.08	5820.19	5.35%
λ_1	T_1	λ_2	T_2	C_{het}	C_{hom}	ΔB
0.3	10^{-3}	0.7	10^{-4}	1831.92	1729.00	-5.95%
0.3	10^{-4}	0.7	10^{-3}	1714.34	1729.00	0.85%
0.3	10^{-4}	0.7	10^{-5}	2032.31	2022.25	-0.50%
1.5	10^{-3}	1.5	10^{-4}	3892.55	3947.93	1.40%
1.5	10^{-4}	1.5	10^{-3}	3892.55	3947.93	1.40%
1.5	10^{-4}	1.5	10^{-5}	4422.17	4147.31	-6.63%
1.5	10^{-3}	2.5	10^{-4}	5218.46	5512.15	5.33%
1.5	10^{-4}	2.5	10^{-3}	5159.25	5512.15	6.40%
1.5	10^{-4}	2.5	10^{-5}	5509.08	5820.19	5.35%

TABLE 5 HETEROGENEOUS SCENARIOS.

The results confirm that our proposal can be extended to heterogeneous scenarios, as most configurations yield additional cost savings –up to 6.4% in some cases. However, these gains remain moderate due to two main factors. First, traffic is isolated across server types, which prevents potential multiplexing gains. Second, our homogeneous design already performs well, leaving limited room for further improvement except in specific scenarios. As discussed in Section 6, our ongoing work focuses on developing novel analytical models to more effectively address the design of heterogeneous server farms.

6. Summary and Conclusions

Designing server farms that can automatically scale to meet changing demands is a complex task—especially in 5G and beyond networks, where reliability and latency are critical. The challenge lies in balancing performance and cost, while accounting for practical constraints such as server boot-up delays, hardware failure risks, and finite lifespans. Traditional approaches often overlook these factors or take a conservative path, over-provisioning resources to ensure service guarantees—at the expense of efficiency and cost.

In this paper, we introduced a comprehensive framework for cost- and reliability-aware auto-scaling of virtualized server infrastructures. By combining queueing theory, cost modeling, and server selection, our method identifies deployment strategies that minimize infrastructure and operational expenses without compromising service reliability. It accounts for real-world constraints like startup latency and failure probabilities, making it particularly suited for demanding applications such as industrial automation, autonomous vehicle coordination, and mission-critical IoT services.

Beyond its theoretical contribution, this framework lays the groundwork for the development of a dedicated auto-scaling rApp within the O-RAN architecture, specifically hosted in the Non-RT RIC. Such an rApp would use historical traffic data and service KPIs to forecast demand and compute optimal provisioning plans for the virtualized RAN (vRAN). These recommendations could be pushed to the O-Cloud orchestrator through the O2 interface, or dynamically adjusted via A1 policies in coordination with the Near-RT RIC. This approach enables intelligent, adaptive resource allocation, ensuring that critical network functions receive the computing power they need—no more, no less—enhancing both reliability and cost-efficiency.

As part of our ongoing work, we are extending this framework in several directions. First, we aim to support heterogeneous and multiplexed resource sharing, moving beyond service-class-specific deployments to exploit cross-service synergies. Second, we are integrating end-to-end latency modeling, taking into account RAN performance, VNF chaining, and network topology to better meet URLLC (Ultra-Reliable Low-Latency Communications) requirements. Finally, we are implementing the solution in Linux-based test environments, incorporating load-balancing mechanisms and leveraging tools like ACHO for seamless service migration. These enhancements are key steps toward deploying a practical, scalable, and vendor-neutral auto-scaling solution that fits naturally within the evolving O-RAN and vRAN ecosystems.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

References

[1] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network slicing to enable scalability and flexibility in 5g mobile networks," IEEE Communications Magazine, vol. 55, no. 5, pp. 72–79, 2017.

[2] M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, and R. Perez, "The case for serverless mobile networking," in 2020 IFIP Networking Conference (Networking), 2020, pp. 779–784.

[3] Y. Ren, T. Phung-Duc, J. Chen, and Z. Yu, "Dynamic Auto Scaling Algorithm (DASA) for 5G Mobile Networks," in Proceedings of the IEEE Global Communications Conference (GLOBECOM 2016), Washington DC, USA, Dec. 2016.

[4] Č. Stefanović, "Industry 4.0 from 5g perspective: Use-cases, requirements, challenges and approaches," in 2018 11th CMI International Conference: Prospects and Challenges Towards Developing a Digital Economy within the EU. IEEE, 2018, pp. 44–48.

[5] J. Ortin, P. Serrano, J. Garcia-Reinoso, and A. Banchs, "Analysis of scaling policies for nfv providing 5g/6g reliability levels with fallible servers," IEEE Transactions on Network and Service Management, vol. 19, no. 2, pp. 1287–1305, 2022.

[6] J. Perez-Valero, A. Banchs, P. Serrano, J. Ortín, J. GarciaReinoso, and X. Costa-Pérez, "Energy-aware adaptive scaling of server farms for nfv with reliability requirements," IEEE Transactions on Mobile Computing, vol. 23, no. 5, pp. 4273– 4284, 2024.

[7] J. Liu, Z. Jiang, N. Kato, O. Akashi, and A. Takahara, "Reliability evaluation for nfv deployment of future mobile broadband networks," IEEE Wireless Communications, vol. 23, no. 3, pp. 90–96, 2016.

[8] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliabilityaware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," IEEE Transactions on Network and Service Management, vol. 14, no. 3, pp. 554–568, 2017.

[9] Y. Ren, T. Phung-Duc, J.-C. Chen, and Z.-W. Yu, "Dynamic auto scaling algorithm (dasa) for 5g mobile networks," in 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6.

[10] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," Performance Evaluation, vol. 67, no. 11, pp. 1123–1138, 2010, performance 2010. [Online]. Available: https://www.sciencedirect.com/science/ article/pii/S0166531610000957

[11] A. Song, W. Wang, and J. Luo, "Stochastic modeling of dynamic power management policies in server farms with setup times and server failures," International Journal of Communication Systems, vol. 27, no. 4, pp. 680–703, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/ 10.1002/dac.2761

[12] J. Burge, P. Ranganathan, and J. L. Wiener, "Cost-aware scheduling for heterogeneous enterprise machines (cash'em)," in 2007 IEEE International Conference on Cluster Computing, 2007, pp. 481–487.

[13] K. Goseva-Popstojanova and K. Trivedi, "Failure correlation in software reliability models," in Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443), 1999, pp. 232–241.

[14] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah, "Worth their watts? - an empirical study of datacenter servers," in HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, 2010, pp. 1–10.

[15] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in Proceedings of the 5th

USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'08. USA: USENIX Association, 2008, p. 337–350.

[16] 5G Americas, ""Vehicular Connectivity: C-V2X and 5G," Technical White Paper, 2021.

[17] S. K. Singh, Linux Yourself: Concept and Programming, 1st ed. Chapman and Hall/CRC, 2021. [Online]. Available: https://doi.org/10.1201/9780429446047

[18] G. Garcia-Aviles, C. Donato, M. Gramaglia, P. Serrano, and A. Banchs, "Acho: A framework for flexible re-orchestration of virtual network functions," Computer Networks, vol. 180, p. 107382, 2020.

[19] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A measurement-based characterization of the energy consumption in data center servers," IEEE Journal on Selected Areas in Communications, vol. 33, no. 12, pp. 2863–2877, 2015.

[20] M. Ananth and R. Sharma, "Cloud management using network function virtualization to reduce capex and opex," in 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2016, pp. 43–47.

[21] L. Kleinrock, Theory, Volume 1, Queueing Systems. USA: Wiley-Interscience, 1975.

[22] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. John Wiley & Sons, 2006.

[23] K. Rupp, R. Schill, J. Süskind, P. Georg, M. Klever, A. Lösch, L. Grasedyck, T. Wettig, and R. Spang, "Differentiated uniformization: A new method for inferring markov chains on combinatorial state spaces including stochastic epidemic models," Computational Statistics, pp. 1–21, 2024.

[24] J. Perez-Valero, J. Garcia-Reinoso, A. Banchs, P. Serrano, J. Ortin, and X. Costa-Perez, "Performance trade-offs of auto scaling schemes for nfv with reliability requirements," Computer Communications, vol. 212, pp. 251–261, 2023.

[25] T. R. Robbins, D. J. Medeiros, and T. P. Harrison, "Does the erlang c model fit in real call centers?" in Proceedings of the Winter Simulation Conference. IEEE, 2010, pp. 2853–

[26] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 577–578.

[27] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, ser. MGC '10. New York, NY, USA: Association for Computing Machinery, 2010.

[28] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," Computing, vol. 98, no. 7, pp. 751–774, Jun. 2014.

[29] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in 2008 Real-Time Systems Symposium, 2008, pp. 303–312.

[30] W. E. Smith, K. S. Trivedi, L. A. Tomek, and J. Ackaret, "Availability analysis of blade server systems," IBM Systems Journal, vol. 47, no. 4, pp. 621–640, 2008.

[31] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009, pp. 365–371.

[32] R. d. S. Matos, P. R. M. Maciel, F. Machida, D. S. Kim, and K. S. Trivedi, "Sensitivity analysis of server virtualized system availability," IEEE Transactions on Reliability, vol. 61, no. 4, pp. 994–1006, 2012.

[33] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," Performance Evaluation, vol. 67, no. 11, pp. 1123 – 1138, Nov. 2010.

[34] O. Adamuz-Hinojosa, L. Zanzi, V. Sciancalepore, A. GarciaSaavedra, and X. Costa-Pérez, "Oranus: Latency-tailored orchestration via stochastic network calculus in 6g o-ran," in IEEE INFOCOM 2024 - IEEE Conference on Computer Communications, 2024, pp. 61–70.