Digital Object Identifier

A Robust Game Approach for On Spot Price Cloud Markets in Microservice-based Applications

HAMTA SEDGHANI[1,4], MAURO PASSACANTANDO[2], RICCARDO LANCELLOTTI[3], MINA ZOLFY LIGHVAN[4], DANILO ARDAGNA[1]

[1]Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy (e-mail: name.lastname@polimi.it)
[2]Department of Business and Law, University of Milano-Bicocca, Milan, Italy (e-mail: mauro.passacantando@unimib.it)
[3]Department of Engineering "Enzo Ferrari" University of Modena and Reggio Emilia, Modena, Italy (e-mail: riccardo.lancellotti@unimore.it)
[4]Department of Electronics and computer, University of Tabriz, Tabriz, Iran (e-mails: mzolfy@tabrizu.ac.ir)

Corresponding author: Hamta Sedghani (h.sedghani@tabrizu.ac.ir, hamta.sedghani@polimi.it).

ABSTRACT

The evolution and widespread adoption of virtualization, service-oriented architectures, autonomic computing, and utility computing have converged, giving rise to Cloud Computing, which enables the on-demand delivery of software, hardware, and data as services. As Cloud-based services become more numerous and dynamic, developing efficient service provisioning policies is increasingly challenging due to the impact of estimation errors in the operating parameters of applications such as execution time or request rate for each application and for each service. In this paper, we take the perspective of Software as a Service (SaaS) providers which host their applications at an Infrastructure as a Service (IaaS) provider. Each SaaS needs to comply with quality of service requirements, specified in Service Level Agreement (SLA) contracts with the end-users, which determine the revenues and penalties based on the achieved performance level. SaaS providers should maximize their revenue and minimize their cost by deciding how many VMs for each class (reserved, on-demand or spot) are to be used, while IaaS must decide the price of VMs in order to maximize his revenue. The problem is modeled using a Generalized Nash Equilibrium Problem (GNEP). The uncertainty in the application parameters, that are not known until run time, is specifically addressed in this paper using a cardinality-constrained robust optimization approach to formulate a robust version of the SaaS problem. Furthermore, we propose a best-reply algorithm to identify a robust GNE, ensuring that the solution remains feasible even under the worst-case scenarios of uncertain parameters. An analytical model is developed to derive the equilibrium. For validation, the analytical model results are compared with simulations that reflect real situations, characterized by more realistic parameter distributions and limited buffer capacities.

INDEX TERMS Cloud market pricing, Generalized Nash Equilibrium, Robust game, Microservice-based applications.

I. INTRODUCTION

Web application architecture is evolving from traditional monolithic, multi-tier models to more agile, loosely-coupled microservices (MS) [1]. Gartner [2] predicted that by 2025, over 90% of new digital applications in large enterprises will be developed using MSs.

MSs architecture offers numerous advantages, including faster delivery, enhanced scalability, and reduced development and maintenance costs. By decomposing applications into smaller, independent services, each typically running in its own container, MSs provide flexibility and modularity, making it easier to scale and maintain complex systems [1]. This architecture is particularly beneficial for large-scale software systems. By decoupling applications into dozens or hundreds of services that interact via lightweight communication protocols, organizations can quickly respond to business demands, achieve fault isolation, and simplify deployment and maintenance. However, the complexity of interactions between services can exacerbate traditional software engineering challenges such as configuration tuning,

operational complexity, and performance management [3].

Ensuring performance guarantees in MS-based applications is crucial due to their distributed nature and their impact on user experience and business operations [4]. Many businesses operate under Service Level Agreements (SLAs) that include performance metrics like response time [5], [6]. Failure to meet these metrics can result in financial penalties and damage to reputation, especially in critical industries like finance and healthcare, where high performance is essential to maintaining uninterrupted operations [3], [6], [7].

As the number of MSs grows, maintaining performance and troubleshooting becomes increasingly difficult. Resource orchestration involves creating software containers on cloud infrastructure nodes. These containers provide isolated environments for running MSs within Virtual Machines (VMs). Scalability is a key feature of MSs architecture, allowing applications to efficiently handle increases in user base, data volume, or transaction rates. MSs enable independent scaling of each service, allowing for dynamic resource allocation that meets varying demands cost-effectively. Optimal orchestration solutions are needed to manage resources efficiently, avoid over-provisioning, and minimize costs [1], [6].

In a cloud computing ecosystem, SaaS (Software-as-a-Service) providers deliver application-level services to end-users, such as customer relationship management, enterprise resource planning, or streaming services. These applications rely on underlying computational resources to process requests, manage data, and ensure service quality. On the other hand, IaaS (Infrastructure-as-a-Service) providers supply the foundational infrastructure, including VMs, storage, and networking capabilities, that SaaS providers leverage to host and run their services. In the modern cloud ecosystem, MS-based software is increasingly offered as a service under the SaaS model [2]. In this paradigm, applications are available over the web, providing Quality of Service (QoS) guarantees to end users [8].

In our work, we focus on QoS to develop a solution that ensures high-quality service delivery. Specifically, we aim to keep the response time of MS requests below specific thresholds, which vary for each service within the application. In the SaaS-IaaS paradigm, IaaS providers offer various types of VMs, including spot instances, which are cost-effective but have limited availability. SaaS providers often compete for these spot instances to minimize costs [9].

Developing efficient service provisioning policies is a significant challenge in cloud research, critical to the overall efficiency of the framework [10]–[12]. In this context, game theory provides valuable insights into service provisioning [13], [14], with applications in resource management, pricing [15], [16], and competition among providers [17]–[20]. One widely used concept in game theory is Nash Equilibrium, where no player can benefit from changing their strategy if others' strategies remain unchanged.

In this work, we consider the perspective of SaaS providers hosting applications on an IaaS platform. Each SaaS provider aims to maximize profit while adhering to QoS requirements specified in SLAs with end users. These SLAs determine revenues and penalties based on performance levels. Profit for a SaaS provider is calculated as the revenue from SLAs minus the costs of using IaaS resources. However, SaaS providers compete selfishly for these resources, while the IaaS provider seeks to maximize its revenue.

To model the behavior of SaaS and IaaS providers in this competitive scenario, we employ the concept of Generalized Nash Equilibrium (GNE) [21], an extension of the classical Nash Equilibrium. However, traditional GNE models often assume that critical parameters, such as service rates and execution times, are known and remain fixed. While this assumption simplifies the mathematical formulation, it can lead to suboptimal or impractical solutions when applied to real-world systems, as these parameters are often subject to variability and uncertainty [22]. To address this challenge, we model the problem as a Robust Game, accounting for uncertainties in key parameters derived from future predictions and runtime monitoring. By incorporating these uncertainties, we aim to develop a robust solution that remains effective even under worst-case scenarios. To the best of our knowledge, the problem of ensuring stable performance even in presence of workload estimation errors is a rather novel problem that has seldom been addressed in literature. The approach to model uncertainty using the number of wrongly estimated parameters is a unique and original contribution of this study. In summary, the main contributions of this paper are as follows:

- We propose a model to capture the interaction between IaaS and SaaS providers as a Generalized Nash Equilibrium Problem (GNEP). In this framework, SaaS providers bid and compete for infrastructural services, while the IaaS provider determines the quantity and pricing of spot VM instances allocated to the SaaS providers application services.
- We formulate a robust GNEP model to mitigate the impact of uncertain parameters in the SaaS problem and we propose a best-reply algorithm to solve this game. The robust GNEP model relies on a cardinality-constrained robust optimization approach as in [23].
- We validate the effectiveness of the proposed analytical model using a simulator. Our tests encompass several scenarios to demonstrate the robustness of the approach. In particular, we consider also lognormal distributions with high variance and we take into account the presence of finite-size buffers and timeouts in the request service model, as it is common in real cloud applications. Under these realistic conditions, we demonstrate how performance is affected depending on the different levels of robustness.

The remainder of this paper is organized as follows: Section II discusses related works. Section III introduces the system model and defines the problem. Section IV formulates the Robust Game. Section V proposes a best-reply algorithm to solve the game. Experimental results are discussed in

Section VI, while conclusions are finally drawn in Section VII.

II. RELATED WORK

Game Theory has been applied to numerous problems in this context. Comprehensive studies on incentive approaches for resource management in cloud are presented in [24] and [25], grounded in artificial intelligence, game theory, and blockchain to foster an economically sustainable cloud ecosystem. It delves into various incentive mechanisms designed to optimize resource allocation in cloud environments, addressing challenges such as resource wastage, load balancing, and QoS violations. Recently, a utility model has been used to examines QoS as a key factor in the equilibrium between users and service providers in [26]. It introduces reputation cost to balance short-term and long-term revenue for providers. Using incomplete information evolutionary equilibrium theory, the paper establishes utility functions and a utility game for both parties. It identifies a balanced QoS solution where strategy changes result in utility loss. Finally, it presents a QoS optimization algorithm to achieve a near-optimal solution that balances user satisfaction and provider profit. Similarly, [27] investigates a hierarchical resource allocation scheme to provide high-quality computing services to end users while balancing the benefit requirements of all participants in the cloud-network convergence service system. The problem is formulated as a two-layer Stackelberg game, including a cloud-edge subgame and an edge-end subgame and a backward induction method is used for game analysis, proving the Stackelberg equilibrium. Convex optimization is employed to derive the optimal resource price response function for the edge layer and the optimal offloading response function for end users. A gradient-based dynamic pricing algorithm is designed to achieve optimal pricing in the cloud and optimal resource requests in the edge layer.

Moreover, efficient resource management for AI applications continues to be a major challenge, as it requires balancing dynamic resource demands and ensuring high performance in the computing continuum. To address this challenge, [28] introduces a design-time tool that uses a Mixed Integer Non-Linear Programming (MINLP) model to optimize resource selection and component placement for AI applications. This approach addresses the pre-deployment phase and focuses on optimizing execution costs and performance constraints. By utilizing various heuristic algorithms, the tool explores different deployment alternatives, ensuring efficient resource allocation within the continuum based on performance and technological constraints. In contrast, [29] employs a Stackelberg game model to optimize real-time resource allocation in Mobile Edge Cloud (MEC). This approach models the interaction between a leader (MEC platform) and followers (mobile users), aiming to balance the platform's resource provisioning and users' computational demands. The game-theoretic framework facilitates a dynamic resource allocation strategy that adapts to the changing demands of users and AI applications. Likewise, authors in [30] introduces the RLPRAF framework, which leverages reinforcement learning (RL) to proactively allocate cloud resources, optimizing resource provisioning and management in real-time. Unlike reactive systems, RLPRAF anticipates future resource demands based on historical data and workload predictions, reducing Service Level Agreement (SLA) violations and improving response times. By continuously learning from the dynamic cloud environment, the framework adapts to changing workloads and adjusts resource allocation in real-time, ensuring efficient resource utilization. The proposed system aims to enhance cloud service delivery by minimizing operational costs, improving resource efficiency, and maintaining high-QoS.

Unlike the aforementioned literature focusing on QoS, some works emphasize improving social welfare. For example, authors in [16] proposed a Dynamic Combinatorial Double Auction model to enhance social welfare and resource utilization, where the service providers and cloud users bid for various resource combinations dynamically. They used a greedy approximation method for winner determination and a truthful payment scheme to handle the complexity of combinatorial auctions. The model is proven to be approximately efficient, incentive compatible, individually rational, and budget-balanced, ensuring fairness and balanced resource allocation while considering both parties' interests and resource scarcity.

Some works consider a multiple-IaaS system, such as [31] that presents a resource allocation service aimed at optimizing user requests for cloud resources across multiple IaaS providers. The proposed platform acts as a broker, offering resource scheduling services based on an SLA that includes reliability, processing, and trust. This approach enables handling diverse user requests, better aligning with user needs, reducing scheduling costs, and avoiding QoS violations. Likewise, authors in [32] focused on preventing QoS violations, rather than reacting, for managing dynamic in multi-IaaS applications. They introduced capacity allocation algorithms designed to minimize execution costs while meeting average response time through joint load balancing and receding horizon capacity allocation techniques for handling multiple request classes. Differently, as the starting point of this paper, authors in [33] considered a GNEP for service provisioning, taking the perspective of SaaS providers hosting their applications at a single IaaS provider. They proposed two solution methods based on best-reply dynamics, proving their convergence to a generalized Nash equilibrium in finite iterations, and developed a distributed algorithm for runtime IaaS resource allocation among competing SaaS providers.

Unlike most studies that assume the resource demands are given in advance, this assumption may be unrealistic because the exact resource demands of a request implementation are unknown until it is completed [34]. There are a few other studies that considered the uncertainty of the other metrics for service provisioning and resource allocation such as computing resource, execution time, throughput. For exam-

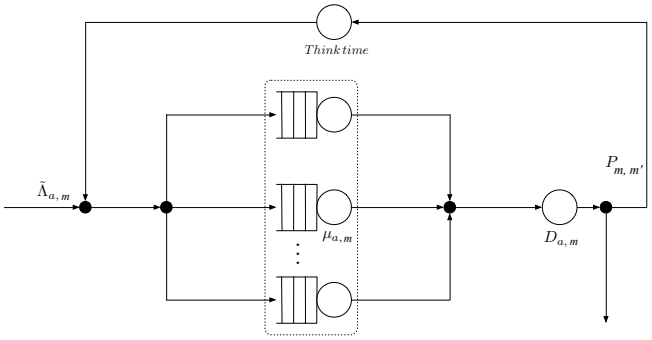


FIGURE 1: System Performance

ple, [35] addresses the dynamic nature of resource demands during execution and aims to optimize resource allocation in mobile edge computing networks to maximize network service provider profit. It tackles the robust service function chain placement (RSFCP) problem under uncertain resource demands, formulates it as a Quadratic Integer Programming (QIP) problem, proves its NP-hardness, and develops a near-optimal approximation algorithm using the Markov approximation technique. To address the uncertainty in the task offloading process in 5G environments, the authors in [36] considered both resource consumption and task reward uncertainties. They developed a multi-armed bandit-based online learning algorithm, demonstrating that its regret and violations are bounded sub-linearly.

In contrast to the aforementioned studies, in this paper we study service provisioning problem in cloud systems under the assumption that both execution time and service rate of each request are affected by uncertainty and propose a cardinality-constrained robust optimization approach to ensure the solution remains feasible even under the worst-case scenario of these uncertain parameters. To the best of our knowledge, none of the literature so far considered such uncertain parameters for service provisioning in the cloud.

III. PROBLEM STATEMENT

In this section, we present the problem and assumptions addressed in this paper. For the sake of clarity, the relevant symbols of the proposed model are summarized in Table 1. In particular, we consider SaaS providers that offer multiple applications built through MSs. The hosted applications can be heterogeneous in terms of resource demands, workload intensities, and QoS requirements. We denote the set of SaaS providers by $\mathcal{S}$, the set of applications of all the SaaS providers by $\mathcal{A}$ and the set of applications offered by the j-th SaaS by $\mathcal{A}_j$. Each application $a \in \mathcal{A}_j$ is built through multiple MSs (e.g., login, browse, buy, etc.) and $\mathcal{M}_a$ denotes the set of services supported by application a.

For each MS $m \in \mathcal{M}_a$, an SLA contract is established between the SaaS provider and its end users which is a commitment, detailing specific aspects of the service such as quality, availability, and responsibilities. These terms are mutually agreed upon by both the service provider and the service user. In our framework, this contract specifies SLA levels which determine the per request revenue (or penalty) and it is expressed in terms of average response time $R_{a,m}$. A threshold $\bar{R}_{a,m}$ is established so if the response time of a request exceeds this threshold, then the SaaS will pay a penalty $\nu_{a,m}$.

In the following, we will consider the run-time resource provisioning at a single IaaS provider. MSs, running in containers, are hosted in VMs, which are dynamically instantiated by the IaaS provider, and multiple VMs implementing the same MS can run in parallel.

Each SaaS provider must determine the optimal number of VMs for each application, performing resource allocation based on predictions of MS workloads. The prediction and control, specifically starting and stopping VMs, are performed periodically with a time interval T. The interval T can range from a few minutes to one hour (e.g., for providers who bill VMs on hourly basis). The SaaS needs also an estimate of the future performance of each VM in order to determine application average response time. In the following, we model (see Figure 1), as a first approximation, each microservice as an M/G/1 queue in tandem with a delay center [37], as done in [38]–[40]. We assume (as common in microservice based systems and containerized systems [41]) the requests are served according to the processor sharing scheduling policy . As discussed in [39], the delay center allows to model network delays and/or protocol delays introduced in establishing connections, etc. Performance parameters are also continuously updated at run-time (see [39] for further details) in order to capture transient behavior, VMs network and I/O interference [42], and performance time of the day variability of the considered Cloud provider [43]. User sessions begin with a class m MS request arriving from an exogenous source with rate $\tilde{\Lambda}_{a,m}$. The analysis of actual e-commerce site traces (see for example [44]) or cloud-based applications [45], [46] has shown that Cloud application workload usually follows a Poisson distribution, hence we assume that the exogenous arrival streams are Poisson processes. The MS request either returns to the system as a class $m^{'}$ request with a probability $p_{m,m'}$ or it completes with probability $1 - \sum_{l \in \mathcal{M}_a} p_{m,l}$. For example, consider a user on an e-commerce website, she/he will continuously sends requests to the server, for instance at first a login request, then with a certain probability she/he will may buy something (a buy request) or just search some product (a browse request). This probability depends from the starting and final type of the request, for example if the user sends a logout request then the probabilities are $p_{logout,m} = 0 \; \forall m \in \mathcal{M}_a$ and the process completes with probability 1. Let $\tilde{\Lambda}_{a,m}$ denote the external rate of arrivals for the microservice m requests, $\Lambda_{a,m} = \sum_{l \in \mathcal{M}_a} \Lambda_{a,l} p_{l,m} + \tilde{\Lambda}_{a,m}$, where $\Lambda_{a,m}$ denotes the aggregate rate of arrivals for the microservice m (see Figure 1). Note that, for simplicity and clarity, we assume that the terms $\tilde{\Lambda}_{a,m}$ and $\sum_{l \in \mathcal{M}_a} \Lambda_{a,l} p_{l,m}$ are implicitly embedded in $\Lambda_{a,m}$. As such, these terms are not explicitly referenced in the

remainder of the paper. Multiple VMs can run in parallel to support the same MS. In that case, we suppose, for the sake of simplicity, that the running VMs are homogeneous in terms of RAM and CPU capacity and the workload is evenly shared among multiple instances (see Figure 1), which is common for current Cloud solutions [47].

For the IaaS provider we consider a pricing model similar to the one previously adopted by Amazon AWS and Microsoft Azure [9], [33], [48]. The IaaS providers offers are as follows:

1) *reserved* VMs: SaaS providers apply for a one-time payment for each instance they want to reserve.
2) *on demand* VMs: allows SaaS to access computing capacity without long-term commitments.
3) *on spot* VMs: spawned by a possible unused IaaS capacity and for which SaaS providers bid and compete.

We denote with ρ the time unit cost of reserved VMs. The on spot cost σ is set by the IaaS provider and fluctuates periodically depending on the IaaS time of the day energy costs and also on the supply and demand from SaaS for on spot VMs [49], it could be smaller or larger than ρ. Indeed, SaaS providers compete for the use of on spot VMs specifying the maximum cost $\bar{\sigma}_j$ they are willing to pay per instance hour. The IaaS can also decide not to allocate any on spot instance to a SaaS. Finally, we denote with δ the cost for on demand VMs, δ is strictly greater than ρ, and we assume $\bar{\sigma}_j \leq q_j \rho$ for all j, with $q_j < 1$. Indeed, since the IaaS provider can arbitrarily terminate on spot instances from a SaaS resource pool [49], no one is willing to pay for a less reliable resource a time unit cost higher than on demand instances that provide a higher availability level. We denote with η_j the maximum fraction of resources allocated as on spot VMs for SaaS $j \in \mathcal{S}$ to allow a reasonable reliability for every application a. Finally, the number of reserved VMs for each SaaS provider j, denoted by $\bar{r}_j$, is guaranteed to access through on-time payment, while the total number of VMs available at the IaaS cloud service center is denoted by N.

The goal of SaaS provider j is to determine the number of reserved r_a, on demand d_a, and on spot s_a VMs every hour to be devoted to the execution of all $\mathcal{A}_j$ applications, in order to minimize the operating costs and, at the same time, to satisfy the prediction $\Lambda_{a,m}$ of the arrival rate of the microservice m exposed by application $a \in \mathcal{A}_j$. Let us denote with $\mu_{a,m}$ the maximum service rate for the microservice m requests of application a. If the workload is evenly shared among the VMs, then the average response time for the execution of the microservice m requests is given by:

$$E\left[R_{a,m}\right] = D_{a,m} + \frac{\frac{1}{\mu_{a,m}}}{1 - \sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}(r_a + d_a + s_a)}}, \quad (1)$$

where $D_{a,m}$ denotes the queuing network delay (see Figure 1) and we further assume the VMs are not saturated (i.e., the equilibrium conditions for the M/G/1 queues hold, $\sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}(r_a + d_a + s_a)} < 1$). $X_{a,m}$ is the throughput (acceptance rate) for microservice m of the application a and we have $X_{a,m} \leq \Lambda_{a,m}$, because SaaS providers can decide of accepting or rejecting microservice requests to minimize costs (trade-off between platform cost and rejection penalties [40]), assuming that such decisions are taken according to some independent and identically distributed (i.i.d.) probabilistic law. SaaS providers may incur in penalties $\nu_{a,m} \geq 0$ upon rejection of request executions of the microservice m. To guarantee a minimum availability, SaaSs have to satisfy a minimum throughput $\lambda_{a,m}$.

It is important to note that, parameters $\mu_{a,m}$, $D_{a,m}$ and $\Lambda_{a,m}$ are subject to uncertainty, as they are derived from predictions of future requests and run-time monitoring [22], [33], [48]. Therefore, we assume that the realizations of these parameters lie within a symmetric interval around a nominal value. The main goal of this work is to find a solution that remains feasible even under the worst-case realization of these parameters.

IV. ROBUST GAME FORMULATION

In this section, the robust game formulation is presented. First, we model and analyze the SaaS problem in Section IV-A. Then, we provide the robust formulation of the SaaS Problem subject to Γ-Robustness in Section IV-B. Finally, Section IV-C presents the IaaS problem formulation.

A. SAAS PROBLEM

Each SaaS provider j aims to minimize its cost function:

$$\min_{X,y,r,d} \sum_{a \in \mathcal{A}_j} \left[\rho r_a + \delta d_a + \sigma s_a + T \sum_{m \in \mathcal{M}_a} \nu_{a,m} y_{a,m} \right]$$

where

$$y_{a,m} \geq \Lambda_{a,m} - X_{a,m} \quad \text{and} \quad y_{a,m} \geq 0.$$

The variable $y_{a,m}$ represents the rejection rate for the microservice m included in application a. The cost function includes the fees requested by the IaaS for instances used while the last term represents the penalties incurred when requests are discarded. It is the sum over all the services of the application a of the penalty costs $\nu_{a,m} y_{a,m}$ multiplied by the time control horizon, e.g., $T = 600$ seconds. Note that σ and s_a are IaaS variables (see Section IV-C) hence, they could be deleted from the objective function of the SaaS since they are constants for it but we will keep them just to quantify the overall cost for the SaaS.

Let us consider now the SaaS constraints. The first constraint ensures that the response time of each MS m in application a, given by (1), is lower than the threshold $\bar{R}_{a,m}$ established in the SLA contract. We denote with:

$$A_{a,m} = \bar{R}_{a,m} - D_{a,m}.$$

$A_{a,m}$ is the maximum response time allowed for microservice m of application a. Since the threshold $\bar{R}_{a,m}$ must be always greater than the queuing network delay $D_{a,m}$, we get $A_{a,m} > 0$ and if we replace $E\left[R_{a,m}\right]$ in (1) with $\bar{R}_{a,m}$, we have:

$$A_{a,m} \geq \frac{\frac{1}{\mu_{a,m}}}{1 - \sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}(r_a + d_a + s_a)}}.$$

Then, after some manipulations, we have

$$A_{a,m} - \frac{A_{a,m}}{(r_a + d_a + s_a)} \sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}} \geq \frac{1}{\mu_{a,m}},$$

that is equivalent to

$$\frac{A_{a,m}\mu_{a,m} - 1}{\mu_{a,m}} \geq \frac{A_{a,m}}{(r_a + d_a + s_a)} \sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}}.$$

Finally, we can obtain:

$$r_a + d_a + s_a \geq \left[\frac{A_{a,m}\mu_{a,m}}{A_{a,m}\mu_{a,m} - 1}\right] \sum_{l \in \mathcal{M}_a} \frac{X_{a,l}}{\mu_{a,l}} \quad \forall a \in \mathcal{A}_j, \forall m \in \mathcal{M}_a. \quad (2)$$

The second constraint guarantees that resources are not saturated, i.e. the equilibrium condition for the M/G/1 queues:

$$r_a + d_a + s_a > \sum_{m \in \mathcal{M}_a} \frac{X_{a,m}}{\mu_{a,m}} \qquad \forall a \in \mathcal{A}_j. \quad (3)$$

The other constraints are:

$$X_{a,m} \geq \lambda_{a,m} \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (4)$$

$$y_{a,m} + X_{a,m} \geq \Lambda_{a,m} \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (5)$$

$$\sum_{a \in A_j} r_a \leq \bar{r}_j \quad (6)$$

$$\sum_{a \in \mathcal{A}} (r_a + d_a) \leq N - \sum_{a \in \mathcal{A}} s_a \quad (7)$$

$$\bar{\sigma}_j \leq q_j \rho \quad (8)$$

$$\bar{s}_a \leq \frac{\eta_j}{1 - \eta_j} (r_a + d_a) \qquad \forall a \in A_j \quad (9)$$

$$r_a, d_a, \bar{s}_a, X_{a,m}, y_{a,m} \geq 0 \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (10)$$

Constraints (4) state that the throughput is greater than or equal to the minimum throughput guaranteed by the SaaS. However, since in our model the predicted workload is affected by uncertainty, the predicted throughput could turn out to be greater than $\Lambda_{a,m}$ for some realizations (constraints (5)). For this reason, we imposed $y_{a,m} \geq 0$ in (10). Constraint (6) imposes that the sum of the reserved VMs to allocate to all the applications cannot be greater than the quantity established by the contract. Constraint (7) entails that the allocated VMs are less or equal to the maximum number offered or guaranteed by the IaaS provider, note also that this constraint is a link between different SaaSs. Constraint (8) defines an upper bound for the price the SaaS j is willing to pay to the IaaS for on spot VMs. Finally, constraints (9) is introduced for fault tolerance reasons: since the on spot VMs can be terminated by the IaaS at each instant, it guarantees that the on spot instances are at most a fraction $\eta_j < 1$ of the total capacity allocated for application a at IaaS.

Notice that the variables $\bar{\sigma}_j$ and $\bar{s}_a$ are not included in the SaaS objective function but they have a fundamental role in the IaaS problem. Moreover, they are used in the variant of the SaaS problem involved in the best reply algorithm used to reach the GNE of the game, we will go deeply into this topic in Section V.

System parameters

$\mathcal{S}$	Set of SaaS providers
$\mathcal{A}$	Set of applications of all the SaaS providers
$\mathcal{A}_j$	Set of applications of the SaaS provider j
$\mathcal{M}_a$	Set of MSs provided by application a
$\Lambda_{a,m}$	Overall prediction of the arrival rate for $m \in \mathcal{M}_a$
$\tilde{\Lambda}_{a,m}$	The external rate of arrivals for the microservice m requests
$\lambda_{a,m}$	Minimum arrival rate to be guaranteed for $m \in \mathcal{M}_a$
$\mu_{a,m}$	Maximum service rate for executing $m \in \mathcal{M}_a$
$D_{a,m}$	Queueing delay for executing $m \in \mathcal{M}_a$
$\bar{R}_{a,m}$	Average response time threshold for $m \in \mathcal{M}_a$
$R_{a,m}$	Average response time for $m \in \mathcal{M}_a$
$p_{m,m'}$	The probability that an incoming request m return to the system as request m'
$\nu_{a,m}$	Penalty for rejecting a single request of $m \in \mathcal{M}_a$
ρ	Time unit cost for reserved VMs
δ	Time unit cost for on demand VMs
q_j	Maximum fraction of reserved VMs price for on spot VMs price that SaaS provider j is willing to pay
ω	VM time unit energy cost for IaaS provider
η_j	Maximum fraction of total resources allocated as on spot VMs for SaaS provider j
N	Maximum number of VMs that can be executed at IaaS
$\bar{r}_j$	Maximum number of reserved VMs for SaaS j
T	Control time horizon

SaaS Decision Variables

r_a	Number of reserved VMs used for application a
d_a	Number of on demand VMs used for application a
$\bar{s}_a$	Number of desired on spot VMs for application a
$X_{a,m}$	Throughput for m provided by application a
$y_{a,m}$	Rejection rate for m provided by application a
$\bar{\sigma}_j$	Time unit cost threshold for SaaS j for on spot VM instances

IaaS Decision Variables

s_a	Number of on spot VMs used for application a
σ	Time unit cost offered for on spot VM instances
z_j	1 if IaaS gives on spot VMs to SaaS j; 0 otherwise

TABLE 1: Parameters and decision variables.

Notice that $\bar{R}_{a,m} \gg D_{a,m}$ and $\bar{R}_{a,m} \gg 1/\mu_{a,m}$, i.e., the QoS threshold need to be higher than the queueing network delay $D_{a,m}$ and MS requests service time $1/\mu_{a,m}$. Thus, we can assume that $\bar{R}_{a,m} > D_{a,m} + 1/\mu_{a,m}$, i.e., $A_{a,m}\mu_{a,m} - 1 > 0$. As a consequence, $\frac{A_{a,m}\mu_{a,m}}{A_{a,m}\mu_{a,m}-1} > 1$ holds and constraint (2) is stronger than constraint (3), which can be dropped from the model. Hence, the constraint (2) ensures both the response time is lower than the threshold and the resources are not saturated.

Finally, we note that variables r_a, d_a and $\bar{s}_a$ are assumed to be continuous instead of integer, as in reality they are, since requiring integer variables makes the solution much more difficult (NP-hard). However, if the optimal values of the variables are fractional and they are rounded to the closest integer solution, the gap between the solution of the real integer problem and the relaxed one will be very small (see, e.g., [33]).

B. ROBUST SAAS PROBLEM

As we already mentioned in Section III, parameters $\mu_{a,m}$, $D_{a,m}$ and $\Lambda_{a,m}$ are subject to uncertainty and may vary unpredictably. Traditional deterministic optimization ap-

proaches can lead to infeasible solutions when such variations occur. To address this problem, we follow the so-called cardinality-constrained robust optimization approach developed in [23]. This approach provides a structured way to deal with uncertainty by assuming that at most Γ uncertain parameters can simultaneously deviate from their nominal values. In other words, instead of considering every possible realization of uncertainty (which may be overly conservative), we define a robustness budget Γ that limits the number of uncertain parameters that can change simultaneously. This approach effectively balances robustness and optimality, so that by choosing an appropriate value for Γ, we control the level of protection against uncertainty while avoiding excessive conservatism. This approach also allows for a tractable reformulation of the constraints containing uncertain parameters. Using results from linear programming duality, it is possible to express robust constraints as a system of linear inequalities with additional auxiliary variables. This ensures that the resulting robust formulation remains solvable using standard optimization techniques. Specifically, if we define the uncertain parameters $\mathbf{B} = (B^l_{a,m})$ as follows:

$$B^l_{a,m} := \frac{A_{a,m}\mu_{a,m}}{(1 - A_{a,m}\mu_{a,m})\mu_{a,l}} \quad \forall a \in A_j, \forall l, m \in \mathcal{M}_a,$$

then the SaaS problem can be written as:

$$\min_{X,y,r,d} \sum_{a\in\mathcal{A}_j} \left[\rho r_a + \delta d_a + \sigma s_a + T \sum_{m\in\mathcal{M}_a} \nu_{a,m} y_{a,m}\right] \tag{P1a}$$

subject to

$$r_a + d_a + \sum_{l\in\mathcal{M}_a} B^l_{a,m} X_{a,l} \geq -s_a \qquad \forall a \in \mathcal{A}_j, \forall m \in \mathcal{M}_a \tag{P1b}$$

$$X_{a,m} \geq \lambda_{a,m} \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P1c}$$

$$y_{a,m} + X_{a,m} \geq \Lambda_{a,m} \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P1d}$$

$$\sum_{a\in A_j} r_a \leq \bar{r}_j \tag{P1e}$$

$$\sum_{a\in\mathcal{A}} (r_a + d_a) \leq N - \sum_{a\in\mathcal{A}} s_a \tag{P1f}$$

$$\bar{\sigma}_j \leq q_j \rho \tag{P1g}$$

$$\bar{s}_a \leq \frac{\eta_j}{1-\eta_j}(r_a + d_a) \qquad \forall a \in A_j \tag{P1h}$$

$$r_a, d_a, \bar{s}_a, X_{a,m}, y_{a,m} \geq 0 \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P1i}$$

Since uncertain parameters $\Lambda_{a,m}$ are present only in the right-hand-side of (P1d), the robust version of constraint (P1d) is simply

$$y_{a,m} + X_{a,m} \geq \Lambda^*_{a,m} \qquad \forall a \in A_j, \forall m \in \mathcal{M}_a,$$

where

$$\Lambda^*_{a,m} = \max_{\Lambda_{a,m}\in V_{a,m}} \Lambda_{a,m}$$

is the worst-case value of $\Lambda_{a,m}$ over the uncertainty set $V_{a,m}$.

We now consider the uncertain parameters $B^l_{a,m}$. We assume that each $B^l_{a,m}$ takes value in a symmetric interval

$$\left[\bar{B}^l_{a,m} - \hat{B}^l_{a,m},\ \bar{B}^l_{a,m} + \hat{B}^l_{a,m}\right]$$

defined by a maximum deviation $\hat{B}^l_{a,m}$ from a nominal value $\bar{B}^l_{a,m}$. Moreover, for any $a \in \mathcal{A}_j$ and $m \in \mathcal{M}_a$, we assume that at most Γ parameters $B^l_{a,m}$ can deviate from the nominal value at the same time, where Γ is a given positive integer. We denote the vector $\mathbf{B}_{a,m} = \left(B^l_{a,m}\right)_{l\in\mathcal{M}_a}$ and

$$\begin{aligned} U_{a,m} := \Big\{ \mathbf{B}_{a,m} :\ & \exists\, \gamma \subseteq \mathcal{M}_a \text{ such that } |\gamma| \leq \Gamma, \\ & B^l_{a,m} \in \left[\bar{B}^l_{a,m} - \hat{B}^l_{a,m},\ \bar{B}^l_{a,m} + \hat{B}^l_{a,m}\right] \ \forall l \in \gamma, \\ & B^l_{a,m} = \bar{B}^l_{a,m}\ \forall l \notin \gamma \Big\} \end{aligned}$$

the uncertainty set corresponding to constraint (P1b).

The robust version of constraint (P1b) requires that it holds for any realization of parameters $\mathbf{B}_{a,m}$ in the uncertainty set $U_{a,m}$, i.e., the inequality

$$r_a + d_a + \min_{\mathbf{B}_{a,m}\in U_{a,m}} \left\{ \sum_{l\in\mathcal{M}_a} B^l_{a,m} X_{a,l} \right\} \geq -s_a \tag{11}$$

holds for any $a \in \mathcal{A}_j$ and $m \in \mathcal{M}_a$. Due to the definition of the uncertainty set $U_{a,m}$, the values of $B^l_{a,m}$ can be decomposed into a nominal part (the one that contributes to the first sum) and an uncertain part (the second sum) that must be minimized. Hence, (11) is equivalent to

$$\begin{aligned} & r_a + d_a + \sum_{l\in\mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} \\ & \quad + \min_{\gamma\subseteq\mathcal{M}_a:\ |\gamma|\leq\Gamma} \left\{ \sum_{l\in\gamma} (-\hat{B}^l_{a,m}) X_{a,l} \right\} \geq -s_a. \end{aligned} \tag{12}$$

The minimization problem in (12) can be equivalently written as the following binary linear programming problem:

$$\begin{aligned} \min_{u}\ & \sum_{l\in\mathcal{M}_a} (-\hat{B}^l_{a,m}) X_{a,l} u^l_{a,m} \\ \text{s.t.}\ & \sum_{l\in\mathcal{M}_a} u^l_{a,m} \leq \Gamma \\ & u^l_{a,m} \in \{0,1\} \qquad \forall l \in \mathcal{M}_a. \end{aligned}$$

The latter problem is equivalent to a knapsack problem where all variables have unit weights. Hence, it is equivalent to its continuous relaxation

$$\begin{aligned} \min_{u}\ & \sum_{l\in\mathcal{M}_a} (-\hat{B}^l_{a,m}) X_{a,l} u^l_{a,m} \\ \text{s.t.}\ & \sum_{l\in\mathcal{M}_a} u^l_{a,m} \leq \Gamma \\ & 0 \leq u^l_{a,m} \leq 1 \qquad \forall l \in \mathcal{M}_a. \end{aligned}$$

The Linear programming duality theory guarantees that the optimal value of the latter problem is equal to the optimal value of its dual problem, i.e.,

$$\begin{aligned} \max_{\alpha,\beta}\ & \left[-\Gamma\alpha_{a,m} - \sum_{l\in\mathcal{M}_a} \beta^l_{a,m} \right] \\ \text{s.t.}\ & \alpha_{a,m} + \beta^l_{a,m} \geq \hat{B}^l_{a,m} X_{a,l} \qquad \forall l \in \mathcal{M}_a \\ & \alpha_{a,m} \geq 0 \\ & \beta^l_{a,m} \geq 0 \qquad \forall l \in \mathcal{M}_a. \end{aligned}$$

Hence, inequality (12) is equivalent to

$$\max_{\substack{\alpha_{a,m}+\beta^l_{a,m}\geq\hat{B}^l_{a,m}X_{a,l}\ \forall l\in\mathcal{M}_a \\ \alpha_{a,m}\geq 0 \\ \beta^l_{a,m}\geq 0\ \forall l\in\mathcal{M}_a}} \left[-\Gamma\alpha_{a,m} - \sum_{l\in\mathcal{M}_a} \beta^l_{a,m} \right]$$

$$+ r_a + d_a + \sum_{l \in \mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} \geq -s_a.$$

Since the operator max is redundant in the latter inequality due to the sign $\geq$, the robust version of constraint (P1b) is equivalent to the following system of linear inequalities:

$$r_a + d_a + \sum_{l \in \mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} - \Gamma \alpha_{a,m} - \sum_{l \in \mathcal{M}_a} \beta^l_{a,m} \geq -s_a \quad (13)$$

$$\alpha_{a,m} + \beta^l_{a,m} \geq \hat{B}^l_{a,m} X_{a,l} \quad \forall l \in \mathcal{M}_a \quad (14)$$

$$\alpha_{a,m} \geq 0 \quad (15)$$

$$\beta^l_{a,m} \geq 0 \quad \forall l \in \mathcal{M}_a \quad (16)$$

Therefore, the robust version of the SaaS j problem can be formulated as follows:

$$\min \sum_{a \in A_j} \left[\rho r_a + \delta d_a + \sigma s_a + T \sum_{m \in \mathcal{M}_a} \nu_{a,m} y_{a,m} \right] \quad (P2a)$$

subject to:

$$r_a + d_a + \sum_{l \in \mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} - \Gamma \alpha_{a,m} - \sum_{l \in \mathcal{M}_a} \beta^l_{a,m} \geq -s_a \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \quad (P2b)$$

$$\alpha_{a,m} + \beta^l_{a,m} \geq \hat{B}^l_{a,m} X_{a,l} \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \quad (P2c)$$

$$\alpha_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (P2d)$$

$$\beta^l_{a,m} \geq 0 \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \quad (P2e)$$

$$X_{a,m} \geq \lambda_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (P2f)$$

$$y_{a,m} + X_{a,m} \geq \Lambda^*_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (P2g)$$

$$\sum_{a \in A_j} r_a \leq \bar{r}_j \quad (P2h)$$

$$\sum_{a \in \mathcal{A}} (r_a + d_a) \leq N - \sum_{a \in \mathcal{A}} s_a \quad (P2i)$$

$$\bar{\sigma}_j \leq q_j \rho \quad (P2j)$$

$$\bar{s}_a \leq \frac{\eta_j}{1 - \eta_j} (r_a + d_a) \quad \forall a \in A_j \quad (P2k)$$

$$r_a, d_a, \bar{s}_a, X_{a,m}, y_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \quad (P2l)$$

Constraints (P2b)-(P2e) represent the robust reformulation of constraint (P1b), where the uncertain parameters $B^l_{a,m}$ are decomposed into a nominal part ($\bar{B}^l_{a,m}$) and an uncertainty part ($\hat{B}^l_{a,m}$). This reformulation expresses constraint (P1b) as a system of linear inequalities, enabling the incorporation of uncertainty in a structured and tractable manner. The remaining constraints are equivalent to constraints (4)-(10) as described in detail in Section IV-A. Therefore, the robust version of the SaaS problem is a linear programming problem and can be solved effectively by commercial solvers, such as Gurobi [50]. However, the IaaS problem, as detailed in the following section, is formulated as a MINLP. Due to the inherent complexity of this problem, Gurobi struggles to scale effectively, often requiring several days to solve large-scale instances. Consequently, Gurobi was not used as a benchmark for evaluating the quality of our proposed solution.

C. IAAS PROBLEM

The IaaS goal is to maximize its revenue of the on spot market, hence it must solve the following optimization problem:

$$\max_{s,z,\sigma} \sum_{a \in A} (\sigma - \omega) s_a \quad (P3a)$$

subject to:

$$\sum_{a \in \mathcal{A}} s_a \leq N - \sum_{a \in \mathcal{A}} (r_a + d_a) \quad (P3b)$$

$$\sigma \geq \omega \quad (P3c)$$

$$\sigma - \bar{\sigma}_j \leq M(1 - z_j) \quad \forall j \in \mathcal{S} \quad (P3d)$$

$$\bar{\sigma}_j - \sigma \leq M z_j \quad \forall j \in \mathcal{S} \quad (P3e)$$

$$s_a \leq \bar{s}_a z_j \quad \forall j \in \mathcal{S}, \forall a \in \mathcal{A}_j \quad (P3f)$$

$$s_a \geq 0 \quad \forall a \in \mathcal{A} \quad (P3g)$$

$$z_j \in \{0, 1\} \quad \forall j \in \mathcal{S} \quad (P3h)$$

To reach the goal, IaaS has to periodically set the price σ for on spot VMs, taking into account the energy cost ω (P3a). Moreover, it has to decide the number s_a of on spot VMs to allocate for the execution of any application a.

Constraint (P3b), shared with all the SaaS providers, entails that the total number of on spot resources allocated does not exceed the number of available VMs. Constraint (P3c) guarantees that the on spot price is bigger or equal to the energy cost to avoid negative revenue for the IaaS. Constraints (P3d), (P3e) and (P3f) impose that the on spot VMs are given to SaaS j if and only if it can pay at least σ for them. In particular, (P3f) guarantees that the on spot VMs s_a allocated for any application a are less than or equal to the number $\bar{s}_a$ of desired on spot VMs for its execution.

V. SOLUTION

In this section, we present our solution approach to address the robust game. First, we propose an ad-hoc algorithm to solve the IaaS problem in Section V-A, followed by a general algorithm for solving the robust game in Section V-B.

A. SOLVING THE IAAS PROBLEM

The IaaS formulation is a mixed-integer nonlinear programming problem, where the objective function is neither convex nor concave due to the bilinear term σs_a. However, the special structure of the problem makes it possible to develop an efficient algorithm to find the global optimal solution (see [48]).

Algorithm 1 starts sorting the SaaS providers so that the offered prices $\bar{\sigma}_j$ are in decreasing order (step 1) and determines the number S_{avail} of on spot VMs that the IaaS can offer (step 2). During the execution of the algorithm, σ^* represents the optimal price for the IaaS, $\mathcal{R}^*$ the corresponding optimal revenue, S_{sell} the total number of on spot VMs sold by the IaaS, and t is an index to iterate among the SaaSs. These four values are initialized in step 3. Steps 4-9 find the optimal price σ^*. First, the IaaS computes the maximum number S_{sell} of on spot VMs that it can sell to the first t SaaSs (equal to the minimum between the available capacity S_{avail} and the total number of on spot VMs requested by the first t SaaSs) and the corresponding revenue $\mathcal{R} = \bar{\sigma}_t S_{sell}$ (steps 5-6). Then, σ^* and $\mathcal{R}^*$ are updated accordingly, if

Algorithm 1: Solving the IaaS problem

1 Sort SaaS providers so that $\bar{\sigma}_1 \geq \bar{\sigma}_2 \geq \cdots \geq \bar{\sigma}_{|\mathcal{S}|}$
2 $S_{avail} = N - \sum_{a \in \mathcal{A}} (r_a + d_a)$
// find the optimal price σ^*
3 $\sigma^* = \infty, \mathcal{R}^* = 0, S_{sell} = 0, t = 1$
4 **while** $S_{sell} < S_{avail}$ *and* $t \leq |\mathcal{S}|$ **do**
5 $S_{sell} = \min \left\{ S_{avail}, \sum_{j=1}^{t} \sum_{a \in \mathcal{A}_j} \bar{s}_a \right\}$
6 $\mathcal{R} = \bar{\sigma}_t \, S_{sell}$
7 **if** $\mathcal{R} > \mathcal{R}^*$ **then**
8 $\sigma^* = \bar{\sigma}_t, \mathcal{R}^* = \mathcal{R}$
9 $t = t + 1$
// find the optimal values z^* and s^*
10 **for** $j \in \mathcal{S}$ **do**
11 **if** $\bar{\sigma}_j \geq \sigma^*$ **then**
12 $z_j^* = 1$
13 **for** $a \in \mathcal{A}_j$ **do**
14 $s_a^* = \min \{\bar{s}_a, S_{avail}\}$
15 $S_{avail} = S_{avail} - s_a^*$
16 **else**
17 $z_j^* = 0$
18 $s_a^* = 0 \quad \forall a \in \mathcal{A}_j$

necessary (step 8). While the IaaS capacity is not saturated ($S_{sell} < S_{avail}$) and there are still SaaS providers to consider ($t \leq |\mathcal{S}|$), steps 4-9 are repeated; otherwise σ^* is the optimal cost. The optimal values of z_j^* and s_a^* are assigned in steps 10-18 according to the price offered by SaaS providers and their MS applications requirements.

Notice that the time complexity of Algorithm 1 is $\mathcal{O}(\max\{|\mathcal{S}| \log(|\mathcal{S}|), |\mathcal{A}|\})$ because of the sorting at step 1 and the assignment of s_a at step 14.

B. SOLVING THE ROBUST GAME

We now present the algorithm to find a generalized Nash equilibrium of the robust game (Algorithm 2).

In step 1, the price σ is initialized as 10% higher than the cost of the energy. In steps 2-4, for any SaaS provider j, we solve a variant of the robust problem (P2), called *SaaS_minimal_resources[j]*:

$$\min \sum_{a \in A_j} \left[\rho r_a + \delta d_a + \sigma s_a + T \sum_{m \in \mathcal{M}_a} \nu_{a,m} y_{a,m} \right] \tag{P4a}$$

subject to:

$$r_a + d_a + \sum_{l \in \mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} - \Gamma \alpha_{a,m} - \sum_{l \in \mathcal{M}_a} \beta^l_{a,m} \geq -s_a \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P4b}$$

$$\alpha_{a,m} + \beta^l_{a,m} \geq \hat{B}^l_{a,m} X_{a,l} \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P4c}$$

$$\alpha_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P4d}$$

$$\beta^l_{a,m} \geq 0 \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P4e}$$

$$X_{a,m} = \lambda_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P4f}$$

$$y_{a,m} + X_{a,m} \geq \Lambda^*_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P4g}$$

$$\sum_{a \in A_j} r_a \leq \bar{r}_j \tag{P4h}$$

$$\sum_{a \in \mathcal{A}} (r_a + d_a) \leq N - \sum_{a \in \mathcal{A}} s_a \tag{P4i}$$

$$\bar{\sigma}_j \leq q_j \rho \tag{P4j}$$

$$\bar{s}_a \leq \frac{\eta_j}{1 - \eta_j} (r_a + d_a) \quad \forall a \in A_j \tag{P4k}$$

$$r_a, d_a, \bar{s}_a, X_{a,m}, y_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P4l}$$

$$s_a = 0 \quad \forall a \in A_j. \tag{P4m}$$

In this formulation, the throughput $X_{a,m}$ is not a variable but a parameter imposed equal to $\lambda_{a,m}$, representing the minimum arrival rate that the SaaS must process (see (P4f)). Moreover, only the reserved and on-demand VMs are considered, i.e., all the IaaS variables s_a are fixed to 0 (see (P4m)). The objective of this formulation is to verify that the minimal arrival rate can be guaranteed: in the positive case we proceed with the rest of the algorithm (steps 7-27) or else we notify that there are no sufficient resources (step 29).

If there are enough resources to solve the game (step 6), then for each SaaS j we set $\bar{\sigma}_j$ as half of the maximum offer that SaaS j can make (step 8). Then, in step 9, we solve a second variant of the robust problem (P2), called *Ideal_SaaS[j]*:

$$\min \sum_{a \in A_j} \left[\rho r_a + \delta d_a + \sigma \bar{s}_a + T \sum_{m \in \mathcal{M}_a} \nu_{a,m} y_{a,m} \right] \tag{P5a}$$

subject to:

$$r_a + d_a + \sum_{l \in \mathcal{M}_a} \bar{B}^l_{a,m} X_{a,l} - \Gamma \alpha_{a,m} - \sum_{l \in \mathcal{M}_a} \beta^l_{a,m} \geq -\bar{s}_a \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P5b}$$

$$\alpha_{a,m} + \beta^l_{a,m} \geq \hat{B}^l_{a,m} X_{a,l} \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P5c}$$

$$\alpha_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P5d}$$

$$\beta^l_{a,m} \geq 0 \quad \forall a \in A_j, \forall m, l \in \mathcal{M}_a \tag{P5e}$$

$$X_{a,m} \geq \lambda_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P5f}$$

$$y_{a,m} + X_{a,m} \geq \Lambda^*_{a,m} \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P5g}$$

$$\sum_{a \in A_j} r_a \leq \bar{r}_j \tag{P5h}$$

$$\sum_{a \in \mathcal{A}} (r_a + d_a) \leq N - \sum_{a \in \mathcal{A}} \bar{s}_a \tag{P5i}$$

$$\bar{s}_a \leq \frac{\eta_j}{1 - \eta_j} (r_a + d_a) \quad \forall a \in A_j \tag{P5j}$$

$$r_a, d_a, \bar{s}_a, X_{a,m}, y_{a,m} \geq 0 \quad \forall a \in A_j, \forall m \in \mathcal{M}_a \tag{P5k}$$

This latter problem coincides with (P2), but the constraint (P2j) is removed and the variable s_a controlled by the IaaS is replaced by the SaaS variable $\bar{s}_a$. In other words, the SaaS aims to compute the best strategy assuming that, for any application a, the desired number $\bar{s}_a$ of on spot VMs is equal to the actual number s_a of on spot VMs used.

In step 11, we check if the offer $\bar{\sigma}_j$ of the SaaS is lower than the current price σ of on spot instances: in the positive

case, we set the number of on spot VMs $s_a = 0$ for any $a \in \mathcal{A}_j$, because it means that the offer of the SaaS is lower than the current price set by the IaaS.

In step 13, we calculate the total number S^{tot} of desired on spot VMs by the SaaS providers which offered at least σ, and in the steps 14-15 we distribute the on spot VMs proportionally to the availability at the IaaS.

Then, we iteratively solve each SaaS problem (P2) (steps 20-21) and increase the time unit cost threshold $\bar{\sigma}_j$ by 1% for the SaaS whose desired number of spot instances exceeds the current number of spot instances and whose offer can be increased according to constraint (P2j) (steps 22-23). This approach seeks to enhance IaaS revenue while satisfying the SaaS constraints. Then, we solve the *Ideal_SaaS[j]* problem to compute the best strategy of SaaS j given the new $\bar{\sigma}_j$ (steps 24-26). Finally, we solve the IaaS Problem (P3) through Algorithm 1 which considers all the offers coming from the SaaS providers and establishes the new value of σ and the current number of on spot s_a for all applications. This procedure ends when either all the SaaSs are satisfied in terms of on spot instances, i.e., $s_a = \bar{s}_a$, or the unsatisfied SaaSs cannot increase their offer.

Algorithm 2: Game solution algorithm

1 $\sigma = 1.1\omega$
2 **for** $j \in \mathcal{S}$ **do**
3 Solve SaaS_minimal_resources[j]
4 Rounding of the solution
5 $\bar{N} = \sum_{a \in \mathcal{A}} (r_a + d_a)$
6 **if** $\bar{N} < N$ **then**
7 **for** $j \in \mathcal{S}$ **do**
8 $\bar{\sigma}_j = q_j \rho / 2$
9 Solve Ideal_SaaS[j]
10 Rounding of the solution
11 **if** $\bar{\sigma}_j < \sigma$ **then**
12 $s_a = 0 \quad \forall a \in \mathcal{A}_j$
13 $S^{tot} = \sum_{\substack{j \in \mathcal{S}: \\ \bar{\sigma}_j \geq \sigma}} \sum_{a \in \mathcal{A}_j} \bar{s}_a$
14 **for** $j \in \mathcal{S}$ *such that* $\bar{\sigma}_j \geq \sigma$ **do**
15 $s_a = \bar{s}_a \min\left\{1, \frac{N - \sum_{a \in \mathcal{A}} (r_a + d_a)}{S^{tot}}\right\} \ \forall a \in \mathcal{A}_j$
16 continue = 1
17 **while** *continue* **do**
18 continue = 0
19 **for** $j \in \mathcal{S}$ **do**
20 solve SaaS problem (P2)
21 Rounding of the solution
22 **if** $\exists a \in \mathcal{A}_j s.t. \bar{s}_a > s_a$ *AND* $\bar{\sigma}_j \leq \sigma \leq q_j \rho$ **then**
23 $\bar{\sigma}_j = \min\{1.01\bar{\sigma}_j,\ q_j \rho\}$
24 solve Ideal_SaaS[j]
25 Rounding of the solution
26 continue = 1
27 solve IaaS problem (P3) through Algorithm 1
28 **else**
29 Error : Insufficient Resources

VI. EXPERIMENTAL RESULTS

In this section, we present the setting of the parameters and all the results of the analyses performed to validate our game approach. In Section VI-A, we introduce all the parameters considered. Then, we demonstrate the impact of the parameter Γ on the model results (Section VI-B) and analyze the price of robustness and the number of iterations required to reach the equilibrium in SectionVI-C. We present the results of a daily analysis in SectionVI-D and evaluate the execution time of microservices through simulation by considering long-tail distributions in SectionVI-E. It is noteworthy that in our previous work [33], we compared the proposed game model (characterized by $\Gamma = 0$ since robustness was not a consideration) with threshold-based policies proposed by the literature [51]–[53]. These policies determine the number of VMs to guarantee a specified utilization target, a strategy also implemented by commercial IaaS providers such as AWS [54] The results demonstrated that the game approach resulted in superior social welfare for SaaS (approximately 42% higher Price of Anarchy) and a more favorable allocation for the IaaS provider, leading to cost savings of approximately 62%.

A. PARAMETERS SETTINGS

Microservice applications require efficient scalability, low response times, and dynamic resource allocation to handle varying loads effectively. To ensure our model captures these requirements, we set key parameters and randomly generated instances based on values observed in real-world microservices deployments, as outlined in previous studies [4], [32], [55]. We varied the number of SaaS providers and microservices, each equally ranging from 5 to 80.

Regarding the microservices parameters, as in [32] $\mu_{a,m}$ is selected randomly between 200 and 400 requests per second and $D_{a,m} \in [0.001, 0.05] sec$ (which is in line with the traces obtained in real systems [55]). The total number N of virtual machine provided by IaaS varied between 100 and 1000. We set the penalty for rejecting requests $\nu_{a,m}$ equal to 4.5×10^{-6}\$ and 9×10^{-6}\$ per rejected request (the latter value was considered for $|\mathcal{S}| > 20$, since with more SaaS providers the system is characterised by higher competition and we obtained too large rejection rates). Since the QoS threshold $\bar{R}_{a,m}$ must be higher than $D_{a,m}$ and $1/\mu_{a,m}$, we set $\bar{R}_{a,m} = 10/\mu_{a,m}$, as in [32] and with values similar to the ones reported in [4].

We generated two synthetic traces considering real logs from an Internet website, as shown in Figure 2, where the red trace represents a spiky day, while the blue trace represents a smoother, normal day. These traces are characterised by a

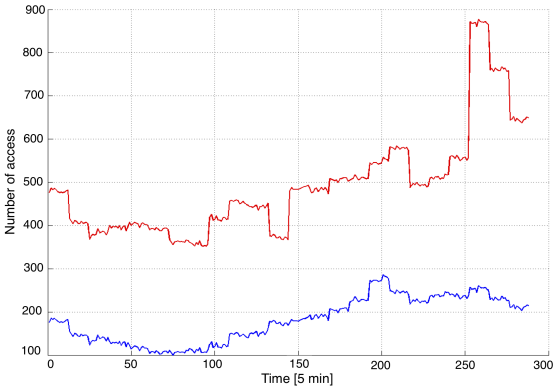


FIGURE 2: Arrivals Traces

Parameters	values
$\|\mathcal{S}\|$	5, 10, 20, 40, 50, 60, 70, 80
Number of apps per SaaS	1
$\|\mathcal{M}_a\|$ *per app*	5, 10, 20, 40, 50, 60, 70, 80
N	100, 500
Γ	0,1,2,3,4,5
$\mu_{a,m}$	$[200, 400]$ req/sec
$D_{a,m}$	$[0.001, 0.05]$ sec
$\nu_{a,m}$	4.5×10^{-6}\$, 4.5×10^{-5}\$
$\bar{R}_{a,m}$	$\frac{10}{\mu_{a,m}}$
$\lambda_{a,m}$	$0.8\Lambda_{a,m}$
$\bar{r}_j$	$\frac{1}{3}\frac{N}{\|\mathcal{S}\|}$
T	3600
ρ	$[0.015, 4]$\$/$hour$
q_j	$[0.25, 0.9]$
δ	4ρ
ω	$[0.05, 0.1]$\$/$hour$
$\hat{\mu}$	$[0.024, 0.42]\mu_{a,m}$
$\hat{D}$	$[0.024, 0.42]D_{a,m}$

TABLE 2: Parameters of experiments.

bi-modal distribution and are representative of real microservices deployments [55]). We randomly picked the peak loads of these traces and introduced random noise proportional to the load (±10% of the value at time t). In Section VI-D, we use these generated traces to monitor the trends of various quantities, such as the number of VMs used by the SaaS providers and the response times of microservices, over an entire day. Specifically, we will simulate the game for each hour of the day, using different values for the arrival rates of the microservices. Finally, we set $\lambda_{a,m} = 0.8\Lambda_{a,m}$ as in [33].

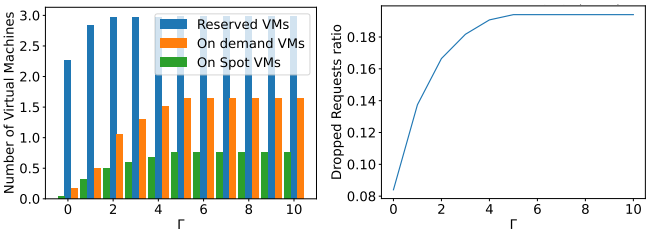
For what concerns the SaaS parameters, we set $\bar{r}_j = \frac{1}{3}\frac{N}{|\mathcal{S}|}$ while q_j is randomly set between 25% and 90% and $\eta_j = 0.25$.

The time horizon T is set to one hour. ρ is selected randomly between 0.015 and 4.00 \$/hour and $\delta = 4\rho$. The energy cost ω is selected randomly between [0.05, 0.1] \$/hour based on the cost in Italy in 2022 [56]. Finally, we randomly initialize the maximum deviation from the nominal values of the arrival rate and delay ($\hat{\mu}$ and $\hat{D}$) within the range of 2.4% to 42% considering the variability that can be achieved by several estimation methods as discussed in [22].

To enhance the robustness of the experiments, we generated 10 random instances using 10 different random seeds. All the results presented in the following sections are averaged over these 10 random instances. Table 2 summarizes the parameter values used in the experiments. These parameters allow us to model realistic microservice behaviors, including peak traffic loads, latency-sensitive responses, and elastic resource allocation, ensuring that our simulation reflects real-world application scenarios.

B. THE EFFECT OF Γ

In this section, we show how the SaaSs respond to parameter uncertainty by varying the value of Γ, which controls the number of uncertain parameters. For this introductory analysis we consider a particular settings with: $|\mathcal{S}| = 1$, $|\mathcal{M}_a| = 5$, $\Gamma = 0, \ldots, 10$ and $N = 10$ to study the effect of the deviation from the nominal value of the parameters. The results, reported in Figure 3 are averaged over 10 random



(a) Number of VMs (b) Rejected requests ratio

FIGURE 3: The effect of Γ for $|\mathcal{S}| = 1$.

instances. Figure 3a shows that enhancing the robustness of the solution leads to an increase in the number of VMs requested by the SaaS. Initially, the SaaS rarely instantiates on-spot and on-demand VMs, preferring to use the available reserved VMs. As Γ increases and the reserved VMs reach $\bar{r}_j$, the SaaS demands additional resources to protect against parameter uncertainty, leading to an increased number of on-spot and on-demand VMs.

Figure 3b shows the dropped request ratio, which is the number of rejected requests (i.e., $\sum_{a\in\mathcal{A}_j}\sum_{ms\in\mathcal{M}_a} y_{a,m}$) over the total requests $\Lambda_{a,m}$. Since the minimum arrival rate $\lambda_{a,m}$ is equal to $0.8\Lambda_{a,m}$, the dropped request ratio is always less than 0.2. Note that, the number of VMs and the dropped request ratio remain unchanged when $\Gamma \geq |\mathcal{M}_a|$.

C. THE PRICE OF ROBUSTNESS

In this section, we perform the analysis of the Price of Robustness (PoR), that we define, for a particular value of Γ, as follows:

$$PoR^{\Gamma=n} = \frac{\sum_{j\in\mathcal{S}} \Theta_j^{\Gamma=n}}{\sum_{j\in\mathcal{S}} \Theta_j^{\Gamma=0}} \qquad n = 0, \ldots, |\mathcal{M}_a|,$$

where Θ_j is the objective function (P1a) of SaaS j. This metric evaluates the losses incurred by the SaaSs at different levels of robustness compared to the non-robust solution, averaged over a 24-hour period and 10 random instances.

$\|\mathcal{S}\|$	$\|\mathcal{M}_a\|$	$\Gamma=1$	$\Gamma=2$	$\Gamma=3$	$\Gamma=4$	$\Gamma=5$
5	5	1.8450	2.4112	2.7316	2.9385	3.0881
10	10	1.4577	1.9501	2.4549	3.0075	3.5006
20	20	1.4043	1.7800	2.1215	2.4280	2.7101
30	30	1.1321	1.2469	1.3530	1.4445	1.5310
40	40	1.1020	1.1942	1.2813	1.3644	1.4422
50	50	1.0794	1.1624	1.2463	1.3324	1.4222
60	60	1.1310	1.2602	1.3965	1.5387	1.6716
70	70	1.1120	1.2206	1.3308	1.4489	1.5697
80	80	1.0984	1.1954	1.2927	1.3937	1.4984

TABLE 3: Price of Robustness, spiky (average values across a 24-hour period and 10 random instances).

$\|\mathcal{S}\|$	$\|\mathcal{M}_a\|$	$\Gamma=0$	$\Gamma=1$	$\Gamma=2$	$\Gamma=3$	$\Gamma=4$	$\Gamma=5$
5	5	1	5	20	24	26	28
10	10	8	24	38	47	51	55
20	20	2	4	6	10	14	16
30	30	5	9	14	22	25	31
40	40	5	9	12	18	21	24
50	50	6	8	10	15	18	19
60	60	4	4	4	6	8	13
70	70	4	4	4	6	8	12
80	80	4	4	6	10	15	20

TABLE 4: Game iterations, spiky load (average values across a 24-hour period and 10 random instances).

Results are reported in Table 3, under spiky traces. Given that spiky traces represent the most severe scenario and offer a more comprehensive assessment, while the outcomes for smooth traces are comparable, we have excluded the latter due to space constraints.

The average PoR increases with Γ since, as the protection levels rise, SaaSs need to allocate more instances and reject more requests. Regarding $|\mathcal{S}| > 20$, since we increased the rejection penalty by a factor of 2, SaaS providers strive to avoid rejections by utilizing more on-demand VMs. This results in a lower PoR compared to smaller cluster sizes. Similarly, the number of iterations required to achieve game equilibrium and the execution time to solve the game tend to increase with the number of Γ, as shown in Table 4 and Table 5, respectively.

It is worth noting that, given the high cost of robustness, SaaS providers should prioritize protecting only the most relevant microservices, such as the most profitable, the *gold* request types, or those most frequently used. Additionally, it is noteworthy that the execution time for solving a single problem of the largest scale—i.e., with 80 SaaS instances and 80 microservices—ranges from approximately 5 to 42 seconds as Γ varies from 0 to 5 (see last row in Table 5), which demonstrates the practical applicability of our proposed method in real-world scenarios.

D. DAILY ANALYSIS (ANALYTICAL RESULTS)

In this section, we analyze the metrics over a full 24-hour period to simulate a real-world scenario where SaaS platforms must handle requests continuously throughout the day. The metrics we aim to analyze include the number of allocated instances, the preemptively dropped requests, the price of robustness, and the throughput as calculated by the analytical

$\|\mathcal{S}\|$	$\|\mathcal{M}_a\|$	$\Gamma=0$	$\Gamma=1$	$\Gamma=2$	$\Gamma=3$	$\Gamma=4$	$\Gamma=5$
5	5	0.02	0.06	0.17	0.20	0.21	0.23
10	10	0.18	0.43	0.66	0.91	0.9	1.00
20	20	0.14	0.33	0.47	0.56	0.68	0.77
30	30	0.57	0.92	1.26	1.54	2.18	2.49
40	40	0.89	1.78	2.64	3.06	3.06	3.37
50	50	1.96	2.88	3.51	4.20	4.66	5.55
60	60	1.70	2.74	3.29	4.18	5.34	5.89
70	70	2.03	3.66	8.70	10.61	9.51	8.17
80	80	5.64	7.73	26.22	27.85	32.62	42.89

TABLE 5: Average execution time (seconds) to solve the game, spiky load (average values across a 24-hour period and 10 random instances).

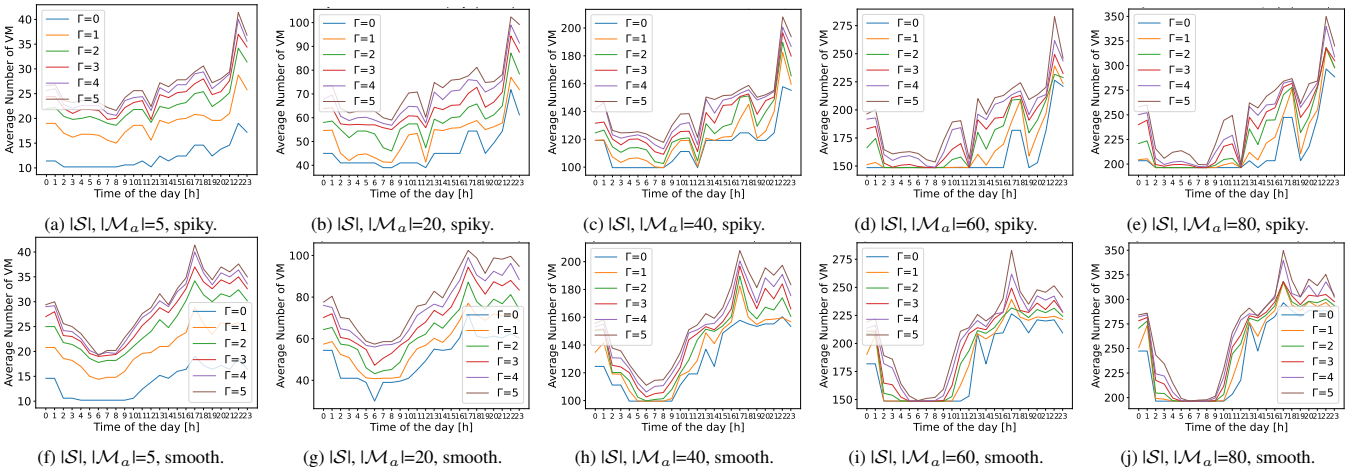
model. We considered up to 80 SaaSs, each with up to 80 microservices[1] and adjusted the arrival rate to follow the patterns shown in Figure 2 over the entire 24-hour period.

First, we analyzed the average total number of VMs used by the SaaS platforms in both the smooth and spiky scenarios, depicted in Figure 4. As anticipated, the number of VMs closely follows the arrival patterns, as an increase in job arrivals necessitates the allocation of additional VMs. Moreover, the number of VMs will increase with Γ, regardless of the time of day or the overall load. This is entirely reasonable, as increasing robustness requires SaaS platforms to instantiate more servers to safeguard against deviations in microservice parameters from their nominal values.

Next, we aim to verify whether these observations hold true for other metrics, such as throughput and the average number of dropped requests. As shown in Figure 5, the throughput closely mirrors the arrival rate, in both spiky and smooth traces. However, it decreases as Γ increases. This decline can be attributed to the fact that higher levels of robustness require each microservice to utilize more VMs (see Figure 4). This leads to increased costs for the SaaS platforms, prompting them to drop some requests. Additionally, we observe that the difference in acceptance rates between the highest levels of protection is smaller compared to the lower values of Γ.

As it is shown in Figure 6, the ratio of dropped requests rises as the SaaS platforms increase their level of protection. Notably, when robustness is not considered ($\Gamma = 0$), the SaaS platforms accept all incoming jobs if the reserved resources are sufficient or the penalty of rejection is small enough (see Figures 6f and 6a). However, as the number of SaaS platforms and microservices increases, the proportion of dropped requests rises, reaching a peak drop rate during most hours for higher Γ values, and even during peak hours when $\Gamma = 0$ (see Figures 6g and 6b). As mentioned in Section VI-A, we increased N and the penalty for rejection when $|\mathcal{S}| > 20$ to mitigate high rejection rates. This adjustment allowed SaaS platforms to once again accept all incoming jobs when $\Gamma = 0$ (see 6h and 6c). However, as the number of SaaS platforms and microservices continues to grow, we observe a return to high drop rates, particularly during peak hours (see 6i, 6d, 6j and 6e).

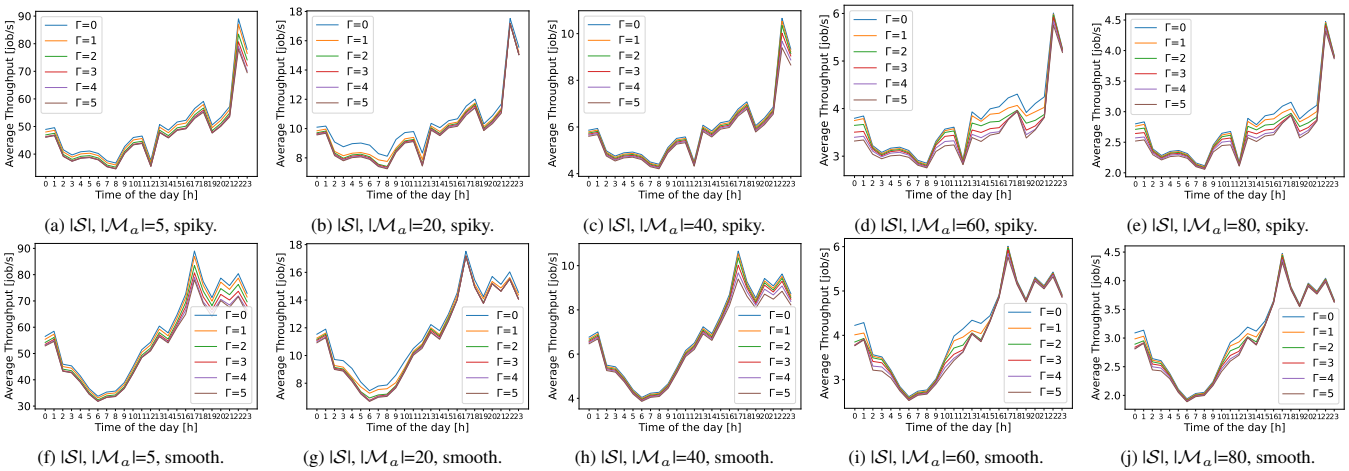
[1]Note that in production systems, e.g. at Alibaba [57], 95% of call graphs, i.e., application invocation patterns, include at most 80 micorservices



(a) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, spiky. (b) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, spiky. (c) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, spiky. (d) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, spiky. (e) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, spiky.

(f) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, smooth. (g) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, smooth. (h) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, smooth. (i) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, smooth. (j) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, smooth.

FIGURE 4: Daily server allocation, considering spiky and smooth load and varying number of SaaSs.



(a) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, spiky. (b) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, spiky. (c) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, spiky. (d) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, spiky. (e) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, spiky.

(f) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, smooth. (g) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, smooth. (h) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, smooth. (i) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, smooth. (j) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, smooth.

FIGURE 5: Daily average throughput, considering spiky and smooth load and varying number of SaaSs.

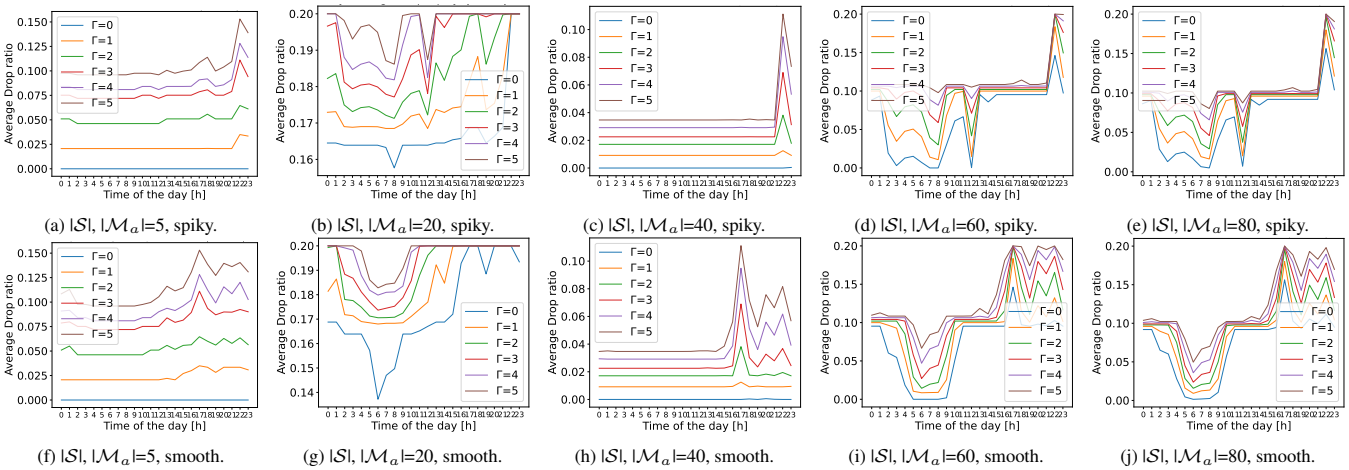
Figure 7 reports the price of robustness during the 24 hours. As mentioned in Section VI-B, the PoR increases with Γ, however, in certain instances, this metric does not strictly rise with the level of protection. These deviations are due to the discretization of the number of instances, which is a factor in the objective function of the SaaS providers.

As can be deduced from the aforementioned figures, even during peak periods, the behavior remains consistent: increasing the level of robustness leads to an increase in the average number of VMs used, the mean dropped requests per second, and the PoR, while the throughput decreases.

It is important to note that, to the best of our knowledge, no state-of-the-art methods account for the impact of uncertainty on both the execution time and service rate of each request, leaving no baseline that incorporates Γ as effectively as our approach. However, our approach enables the quantification of Γ and the associated price of robustness. While increasing Γ improves system utilization by proactively rejecting requests under unfavorable conditions, it also leads to higher costs. The results indicate that Γ values should be limited to one or two, as higher values significantly increase the cost of robustness without yielding substantial improvements in average throughput.

E. VALIDATION THROUGH SIMULATION

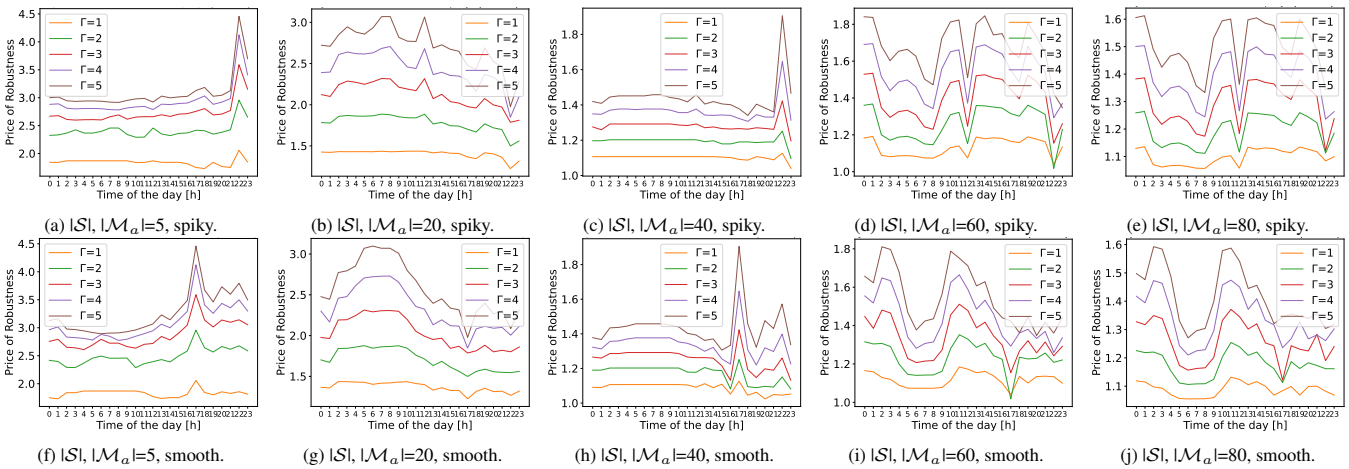
In the following analysis, we relied on a simulator for validating the game model solution and conducted a more in-depth exploration of the analytical model behavior. To mitigate prolonged simulation times, we limited our analysis to spiky workloads (see Figure 8) and reduced the number of SaaS instances and microservices. We performed a series of analyses using an increasing number of SaaSs (5, 10, 20, 40, 50), each configured with one application and 5 microservices. All experiments were carried out across the full range of Γ values (from 0 to 5). As in the previous section, we generated 10 random instances for all simulations and the results presented in the following sections are the averages across these 10 random instances. To ensure that the simulation results accurately represent a steady-state scenario, we removed the initial 10 minutes of the simulation data, which may exhibit transient behavior. We evaluated each instance over a 24-hour time horizon. At the largest



(a) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, spiky. (b) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, spiky. (c) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, spiky. (d) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, spiky. (e) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, spiky.

(f) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, smooth. (g) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, smooth. (h) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, smooth. (i) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, smooth. (j) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, smooth.

FIGURE 6: Average dropped ratio, considering spiky and smooth load and varying number of SaaSs.



(a) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, spiky. (b) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, spiky. (c) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, spiky. (d) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, spiky. (e) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, spiky.

(f) $|\mathcal{S}|$, $|\mathcal{M}_a|$=5, smooth. (g) $|\mathcal{S}|$, $|\mathcal{M}_a|$=20, smooth. (h) $|\mathcal{S}|$, $|\mathcal{M}_a|$=40, smooth. (i) $|\mathcal{S}|$, $|\mathcal{M}_a|$=60, smooth. (j) $|\mathcal{S}|$, $|\mathcal{M}_a|$=80, smooth.

FIGURE 7: Price of robustness, considering spiky and smooth load and varying number of SaaSs.

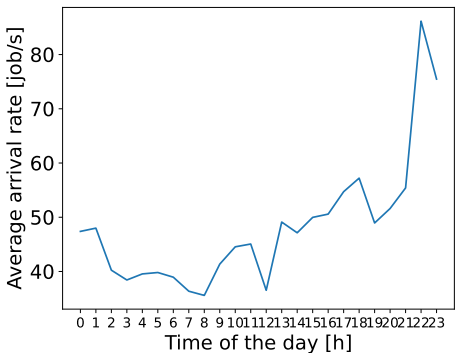


FIGURE 8: Average arrival rate in the simulator analysis.

scale each simulation required approximately one week to complete on a Linux server machine with 20- cores Intel(R) Xeon(R) CPU 2.20GHz and 48 GB memory. Note that all the SaaSs are different agents with different microservices, each having different arrival rates that were randomly assigned.

We analyzed two different scenarios. In the first scenario, we considered the results obtained by the game solution (based on the analytical model which response times estimates are exact only when service demands follow an exponential distribution), vice versa, in the second, more realistic scenario, the decisions taken by the game model are evaluated through the simulator which considers service demand following a lognormal distribution with a standard deviation equal to four times the mean value, as in [43]. Additionally, the buffer length was set to a finite size of 20 requests, in accordance with [32]. In case a request response time exceeds its threshold, it is dropped by the simulator. For space reasons we omit a sensitivity analysis with respect to the execution time threshold and to the request queue length, relying on values that have been taken from the scientific literature [32], [43]. The main simulation parameters are summarized in Tab.6

We analysed the number of VMs allocated, the drop ratio, the average number of jobs that time out and compared the average response time obtained through the analytical model with the one reported by the simulator. We expect all metrics to closely follow the arrival rate, as they are directly correlated. Fewer jobs to process result in shorter queues and,

Parameter	values
Repetitions	10
Confidence interval	95%
Warm-up	10 minutes
Simulation duration	24 hours
Execution time	lognormal CoV=4
Response time threshold	10 × Execution time
Mar requet queue	20

TABLE 6: Simulation Parameters

consequently, lower response times. Similarly, with fewer requests, the likelihood of jobs timing out decreases, and the SaaS instances have more computational resources available to handle each arrival efficiently.

The results in Figure 9 indicate that the average response time for each microservice request remains quite consistent across $\Gamma = 1, 2, 3, 4, 5$. When $\Gamma = 0$, where parameter uncertainty is entirely ignored, the response time obtained in simulation is slightly higher than for any other protection level. We justify this result since in the analytical model the buffer size is ignored and an accepted microservice request occupies resources until the end of its execution (note that in reality an individual request might exceed its threshold since the analytical model only captures the "average" system behaviour). In contrast, evaluating the game model results with the simulator, if a job arrives and all microservices are busy with full queues (each accommodating 20 requests in the limited buffer), the job is immediately dropped. Additionally, the requests under lognormal distribution have higher variability and the simulator discards requests after a timeout (set equal to the response time threshold). Consequently, the observed increase in the drop rate (see Figure 10) is expected and justified due to buffer size limitations and the resulting request rejection from the timeout. However, it is important to note that the differences among various levels of Γ are more pronounced. Specifically, the non-robust solution evaluated through simulation achieves a higher average drop rate (about twice) compared to the value that can be expected through the analytical model.

Despite the differences in outcomes between the analytical model and the simulation, the results remain favorable in both scenarios, with the violation rate consistently below 3% for all values of Γ except for $\Gamma = 0$. When accounting for robustness, the drop rate rapidly approaches zero and increasing the number of SaaS instances does not change it considerably. Unlike the response time, the differences between various levels of Γ are subtle but noticeable. Specifically, $\Gamma = 1$ consistently stands out from the others, while $\Gamma = 2$ and $\Gamma = 3$ tend to result in a higher violation rate compared to the highest levels of conservatism. It is also noteworthy that $\Gamma = 4$ and $\Gamma = 5$ are nearly indistinguishable from each other.

The average total number of VMs, shown in Figure 11, closely aligns with the arrival patterns, as an increase in job arrivals demands the allocation of additional VMs. Furthermore, the number of VMs will rise with Γ, irrespective of the time of day or the overall load.

We summarize the impact of guaranteeing stable performance in the presence of system parameter errors in Table 7. For each considered scenario and for each level Γ of protection, we show the worst case scenario that is the ratio between the maximum number of additional VMs required and the number of VMs required when no protection is applied (that is $\Gamma = 0$). The table shows that the proposed mechanism can easily protect against a limited level of error: for example for $\Gamma = 1$ the number of additional VMs required is in the range 30%-55%. Additionally, the costs overhead decreases as the number of SaaSs increases.

Scenario	$\Gamma = 1$	$\Gamma = 2$	$\Gamma = 3$	$\Gamma = 4$	$\Gamma = 5$
$\|\mathcal{S}\|$=5	55%	89%	104%	118%	127%
$\|\mathcal{S}\|$=10	44%	88%	113%	124%	134%
$\|\mathcal{S}\|$=20	49%	89%	108%	128%	143%
$\|\mathcal{S}\|$=40	40%	75%	92%	107%	116%
$\|\mathcal{S}\|$=50	38%	75%	95%	112%	122%

TABLE 7: Worst case scenarios comparison: additional VMs ratio. Spiky load.

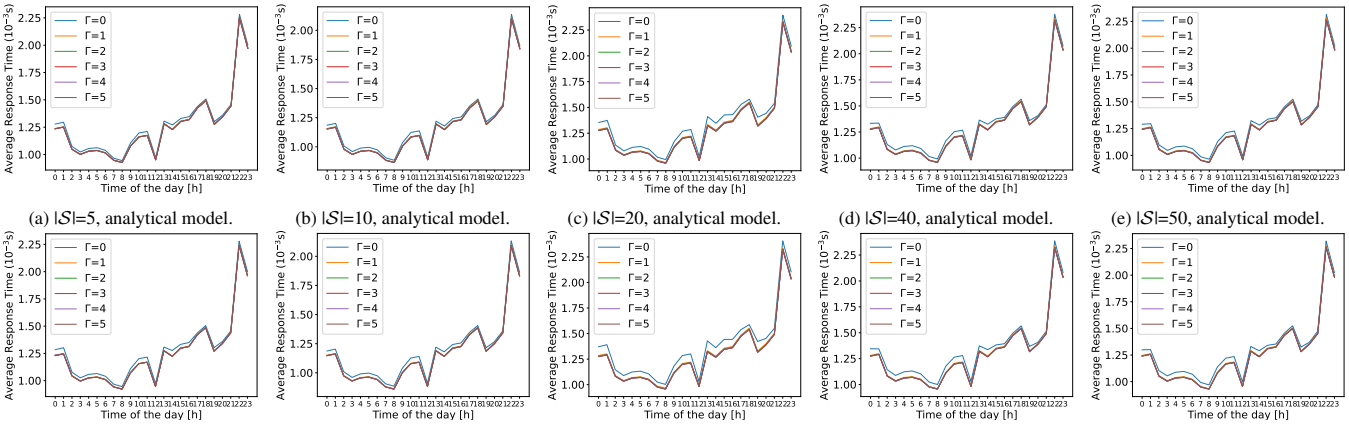
VII. CONCLUSION

This paper formulates the resource allocation problem for multiple SaaS providers competing in an IaaS system as a game with uncertain parameters, addressing it through a robust Generalized Nash Equilibrium approach. By incorporating Γ-Robustness, providers can adjust conservatism levels to suit their business needs. Analytical solutions demonstrate that robust models enhance service quality and mitigate unfeasibility risks under runtime uncertainties. Numerical and simulation results validate the method's effectiveness, even in realistic, large-scale scenarios. While the approach shows promise for optimizing SLA contracts, further work is needed to explore varying buffer sizes and demand distributions.

For future work, our robust approach can be extended to multi-tenant cloud environments, where multiple tenants share the same infrastructure and compete for resources. This extension would involve addressing the additional complexities introduced by tenant isolation, resource fairness, and priority constraints, while maintaining the robustness of the solution under uncertain conditions. Additionally, a promising direction is to explore the application of our robust game framework to resource allocation for AI models, in particular for the inference case , where the computational demands and performance variability of AI workloads introduce unique challenges. Incorporating the uncertainties inherent in AI tasks, such as dynamic workload sizes and fluctuating data processing rates, would allow us to develop solutions that optimize resource utilization while ensuring model performance and reliability.

ACKNOWLEDGMENT

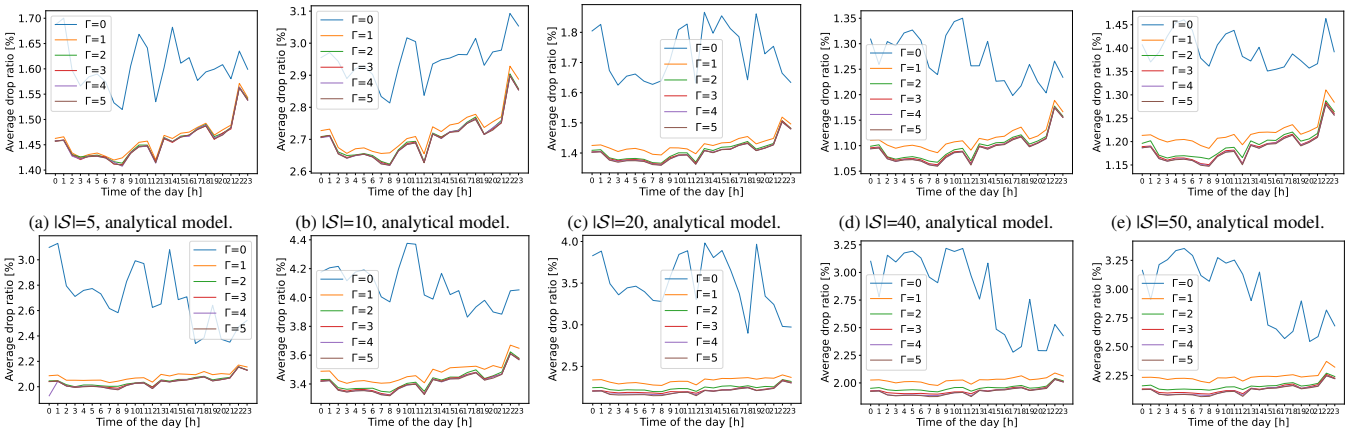
We would like to extend our sincere gratitude to Dr. Sara Mattia for the insightful discussions and valuable contribu-



(a) $|\mathcal{S}|$=5, analytical model. (b) $|\mathcal{S}|$=10, analytical model. (c) $|\mathcal{S}|$=20, analytical model. (d) $|\mathcal{S}|$=40, analytical model. (e) $|\mathcal{S}|$=50, analytical model.

(f) $|\mathcal{S}|$=5, simulation. (g) $|\mathcal{S}|$=10, simulation. (h) $|\mathcal{S}|$=20, simulation. (i) $|\mathcal{S}|$=40, simulation. (j) $|\mathcal{S}|$=50, simulation.

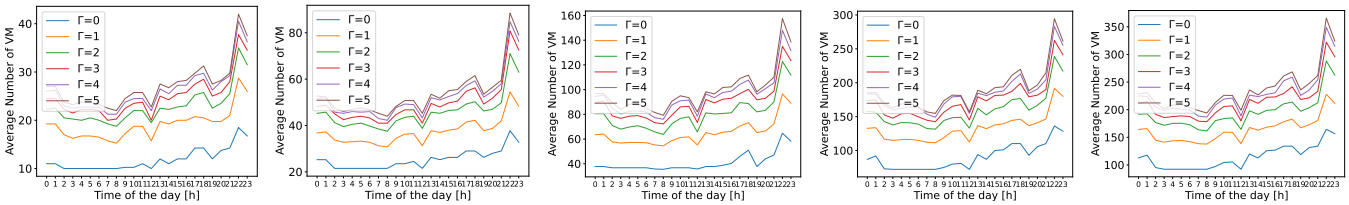
FIGURE 9: Average response time obtained by the analyitical model and the simulation varying number of SaaSs with $|\mathcal{M}_a|$=5 under spiky load.



(a) $|\mathcal{S}|$=5, analytical model. (b) $|\mathcal{S}|$=10, analytical model. (c) $|\mathcal{S}|$=20, analytical model. (d) $|\mathcal{S}|$=40, analytical model. (e) $|\mathcal{S}|$=50, analytical model.

(f) $|\mathcal{S}|$=5, simulation. (g) $|\mathcal{S}|$=10, simulation. (h) $|\mathcal{S}|$=20, simulation. (i) $|\mathcal{S}|$=40, simulation. (j) $|\mathcal{S}|$=50, simulation.

FIGURE 10: Average drop ratio obtained through the analytical model and the simulation varying number of SaaSs with $|\mathcal{M}_a|$=5 and spiky load.



(a) $|\mathcal{S}|$=5. (b) $|\mathcal{S}|$=10. (c) $|\mathcal{S}|$=20. (d) $|\mathcal{S}|$=40. (e) $|\mathcal{S}|$=50.

FIGURE 11: Average number of VMs varying number of SaaSs with $|\mathcal{M}_a|$=5 and spiky load.

tions during the initial talks on the modeling of the robust game.

REFERENCES

[1] L. Carvalho, T. E. Colanzi, W. K. G. Assunção, A. Garcia, J. A. Pereira, M. Kalinowski, R. M. de Mello, M. J. de Lima, and C. Lucena, "On the usefulness of automatically generated microservice architectures," IEEE Transactions on Software Engineering, vol. 50, no. 3, pp. 651–667, 2024.

[2] Gartner Group, "Microservices Architecture: Have Engineering Organizations Found Success?" 2023. [Online]. Available: https://www.gartner.com/peer-community/oneminuteinsights/microservices-architecture-have-engineering-organizations-found-success-u6b

[3] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," IEEE Transactions on Mobile Computing, vol. 20, no. 3, pp. 939–951, 2021.

[4] V. M. Mostofi, E. Krul, D. Krishnamurthy, and M. Arlitt, "Trace-driven scaling of microservice applications," IEEE Access, vol. 11, pp. 29 360–29 379, 2023.

[5] R. Samani, B. Honan, and J. Reavis, "Chapter 2 - selecting and engaging with a cloud service provider," in CSA Guide to Cloud Computing. Syngress, 2015, pp. 23–34.

[6] Y. Gao, G. Casale, and R. Singhal, "Performance modeling of distributed data processing in microservice applications," ACM Trans. Model. Perform. Eval. Comput. Syst., aug 2024. [Online]. Available: https://doi.org/10.1145/3687265

[7] H. Sedghani, D. Ardagna, M. Matteucci, G. Fontana, G. Verticale, F. Amarilli, R. Badia, D. Lezzi, I. Blanquer, A. Martin, and K. Wawruch, "Advancing design and runtime management of ai applications with ai-

sprint (position paper)," in 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 2021, pp. 1455–1462.

[8] U. Nnaji, E. O. Azubuike, E. Adetiba, and D. Akinola, "Development of an automated service level agreement negotiation framework for saas cloud e-marketplace," in 2024 International Conference on Science, Engineering and Business for Driving Sustainable Development Goals (SEB4SDG), 2024, pp. 1–5.

[9] X. Wu, F. D. Pellegrini, G. Gao, and G. Casale, "A framework for allocating server time to spot and on-demand services in cloud computing," ACM Trans. Model. Perform. Evaluation Comput. Syst., vol. 4, no. 4, pp. 20:1–20:31, 2019. [Online]. Available: https://doi.org/10.1145/3366682

[10] Z. Sun, G. Sun, Y.Liu, J. Wang, and D. D. Cao, "Bargain-match: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks," IEEE Transactions on Mobile Computing, vol. 23, no. 2, pp. 1655–1673, 2024.

[11] H. Wu, J. Nie, Z. Xiong, Z. Cai, T. Zhou, C. Yuen, and D. Niyato, "A game-based incentive-driven offloading framework for dispersed computing," IEEE Transactions on Communications, vol. 71, no. 7, pp. 4034–4049, 2023.

[12] L. Xie, S. Meng, W. Yao, and X. Zhang, "Differential pricing strategies for bandwidth allocation with lfa resilience: A stackelberg game approach," IEEE Transactions on Information Forensics and Security, vol. 18, pp. 4899–4914, 2023.

[13] H. Sedghani, D. Ardagna, M. Passacantando, M. Z. Lighvan, and H. S. Aghdasi, "An incentive mechanism based on a Stackelberg game for mobile crowdsensing systems with budget constraint," Ad Hoc Networks, vol. 123, 2021, paper 102626.

[14] H. Sedghani, M. Z. Lighvan, H. Aghdasi, M. Passacantando, G. Verticale, and D. Ardagna, "A Stackelberg game approach for managing AI sensing tasks in mobile crowdsensing," IEEE Access, vol. 10, pp. 91 524–91 544, 2022.

[15] A. Bandopadhyay, S. Swain, R. Singh, P. Sarkar, S. Bhattacharyya, and L. Mrsic, "Game-theoretic resource allocation and dynamic pricing mechanism in fog computing," IEEE Access, vol. 12, pp. 51 704–51 718, 2024.

[16] Q. Li, X. Jia, C. Huang, and H. Bao, "A dynamic combinatorial double auction model for cloud resource allocation," IEEE Transactions on Cloud Computing, vol. 11, no. 3, pp. 2873–2884, 2023.

[17] W. Fan, M. Hua, Y. Zhang, Y. Su, X. Li, B. Tang, F. Wu, and Y. Liu, "Game-based task offloading and resource allocation for vehicular edge computing with edge-edge cooperation," IEEE Transactions on Vehicular Technology, vol. 72, no. 6, pp. 7857–7870, 2023.

[18] Z. Xu, L. Zhou, S. Chau, W. Liang, H. Dai, L.Chen, W. Xu, Q. Xia, and P. Zhou, "Near-optimal and collaborative service caching in mobile edge clouds," IEEE Transactions on Mobile Computing, vol. 22, no. 7, pp. 4070–4085, 2023.

[19] T. Yoshihiro and S. Hosio, "Simulation-based iot stream data pricing incorporating seller competition and buyer demands," IEEE Access, vol. 11, pp. 16 213–16 225, 2023.

[20] S. Wu and L. Song, "Effects of competition and reputation on auditing cloud service provider's security commitment," IEEE Systems Journal, vol. 17, no. 1, pp. 306–313, 2023.

[21] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," Annals of Operations Research, vol. 175, no. 1, pp. 177–211, 2010.

[22] S. Spinner, G. Casale, F. Brosig, and S. Kounev, "Evaluating approaches to resource demand estimation," Performance Evaluation, vol. 92, pp. 51–71, 2015.

[23] D. Bertsimas and M. Sim, "The price of robustness," Operations research, vol. 52, no. 1, pp. 35–53, 2004.

[24] F. Yunlong and L. Jie, "Incentive approaches for cloud computing: challenges and solutions," Journal of Engineering and Applied Science, vol. 71, no. 51, pp. 1–18, 2024.

[25] R. Jeyaraj, A. Balasubramaniam, M. A. Kumara, N. Guizani, and A. Paul, "Resource management in cloud and cloud-influenced technologies for internet of things applications," ACM Comput. Surv., vol. 55, no. 12, Mar. 2023.

[26] Y. Wang, J. Zhou, and X. Song, "A utility game driven qos optimization for cloud services," IEEE Transactions on Services Computing, vol. 15, no. 5, pp. 2591–2603, 2022.

[27] T. Lyu, H. Xu, F. Liu, M. Li, L. Li, and Z. Han, "Two layer stackelberg game-based resource allocation in cloud-network convergence service computing," IEEE Transactions on Cognitive Communications and Networking, pp. 1–1, 2024.

[28] H. Sedghani, F. Filippini, and D. Ardagna, "Space4ai-d: A design-time tool for ai applications resource selection in computing continua," IEEE Transactions on Services Computing, vol. 17, no. 6, pp. 4324–4339, 2024.

[29] R. Sala, H. Sedghani, M. Passacantando, G. Verticale, and D. Ardagna, "Ai applications resource allocation in computing continuum: A stackelberg game approach," IEEE Transactions on Cloud Computing, pp. 1–18, 2024.

[30] R. Panwar and M. Supriya, "Rlpraf: Reinforcement learning-based proactive resource allocation framework for resource provisioning in cloud environment," IEEE Access, vol. 12, pp. 95 986–96 007, 2024.

[31] C. A. Vieira, L. F. Bittencourt, T. A. L. Genez, M. L. M. Peixoto, and E. R. M. Madeira, "Raaas: Resource allocation as a service in multiple cloud providers," Journal of Network and Computer Applications, vol. 221, p. 103790, 2024.

[32] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for qos-driven control of multi-iaas applications," IEEE Transactions on Cloud Computing, vol. 9, no. 2, pp. 418–434, 2021.

[33] D. Ardagna, B. Panicucci, and M. Passacantando, "Generalized Nash equilibria for the service provisioning problem in cloud systems," IEEE Transactions on Services Computing, vol. 6, pp. 429–442, 2013.

[34] D. Shmoys, J. Wein, and D. Williamson, "Scheduling parallel machines on-line," in [1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science, 1991, pp. 131–140.

[35] J. Li, W. Liang, and Y. Ma, "Robust service provisioning with service function chain requirements in mobile edge computing," IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 2138–2153, 2021.

[36] X. Zhang, R. Z. Z. Zhou, J. Lui, and Z. Li, "An online learning-based task offloading framework for 5g small cell networks," in Proceedings of the 49th International Conference on Parallel Processing, ser. ICPP '20, 2020.

[37] 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.

[38] D. A. Menascé and V. Dubey, Utility-based QoS Brokering in Service Oriented Architectures. IEEE ICWS Proc., pages 422-430, 2007.

[39] S. Kumar, V.Talwar, V. Kumar, P. Ranganathan, and K. Schwan, vManage: loosely coupled platform and virtualization management in data centers. ICAC2009 Proc., 2009.

[40] J. Almeida, D. Ardagna, I. Cunha, C. Francalanci, and M. Trubian, Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing, 2010.

[41] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. Institute of Electrical and Electronics Engineers, 2018, p. 1591.

[42] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, Performance analysis of network i/o workloads in virtualized data centers. Services Computing, IEEE Transactions on, 6(1):48-63, 2013.

[43] A. Croll, "Cloud performance from the end user perspective," 2017.

[44] R. Birke, A. Podzimek, L. Y. Chen, and E. Smirni, "State-of-the-practice in data center virtualization: Toward a better understanding of vm usage," in 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2013, pp. 1–12.

[45] T. Deng, "Analysis of user behavior in cloud broker," in 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 157–160.

[46] Z. Yao and I. Papapanagiotou, "A trace-driven evaluation of cloud computing schedulers for iaas," in 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6.

[47] http://aws.amazon.com/elasticbeanstalk/, "Amazon inc. aws elastic beanstalk."

[48] M. Passacantando, D. Ardagna, and A. Savi, "Service provisioning problem in cloud and multi-cloud systems," INFORMS Journal on Computing, vol. 28, pp. 265–277, 2016.

[49] "Amazon inc. amazon elastic cloud."

[50] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[51] L. Cherkasova and P. Phaal, "Session-based admission control: a mechanism for peak load management of commercial web sites," IEEE Transactions on Computers, vol. 51, no. 6, pp. 669–685, 2002.

[52] A. Wolke and G. Meixner, "Twospot: A cloud platform for scaling out web applications dynamically," in Towards a Service-Based Internet, E. Di Nitto and R. Yahyapour, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 13–24.

[53] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. Mckee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000

islands: an integrated approach to resource management for virtualized data centers," Cluster Computing, vol. 12, no. 1, p. 45–57, 2009.
[54] Amazon, "Auto scaling groups." [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-groups.html
[55] A. Bauer, H. Pan, R. Chard, Y. Babuji, J. Bryan, D. Tiwari, I. Foster, and K. Chard, "The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud," Future Gener. Comput. Syst., vol. 153, no. C, p. 558–574, may 2024. [Online]. Available: https://doi.org/10.1016/j.future.2023.12.007
[56] "Average monthly electricity wholesale price in italy from january 2019 to january 2023," https://www.statista.com/statistics/1267548/italy-monthly-wholesale-electricity-price/.
[57] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, J. He, and C. Xu, "An in-depth study of microservice call graph and runtime performance," IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 12, pp. 3901–3914, 2022.

HAMTA SEDGHANI received her B.Sc. degree from Iran University of Science and Technology, Tehran, Iran and her M.S and Ph.D. degrees from University of Tabriz, Tabriz, Iran in Computer Engineering. Currently, she is post-doc researcher at Politecnico di Milano. Her current interests include game theory and optimization for resource management in computing continuum.

MAURO PASSACANTANDO received the M.S. and Ph.D. degrees in Mathematics from University of Pisa, Italy. He was an Assistant Professor and then an Associate Professor of Operations Research at the University of Pisa from 2002 to 2022. He is currently an Associate Professor of Operations Research (qualified for Full Professorship) with the Department of Business and Law, University of Milano-Bicocca. His research is mainly devoted to variational inequalities and equilibrium problems. In recent years, his work has also focused on warehouse management and green logistics problems. He has published more than 80 peer-reviewed papers in international journals, book chapters and conference proceedings.

RICCARDO LANCELLOTTI is an associate professor at the Department of Engineering "Enzo Ferrari" in the University of Modena and Reggio Emilia since 2015. He received the Laurea Degree in computer engineering from the University of Modena and Reggio Emilia in 2001 and the Ph.D. in computer engineering from the University of Roma "Tor Vergata" in 2003. In 2003, he spent eight months at the IBM T.J. Watson Research Center as a visiting researcher. His research interests include geographically distributed systems, fog and cloud computing. On these topics he published more than 90 papers on international journals and conferences. He is a member of IEEE and of ACM.

MINA ZOLFY LIGHVAN received her B.Sc. degree in Computer Engineering (hardware) and M.Sc. degree in Computer Engineering (Computer Architecture) from ECE faculty, university of Tehran, Iran in 1999, 2002 respectively. She received Ph.D. degree in Electronic Engineering (Digital Electronic) from Electrical and Computer Engineering faculty of University of Tabriz, Iran. She is an associate professor in Computer engineering department of ECE faculty in University of Tabriz.

DANILO ARDAGNA is Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. He received a Ph.D. degree in computer engineering in 2004 from Politecnico di Milano from which he also graduated in December 2000. His work focuses on the design, prototype and evaluation of optimization algorithms for resource management of cloud computing and AI systems.