Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Joint admission control and resource allocation in virtualized servers

Jussara Almeida[a], Virgílio Almeida[a], Danilo Ardagna[b,*], Ítalo Cunha[a], Chiara Francalanci[b], Marco Trubian[c]

[a] Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, MG 31270-010, Brazil
[b] Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy
[c] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135, Milan, Italy

ARTICLE INFO

Article history:
Received 31 December 2008
Received in revised form
29 June 2009
Accepted 26 August 2009
Available online 6 September 2009

Keywords:
Resource allocation
Admission control
Quality of service
Optimization

ABSTRACT

In service oriented architectures, Quality of Service (QoS) is a key issue. Service requestors evaluate QoS at run time to address their service invocation to the most suitable provider. Thus, QoS has a direct impact on the providers' revenues. However, QoS requirements are difficult to satisfy because of the high variability of Internet workloads.

This paper presents a self-managing technique that *jointly* addresses the resource allocation and admission control optimization problems in virtualized servers. Resource allocation and admission control represent key components of an autonomic infrastructure and are responsible for the fulfillment of service level agreements. Our solution is designed taking into account the provider's revenues, the cost of resource utilization, and customers' QoS requirements, specified in terms of the response time of individual requests.

The effectiveness of our joint resource allocation and admission control solution, compared to top performing state-of-the-art techniques, is evaluated using synthetic as well as realistic workloads, for a number of different scenarios of interest. Results show that our solution can satisfy QoS constraints while still yielding a significant gain in terms of profits for the provider, especially under high workload conditions, if compared to the alternative methods. Moreover, it is robust to service time variance, resource usage cost, and workload mispredictions.

1. Introduction

In the Service-Oriented Architecture (SOA), Quality of Service (QoS) requirements can be difficult to satisfy, especially due to the high variability of Internet workloads. The workload of Internet applications can vary by orders of magnitude within the same business day [10]. Such variations cannot be accommodated with traditional capacity planning and allocation practices, but require autonomic computing self-managing techniques [29], which dynamically allocate resources among different services on the basis of short-term demand estimates.

This dynamic allocation of computing capacity is enabled by the *virtualization* of resources [16]. A virtualization mechanism provides service differentiation and performance isolation for multiple Web services sharing the same physical resources (i.e., CPUs, disks, communication networks). Service differentiation is obtained by partitioning physical resources into multiple virtual resources, thus creating independent *virtual machines* (VMs). Each VM is dedicated to a single Web service and is allocated a varying fraction of capacity depending on the Web service's QoS requirements and current workload. This also prevents the contention for resources among different Web services, thus providing performance isolation.

Virtualized environments are managed by autonomic controllers which usually include: (i) a *monitor*, which measures the workload and the performance metrics of each Web service, (ii) a *predictor*, which forecasts future system workload conditions based on past monitoring data, (iii) a *resource allocator*, which calculates the capacity required by different Web services from incoming workload predictions, and (iv) an *admission controller*, which decides whether to accept or reject service requests according to the server's ability to guarantee QoS requirements given its current workload and capacity.

This paper focuses on the resource allocation and admission control problems in a virtualized platform hosting a number of Web applications. Resource allocation and admission control represent the components of a virtualized environment that are responsible for Service Level Agreement (SLA) fulfillment. Decisions on resource allocation and admission control have clear

* Corresponding author.
E-mail addresses: jussara@dcc.ufmg.br (J. Almeida), virgilio@dcc.ufmg.br (V. Almeida), ardagna@elet.polimi.it (D. Ardagna), cunha@dcc.ufmg.br (Í. Cunha), francala@elet.polimi.it (C. Francalanci), trubian@dsi.unimi.it (M. Trubian).

0743-7315/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2009.08.009

mutual interrelations. Different admission policies lead to different workload patterns and, vice versa, different resource allocation strategies lead to varying SLA violation patterns depending on the workload. Despite these interrelations, previous literature has considered each decision problem in isolation [43,12,36]. The joint admission control and resource allocation problem is complex. Furthermore, the solution has to be evaluated according to the short-term planning horizon of autonomic computing, i.e., decisions must be made within strict time constraints (in the range of 5–10 min [4,40,7]). This time issue represents a major challenge and the main reason as to why resource allocation and admission control have been addressed as separate decision problems [43,12,36].

The main contribution of this paper is a *time efficient* optimization model that considers resource allocation and admission control as a *joint* decision problem. The model includes:

- the provider's revenue;
- the provider's cost, that is, the cost of utilization of servers resources;
- customers' QoS requirements in terms of response time of *individual requests*.

The optimization objective is to maximize the provider's revenue, while satisfying customers' QoS requirements and minimizing the cost of resource utilization. Traditional resource management approaches (e.g., see [29,43]) focus on the optimization of performance measured by *average response time* and system utilization. In contrast, this paper's optimization model considers the *tail distribution* of request response times, thus providing guarantees on the response time observed by *each request*.

We solve the optimization problem by means of an analytical queuing-based solution of a performance model. Analytical solutions allow a time efficient evaluation of performance metrics and, thus, reduce computation time. The paper proposes an efficient algorithm to determine the global optimum of the optimization problem, which can solve large problem size instances within very strict time constraints, suitable for on-line implementation.

The effectiveness of the resource allocation and admission control policies identified by the optimization model is tested through simulation in a number of different scenarios of interest. Results show that resource allocation and admission control policies satisfy QoS constraints. Furthermore, compared to state-of-the-art resource management techniques, our joint solution can yield a significant profit increase for the provider.

The rest of this paper is organized as follows. Section 2 discusses previous literature. Section 3 presents our resource management approach and discusses the characteristics of the reference system infrastructure considered in this paper. The performance model designed to evaluate QoS and the optimization techniques applied to solve the joint resource allocation and admission control problem are discussed in Sections 4 and 5, respectively. Section 6 demonstrates the economic efficiency of our solutions by presenting simulation results. Conclusions are finally drawn in Section 7, along with a discussion on possible directions for future work.

2. Related work

Several research contributions have previously addressed the issue of SLA management in cluster-based service centers. In general terms, they consider four main problems that must be addressed to enforce system management policies: (1) admission control, (2) resource allocation, (3) load balancing, and (4) power consumption optimization. Existing mechanisms and solutions fall into one of four main categories, namely, (1) the control theory-based feedback loop, (2) machine learning adaptive techniques, (3) reputation-based approaches, and (4) utility-based optimization techniques. We summarize the main results in each category, emphasizing existing utility-based optimization solutions, which more closely relate to our approach.

The main advantage of the control theory-based feedback loop is that it guarantees system stability. Furthermore, upon a change in workload, these mechanisms can accurately model transient behavior and can adjust the system's configuration within a constant time, which can be fixed a priori. The adoption of control-oriented solutions is quite recent and their scalability has not yet been demonstrated. Kamra et al. [18] have proposed a non-invasive admission control system based on a control-theoretic approach that prevents overload and enforces absolute client response times in multi-tiered Web sites. This approach is architecture independent, as it works with any server model, process, thread, or event based. Other research contributions [1] use feedback control to limit the utilization of bottleneck resources by means of admission control or resource allocation. While most control theoretic approaches adopt system identification techniques to build linear time invariant models and then apply classical proportional integral differential control, authors in [32] have developed linear parameter varying models for performance control by adopting server's CPU dynamic voltage and frequency scaling (DVFS) as control variable.

Machine learning adaptive techniques base their learning sessions on live systems, without a need for an analytical model of the system. A recent research contribution [41] has proposed a resource allocation technique that determines the assignment of physical servers to applications that maximizes the fulfillment of SLAs. Kephart et al. [19] have applied machine learning to coordinate multiple autonomic managers with different goals. In particular, their work has integrated an autonomous performance manager with a power manager in order to satisfy performance constraints while minimizing energy consumption costs by exploiting server DVFS. A recognized advantage of machine learning techniques is that they accurately capture system behavior without any explicit performance or traffic model and with little built-in system-specific knowledge. However, training sessions tend to extend over several hours [19]. Furthermore, the proposed techniques usually consider a limited set of managed applications and apply separate actuation mechanisms.

Reputation-based approaches have been proposed mainly in the grid and Peer-to-Peer (P2P) research communities, with the goal of developing distributed admission control, load balancing and resource selection policies. In [38], the authors have addressed the inherent unreliability and instability of nodes in large grid and P2P infrastructures. Several reputation-based algorithms, which update the reputation metric by evaluating the run-time reliability of participating nodes, have been developed with the goal of maximizing system throughput and task completion probability. A reputation model specific for the resource selection in economic grids has been discussed in [23]. Authors in [37] have proposed distributed reputation-based policies specific for P2P systems, in which peers receive incentives to share their resources (i.e., transmission capacity) by uploading content to serve other peers' needs, instead of behaving selfishly. These policies enforce peer collaboration, thus maximizing the global transmission rate. The distributed, and thus scalable, architecture of most reputation-based mechanisms constitutes their main advantage with respect to other centralized solutions. However, the reputation models are often very application-specific, and, thus, cannot be generalized to other domains. Furthermore, if participating nodes cannot update reputation locally, an additional infrastructure for reputation notification has to be introduced [17].

Finally, utility-based approaches have been introduced to optimize the degree of users' satisfaction by expressing their goals in terms of user-level performance metrics. Typically, the system is modeled by means of a performance model embedded within an optimization framework. Optimization can produce global optimal solutions or sub-optima by means of heuristics, depending on the complexity of the optimization model, thus providing scalability to reasonably large size problem instances.

In most previously proposed utility-based approaches, optimization is typically applied to each of the four actuators in isolation. With respect to admission control, most existing solutions [12,44] make session-based dynamic decisions, aiming at preventing system overload and obtaining service differentiation. Admission control is considered effective if the admitted requests are served according to their SLA contracts. Utility is expressed in terms of the number of served sessions. Resource allocation, in turn, is typically viewed as a separate optimization activity that operates under the assumption that servers are protected from overload, and that SLAs can be satisfied with an efficient allocation of computing resources across different *classes of requests*. A request class is defined as a subset of requests that is homogeneous with respect to the specific set of applications required to support their executions, their SLA contracts (which specify performance requirements, price, penalties, etc.), and workload profiles (i.e., resource demands).

Other studies focus on service differentiation in server farms by physically partitioning the farm into separate clusters, each serving one of the request classes (e.g., see [5,52]). For example, the work in [44] aims at maximizing the revenues from SLA of multi-class systems where each class of request is assigned to a number of dedicated homogeneous servers in order to provide performance guarantees. The number of servers is evaluated by modeling each physical server by means of a G/G/1 queue, although the solution is determined by a very simple greedy approach. In contrast, new technology trends advocate the sharing of physical resources among multiple request classes [7,29,40]. Pacifici et al. [29,43] have proposed statistical multiplexing techniques which allow the instantaneous sharing of unused resource capacities, and allocate a varying number of threads to each request class. A fundamental difference between Pacifici's as well as a number of other related strategies [45,11] and this paper's approach is that they express utility in terms of *average response time*. In sharp contrast, we express utility in terms of the *probability distribution of response times*. Therefore, our focus is on the performance observed by *individual user requests*, instead of average performance, which may not be very meaningful for very heterogeneous and highly variable workloads. Moreover, this paper's approach exploits the virtualization mechanism and, hence, we manipulate the fraction of capacity to be devoted to each virtual machine, instead of the number of threads, in order to adjust capacity.

The authors in [8] have observed that power consumption represents a serious concern in the design and operation of large service centers. Power consumption is known for involving high expenses related to electricity, in addition to the fixed costs related to the design of cooling systems. To the best of our knowledge, [50] represents the first research contribution that tackles energy consumption and SLA performance management as a joint optimization problem. This paper and other early techniques for power management turn unused servers off during periods of light load. More recent approaches [32,19,22] have also considered adjusting the frequency scaling of servers as a means to reduce consumption. In this work, we include the cost of resource usage in the objective function of our optimization model, and provide the fraction of unused capacity as an output to drive power consumption decisions.

Authors in [33] have presented the result of a server consolidation project on blade servers based on a power budget mechanism. Processors' frequencies are throttled in order to achieve CPU utilization and energy saving goals. Nevertheless, the approach cannot provide, a priori, any SLA guarantee. In [39], a virtualization framework able to support heterogeneous workloads (batch and web interactive system) is presented, but energy saving policies are not considered. Authors in [53] have presented a multi-layer and multi-time scale solution for the management of virtualized systems, but they also do not consider the trade-off between performance and system costs. Autonomic management is implemented also in current virtualization products (e.g., VMWare DRS or Virtuoso VSched [39]) with the aim of equalizing the use of resources, instead of determining performance and energy trade-offs.

In [7], we have provided a joint solution for the problem of VM placement, load balancing and resource allocation of multiple VMs. Unlike this work, that solution acts at larger time scales (at least half an hour as opposed to a few minutes), and the admission control problem has not been considered. On a parallel effort, we have also proposed an alternative autonomic capacity management framework for virtualized infrastructures [3,14,15]. The framework consists of an optimization model that embeds an analytical queuing-based performance model as well as a pricing model. The performance model, proposed as a proof of concept in [3], was further extended to multi-tier platforms in [14]. The framework was also extended to include costs related to energy consumption (similarly to our approach in this paper) and security management [15]. Nevertheless, in contrast to our current effort, that solution focuses only on the resource allocation problem, and, thus, does not address the admission control problem. This paper builds greatly on the solution we proposed in [4]. In comparison with that preliminary work, we here adopt a much more accurate performance model and provide a deeper experimental analysis. Our ongoing research on energy management in discussed in [6].

The four categories of existing SLA management approaches are summarized in Table 1.

Out of the available resource allocation and admission control mechanisms available in the literature, we selected two techniques, namely, the resource allocation strategy proposed in [36] and the admission control scheme proposed in [12], to compare against the joint solution proposed in this paper. These two techniques were chosen because: (1) they are top-performing state-of-the-art techniques addressing each individual problem, (2) they are both based on utilization thresholds, and thus can be easily integrated into a coherent framework, and (3) they allow a comparison between model-based solutions (such as the one here proposed) and utilization-driven techniques.

The workload management strategy proposed in [36] provides one of the best-performing techniques for resource allocation. Designed for an autonomic virtualized service center, it addresses the issue of choosing per-VM values of resource allocation control parameters. The strategy aims at keeping resource (i.e., CPU) utilization between two thresholds $U^{\min}$ and $U^{\max}$. Whenever utilization drops below $U^{\min}$, CPU allocation is decreased by a pre-defined value, but it never drops below a minimum allocation. Similarly, whenever utilization increases above the upper-threshold $U^{\max}$, CPU allocation is increased by the same value, limited to a maximum allocation. An optimized fast allocation policy is also proposed in [36], in which CPU allocation is increased to the maximum in one step whenever workload is observed to grow rapidly. The analysis presented in Section 6.2 considers this fast allocation policy. We note that the policies proposed in [36] allocate resources for multiple VMs independently, i.e., allocation decisions are made for each VM separately. Thus, they do not handle the trade-offs and conflicts that arise when multiple workloads (i.e., VMs) compete for resources in a shared infrastructure.

We also consider the admission control mechanism proposed in [12]. The mechanism operates in pre-defined time intervals, and

Table 1
Comparison of alternative categories of SLA management solutions in literature.

Control policy category	Research contribution	Problems addressed	Strengths	Weaknesses
Control theory feedback loop	[1,18,32,22]	Admission Control; Resource Allocation; Power Optimization;	System stability guarantees; Accurate transient behavior modeling; Effective on short term time scales (s); Upper bound on the effects of configuration changes;	Scalability is yet to be demonstrated;
Machine learning	[41,19]	Resource Allocation; Power Optimization;	Acquire system model from running applications;	Training sessions extend over several hours; Limited set of managed applications; Separate actuation mechanisms;
Reputation based	[38,17,37,23]	Admission Control; Load Balancing; Resource Allocation;	Distributed Architecture; Scalability;	Application specific; May require reputation notification infrastructure;
Utility-based optimization	[52,12,5,44] [50,29,45,4] [3,7,40,11] [39,53,14,15]	Admission Control; Load Balancing; Resource Allocation; Power Optimization;	Global optimal (or sub-optima) solutions; Scalability;	Performance parameters estimated at run-time; Effective on medium term time scales (several minutes);

is based on a target maximum CPU utilization and on predictions of utilization for the next interval. If the predicted utilization grows above the target maximum, the admission controller rejects all new sessions for the next interval, favoring the service of requests from already admitted sessions. The prediction of future CPU utilization is based on the moving average of utilization in the last n intervals. The paper also proposes a self-tunable hybrid mechanism that dynamically adjusts the moving average as a function of the number of aborted requests and refused connections. The efficiency of the mechanism is further improved by considering a prediction of the number of sessions that the server is able to process. In Section 6.2, we consider this final *predictive* admission control mechanism. However, since our solution is designed for workloads composed of requests, instead of sessions, the mechanism is applied to *request admission control*.

Finally, we note that both techniques presented in [12,36] make decisions based on target utilizations, which, in general, cannot provide any QoS guarantee. In contrast, our joint solution makes resource allocation and admission control decisions based on the performance (i.e., per-request response time) achieved by applications.

3. Autonomic resource management

This section provides an overview of our autonomic computing approach for the resource management of virtualized Web service infrastructures. Section 3.1 discusses the target virtualized environment. The main components of our autonomic resource management framework are introduced in Section 3.2, whereas the main assumptions of the design of our joint resource allocation policies are presented in Section 3.3.

3.1. Target virtualized environment

We assume that the reference service center offers multiple transactional Web services, and each service represents a different Internet application or a different instance of the same application. Indeed, multiple Web service instances with different quality profiles can be offered by the provider to meet varying customer needs. In either case, the hosted Web services can be heterogeneous with respect to resource demands, workload intensities and QoS requirements. Services and service instances with different quality and workload profiles are categorized into independent Web service (WS) classes.

An SLA contract, associated with each WS class, is established between the service provider and its customers. It specifies the QoS levels the provider must meet while responding to customer requests for a given service class, as well as the corresponding pricing scheme. We assume that the provider gains full revenues from the requests served within a response time threshold specified in the SLA contracts. Otherwise, revenues are lost.

Fig. 1 shows the architecture of the service center under study. The system includes a set of heterogeneous servers, each of which runs a Virtual Machine Monitor (VMM), such as VMWare or Xen [16]. Each server's physical resources (i.e., CPU, disks, communication network) are partitioned among multiple virtual machines, each running a dedicated Web service application. Virtualization enables the flexible reduction and expansion of the resource capacity assigned to each VM (and, thus, to its hosted WS class). As a first approximation, we assume that, once a capacity assignment has been performed, each VM is guaranteed its resources, regardless of the actual load of other VMs. In other words, we assume that virtualization provides performance isolation and facilitates service differentiation, preventing the contention for resources among services (or WS classes).

As in [29,36,43,10], among the server's many resources, we consider CPU as representative for the resource allocation problem. Each VM hosts a single Web service application, and multiple VMs implementing the same WS class can run in parallel on different hosts. Under light load conditions, servers can be turned off or put in a stand-by state and moved to a free server pool in order to save energy costs. They can also be moved back into an active state and re-introduced into the running infrastructure during load peaks.

Our system is based on a hierarchical architecture. At a higher level, a Layer 1 (L1) controller is responsible for establishing the set of servers in active state,[1] the VM to physical server assignment,

[1] Servers that are running and, thus, that are neither in stand-by state nor turned-off and allocated to the free server pool.

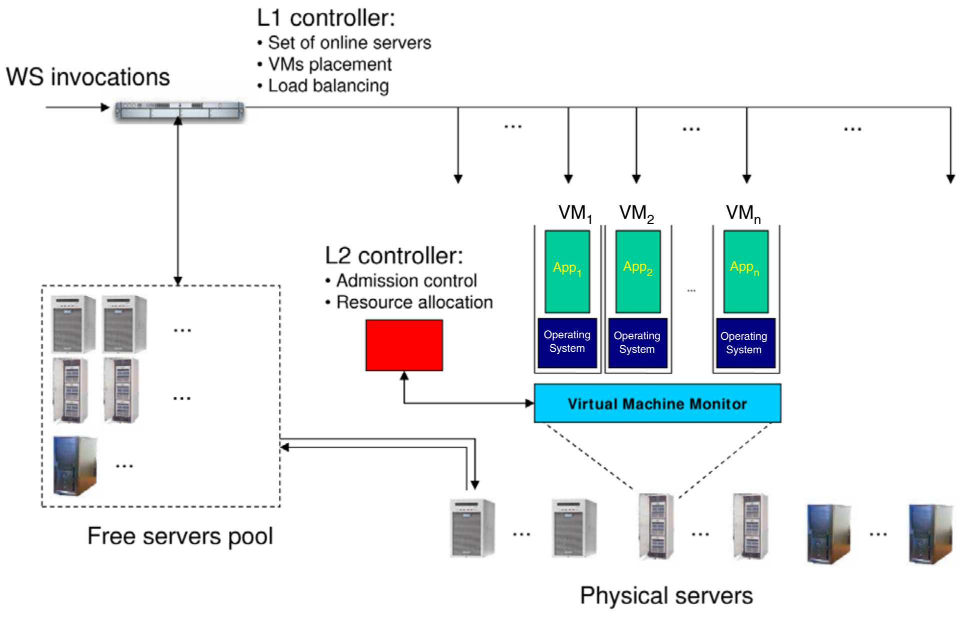


Fig. 1. Target virtualized environment.

and the load balancing across multiple VMs implementing the same WS class. At a lower level, Layer 2 (L2) controllers implement admission control and resource allocation policies. The L1 controller is centralized, and is hosted on a dedicated server which acts as a network dispatcher, as in [22]. L2 controllers, in turn, are installed and run locally at each server. Moreover, L1 and L2 controllers work on different time scales. Since the actions taken by the L1 controller, namely, servers switching and VMs placement, introduce a significant system overhead, they should be performed about every half an hour. In contrast, the L2 admission control and resource allocation decisions have shorter time requirements, and can be performed every few minutes. Overall, an L2 controller is responsible for locally managing, at a finer grain time scale, the VMs allocated by L1 to its physical server. In the remainder of this paper, we will focus on the design of the L2 controllers. The implementation of our L1 controller is described in [7]. Alternatively, the solutions proposed in [40,22,50] could also be adopted.

3.2. *Autonomic framework*

Our L2 controller combines a *performance model* and an *optimization model*, and has the goal of maximizing revenues from SLAs, while minimizing the total cost associated with the use of resources, including costs related to hardware, power, air conditioning, etc. In order to achieve these goals, L2 controllers can choose to dynamically adjust the fraction of capacity assigned to each VM and/or to limit the incoming workload by serving only the subset of requests that maximizes profits, i.e., revenues minus costs. For example, an L2 controller may choose to reject a fraction of incoming requests and reduce the capacity of the corresponding VM, if this choice leads to a cost reduction that is greater than the revenue losses due to request rejections.

Resource allocation is performed periodically, every 5 to 10 min [29,7], depending on workload profiles. At the end of each period, on the basis of a prediction of future workloads [9,2], the *performance model* produces an estimate of the future performance of each VM, i.e., determines future SLA violations for each WS class. The *optimization model* uses these estimates to determine the fraction of capacity to be assigned to each VM (i.e., resource allocation) as well as the number of requests effectively served for each class (i.e., admission control) that maximizes profits.

3.3. *Design assumptions*

In designing the performance model and the optimization model of our L2 controller, we made a few assumptions driven by the goal of meeting a reasonable trade-off between accuracy and practical solution time.

First, we assume an open request-based workload model in which WS requests arrive according to a Poisson process, as observed in several real systems [31,13] and assumed by most previous solutions [25,47,34]. This assumption greatly simplifies the model, thus shortening solution time. However, one might raise the issue of whether it is adequate to certain e-commerce and enterprise applications, whose workloads are more accurately described by independent arrivals of *user sessions*, that is, sequences of inter-dependent requests [21]. To address this issue, we rely on recent results from [51], which show that the performance (i.e., response time) and resource requirements of *session-based systems* can be accurately captured by a simplified model based on the assumption of *independent request arrivals*, such as the one proposed here, provided that the distribution of different request types is the same as in the original session-based system. As in [51], we assume this distribution can be estimated by continuously monitoring the system.

Moreover, we assume that each WS request has a service demand that is exponentially distributed, with different WS classes possibly exhibiting different service time distributions. This assumption also helps in simplifying our performance model, thus reducing solution time. However, it may be too restrictive, thus hurting accuracy and solution cost-effectiveness in several real scenarios in which applications observe a much higher variability in service times. Thus, we also assess, in Section 6.3.1, the impact of relaxing this assumption by evaluating our solution in scenarios in which service times follow heavy-tailed distributions with various degrees of variability.

Finally, we consider the fraction of server capacity assigned to each VM as a continuous variable, and assume that both performance and costs associated with WS classes are continuous and linear. This representation is an idealization of the joint adoption of VMMs and the DVFS mechanism [22] implemented in modern servers, which allows the dynamic change of supply voltage and CPU operating frequency. Other approaches proposed

Table 2
Autonomic framework: parameters and decision variables.

Symbol	Description
SLA parameters	
ω_i	Price of class i requests served with the QoS level specified in the SLA.
R_i^{SLA}	Response time threshold guaranteed for class i requests.
System parameters	
N	Number of hosted WS classes (and VMs).
υ_i	Maximum utilization planned for VM i ($0 \leq \upsilon_i < 1$).
D_i	Average service demand of class i requests on a VM to which the full capacity is allocated.
C	Cost per time unit associated with the use of total physical capacity.
Input from short-term workload predictor and monitoring	
λ_i	Predicted arrival rate of class i requests for the next period.
Decision variables	
X_i	Throughput of class i requests for the next period.
f_i	Fraction of server total physical capacity assigned to VM i for the next period.

in the literature [22,19] assume that the server performance (i.e., request service times) vary linearly with the CPU operating frequency. Furthermore, as it will be shown by Theorem 1 and Property 1 (Section 5), our optimization problem is convex. Thus, even if in reality the operating frequency can be varied only in a limited set of values (usually 5 or 6 with the current technology), modeling server capacity by continuous variables is not a limitation, i.e., we can find the best set of scheduling parameters for the VMs and global optimal CPU operating frequency (see Fig. 11(b) in Appendix).

The cost of resource usage depends on several parameters such as CPU utilization, cooling overhead, time of day energy cost, and others. We adopt a linear model for resource cost as a first approximation. The adoption of energy-proportional computing systems,[2] characterized by linear costs, is also advocated in the literature [8]. Given the state-of-the-art of current hardware technology, designing servers that exhibit this property is an open issue. However, some recent proposals [42] based on advanced resource allocation techniques, like the one proposed in this paper and in our previous work [7,6], have shown how energy-proportional systems can be obtained by off-the-shelf hardware components.

The next section presents the performance model of our dynamic resource allocation framework, while Section 5 formalizes the dynamic resource allocation problem by means of a non-linear optimization model. Both sections make use of the input and output parameters as well as of the decision variables defined in Table 2.

4. System performance model

This section presents our analytical queuing model that predicts the performance metrics of each VM. This model is a core component of the optimization solution presented in Section 5 to solve the dynamic resource management problem.

Our solution addresses the need to guarantee the response time observed by each *individual request*, thus specifying guarantees on the response time *tail distribution*. In other words, the goal of our performance model is to estimate the *probability* that a class i request experiences a response time that violates the SLA contract of the corresponding WS class, i.e., that is longer than the R_i^{SLA} threshold, given the (accepted) service load (i.e., the class throughput X_i), and the fraction of the server physical capacity assigned to the VM hosting the class, f_i. Thus, given the response time of a *single* class i request, R_i, our performance model estimates the probability $P(R_i \geq R_i^{SLA})$.

[2] Systems in which no power is used during idle periods, low power is used during lightly loaded periods, and proportionately higher power at higher loads.

This new type of SLA model, proposed as a means for providers to offer more attractive contracts to their customers [24], is in sharp contrast to traditional SLA contracts, assumed by most previous work [43,50,7], which provide guarantees only on the *average response time* of each request class. Guarantees only on *average* performance may not be very helpful in the context of current Internet workloads, which exhibit high variability. Moreover, providing guarantees on a per-request basis may be more attractive in the particular context of SOA systems, in which customers access a service only a few times (possibly only once), and, thus, may be interested mainly in their own service execution response times, as opposed to in an average across requests from multiple customers.

We model each VM as a single queue. This scenario provides a proof of concept, allowing us to evaluate the overall applicability and effectiveness of our framework. Moreover, given the assumptions of Poisson request arrivals and exponentially distributed service times discussed in Section 3.3, each VM is modeled as an M/M/1 open queue with FCFS scheduling, as this discipline has been frequently considered a reasonable abstraction for transactional service centers [27]. We adopt analytical models in order to obtain an indication of system performance, as in [29,43,44]. There is a trade-off between the accuracy of the model and the time required to estimate system performance, which has to be evaluated with strict time constraints. More accurate performance models have been provided in the literature for Web systems (e.g., [14,15, 35]). However, due to the analysis complexity, these models can deal with only small size models, based on a limited number of queues, and thus cannot be adopted here.

We assume that requests belonging to the same WS class i ($i = 1 \ldots N$) are statistically indistinguishable and, thus, have the same average service demand. We use D_i to indicate the average service demand in a virtualized server when WS class i receives the whole physical capacity of the server. D_i can be evaluated by continuously monitoring the system (e.g., see [28]), and it is assumed that the monitoring component can also take the overhead introduced in the system by the VMM layer into account [49]. The average service demand of class i requests when the server hosts multiple VMs can then be estimated by inflating D_i by the fraction of server physical capacity assigned to VM i, that is, $\frac{D_i}{f_i}$ [50].

In order to estimate the probability of response time violations for WS class i requests, we need the probability distribution of class i response times. Exact expressions for the response time distribution exist only for specific types of queues (for M/M/1 in particular [20]). On the other hand, a number of simpler approximations is available in the literature, some of which have been shown to yield results very close to the exact expressions [3,14].

Markov's Inequality [30,20] provides an *upper-bound* on the probability that the response time of a class i request, R_i, exceeds the threshold R_i^{SLA}. This upper-bound depends only on the average response time experienced by class i requests, $E[R_i]$, and can be computed as $P(R_i \geq R_i^{SLA}) \leq E[R_i]/R_i^{SLA}$. However, Markov's Inequality is known for providing somewhat loose upper-bounds, thus leading to allocation decisions that are too conservative, far from optimal, and thus cost-ineffective [3,14].

Chebyshev's Inequality [20], on the other hand, provides a much tighter upper-bound based on estimates of response time variance, $Var[R_i]$, in addition to estimates of the average response time $E[R_i]$. The average response time $E[R_i]$ can be estimated as follows [27]:

$$E[R_i] = \frac{D_i/f_i}{1-(D_i/f_i)X_i} = \frac{D_i}{f_i - D_iX_i}. \tag{1}$$

For M/M/1 queues, the response time variance is given by $Var[R_i] = \frac{(D_i/f_i)^2}{(1-(D_i/f_i)X_i)^2}$. Chebyshev's Inequality can be used by first computing the average and variance of class i response times, and then applying these metrics to estimate the probability that a class i request violates its response time SLA as follows:

$$P(R_i \geq R_i^{SLA}) \leq \frac{Var[R_i]}{(R_i^{SLA} - E[R_i])^2}$$

$$P(R_i \geq R_i^{SLA}) \approx \min\left(\frac{Var[R_i]}{(R_i^{SLA} - E[R_i])^2}, 1\right). \tag{2}$$

In this paper, we extend our preliminary work presented in [4] into three directions. First, we adopt the Chebyshev's Inequality instead of the Markov's Inequality, as it meets a much better trade-off between simplicity and precision. In particular, [3,14] shows that it yields very similar results if compared to the exact distribution for M/M/1 queues, in spite of the much simpler mathematical expression. Second, we obtain a convex formulation of the optimization problem with guaranteed global optimum. Finally, we provide an extensive experimental analysis and a comparison with other previous approaches from the literature.

Note that our estimate of the probability of response time violation for WS class i depends on the fraction f_i of the server capacity assigned to the corresponding VM, as well as on the VM's achieved throughput X_i, which represent the two decision variables of our optimization model (see next section). VM i's throughput, X_i, is constrained not only by the (predicted) arrival rate of class i requests, λ_i, but also by the maximum arrival rate that VM i can support before saturation (utilization equal to 100%), i.e., $X_i < \frac{1}{D_i/f_i}$ (or $X_i < \frac{f_i}{D_i}$). In order to avoid the performance instability common to systems running close to saturation, we introduce an upper-bound υ_i ($0 \leq \upsilon_i < 1$), up to which the utilization of VM i is planned. An upper-bound on throughput X_i can then be computed as $X_i \leq \min\left(\lambda_i, \frac{\upsilon_i f_i}{D_i}\right)$.

5. Joint admission control and resource allocation problem

In our model, the SLA is defined by considering the response time of each Web service invocation. If the response time of a service invocation is above a given threshold R_i^{SLA}, then, the SLA is violated, and the customer will not pay for the Web service. Vice versa, if the response time is lower than R_i^{SLA}, the customer will pay ω_i to the provider. With this payment scheme, penalties are modeled as missed payments due to a high response time. In the case of a rejected request, the customer is not supposed to pay for the Web service invocation either, since a rejected request can be considered a particular type of QoS violation (availability violation).

Let us denote by T the control horizon time interval, and by C the cost per time unit associated with the use of server capacity. The decision variables are X_i, class i throughput, and f_i, the computing capacity fraction assigned to VM i for the next period. The provider's goal is to minimize revenue loss from SLA violations (rejections or response time violations) and the cost of system resource usage, thus maximizing profits. Formally, the objective is to minimize:

$$\sum_{i=1}^{N} T \cdot \left\{\omega_i \left[(\lambda_i - X_i) + P(R_i \geq R_i^{SLA})X_i\right] + Cf_i\right\}. \tag{3}$$

The quantities $(\lambda_i - X_i)T$ and $P(R_i \geq R_i^{SLA})X_iT$ are equal to the numbers of SLA violations due to the admission control system and due to response time violations, respectively, predicted for the next control interval T. The term Cf_iT indicates the cost of use of the computing resources (including energy and cooling) required to serve the admitted class i WS requests in the control time horizon.

After replacing Eq. (2) and dropping the term $\sum_{i=1}^{N} \omega_i\lambda_iT$, which is a constant with respect to decision variables f_i and X_i, the optimization problem can be formulated as:

$$\text{(P1)} \quad \min \sum_{i=1}^{N}\left[-\omega_iX_i + Cf_i + \omega_iX_i \min\left(\frac{Var[R_i]}{(R_i^{SLA} - E[R_i])^2}, 1\right)\right]$$

$$D_iX_i \leq f_i\upsilon_i \quad \forall i \tag{4}$$

$$X_i \leq \lambda_i \quad \forall i \tag{5}$$

$$\sum_{i=1}^{N} f_i \leq 1 \tag{6}$$

$$X_i, f_i \geq 0 \quad \forall i. \tag{7}$$

Constraint family (4) entails that the overall utilization of system resources dedicated to serve class i requests is below the planned threshold υ_i, and guarantees that the queuing network model is in equilibrium. Constraint family (5) entails that class i throughput is less than or equal to the predicted incoming workload, while constraint (6) guarantees that at most 100% of the server capacity is used in the next period.

Note that, the term $\min\left(\frac{Var[R_i]}{(R_i^{SLA}-E[R_i])^2}, 1\right)$ can be re-written as $\min\left(\frac{\frac{D_i^2}{(f_i-X_iD_i)^2}}{(R_i^{SLA} - \frac{D_i}{f_i-X_iD_i})^2}, 1\right)$, while the objective function can be re-stated as:

$$\sum_{i=1}^{N}\left[-\omega_iX_i + Cf_i + \omega_iX_i \frac{\frac{D_i^2}{(f_i-X_iD_i)^2}}{(R_i^{SLA} - \frac{D_i}{f_i-X_iD_i})^2}\right].$$

The change in the objective function can be applied because, in any optimal solution, the following condition holds:

$$\tau_i = \frac{\frac{D_i^2}{(f_i-X_iD_i)^2}}{(R_i^{SLA} - \frac{D_i}{f_i-X_iD_i})^2} \leq 1 \quad \forall i. \tag{8}$$

Indeed, if, by contradiction, in any local or global optimum for a given class $\bar{i}$ with $X_{\bar{i}} > 0$, the value $\tau_{\bar{i}}$ is strictly greater than 1, then the optimum can be improved by setting $X_{\bar{i}} = f_{\bar{i}} = 0$, while satisfying all the constraints. Then, the following optimization problem can be considered:

$$\text{(P2)} \quad \min \mathcal{F}(\mathbf{X}, \mathbf{f}) = \sum_{i=1}^{N}\left[-\omega_iX_i + Cf_i + \omega_iX_i \frac{\frac{D_i^2}{(f_i-X_iD_i)^2}}{(R_i^{SLA} - \frac{D_i}{f_i-X_iD_i})^2}\right]$$

s.t. constraints from (4) to (7).

(P2) has a non-linear convex objective function with linear constraints, as demonstrated by the following Theorem. Hence, the global optimal solution can be determined.

Theorem 1. *The objective function of problem (P2) is convex in the feasible solution set.*

Proof. See Appendix. □

5.1. Optimization technique

(P2) is a general nonlinear optimization problem and has to be solved within the short-term planning horizon of autonomic computing (i.e., 5–10 min [5,40,7]). In such a short optimization time, commercial nonlinear optimization tools cannot solve full size instances. For realistic problems of a reasonable size, a decomposition approach has been considered.

The optimization problem includes two distinct families of variables: X_i, which determine the optimal admission control for the next control horizon, and f_i, which determine the optimal allocation of system resources. In similar contexts [50,4], the Fixed Point Iteration technique (referred to as FPI throughout the rest of the paper) has been proved efficient in providing optimal solutions. FPI identifies the optimal value of a subset of variables, while the value of the other variables is fixed, and stops when the difference between two subsequent values of the objective function is lower than a given threshold ϵ_{FPI}.

Our approach is reported in Algorithm 1. First, variables f_i are randomly initialized to non-negative values to satisfy constraint family (6) as an equality (step 1). Then, the values of variables f_i are held fixed, and the optimal admission control variables X_i are identified (step 4). In the next step, the values of X_i are held fixed, and the optimal resource allocation variables f_i are determined (step 6).

The next two sections present the solutions of the admission control and resource allocation.

```
1 $\mathbf{f} = BuildRandomSolution$
2 $\Delta = \infty$
3 while $\Delta > \epsilon_{FPI}$ do
4     $\mathbf{X} = optAC(\mathbf{f})$;
5     $g_1 = Eval(\mathbf{X}, \mathbf{f})$
6     $\mathbf{f} = optRA(\mathbf{X})$
7     $g_2 = Eval(\mathbf{X}, \mathbf{f})$
8     $\Delta = |g_2 - g_1|$
9 end
```

Algorithm 1: Optimization Procedure.

5.2. The admission control sub-problem

If the capacity allocation of system resources in the virtualized environment is held fixed, i.e. $f_i = \overline{f}_i$, then the admission control sub-problem can be formulated as follows:

$$(\text{P3}) \quad \min \sum_{i=1}^{N} \left[-\omega_i X_i + C\overline{f}_i + \omega_i X_i \frac{\frac{D_i^2}{(\overline{f}_i - X_i D_i)^2}}{(R_i^{SLA} - \frac{D_i}{\overline{f}_i - X_i D_i})^2} \right]$$

$$0 \le X_i \le U_i = \min\left[\frac{\overline{f}_i v_i}{D_i}, \lambda_i\right] \quad \forall i \tag{9}$$

where X_i are the decision variables. (P3) is separable, and N admission control sub-problems (P3$_i$) of a single variable X_i can be solved independently. Each sub-problem can be formulated as:

$$(\text{P3}_i) \quad \max g_i(X_i) = X_i - X_i \frac{\frac{D_i^2}{(\overline{f}_i - X_i D_i)^2}}{(R_i^{SLA} - \frac{D_i}{\overline{f}_i - X_i D_i})^2}$$

$0 \le X_i \le U_i$.

Note that if the resource allocation is fixed, the optimal solution of the admission control sub-problem is independent of price ω_i.

(P3$_i$) has a non-linear objective function with the independent variable bounded in the interval $[0, U_i]$. The solution can be obtained by determining the unique stationary point p_i of $g_i(X_i)$. If p_i is feasible, then it is the optimal solution. Otherwise, the optimal solution is the best between the two extremes of the feasibility interval. This property follows by the convexity of the global problem (P2).

5.3. The resource allocation sub-problem

If the admission control problem has been solved, and the throughput for each WS class i is held fixed at $X_i = \overline{X}_i$, then the resource allocation sub-problem can be formulated as:

$$(\text{P4}) \quad \min h(\mathbf{f}) = \sum_{i=1}^{N} \left[-\omega_i \overline{X}_i + Cf_i + \omega_i \overline{X}_i \frac{\frac{D_i^2}{(f_i - \overline{X}_i D_i)^2}}{(R_i^{SLA} - \frac{D_i}{f_i - \overline{X}_i D_i})^2} \right]$$

$$f_i \ge \frac{D_i \overline{X}_i}{v_i} \quad \forall i \tag{10}$$

$$\sum_{i=1}^{N} f_i \le 1 \tag{11}$$

where f_i represent the only decision variables. Note that a feasible solution exists for problem (P4) and can be identified by setting $f_i = \frac{D_i \overline{X}_i}{v_i}$ for each i, since the condition (11) is satisfied by the solution of each problem ($P3_i$).

$h(\mathbf{f})$ is convex since $\mathcal{F}(\mathbf{X}, \mathbf{f})$ is convex. Let $f_i^{(1)} = \frac{D_i \overline{X}_i}{v_i}$ be the lower bound of class i fraction of capacity. The first partial derivatives of the objective function $h(f_i)$ are:

$$\frac{\partial h}{\partial f_i} = C + \frac{2D_i^2 R_i^{SLA} \omega_i \overline{X}_i}{(-f_i R_i^{SLA} + D_i(1 + R_i^{SLA} \overline{X}_i))^3}.$$

The first order stationary condition provides only one real root:

$$f_i^{(2)} = \frac{D_i}{R_i^{SLA}} + \sqrt[3]{\frac{2D_i^2 \omega_i \overline{X}_i}{C(R_i^{SLA})^2}} + D_i \overline{X}_i.$$

Let us denote by $\overline{N} \subseteq [1, N]$ the set of indexes i, such that $f_i^{(2)} < f_i^{(1)}$ and let $F = 1 - \sum_{i \in \overline{N}}^{N} f_i^{(1)}$. If the stationary point is feasible with respect to linear constraints (10) and (11), then it is also the global optimal solution. Otherwise, for all $i \in \overline{N}$ the optimal solution is obtained by setting $f_i = f_i^{(1)}$, i.e., class i is not profitable with respect to the cost of resource usage, and we set the capacity devoted to the corresponding VM equal to the lower bound. For the remaining VMs, two cases are possible:

- if $\sum_{i \notin \overline{N}} f_i < F$, we are in light load conditions, and the optimal solution can be found by setting the variables equal to their stationary values;
- otherwise, we are in high load conditions, and the optimal solution is obtained by using the whole server capacity, i.e., constraint (11) is satisfied as an equality, and $\sum_{i \notin \overline{N}} f_i = F$.

This property is demonstrated by the following theorem.

Theorem 2. *In the optimal solution of problem (P4), the capacity for WS class i is given $f_i = f_i^{(1)}$, for each $i \in \overline{N}$. For each $i \notin \overline{N}$, f_i is either $f_i^{(2)}$ or it belongs to the plane $\sum_{i=1}^{N} f_i = 1$.*

Proof. See Appendix. □

5.4. Optimization technique implementation

We have developed an ad-hoc controller which implements the procedure reported in Algorithm 1. Since the objective function of problem (P3) is concave and continuously differentiable in the feasibility region, its solution is efficiently obtained by applying standard line search techniques, such as bi-section [26]. The solution of problem (P4) is obtained either by setting the variables equal to their stationary points if feasible, or by applying an ad-hoc gradient technique [26] that determines, at each step, the optimal resource allocation between two WS classes.

The complexity of the solution of each admission control sub-problem is $O(1)$. Hence, the complexity of the solution of problem (P3) is $O(N)$. The complexity of the solution of problem (P4) is likewise $O(N)$. Overall the FPI procedure complexity is $O(n_{FPI} \times N)$, where n_{FPI} is the number of iterations required by the procedure to converge within a factor ϵ_{FPI}. In Section 6.1, we will show that the number of iterations required to converge within a precision of 10^{-3} is typically within 5 and 15. Therefore, our approach has an efficient solution with time complexity that increases linearly with the number of hosted WS classes, within a reasonably small constant factor.

6. Experimental results

We extensively evaluated our dynamic resource management solution, aiming at assessing both its scalability and cost-effectiveness in comparison with alternative existing solutions. Towards that goal, we consider different representative and realistic scenarios, built from a variety of system and workload configurations.

Section 6.1 presents the most representative experimental results on the scalability of our optimization algorithm. A cost-benefit evaluation of the proposed framework, comparing it with alternative strategies built from top-performing state-of-the-art resource allocation and admission control techniques [36,12], is presented in Section 6.2. Finally, a discussion on the sensitivity of our approach to two key assumptions, namely, exponential service times and exact prediction of incoming workloads, is presented in Section 6.3.

6.1. Performance of the optimization algorithm

This section reports results on the time efficiency of our optimization algorithm for a growing number of WS classes. We ran experiments with various configurations of the model parameters, evaluating the algorithm execution time for a wide range of problem instances. The set of experiments conducted to assess the time efficiency and scalability of our algorithm, drawn from the literature [3,9,50], is described next.

The algorithm is tested on a wide set of randomly generated problem instances. The number N of WS classes is varied between 100 and 1000, a range that captures the sizes of current installations of VMM monitors, such as VMWare ESX and Xen, which run up to 170 VM on 32 physical cores machines [48].

For each value of N, 20 test cases are generated. In each test case, the values of D_i are uniformly randomly generated in the range [0.1, 3] seconds. R_i^{SLA} is set proportional to the average service demand $R_i^{SLA} = \alpha_i \times D_i$, where α_i is a randomly generated dimensionless constant in the range [10, 30], as in [50, 7]. Furthermore, for each test case, 7 problem instances are built as follows. First, the fraction of the incoming workload for each WS class is randomly generated. Then, the total incoming workload, i.e., $\sum_{i=1}^{N} \lambda_i$, is varied in such a way that the total system utilization ranges between 20% and 80% at steps of 10%. By doing so, we are able to test our solution performance in both lightly and heavily loaded conditions, as in [50].

The price of each class i request is set to $\omega_i = \beta_i \times D_i$, where β_i is a constant expressed in \$ /s, uniformly randomly generated in the interval [1, 5]. With these values of ω_i, the revenues obtained by using a single CPU for one hour varies between \$1 and \$10, according to the current commercial fees (see, for example, the

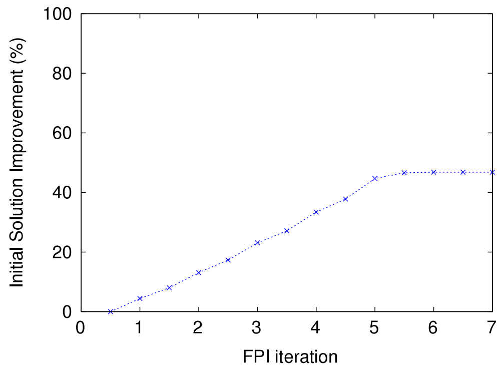


Fig. 2. FPI execution trace.

Sun Utility Computing infrastructure[3] or *Amazon Elastic Cloud*[4]). The planned maximum utilization of each VM, v_i, is varied between 60% and 90%, as in [36,12]. The cost C, per time unit, associated with the use of system resources is varied between 10% and 30% of the maximum revenues per unit of capacity, as in [50].

Overall, $20 \times 10 \times 7 = 1400$ problem instances are built and analyzed. The FPI threshold ϵ_{FPI} (the stopping criterion) is set to 10^{-3}.

Fig. 2 shows a representative execution trace of the FPI procedure for one of the 1400 problem instances analyzed. Note that, at every iteration the FPI performs two steps, i.e., one optimization for X variables (half iterations in Fig. 2 plot) and one for f variables. Results show that the initial solution of FPI is improved by about 40%–50%. The number of iterations n_{FPI} required to converge within the precision $\epsilon_{FPI} = 10^{-3}$ is typically between 5 and 15. For all cases analyzed, we found that the optimal solution can be determined in about 20 s on a general purpose PC, even for very large systems (up to 1000 WS classes). Thus, we believe that the FPI algorithm is very efficient and can be applied in real-world practical scenarios.

6.2. Cost-effectiveness of the dynamic resource management framework

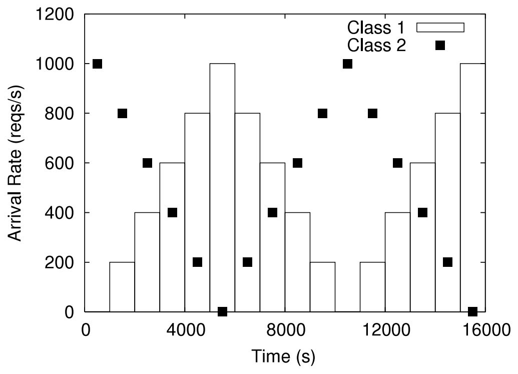
This section presents an evaluation of the economic efficiency of our dynamic resource management framework. The evaluation is performed through simulation, by comparing results against previous solutions available in the literature. The main metric of interest is the provider's total profit (i.e., revenues minus costs) obtained with each strategy, according to the pricing scheme discussed in Section 5. Complementary, we also quantify the fraction of SLA violations, which not only has direct impact on the provider's profits but also reflects (at least partially) the quality of service experienced by customers.

We consider five scenarios of interest in our evaluation. In scenarios 1 and 2 (Section 6.2.1), built from synthetic workload traces, we compare our framework against alternative solutions based on the resource allocation strategy proposed in [36] and the admission control scheme proposed in [12]. These comparisons aim at highlighting the *main trade-offs* and benefits from our framework. In scenarios 3, 4, and 5 (Section 6.2.2), we further evaluate our strategy, comparing against previous solutions, for more *realistic* workload configurations built from traces of requests to a *real Web server system*. The five scenarios are summarized in Table 3, which shows, for each them, the number of WS classes (N), the workload type as well as a brief description of other aspects of the experimental setup.

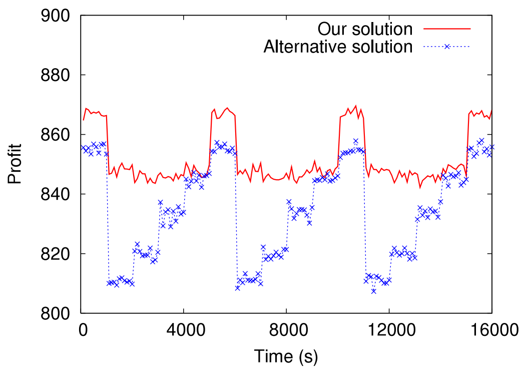
[3] http://www.sun.com/service/sungrid/index.jsp.
[4] http://aws.amazon.com/ec2/.

Table 3
Scenarios for evaluation of the cost-effectiveness of our solution.

Scenario	N	Workload	Description
1	2	Synthetic	Our resource allocation solution vs. [36].
2	2	Synthetic	Our joint solution vs. alternative ones. Scenario 1 with doubled arrival rates.
3	12	Built from real access traces	Our joint solution vs. alternative ones. Customers from the same timezone.
4	12	Built from real access traces	Our joint solution vs. alternative ones. Scenario 3 with higher infrastructure costs.
5	12	Built from real access traces	Our joint solution vs. alternative ones. Scenario 3 with 6 classes shifted by 2 h.



(a) Workload profiles.



(b) Provider's profit.

Fig. 3. Resource allocation: Our solution vs. the alternative one [36] (Scenario 1).

We have built an event-driven simulator that takes as input the workload traces of N WS classes [14]. The simulator is coupled with the optimization model solver based on Algorithm 1, which is called, at the end of each control interval, to calculate the resource allocation f_i and the accepted throughput X_i for each class i, for the next interval. During each interval, per-request response time as well as per-class throughput and resource allocation are collected and used to compute the provider's profit. Our simulator employs a fair admission control mechanism, which accepts a class i request with probability X_i/λ_i. This is a conservative approach compared to other mechanisms that aim at minimizing the inter-arrival time variance [20]. We also note that the assumption of Poisson arrivals holds also for requests that are accepted into the system.

Since our current focus is on the economic efficiency of each strategy, we here consider a best-case scenario to understand key trade-offs. In particular, we assume that there is no time limitation for adapting the system, and that an ideal workload forecasting is used, i.e., a perfect prediction of the number of requests to process in the next interval is available. The selection and evaluation of a practical workload forecasting method, among several existing techniques [2] with a varying degree of accuracy, is left for future work. Nevertheless, in Section 6.3.2, we evaluate the sensitivity of our approach to errors in the prediction of the incoming workload.

In all experiments, we assume that: (i) the maximum planned utilization for all VMs is 90%, (ii) the price for serving each class i request within the pre-determined SLA is 1, and (iii) the response time SLA for each class is equal to 30 times the class average service demand. In other words, we set $\upsilon_i = 0.90$, $\omega_i = 1$, and $R_i^{SLA} = 30 \times D_i$, for all i. We also assume that the cost per time unit associated with the infrastructure, C, is a fraction (10%, unless otherwise noted) of the maximum achievable profit per unit of capacity, which is computed as the product of the price for serving a class i request, ω_i, by the maximum theoretical throughput rate, given by $X_i \leq 1/D_i$ [20]. Thus, $C = 0.1 \times \omega_i \times 1/D_i$ (see also [50,4]).

The following sections present the most significant results for all scenarios. The reported results are averages of 5 runs with standard deviations under 4% of the means.

6.2.1. Synthetic workloads

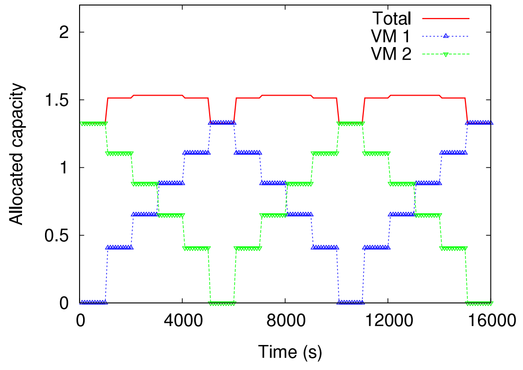
This section discusses results for the first two scenarios, both built from two WS classes ($N = 2$) with synthetic workloads. For the sake of simplicity, we consider classes characterized by homogeneous service demands with average values equal to 1 ms (i.e., $D_i = 0.001$) and heterogeneous workloads. However, similar results were obtained for heterogeneous D_i as well.

In scenario 1, requests from each class arrive according to the periodic step-like non-homogeneous Poisson processes shown in Fig. 3(a). Arrival rates vary from 0 to 1000 requests per second, with step and period of 1000 and 10 000 s, respectively. Both workloads have identical profiles with a shift in their periods. This is an interesting scenario for dynamic and self-adaptive allocation strategies, which are able to reassign idle capacity from underloaded to overloaded VMs in order to satisfy SLA requirements. We assume the resource allocation strategies are called at the end of each control interval, which coincides with time instants when per-class request rates change.

In this scenario, we focus on the resource allocation problem, comparing our solution against the method proposed in [36], here referred to as simply *the alternative*. Recall that this resource allocation method aims at determining the resource (i.e., capacity) needed by each VM i in order to meet a target utilization range ($U_i^{\min}$, $U_i^{\max}$), given the incoming workload. Thus, it does not perform admission control. Moreover, in contrast to our solution, resource allocation is performed separately and independently for each VM.

Therefore, in order to make a fair comparison, we take the following actions. First, we turn the admission control component of our framework off, i.e., we admit all incoming requests. Second, we run the alternative resource allocation method [36], collecting the maximum capacity, $\mathcal{C}_{\max}$, *simultaneously* allocated by the method for *all* VMs.[5] We then calculate the provider's revenues from requests served within pre-defined SLA requirements as

[5] We note that $\mathcal{C}_{\max}$ is measured as a factor of the total capacity of the physical server.



(a) Our resource allocation.

(b) Alternative resource allocation [36].

Fig. 4. Allocated capacity (Scenario 1).

Table 4
Economic efficiency of resource management approaches (Scenarios 1 and 2).

Scenario	Solution	Metric	Target utilization range			
			0.4–0.6	0.5–0.7	0.6–0.8	0.7–0.9
1	Ours	Profit Gains over Alternative	5%	2%	1%	5%
		Avg. Allocated Capacity	1.55	1.47	1.42	1.31
		% SLA Violations	0.07	0.05	0.03	0.1
		Avg. Throughput (req/s)	1000	1000	1000	1000
	Alt. [36]	Avg. Allocated Capacity	1.93	1.62	1.40	1.24
		% SLA Violations	0.2	0.4	1.1	5.0
		Avg. Throughput (req/s)	1000	1000	1000	1000
2	Ours	Profit Gains over Alt. A	39%	38%	39%	41%
		Profit Gains over Alt. B	22%	20%	19%	5%
		Profit Gains over Alt. C	23%	37%	52%	58%
		Avg. Allocated Capacity	2.9	2.77	2.5	2.22
		% SLA Violations	0.04	0.04	0.5	1.0
		Avg. Throughput (req/s)	1996	1996	1984	1843
	Alt. A	Avg. Allocated Capacity	2.5	2.2	1.9	1.68
		% SLA Violations	0.1	0.3	1.4	9.1
		Avg. Throughput (req/s)	1448	1444	1430	1319
	Alt. B	Avg. Allocated Capacity	2.5	2.47	2.43	2.22
		% SLA Violations	0.1	0.1	1.2	1.2
		Avg. Throughput (req/s)	1668	1713	1728	1756
	Alt. C	Avg. Allocated Capacity	2.5	2.2	1.9	1.68
		% SLA Violations	8	11	12	30
		Avg. Throughput (req/s)	1631	1463	1308	1180

well as the costs associated with the total capacity allocated. In comparison, we run our method using $\mathcal{C}_{\max}$ as the total system capacity.[6] We then compare the total provider's profits achieved by both methods. By performing such comparison, we are, in turn, comparing the economic efficiency of the two resource allocation strategies given a *fixed total system capacity*.

Fig. 3(b) shows the provider's profits over time for both strategies, when the alternative resource allocation technique [36] targets the system utilization range $U_i \in (0.5, 0.7)$. The capacity assigned to each VM as well as the total system capacity effectively used by our method and by the alternative solution are shown in Fig. 4(a) and (b), respectively. Since this scenario is built by allocating as much capacity to each VM as it needs to reach the target utilization range, the load is relatively low for the system configuration. Thus, most requests that arrive (and thus are admitted into the system) will be served within their SLA requirements. In other words, there is little room for benefit from any one strategy over the other.

Nevertheless, this scenario highlights some fundamental issues and trade-offs that are worth noting with respect to the allocation strategies applied by both methods. By taking the request response time distribution into account to share available resources among hosted VMs, our optimizer can make a better use of the total capacity. In fact, it allocates fewer resources than the alternative method, thus saving infrastructure costs, while still yielding a (slightly) smaller number of SLA violations. The alternative solution, which takes into account only the infrastructure utilization and makes independent resource allocation decisions, allocates more resources (see Fig. 4). It also yields a larger number of SLA violations. As a result, it leads to a (slightly) lower total profit for the provider (2%, on average, in this light load scenario). Similar results are also obtained for different target utilization ranges, as shown in Table 4, which presents the overall profit gains of our solution over the alternative one, the average allocated capacity, the average fraction of requests for which SLAs are violated as well as the average fraction of requests which were served within their SLA requirements (i.e., average throughput) for each method.

In scenario 2, the two WS classes have the same workload profiles as in scenario 1, but the workload intensities are twice as heavy, in each step, for both classes. In other words, we double the arrival rates while keeping average service demands and SLAs equal to scenario 1. In this heavier load scenario, we consider our joint resource allocation and admission control solution,

[6] This is equivalent to inflating average service demands D_i by $\mathcal{C}_{\max}$ in our performance and optimization models as well as response time SLA expressions.