Self-Adaptive SLA-Driven Capacity Management for Internet Services

Bruno Abrahao, Virgilio Almeida and Jussara Almeida
Computer Science Department
Federal University of Minas Gerais, Brazil

Alex Zhang, Dirk Beyer and Fereydoon Safai
Intelligent Enterprise Technology Lab.
HP Labs, Palo Alto, CA, USA

Abstract— **This work considers the problem of hosting multiple third-party Internet services in a cost-effective manner so as to maximize a provider's business objective. For this purpose, we present a dynamic capacity management framework based on an optimization model, which links a cost model based on SLA contracts with an analytical queuing-based performance model, in an attempt to adapt the platform to changing capacity needs in real time. In addition, we propose a two-level SLA specification for different operation modes, namely, normal and surge, which allows for per-use service accounting with respect to requirements of throughput and tail distribution response time. The cost model proposed is based on penalties, incurred by the provider due to SLA violation, and rewards, received when the service level expectations are exceeded. Finally, we evaluate approximations for predicting the performance of the hosted services under two different scheduling disciplines, namely FCFS and processor sharing. Through simulation, we assess the effectiveness of the proposed approach as well as the level of accuracy resulting from the performance model approximations.**

I. Introduction

With the exponential growth in popularity over the past few years, Internet services have become a popular solution for businesses to offer their traditional services online to customers and to deploy internal business operations in a distributed fashion. In light of this, businesses are increasingly relying on computing-based capacity outsourcing [1] as a financially attractive approach to host their services. In this scenario, businesses sign SLA contracts [2] with a provider that hosts a large diversity of Internet services in shared Internet data centers (IDC). On the other hand, in order to be profitable, providers direct their efforts to manage resources in the most cost-effective way while satisfying the established customers' service level requirements.

In this work, we consider a scenario where a number of different third-party transactional Internet services are hosted in a shared platform which employs mechanisms to provide service differentiation. The focus of our work is on the capacity management for IDCs in such a way as to explore the available resources to the provider's best advantage so that a business goal is maximized.

The emergence of new customer demands pose unprecedented operational challenges to this problem. First, in order to stay competitive, instead of simply agreeing on satisfying an average response time requirement, providers have been offering contracts that promise the satisfaction of a tail distribution requirement of response time [3], meaning that an upper bound on the probability of the end-to-end response times exceeding a given threshold is required. Moreover, in addition to establishing end-to-end response time requirements, customers have also been demanding from the providers a guarantee on the throughput achieved by their services. Last but not least, there has been a recent outburst of interest in service contracts in which customers pay only for actual use [4], thereby forcing providers to review the traditional SLA contracts and proposing more flexible service accounting schemes.

The above facts have direct impact on the capacity management of Internet services. First, the performance modeling of applications requires more complex models which, in some cases, cannot be solved efficiently, or a solution that captures the relevant characteristics of the system is not known. On the other hand, the satisfaction of the performance requirements has strong implications on the provider's revenue [2]. As a consequence, the capacity management model should be driven with respect to a cost model, based on per-use accounting, in which capacity need conflicts are handle in light of the provider's financial objective, instead of simply meeting the customers' performance requirements. Finally, to make matters more complicated, these services exhibit workloads that may present great fluctuations over time, which changes the service capacity needs during their operation dramatically. This clearly calls for a dynamic approach to self-adapt the system in real time, as a response to such changes.

A. Research Contributions

This paper presents a model for self-adaptive capacity management in shared environments, driven by a cost model based on SLA contracts. Due to the variability in level exhibited by the workloads, we propose a two-level SLA specification, namely, normal and surge operation modes, which allows customers to pay for extra capacity (than that normally required) only when needed. In addition to the proposed SLA contract, we also propose a cost model based on penalties, incurred by the provider as a result of service level requirement violations, and rewards, paid by customers when their expectations, expressed as requirements of throughput subject to a response time guarantee, is exceeded.

In order for the IDC to respond to workload changes and, consequently, adapt to changing capacity needs, a real-time self-adaptive framework for managing capacity within an IDC

is proposed here. Our approach is based on an optimization model, which, with the objective of maximizing the net result from penalties and rewards, links the proposed cost model with a queuing-based performance model that predicts the performance of the hosted Internet services.

The computation of the probability distribution of response times, required when predicting the capacity needs with respect to the tail distribution requirements, is also challenging when dealing with queuing models. This is because i) the exact results for this metric are only available for special types of queues, most of which do not realistically capture the characteristics of the system considered, and ii) some of the available results make the optimization problem hard to solve in real time. As a consequence, approximations are often needed. Therefore, we propose and evaluate different approximations to express the probabilistic response time requirement, comparing the level of accuracy resulting from each of them in the context of our problem formulation under two different service scheduling discipline, namely fist come first served (FCFS) and processor sharing.

Last, we assess the effectiveness of the proposed approach through discrete event simulation of the multi-service IDC along with the resulting cost implications on the financial affairs of the provider.

This paper is organized as follows. Section II describes the environment considered and the cost model proposed. The self-adaptive framework for capacity management is presented in Section III. Section IV describes the parts that comprise the capacity management model, and the experimental analysis is presented in Section V. Finally, Section VI discusses related work, and Section VII offers our conclusion.

II. Environment Description

This section provides an overview of the environment and the cost model upon which our autonomic capacity management framework is built.

A. Hosting Platform Description

We consider a scenario where a provider hosts multiple third-party transactional Internet services in a shared IDC.

A key feature of IDCs is the ability to provide performance isolation by preventing the direct contention for resources between different services. Accordingly, a fair sharing policy or a virtualization scheme [5], [6] is employed so as to provide service differentiation. These mechanisms partition the physical resources (i.e., processing, storage and communication resources) into multiple isolated virtual ones, each running at a fraction of its corresponding physical resource capacity. Hence, instead of using the physical resources directly, the hosted Internet services demand service from a pool of virtual resources, created and maintained by an intervening *virtualization layer*.

Typical Internet services are usually composed of different transaction types, subjected to different workloads, with different service demands on the resources and executed by independent software components. We denote these components as *application classes*. Under the assumption that the classes are independent, they are analyzed here as independent applications.

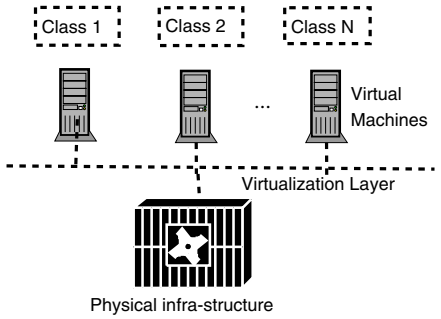


Fig. 1. Internet service hosting platform.

The hosting platform considered is depicted in Figure 1. The virtualization layer creates N isolated virtual machines (VM) (also known as resource containers or application environments [5], [7]) on top of the physical infra-structure. Each VM is composed of a set of virtual instances of each of the IDC's resources and is dedicated to serving a single application class only. This hosting model isolates classes one from another, each using the virtual machine as if it were a dedicated server, working at a fraction of the total (physical) capacity.

Virtualization allows the physical resources to be proportioned to application classes, each receiving at least as much capacity as has been assigned to it, regardless of the load imposed by other classes. Thus, it enables the IDC to flexibly contract or expand the resource capacities assigned to application classes. Hence, we define the capacity allocation decision as the determination of the fractions of server capacity each VM i ($i = 1, .., N$) obtains from the corresponding physical one.

Last, we assume that the VMs employ an admission control scheme [8] that rejects requests for various purposes. For example, the VMs may drop some requests to avoid service instability due to capacity limitations or to guarantee that the requirements of response time are met.

In this analysis, we focus on the dynamic capacity management model within the server nodes, and we make use of a high level of abstraction, by considering each VM as a single resource. We left the extension of the managing each of the VM's devices (i.e. CPU, disk, etc) independently, as well as service replication and multi-tiered services for future work.

B. Cost Model

The service contracted by customers must meet certain performance expectations which the parties agree upon in the SLA contracts. This work focuses on the requirements of throughput and response time. In addition to agreeing on the requirements in the SLA contracts, the provider also establishes a service rate to charge customers proportional to the strictness of their service level requirements. For instance,

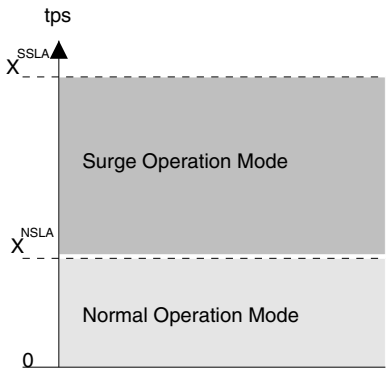


Fig. 2. Example of an arbitrary valid throughput scale with the ranges of normal and surge operation modes.

customers who run critical businesses are willing to pay more so as to obtain high throughput and/or short response times whereas other customers sign up for service with loose requirements but lower costs.

The Internet offers plenty of examples of services which usually receive low to moderate load, but occasionally receive an exceptionally large surge of requests. Common examples are the online news services which expect a predictable number of users during normal operation but, whenever certain special events occur, a sudden surge of clients overload the site, changing the capacity needs dramatically [9].

Due to the highly dynamic nature of Internet workloads, we propose contracts with two levels of requirements, which correspond to two different operation modes, namely, normal and surge. In the normal operation mode, customers contract the service level which satisfies their needs for the majority of operation time whereas in the surge operation mode, a higher service level limit is established, up to which the provider has an incentive to assign extra capacity to services so as to accommodate occasional workload peaks. Traditional contracts with a single performance target would require customers whose workload presents high peak-to-mean ratio to pay for the service level needed to satisfy both the average and the peak of demand during the entire operation, even though only part of the capacity the customer pays is actually utilized most of the time. Therefore, from the business standpoint, the two-level SLA approach can be advantageous both to customers who pay for extra capacity only when needed, and to providers who are able to offer more attractive service plans by operating with more flexibility.

There are several ways of measuring the service level provided to a customer. In this work, the SLA performance requirements quantify the VM's ability to process transactions, provided the quality of the processed transaction response times satisfy a given requirement. Accordingly, it is considered the tail distribution response time requirement which states that the response time of the transactions from class i must not exceed a given threshold R_i^{SLA} for more than $\alpha_i \times 100\%$ of the time, or equivalently, $P(R_i > R_i^{SLA}) \leq \alpha_i$, where R_i is the response time of a single transaction of class i. Moreover, in light of the above reasoning, the proposed SLA contracts contain performance targets for the normal operation mode, under NSLA requirements, and for the surge operation mode, under SSLA requirements.

In order to present the operation modes, let us first introduce some definitions. The achieved performance of applications is computed at the end of periodic intervals so that the payoff of the service's execution (i.e., the net result from penalties and rewards) can be determined. In addition, we refer to the the actual processing rate in which transactions are performed within the response time requirement as the *valid throughput*. Accordingly, the remaining transactions which violate the response time requirement are not considered in the SLA accounting process.

Figure 2 displays an example of an arbitrary scale that shows the range of the normal and the surge operation modes. The NSLA requirement states that the valid throughput of application class i should be at least X_i^{NSLA} transactions per second, which is the upper limit to the normal operation mode. Nonetheless, violations of the NSLA requirements may occur, in which case the provider agrees to refund part of the service charge to customers, which is proportional to the difference between the NSLA throughput requirement and the saturated valid throughput, provided this difference has been caused by capacity limitations. Conversely, the SSLA target is established in order to prevent missing service or unavailability in case of occasional workload peaks. Thus, the parties establish a limit on the throughput, X_i^{SSLA} ($X_i^{SSLA} \geq X_i^{NSLA}$), which is the upper limit to the surge operation mode, up to which the owner of class i agrees to pay a reward to the provider whenever the valid throughput over the interval exceeds X_i^{NSLA}. Accordingly, the rewards are proportional to the extra valid throughput achieved.

Given the framework described, we define the provider's business objective as the provision of capacity to VMs so as to maximize the net result from penalties and rewards.

Finally, it should be emphasized that this approach is not restricted to only two SLA levels. Thus, one may define multiple levels which specify multiple performance targets so as to account for different levels of demands.

III. Self-Adaptive Framework

In order to provide the system with the ability to adapt itself in response to changes, we propose the closed control loop presented in Figure 3. The heart of the model is the *capacity manager* component which, given the expected workload of each VM, the SLA requirements, and the system characteristics of the classes, reconfigures the IDC with the goal of maximizing the provider's business objective. Therefore, the capacity manager component is configured with SLA and system parameters of each application class (discussed in Section II-B and summarized in Table I of Section IV-A) (1). These values are updated whenever contract changes occur (i.e., a new application class is introduced or removed from the data center, or the requirements change).

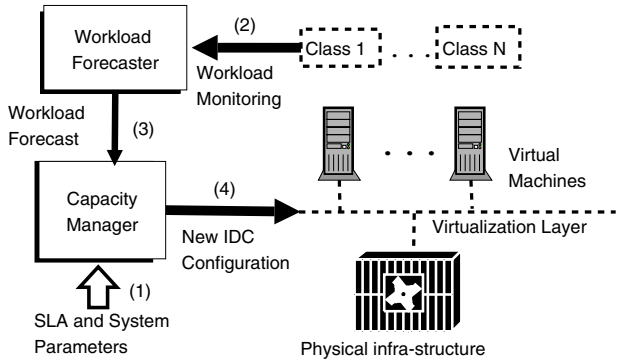


Fig. 3. Closed Control Loop

The *workload forecaster* component is responsible for constantly monitoring the intensity of the workload of each class over time (2). It contains a local storage facility (i.e., a database) to store the past behavior of the workload so that the logged data can be used for periodic analysis. Accordingly, it uses one of the existing workload forecasting methods [10] to predict, based on past observed behavior, the workload that each class is likely to receive in the near future. Consequently, the predicted workload is also used as input by the capacity manager (3).

After determining a capacity allocation decision, the capacity manager component sends the new IDC configuration to the system (4), adjusting the virtual resource mappings in the virtualization layer into new values at the physical infrastructure and the admission control parameters for each VM.

We refer to the interval between consecutive controller interventions as the *controller interval*. The durations of these intervals may be fixed or variable depending on the characteristics of the system and its services. For example, services whose workload present a constant workload behavior can make use of adaptations triggered by unexpected changes in demand. On the other hand, the capacity manager invocations can occur at periodic intervals of time, which are usually configured for periods in which the workload of services is unlikely to change. Moreover, limitations on the time required to adapt the system might define the minimum length of the interval. Finally, we consider that the payoff for the execution of the services (the net result from penalties and rewards) during each controller interval is computed at the end of the corresponding time period.

The capacity manager component presented in this work is based on an optimization model which links the cost model to a performance model (discussed in Section IV-D) and determines, in light of the SLA specifications and the predicted workload for each application class, a capacity allocation decision that maximizes the provider's business objective for the next controller interval.

IV. Capacity Management Model

This section describes the capacity management model for Internet services. We first describe the model parameters and its assumptions (Section IV-A), the penalty and the reward accounting scheme (Section IV-B), and, finally, we present the optimization model for capacity allocation (Section IV-C).

TABLE I
Model Parameters

Symbol	Description
SLA parameters	
X_i^{NSLA}	valid throughput requirement (in tps) for class i in normal operation mode.
X_i^{SSLA}	valid throughput upper limit (in tps) for class i in surge operation mode.
R_i^{SLA}	response time threshold (in seconds) for class i.
α_i	limit on probability of response time being greater than R_i^{SLA} for class i.
c_i	penalty cost for a unit of throughput NSLA saturation for class i.
π_i	reward price for a unit of extra valid throughput above X_i^{NSLA} for class i.
System parameters	
N	number of application classes, and thus number of VMs.
υ_i	maximum utilization planned for VM i.
$E[S_i]$	average service time (in seconds) of class i transactions on the physical server.
Workload Forecaster's estimates	
λ_i^*	predicted arrival rate (in tps) of class i for next controller interval.
$P\{Z_i\}$	probability of capitalizing rewards from class i during next controller interval.

A. Main Model Parameters and Assumptions

The main model parameters are presented in Table I. Some of the parameters are extracted directly from the SLA contracts while others are system parameters and estimates made by the workload forecaster component, used as inputs to the capacity manager.

The SLA parameters can be understood in light of the discussion presented in Section II-B. The system parameters consist of: 1) the number N of VMs maintained by the virtualization layer, thereby the number of application classes hosted by the provider, 2) the maximum utilization υ_i the provider allows VM i to reach, and 3) the average service time of transactions from class i on the physical server.

Parameter υ_i is introduced because both the mean and the variance of response times can increase without limit when the utilization approaches 100%. Thus, in order to maintain a certain level of stability in the VMs, a fractional upper limit υ_i ($0 \leq \upsilon_i < 1$) is specified, up to which the utilization of VM i is planned.

We assume that transactions from the same application class are statistically indistinguishable and, thus have the same average service time on the physical server. Therefore, $E[S_i]$ is used here as a model parameter to indicate the mean service time of class i, which can be approximated by the arithmetic

average of the observed values of S_i, measured in a pre-production execution of the applications on the physical server.

Since each VM receives a guaranteed fraction of time from the physical server, taking f_i as the fraction of the server assigned to the VM i, we approximate the average service time on the VM as $E[S_i]/f_i$. Accordingly, the capacity allocation decision is the determination of the capacity fractions (i.e., $f_i, i = 1..N$) to be assigned to each VM i. Therefore, f_i is the main decision variable of the proposed problem.

The last two rows of Table I present the outputs of the workload forecaster component. As mentioned in Section III, the capacity manager component receives from the workload forecaster component an estimate λ_i^* of the arrival rate of requests during the next controller interval for each application class i.

The capacity optimizer component can be invoked in response to arrival rate deviations from the above estimate. However, when considering fixed controller intervals, events that are significantly shorter than the interval could mislead the manager. For example, a surge of requests with short duration can cause the capacity manager to seize the capacity needed by other classes to meet their NSLA requirements so as to provide extra capacity to a more profitable one. Clearly, this would incur penalties throughout the controller interval whereas the reward would be capitalized during a tiny fraction of the time.

In order to minimize this undesired effect, the workload forecaster component may also produce the estimated probability of a class receiving a surge of requests, that is, an arrival rate greater than X_i^{NSLA}, during the next controller interval. Thus, parameter $P\{Z_i\}$ indicates the certainty level to bet on reward capitalization for VM i. If the system is allowed to adapt to workload changes, this parameter can be bypassed by setting it to 1.

We consider λ_i as the arrival rate of class i over the controller interval. However, due to capacity limitations, some of the requests may be rejected, in which case the actual throughput of the class becomes smaller than λ_i. Moreover, some of the processed transactions may violate the response time requirement, in which case they are not counted (for SLA purposes to compute the valid throughput) as processed. In order to differ these cases, X_i is introduced to denote the actual throughput, and μ_i to denote the valid throughput of class i, over the controller interval.

B. Penalty and Reward Accounting

This section describes how penalties and rewards are computed and how the payoff for the execution of services is calculated at the end of each controller interval.

We start by defining Q_i as the relative frequency of transactions with response times below R_i^{SLA} at the end of the controller interval. Thus, if the requirement has been violated, that is $Q_i < (1 - \alpha_i)$, then $\mu_i < X_i$, and the valid throughput is given by the transactions processed within the time limit plus the tolerance allowed: $\mu_i = Q_i X_i + \alpha_i(Q_i X_i) = (1 + \alpha_i)(Q_i X_i)$. Otherwise, when $Q_i \geq (1 - \alpha_i)$, all the transactions processed during that interval were within the response time requirement and, thus, $\mu_i = X_i$.

TABLE II
VALUES OF PENALTY AND REWARD

	Symbol	Condition	Value
Penalty	Y_i	$\mu_i < X_i^{NSLA}$	$min\{X_i^{NSLA}, \lambda_i\} - \mu_i$
Reward	Z_i	$\mu_i \geq X_i^{NSLA}$	$min\{X_i^{SSLA}, \mu_i\} - X_i^{NSLA}$

Upon defining the response time tail distribution requirement as a QoS condition, the NSLA saturation is considered here as the VM's inability to achieve the valid throughput X_i^{NSLA} requirement. Consider Y_i to be the penalty incurred by the provider due to violating the performance requirement of VM i. As mentioned in Section II-B, the penalties are proportional to the magnitude of the difference between the NSLA throughput requirement and the valid throughput. However, the throughput may have been smaller than the requirement due to capacity limitations or due to an arrival rate smaller than X_i^{NSLA}. Since the provider should be penalized only for the former case, a penalty is incurred whenever $\mu_i < X_i^{NSLA}$ and $\mu_i < \lambda_i$. In this case the penalty value is $Y_i = \lambda_i - \mu_i$. However, since the NSLA target requires the provider to process at least X_i^{NSLA} transactions over the controller interval, if $\lambda_i \geq X_i^{NSLA}$, the incurred penalty should be $Y_i = X_i^{NSLA} - \mu_i$. Thus, the penalty cases just described can be expressed as $Y_i = min\{X_i^{NSLA}, \lambda_i\} - \mu_i$.

Conversely, if the valid throughput of an application class has exceeded the throughput NSLA requirement, $\mu_i \geq X_i^{NSLA}$, the provider is able to capitalize rewards proportionally to the magnitude of the difference between the extra valid throughput and the NSLA throughput requirement. However, the reward value cannot exceed the limit X_i^{SSLA}. Let us denote Z_i to be the reward value paid by class i owner and express the reward case as $Z_i = min\{X_i^{SSLA}, \mu_i\} - X_i^{NSLA}$.

The above conditions determine the values of the SLA penalties and rewards computed at the end of the controller interval. The penalty or reward value for any other possible condition is 0. All penalty and reward cases are summarized in Table II.

C. Optimization Model Formulation

The optimization model evaluates the estimated net result from penalties and rewards at the beginning of each controller interval, using the workload predicted by the workload forecaster component in order to provide a capacity allocation decision which maximizes the provider's business objective.

The model objective function expresses the sum over all application classes of the expected payoff for their execution as follows:

$$\textit{Maximize} \sum_{i=1}^{N} -c_i Y_i + \pi_i Z_i P\{Z_i\} \tag{1}$$

The whole optimization model is depicted in Figure 4 and in order to refer to the model constraints, they are indexed

$$
\begin{aligned}
\textit{max} \quad & \textstyle\sum_{i=1}^{N} -c_i Y_i + \pi_i Z_i P\{Z_i\} \\
\textit{s.t.} \quad & \\
& Y_i \geq min(X_i^{NSLA}, \lambda_i^*) - X_i && (a) \\
& Y_i \geq 0 && (b) \\
& Z_i = \delta_i (min(X_i^{SSLA}, X_i) - X_i^{NSLA}) && (c) \\
& Z_i \geq 0, \quad \delta_i \in \{0,1\} && (d) \\
& \lambda_i^* = \lambda_i^{acc} + \lambda_i^{rej} && (e) \\
& X_i = \lambda_i^{acc} && (f) \\
& P(R_i \geq R_i^{SLA}) \leq \alpha_i && (g) \\
& \rho_i = \lambda_i^{acc} / \lambda_i^{sat} \leq \upsilon_i && (h) \\
& \textstyle\sum_i^N f_i \leq 1 && (i) \\
& 0 \leq f_i \leq 1, \quad \rho_i \geq 0 && (j) \\
& X_i, \lambda_i^{acc}, \lambda_i^{rej}, \lambda_i^{sat} \geq 0 && (k)
\end{aligned}
$$

Fig. 4. Optimization model

using lowercase letters (a) to (k).

Constraints (a) to (d) express the cases shown in Table II, replacing the arrival rate λ_i and the valid throughput μ_i of the past controller interval by the estimate λ_i^* and the actual throughput X_i respectively. Notice that since $Y_i \geq 0$ is stated in constraint (b), Y_i assumes 0 in cases of reward (i.e., when $X_i > X_i^{NSLA}$). In addition, in constraints (c) and (d), an artificial binary variable δ_i is introduced as part of the expression for Z_i. Thus, δ_i is forced to 0 when $X_i < X_i^{NSLA}$, making $Z_i = 0$ in case of penalties. Otherwise, δ_i assumes the value 1.

Due to capacity limitations, the VMs impose a limit on the number of requests that can be processed. However, the arrival rate can be greater than the maximum achievable processing rate in which case some of the requests must be dropped. Therefore, we assume that the VMs employ one of the existing admission control policies [8] which splits the arrival rate a VM receives into $\lambda_i = \lambda_i^{acc} + \lambda_i^{rej}$, where λ_i^{acc} is the rate at which VM i accepts requests, and λ_i^{rej} is the rate at which VM i rejects requests. This is expressed in the optimization model by constraint (e), using λ_i^* as the expected arrival rate.

Constraint (f) expresses the job flow balance condition [11] which states that all accepted requests are actually processed by the VM, thereby assuring that no accepted request is lost.

If the reader will recall, the valid throughput of a VM is subject to the response time requirement satisfaction. Therefore, this is expressed as model constraint (g). Since the expected response time is a function of the VM throughput and the fraction of capacity it receives, $R_i = f(\lambda_i^{acc}, f_i)$, constraint (g) forces the VMs to accept only the transactions that would satisfy the response time requirement for any given value of f_i, with the objective of making $\mu_i = X_i$ at the end of the controller interval.

Notice that α_i expresses a tradeoff between the throughput and the quality of the processed transactions. Indeed, a smaller α_i results in lower throughput but guarantees a higher degree of response time requirement satisfaction, whereas a large α_i increases the processing rate of transaction at the expense of their response times.

The utilization ρ_i of VM i is defined by constraint (h) as the ratio of λ_i^{acc} and the smallest arrival rate λ_i^{sat} at which VM i becomes saturated (see Section IV-D). It is also important to emphasize that, similarly to the expected response time, ρ_i is a function of both the acceptance rate of request and the fraction of capacity the VM receives. Thus, $\rho_i = g(\lambda_i^{acc}, f_i)$. Moreover, constraint (h) guarantees the stability condition by limiting ρ_i to $\upsilon_i \times 100\%$.

A direct consequence of the above constraints is that λ_i^{acc} is limited by constraints: (e), (g), and (h), by the expected arrival rate, the response time satisfaction, and the maximum allowed utilization respectively.

The capacity allocation constraint is expressed in (i). This sum limits the capacity percentages assigned to the VMs to 100%.

Finally, constraints (j) and (k) delimit the domain of the variables.

The optimization model presented in this section can be combined with several performance prediction techniques to estimate the values of the performance-related variables λ_i^{sat} and $P(R_i \geq R_i^{SLA})$, which result in different levels of accuracy. With this objective, in the next section, we discuss methods based on queuing models.

D. Estimating the Performance Metrics

This section presents an analytical queuing model to calculate the values of the performance metrics used in the optimization model presented in Section IV-C.

As discussed earlier, the most challenging part of the performance prediction for the model considered is the estimation of the probability distribution of response times. This is because this exact probability distribution of response time can only be computed for some types of queues which do not directly apply to the system considered, and some of the available expressions are complex and, therefore, limit the optimization model real time computation [12]. Thus, we propose and compare different approximations for this purpose. The following assumptions hold for the queuing analysis:

- Since there is considerable evidence that the arrival process imposed by users follows Poisson [13], [12], [14], we assume Poisson request arrivals. Moreover, for the sake of simplicity, we assume that the service times of application classes are exponentially distributed and left the study of other traffic patterns, which are characteristic of specific applications, as future work.
- We consider two types of queues for modeling transactional service centers, namely M/M/1 and M/G/1 with processor sharing (PS). The former works under FCFS scheduling discipline, which may be more accurate for

modeling, for example, transactional servers with write operations, whereas the latter serves customers in a round robbin fashion with a very small quantum, which closely mimics the behavior of multi-threaded servers. Both queues have been frequently considered as reasonable abstractions for transactional service centers [12], [11].

Let us start by finding the saturating throughput of VM i. Applying the Utilization Law [11] results in the following expression for the utilization: $\rho_i = \frac{E[S_i]}{f_i} \times X_i$. Thus, the saturating throughput is determined to be $X_i = \frac{\rho_i f_i}{E[S_i]} \leq \frac{f_i}{E[S_i]} = \lambda_i^{sat}$. Notice that with this result, constraint (h) of Figure 4 imposes the following limitation on the rate of acceptance of requests: $\lambda_i^{acc} \leq \upsilon_i f_i / E[S_i]$.

Let us now turn our attention to approximations to compute the tail probability distribution of the response time.

We consider Markov's Inequality [15] as a first approximation, requiring only the expected response time of the classes, which is computed using the same expression for both M/M/1 and M/G/1 (PS) queues [16]: $E[R_i] = \frac{E[S_i]/f_i}{1-\rho_i}$. Thus, using the mean response time and guaranteeing the stability condition, it is possible to approximate the tail distribution response time requirement as:

$$P(R_i \geq R_i^{SLA}) \leq \frac{E[R_i]}{R_i^{SLA}} = \frac{1}{R_i^{SLA}} \frac{E[S_i]}{f_i - \lambda_i^{acc} E[S_i]} \leq \alpha_i \tag{2}$$

The advantage of using Markov's Inequality is that it only requires the average response time and can be applied to both M/M/1 and M/G/1 (PS) interchangeably. Nevertheless, Equation 2 often provides a loose bound and, therefore, it could result in overly strict requirements of response time.

Often, it is possible to improve upon Markov's inequality to obtain a tighter bound by using Chebyshev's Inequality [15]. However, this result depends on both the average and the variance of the response time. The latter metric is queue dependent and, for the M/M/1 case, is given by $Var[R_i] = \frac{(E[S_i]/f_i)^2}{(1-\rho_i)^2}$ [16]. For M/G/1 (PS) queues, the computation of the response time variance uses an integration term which makes the optimization problem hard to solve. However, it can be shown that $Var[R_i] \leq \frac{\rho_i (E[S_i^2]/f_i^2)}{(1-\rho_i)^3}$ gives a tight upper bound of the variance for a PS queue [17], especially for large jobs. Notice that the latter equation requires both the first and second moment of the service times, which we assume can be directly measured in a pre-production execution of the application on the physical server.

Given the mean and the variance of the response time of applications, Chebyshev's inequality limits the probability of a transaction's response time being greater than the SLA threshold as:

$$P(R_i \geq R_i^{SLA}) \leq \frac{Var[R_i]}{(R_i^{SLA} - E[R_i])^2} \leq \alpha_i \tag{3}$$

Although Equation 3 often give more precise bounds than Equation 2, it requires more information, such as the queue type and the second moment of service times.

The last proposed approximation uses the response time CDF exact solution for calculating the percentile of the response times for the M/M/1 queue [16] so as to express the tail distribution response time requirement as:

$$P(R_i \geq R_i^{SLA}) = e^{-R_i^{SLA}(f_i/E[S_i])(1-\rho_i)} \leq \alpha_i \tag{4}$$

Notice that Equation 4 is only exact when there is no rejection of requests. Thus, since we are assuming a general admission control policy, this expression is also an approximation. The equivalent result for M/G/1 (PS) queue is still an open problem [17].

In Section V, we compare the level of accuracy provided by each of the approximations, Markov, Chebyshev and Percentile, presented in this section.

V. Experimental Analysis

In this section, we evaluate the efficacy of the capacity management model as well as the approximations proposed to model the classes' performance.

Our results are derived from the discrete event simulation of the environment considered. The simulator executes the classes concurrently, each running on a separate VM. The capacity manager component is called periodically in order to change the settings of each VM, that is, f_i and λ_i^{acc}. We have also instrumented the simulator in order to collect performance measurements, such as throughput, response time, queue length and utilization of VMs.

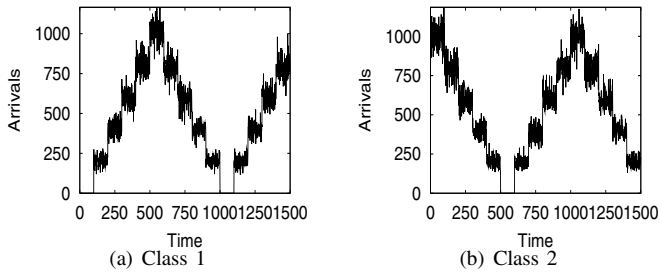
Using the approximations presented in Section IV-D, the optimization model becomes a non-linear problem (due to the non-linear constraint of response time). Thus, we have conducted the experiments using the non-linear solver DONLP2 [18]. Experiments with up to 50 application classes resulted in solution times under one second in a Athlon 2.2 GHz with 512 MB of main memory.

A. Experimental Setup

In the experiments described in this section, as it suffices to assess the quantitative results, we present results for two competing VMs maintained on top of the shared physical infra-structure. We have run experiments varying the number of VMs up to 50, and the qualitative results are similar.

It is known that different admission control policies have different effects on the performance of the system [8]. Since the main concentration in this study is on the resulting effect of employing the capacity management model, for the sake of simplicity, in the following set of experiments, a strict and conservative admission policy based on tokens is employed. That is to say that at each second a limited number of tokens is set up, which transactions need to acquire in order to enter the system. This limit is configured (dynamically) at each VM, using the value of variable λ_i^{acc}, after solving the optimization model.

In the experiments involving adaptation, the selection and evaluation of the workload forecasting method is out of the scope of this discussion. In fact, one can use one of the existing



(a) Class 1 (b) Class 2

Fig. 5. Synthetic periodic step-like Poisson arrival processes submitted to a) application class 1 b) application class 2.

time series forecast methods [10] and obtain results that vary with different degrees of accuracy. Instead, we assume an ideal forecasting, meaning that the future request arrival rate is known *a priori*. We also assume that there is no time limitation for adapting the system.

The set of experiments conducted to assess the adaptive approach were driven by synthetic workloads. Workload generators are used to submit requests following a periodic step-like non-homogeneous Poisson processes [15] whose shape is similar to the ones presented in Figure 5. This figure presents the arrivals submitted to each of the classes where the average arrival rate in the traces ranges from 0 to 1000 arrivals per second, each periodic cycle lasts for 1000 seconds, and each step lasts for 100 seconds. Notice that we introduce a shift of 500 seconds in the arrival process presented in Figure 5(b) with respect to that in Figure 5(a) in order to produce a displacement of their periods. This scenario makes for an interesting analysis of the ability of the system to redeploy its capacity, by assigning the idle capacity of the VM experiencing low workload activity to the overloaded one.

Given the characteristics of the workload, the system is configured to adapt at the end of each workload step, that is 100 seconds, and, since the average request rate is stable over the steps, we bypass $P\{Z_i\}$ by setting it to 1. In addition, as the main focus is on the ability of the autonomic approach to adapt the system in response to workload changes, equal requirements are set up for the classes, which are summarized in Table III. Last, since we aim at studying the performance of the virtual machines only, the virtualization layer allows us to completely abstract the physical topology of the data center.

For the case in which the performance of the classes is approximated as a M/M/1 queue, we simulate a physical server in which both applications are served with exponentially distributed service time whose parameter is $E[S_i] = 10^{-3}$, using FCFS scheduling discipline. For the M/G/1 (PS) case, the applications experience the same service time distribution with the same parameter, but with processor sharing scheduling discipline. Notice that the maximum throughput of each application using the entire physical server capacity is 1000 tps. As a consequence, the server is underprovisioned to serve the two application workloads simultaneously as their average request rates may vary from 0 to 1000 tps each.

TABLE III
SYSTEM CONFIGURATION

υ_i	α_i	X_i^{NSLA}	X_i^{SSLA}	R_i^{SLA}	c_i	π_i
0.95	0.1	500	1000	0.1	1.0	0.5

As a reference, we have run the classes without the capacity manager intervention, referred to here as the *Static* approach. In the latter, the system is started up with the results obtained from the optimal solution obtained from the best capacity allocation model found for each queue (see Section V-B), considering the average value of the workload over the entire simulation period.

B. Numerical Results

In sequence, the effect of the concurrent execution of the classes for the M/M/1 and M/G/1 (PS) modeling cases, using the three approximations, namely, Markov, Chebyshev and Percentile is analyzed.

We present the average payoff over 10 runs of the execution of the classes for each modeling case, throughout a period corresponding to half an hour, along with the 90% confidence interval of the averages in Figure 6.

The results presented in Figure 6 show that for the M/M/1 case, using the Static approach, the payoff for the execution varies approximately from -400 to -200 while using the Markov approximation this value varies between -240 and 100. The payoff obtained from the Chebyshev approximation is slightly smaller than the payoff obtained by the Percentile approach even though the two curves overlap most of the time. These latter two approximations produced values that range approximately from 0 to 200. In the M/G/1 (PS) modeling case, the Static approach produces payoffs varying approximately from -310 to -130, while using the Markov approximation these values range from -125 to 100, and using Chebyshev from -90 to 240.

Notice that the repeating pattern of the curves are produced by the periodic behavior of the workloads submitted to the VMs. In this case, the peaks of payoff values for the static case corresponds to periods of time in which the two workloads approach their average values together. Accordingly, these are periods where the adaptive approaches obtain their smaller payoff, as there is little opportunity to assign idle capacity of one VM to another. Indeed, periods where the peak of the workload submitted to one VM complements the valley of the other are the cases in which the adaptive approaches achieved the highest payoff values.

It is interesting to analyze the performance behavior of the classes by analyzing the achieved response time, queue length and throughput throughout the runs.

Figures 7 and 8 show the CDFs of the queue lengths at both VMs, collected at each transaction departure. The two figures show that, for the M/M/1 case, when the system is adapted using any of the approximations, the queue length consistently remains at nearly 15, whereas when there is no adaptation, the queue lengths vary from 30 to 48 in approximately 75%

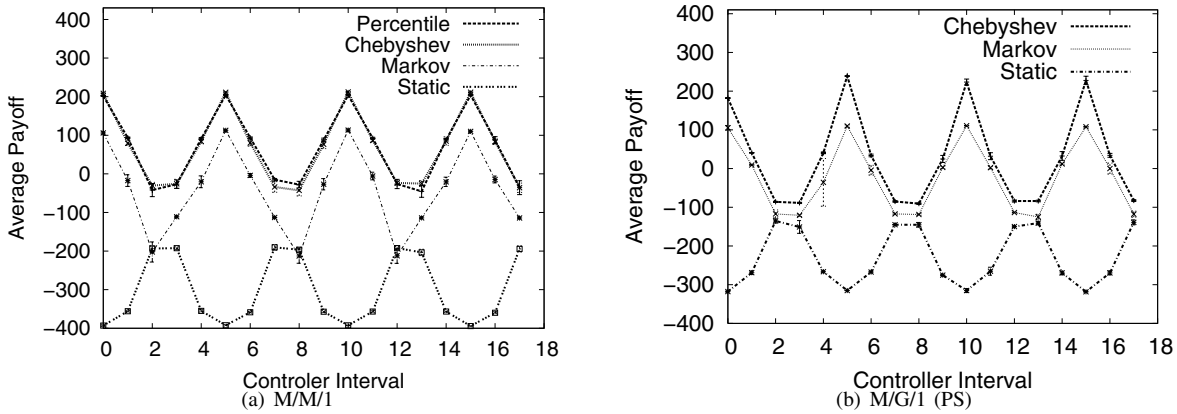


Fig. 6. Average payoff for the execution of applications produced by the adaptive approach, using the Percentile, Chebyshev and Markov approximations, compared with the Static approach a) for the M/M/1 case b) for M/G/1 (PS) case.

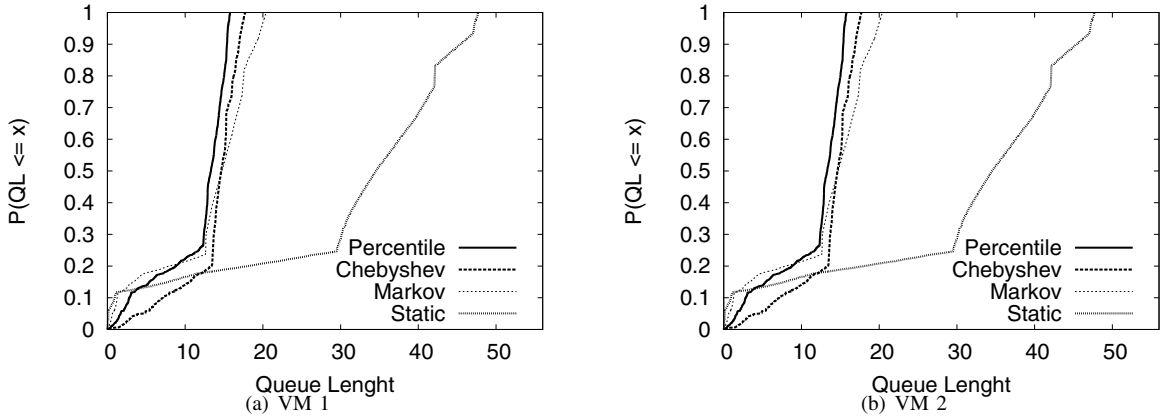


Fig. 7. CDF of the queue lengths over time of a) VM 1 and b) VM 2 for the M/M/1 case

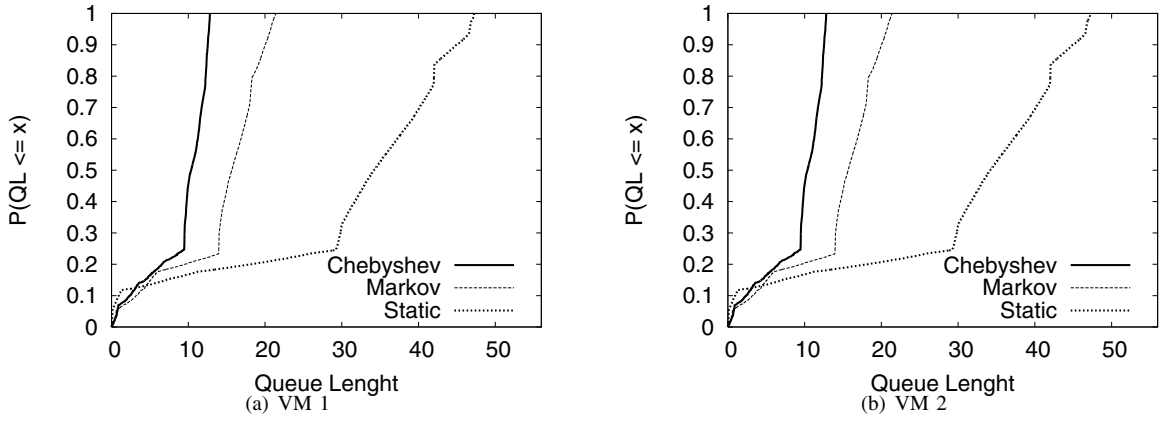
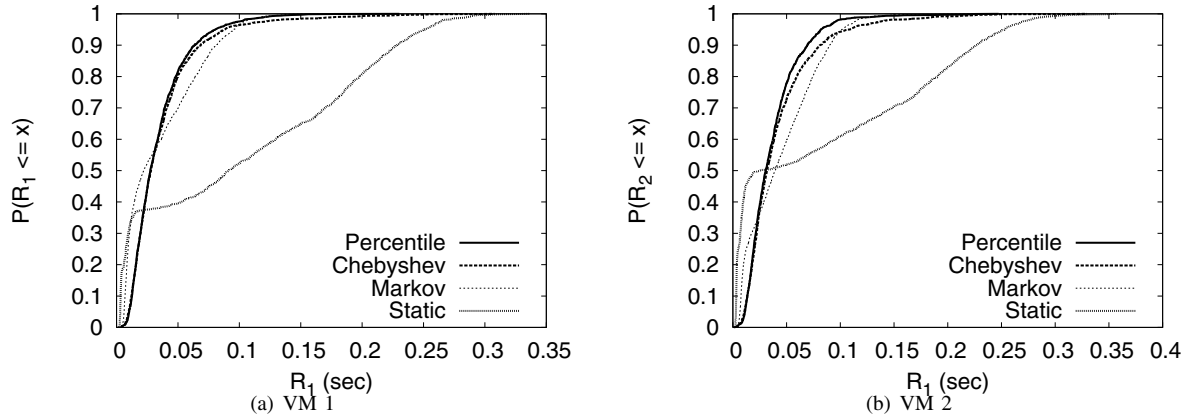


Fig. 8. CDF of the queue lengths over time of a) VM 1 and b) VM 2 for the M/G/1 (PS) case.

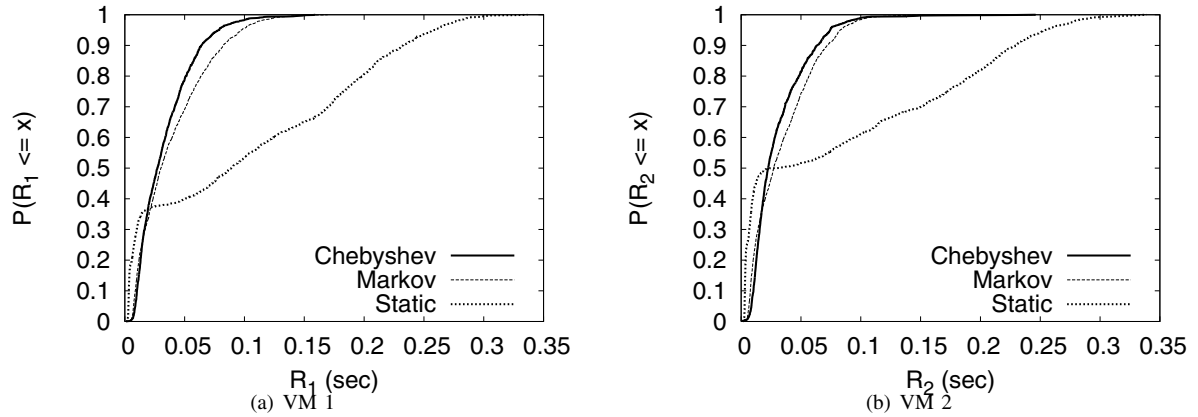
of the cases. For the M/G/1 (PS) case, using the Chebyshev approximation, the queue length stays at 10 most of the time; using the Markov approximation, this value is 15; and when no adaptation takes places, it varies between 30 and 48 in approximately 75% of the cases.

Notice that in the cases where there is adaptation, the queue lengths stay within the theoretical value of the average number of customers for M/M/1 and M/G/1 (PS) queues, given by



(a) VM 1 (b) VM 2

Fig. 9. CDF of the response times of a) VM 1 and b) VM 2 for the M/M/1 case.



(a) VM 1 (b) VM 2

Fig. 10. CDF of the response times of a) VM 1 and b) VM2 for the M/G/1 (PS) case

$Q_i = \frac{\rho_i^2}{1-\rho_i}$. Replacing ρ_i by the maximum utilization values $\upsilon_i = 0.95$, at which we planned the VMs' operation, gives $Q_i = 18.05$.

In order to analyze the satisfaction of the SLA tail distribution response time requirement during the runs, we have plotted the CDFs of the response times for the VMs in Figure 9 for both modeling cases. Notice that the response time requirement is met for both VMs, in the two modeling cases, when using the adaptive approaches with any approximation. Accordingly, the response time of the transactions were shorter than 0.1 seconds more than 90% of the time. By contrast, using the static approach the response time threshold is respected approximately 55% and 60% of the time for the VMs 1 and 2 respectively in the M/M/1 case, and approximately 57% for both VMs in the M/G/1 (PS) cases.

Up to this point, we have shown that the response time requirements of the classes are attained by using any of the approximations for the two modeling cases and, therefore, they are equally effective for guaranteeing the response time requirement satisfaction. In contrast, the resulting throughput explains why one approximation results in higher payoffs than the others. In order to observe this effect, let us analyze the magnitude of the penalties and rewards achieved by each class separately for both modeling cases. We present the CDFs of the magnitude of the penalties and rewards over the controller intervals for each VM, for the M/M/1 modeling case in Figure 11 and for the M/G/1 (PS) case in Figure 12.

Clearly, in all the plots, the static approach resulted in significantly higher penalties and smaller rewards in comparison to the results provided when the system is adapted. In addition, the values resulting from the Markov approximation are also smaller than the ones achieved when using Chebyshev and Percentile. These latter two approaches produced curves that almost coincide. For the M/M/1 case, this is clear in the values corresponding to rewards for both Figures 11(a) and 11(b), and is also observed in the values of penalties in the Figure 11(b) only. This is because, in this experiment, the classes have the same requirements, and, thus, the solver favored VM 1 in these cases, since this decision produces a equally optimal solution as distributing the capacity evenly between the classes. For the M/G/1 (PS) queue, this difference is also observable, especially in reward values. This result

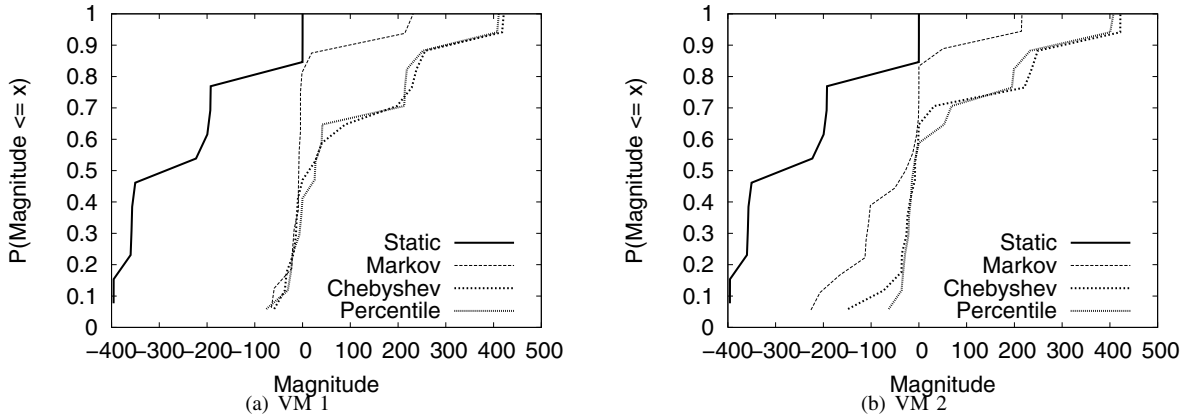


Fig. 11. CDFs of the magnitude of penalties and rewards produced by a) VM 1 and b) VM 2, for the M/M/1 case.

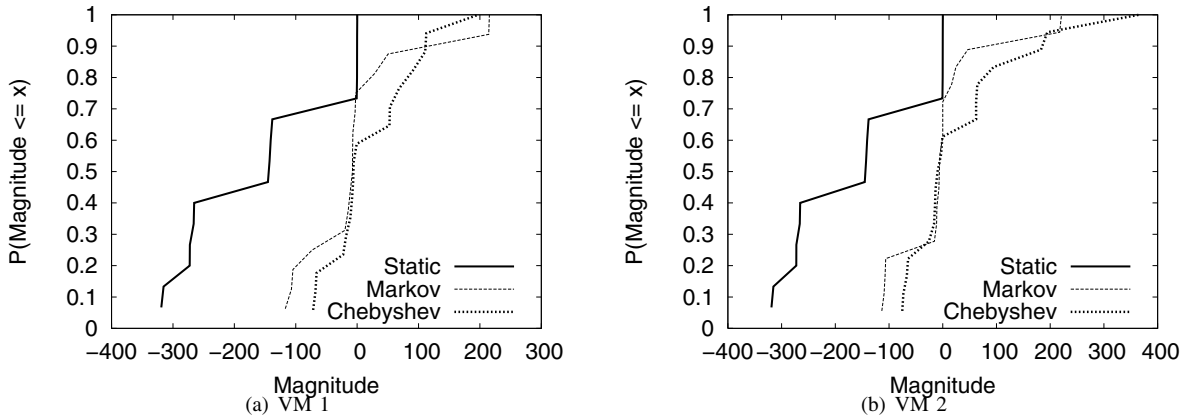


Fig. 12. CDFs of the magnitude of penalties and rewards produced by a) VM 1 and b) VM 2, for the M/G/1 (PS) case.

conforms to Figure 6(b) where the payoff difference produced by Markov and Chebyshev approximations is greater in the higher values while they nearly coincide in the smaller values.

In conclusion, we can see that the use of the adaptive approach proposed is able to significantly increase the net result from penalties and rewards in comparison to the Static approach. We can observe that the Markov approximation is more conservative than the others, because it over-estimates capacity needs due to the response time requirements. On the other hand, the Chebyshev approximation produces average payoffs slightly smaller than the Percentile solution for the M/M/1 case. However, we have observed through other experiments that when the ratio $\frac{\lambda_i}{\lambda_i^{acc}}$ increases, thereby altering the Poisson characteristic of the arrival rate, Chebyshev approximation is able to produce better results than Percentile. Moreover, Chebyshev can also be applied to the M/G/1 (PS) modeling case, producing improvements in comparison with the Markov approximation.

VI. Related Work

There have been several proposals to maximize the capitalization on the strategic advantages that shared data centers may deliver. For instance, Ross and Westerman [1] study the impact of the maturity of both business and technological strands on the revenue of providers; in [2], the authors present evidences, showing that business-oriented design for shared environments reduces the financial risks associated with service level violations. Rappa [19] proposes service contracts for IDCs similar to the ones used for public utilities, wherein customers would pay only for actual use, and shows the potential of these agreements for future computing services.

Regarding the systems issues, the authors of [4] present the main technological challenges in the path to the maturity of data centers, which aim at adapting applications to this new paradigm; Graupner et al [6] discuss the impact of virtualization on the management of shared environments.

The manual management of resources operated by humans has become increasingly unsuitable for modern computing systems. In the light of this fact, autonomic (or self-managing) approaches appeared as a solution for adequately administer the complexity of such systems. In this direction, previous work considered autonomic closed control loops using different techniques for a diversity of purposes. For instance,

a model based on a queuing model that periodically reconfigures the parameters of a web server so as to increase its performance is presented in [20]. The authors in [7] considered mapping the workload level of a data center to its observed influence on the system. Aiming at maximizing the sum of utility functions of the performance, they rely on a state-space based on the past behavior so as to provide information for future capacity allocation decisions. The authors in [21] further revisited the latter framework, proposing a combinatorial search technique together with queuing models to improve the efficiency of the table-driven approach. Other analyses propose a control theoretical approach [22], [3]. However, the approach proposed in this paper is based on an optimization model which is able to express the important properties, constraints and, more importantly, bind the performance of the hosted services to an SLA cost model driven by penalties and rewards. Moreover, unlike ours, these previous work manage resources with respect to deterministic requirements of response time (i.e., maximum average response time).

Static resource management optimization problems in IDCs has been dealt with in several studies. However, they usually have different goals. For example, [23] optimizes the resource allocation in order to minimize network traffic and resource underutilization; studies [12] and [13] deal with the problem of resource allocation for maximizing SLA profits of an e-commerce provider. However, they assume static workload, abstract the whole servers as discrete units of capacity allocation, and work with response time requirements alone. In contrast, our work presents an SLA business model to handle the operational challenges posed by new customer demands and presents different approximations to capture the realistic performance behavior of the services, measured through the processing rate subjected to a response time guarantee.

VII. Conclusions

In this work we have considered a self-adaptive capacity management approach for a multi-service environment driven by a cost model based on SLA contracts with the goal of best exploring the IDC's resources. At the business level, we propose a two-level SLA specification for different operation modes that works with a cost model based on penalties and rewards, which allows the per-use service accounting with respect to the dual SLA requirement of throughput and tail distribution response time. In the system level, we evaluate approximations based on queuing theoretical formulas for predicting the performance of the hosted services under two different scheduling disciplines, namely FCFS and Processor Sharing. The system and business levels are linked by an optimization model which allows the capacity manager to adapt the IDC to changing capacity needs in real time so as to maximize a provider's financial objective.

Finally, we have demonstrated that the use of the proposed self-adaptive model can effectively manage the capacity allocation so as to significantly increase the financial value derived from the IDC. In addition, we have also presented the difference in the level of accuracy resulting from the use of different approximations to express the tail distribution requirement of response time using queuing models.

Acknowledgments

This work was developed in collaboration with Hewlett Packard Brazil R&D (Project CAMPS HP-UFMG-2005).

References

[1] J. W. Ross and G. Westerman, "Preparing for utility computing: The role of IT architecture and relationship management," *IBM Systems Journal*, vol. 1, no. 43, pp. 5–19, 2004.

[2] M. J. Buco, R. N. Chang, L. Z. Luan, C. Ward, J. L. Wolf, and P. S. Yu, "Utility computing SLA management based upon business objectives," *IBM Systems Journal*, vol. 1, no. 43, pp. 159–178, 2004.

[3] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," in *9th IFIP/IEEE Int'l Symposium on Integrated Network Management*, 2005.

[4] J. Wilkes, J. Mogul, and J. Suermondt, "Utilification," Hewlett-Packard Laboratories, Palo Alto, CA, USA, Tech. Rep., 2004, hPL-2004-124.

[5] G. Banga, P. Druschel, and J. Mogul, "Resource containers: A new facility for resource management in server systems," in *3rd USENIX Symposium on Operating Systems Design and Implementation*, 1999.

[6] S. Graupner, R. König, V. Machiraju, J. Pruyne, A. Sahai, and A. V. Moorsel, "Impact of virtualization on management systems," Hewllet-Packard, Tech. Rep. HPL-2003-125, 2003.

[7] W. E. Walsh, G. Tesauro, J. Kephart, and R. Das, "Utility functions in autonomic computing," in *IEEE Internaltional Conf. Autonomic Computing (ICAC'04)*, 2004.

[8] H. G. Perros and K. H. Elsayed, "Call admission control schemes : A review," *IEEE Magazine on Communications*, vol. 34, no. 11, pp. 82–91, 1996.

[9] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," Hewllet-Packard Labs, Tech. Rep. HPL-1999-35R1, 1999.

[10] B. Abraham and J. Ledolter, *Statistical Methods for Forecasting*. John Wiley and Sons, 1983.

[11] D. Menascé, V. Almeida, and L. Dowdy, *Performance by Design*. Prentice Hall, 2003.

[12] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," in *IEEE Int'l Workshop on Quality of Service*, June 2004.

[13] Z. Liu, M. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *ACM Eletronic Commerce Conference*, October 2001.

[14] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, 1995.

[15] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. McGraw-Hill, 2002.

[16] L. Kleinrock, *Queueing Systems Volume I: Theory*. Wiley Interscience, 1975.

[17] S. F. Yashkov, "Processor-sharing queues: Some progress in analysis." *Queuing Systems*, vol. 2, pp. 1–17, 1987.

[18] P. Spellucci, "An SQP method for general nonlinear programs using only equality constrained subproblems," *Mathematical Programming*, vol. 3, no. 82, pp. 413–448, 1998.

[19] M. A. Rappa, "The utility business model and the future of computing services," *IBM Systems Journal*, vol. 1, no. 43, pp. 32–41, 2004.

[20] D. A. Menascé and M. N. Bennani, "On the use of performance models to design self-managing computer systems," *IEEE Distributed Systems*, vol. 6, no. 1, 2005.

[21] M. Bennani and D. Menascé, "Resource allocation for autonomic data centers using analytic performance models," in *IEEE Int'l Conference on Autonomic Computing*, 2005.

[22] Y. Diao, N. Gandhi, J. Hellerstein, S.Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server," in *IEEE/IFIP Network Operations and Management Symp.*, 2002.

[23] X. Zhu, C. Santos, J. Ward, D. Beyer, and S. Singhal, "Resource assignment for large-scale computing utilities using mathematical programming," Hewllet-Packard Labs, Tech. Rep. HPL-2003-243R1, 2003.