Self-Adaptive Capacity Management for Multi-Tier Virtualized Environments

Ítalo Cunha, Jussara Almeida, Virgílio Almeida, Marcos Santos
Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Brazil, 30123-970
{cunha, jussara, virgilio, marcos}@dcc.ufmg.br

Abstract— **This paper addresses the problem of hosting multiple applications on a provider's virtualized multi-tier infrastructure. Building from a previous model, we design a new self-adaptive capacity management framework, which combines a two-level SLA-driven pricing model, an optimization model and an analytical queuing-based performance model to maximize the provider's business objective. Our main contributions are the more accurate multi-queue performance model, which captures application specific bottlenecks and the parallelism inherent to multi-tier platforms, as well as the solution of the extended and much more complex optimization model. Our approach is evaluated via simulation with synthetic as well as realistic workloads, in various scenarios. The results show that our solution is significantly more cost-effective, in terms of the provider's achieved revenues, than the approach it is built upon, which uses a single-resource performance model. It also significantly outperforms a multi-tier static allocation strategy for heavy and unbalanced workloads. Finally, preliminary experiments assess the applicability of our framework to virtualized environments subjected to capacity variations caused by the processing of management and security-related tasks.**

I. Introduction

Modern Internet and Web-based services commonly rely on computing-based outsourcing as a financially attractive approach to host their increasingly popular services [1]. In this scenario, the service provider signs Service Level Agreement (SLA) contracts with an infrastructure provider. In order to be profitable, the service provider demands that a significant fraction of their customer requests are served with a quality that meets specific requirements. On the other hand, the goal of the infrastructure provider, or simply the *provider*, is to devise the most cost-effective strategy for managing their available resources, shared among a number of hosted applications.

The capacity management of this shared infrastructure becomes particularly challenging due to several reasons. Current Web services demand for complex and heterogeneous multi-tier service platforms composed of HTTP servers, application servers and, possibly, a database server. Moreover, application heterogeneity and typically high workload daily fluctuations [2] can not be effectively accommodated with traditional static capacity management strategies. In such scenario, resource virtualization [3], [4], [5] can build a much more cost-effective environment. A virtualization mechanism creates isolated virtual machines (VMs) on top of the physical infrastructure, each one composed of a set of virtual instances of the physical resources, and dedicated to serve a single application.

The design of cost-effective capacity management strategies for the hosting infrastructure is further challenged by new demands from the service providers. There is a growing interest in establishing service contracts where payment is proportional to the resources actually used [6]. Moreover, such contracts should provide guarantees not only on service throughput but also on the response time observed by each request [7]. The latter implies a demand for guarantees on the response time tail distribution, as opposed to the traditional requirements on *average* response time. These requirements imply the need for more complex SLA contracts, business and pricing models, and, ultimately, for more sophisticated capacity management solutions.

Integrated management of such complex networked systems demands analytic models that combine different techniques to represent important facets of the system. In [8], we have proposed a self-adaptive capacity management solution that addresses some of the aforementioned challenges. Our solution combines a pricing model, built from a two-level SLA contract model, a queuing-based performance model, and an optimization model to dynamically allocate the available capacity among the hosted applications, aiming at maximizing the provider's business objectives. As in other previous work [9], [10], [11], [12], our performance model represents the fraction of the physical resources assigned to each application as a single resource (i.e., a single queue). This is a simplified representation of the system, especially for heterogeneous applications running on multi-tier platforms. Thus, it may lead to significant inaccuracies, ultimately impacting the cost-effectiveness of the solution. Other common limitations of previously proposed resource management frameworks include lack of business-oriented goals and use of average performance guarantees [12], [13].

This paper builds on our previous work, and proposes a significantly more cost-effective capacity management framework for multi-tier virtualized systems. The framework assumes the physical infrastructure of each tier is virtualized, assigning one local VM to each hosted application. In this case, the performance model represents each VM as a separate queue, thus capturing the inherent parallelism of multi-tier platforms missed by the single-resource models. In this multi-tier model, the computation of the probability of response time

SLA violations becomes significantly more challenging, which ultimately adds to the optimization model complexity. Two alternative estimates of this probability are proposed, which combined with an extended optimization model yields two self-adaptive multi-tier approaches.

We compare our two approaches against our previous self-adaptive single-resource model as well as a multi-tier static allocation strategy. Simulation experiments with synthetic and realistic workload profiles are used to assess the relative cost-effectiveness of the analyzed strategies in various interesting scenarios. One such scenario models dynamic changes in the available capacity as a consequence of local processing by management and security-related tasks.

Our main conclusions are as follows. First, our multi-tier self-adaptive approaches scale reasonably well to practical scenarios. Second, in case of heavy and unbalanced workloads, it yields significant revenue gains to the provider, if compared with the multi-tier static allocation. Third, the performance inaccuracies introduced by the single-resource model lead to very conservative allocation decisions. As a consequence, the single-resource strategy is outperformed by the multi-tier self-adaptive and *static* approaches by orders of magnitude, even when the hosted applications are homogeneous and have balanced service demands. Finally, our multi-tier self-adaptive approaches are also much more robust than the single-resource model in face of capacity variations due to the local execution of management and security-related tasks.

This paper is organized as follows. Section II discusses related work. Our multi-tier virtualized environment and the pricing model upon which our approaches are built are described in Section III. Section IV presents our self-adaptive framework. Simulation results are presented in Section V. Conclusions and future work are offered in Section VI.

II. Related Work

A significant amount of effort has been dedicated to the design of efficient methods for autonomic resource management in modern computing systems. In particular, some previous work focused on improving system performance by applying admission control and scheduling mechanisms [13] as well as techniques for allocating shared capacity among hosted applications [12]. However, these studies focus only on system performance and lack business-oriented goals. Other related topics include the use of reward-driven request prioritization [14] and the management of grid systems [15].

A considerable amount of work applies analytic queuing models for autonomic capacity management, usually combining admission control and capacity allocation with the objective of maximizing the provider's business objective. Models using M/M/1 and M/G/1 queues with FIFO and processor sharing scheduling are considered in [10], [11], employing different approximations to the response time of M/G/1 queues. The capacity manager proposed in [9] includes operational costs (e.g., energy) into the optimization model aiming at minimizing the revenue losses due to SLA violations and management costs. In [8], we combine an SLA-driven pricing model, a queuing-based performance model, and an optimization model to dynamically allocate available capacity among hosted applications aiming at maximizing the provider's revenues. Models with a single service center (either a M/M/1 or a M/G/1 queue) are used to provide estimates of the response time tail distribution.

A number of previous work, including [16], [17], proposes queuing-based performance models specifically for multi-tier systems. More recent multi-tier models addresses issues such as the caching of responses at tiers [18] and work conservation [19]. In common, these models focus on performance estimation only. They are not coupled to optimization models, and usually provide only *average* performance estimates.

Our approach improves on previous work, combining a more accurate multi-tier performance model and an optimization model, taking into account probabilistic performance guarantees and being driven by the provider's business objective, expressed in a flexible two-level pricing model.

III. Infrastructure Model

This section describes the target platform (Section III-A) of our self-adaptive capacity management framework as well as the pricing model (Section III-B) it is built upon.

A. Virtualized Hosting Platform

We consider a scenario where a provider hosts multiple third-party Web services. Each such Web services may be composed of different request *types*, characterized by different workloads, different service demands on resources, and executed by independent software components. We refer to each such request type as an *application class*, and assume the infrastructure hosts N independent classes from all services.

We consider the provider's infrastructure is composed of multiple (K) tiers, as is the case for many Web services. Each tier is responsible for a specific task in the process of serving a request (e.g., presentation, application and database tiers). Tiers operate in parallel and requests visit tiers in sequence. That is, a request from class i enters tier j, is served, and then leaves the system with probability $p_{i,j}$ or proceeds to tier $j+1$ with probability $(1 - p_{i,j})$ $(i = 1..N, j = 1..K, p_{i,K} = 1)$.

Each tier is hosted on a separate hardware, which is shared by all classes. Such an infrastructure must provide performance isolation between classes, among other desired features. Thus, we assume each tier runs a virtualization mechanism, such as Xen [3] and Denali [4], which provides service differentiation and performance isolation for hosted application classes, simplifying load balancing and allowing the *dynamic* allocation of resources to each class. In fact, such technologies, are currently experiencing a renewed interest as a means for *server consolidation*, improving system security, reliability and availability, reducing costs and providing flexibility.

The considered hosting platform is shown in Fig. 1. On top of each tier physical infrastructure, a virtualization layer creates N isolated virtual machines (VMs), one for each class. Given K tiers, each request from a given application class is served by K VMs, dedicated to that class. This hosting model

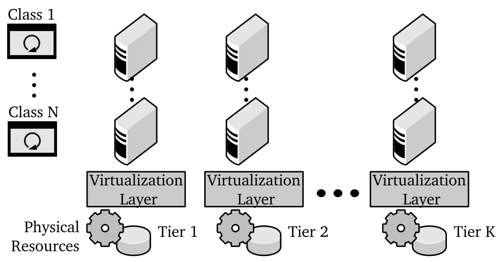


Fig. 1. Multi-Tier Virtualized Service Hosting Platform

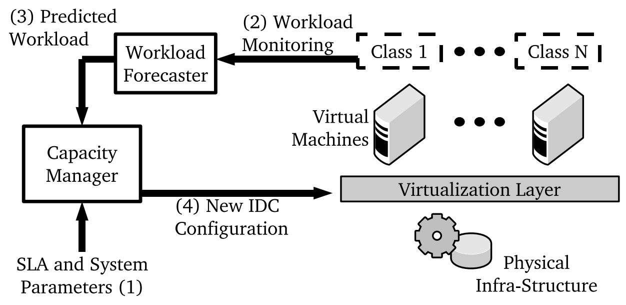


Fig. 2. Self-Adaptive Capacity Management (from one tier's perspective)

isolates classes one from another, each using K VMs as if they were dedicated servers, working at a fraction of the total (physical) capacity.

The virtualization layers allow the provider to dynamically increase or decrease the amount of physical resources dedicated to a class on each tier, independently. Hence, we define the capacity allocation problem as the determination of physical capacity fractions for each class i at each tier j.

We assume that VMs employ admission control schemes to avoid unwanted situations like, service instability due to capacity limitations, security attacks or to guarantee that the requirements of response time are met. In this work, we focus on the dynamic capacity allocation for hosted classes across all tiers, with the goal of maximizing the provider's business objective, described next. By provisioning each tier separately, we can take specific application bottlenecks into consideration.

B. Pricing Model

In [8], we propose a pricing scheme that addresses the high variability of application workloads in online services. Many services which usually receive low to moderate load, are suddenly inundated by an exceptional surge of requests. This phenomenon, known as "flash crowds" generates congestion at the service infrastructure, causing significant delays to customers. 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 following discussion, we refer to the service provider as the *customer*, which establishes a contract with the infrastructure *provider* to host its service.

In the normal operation mode, customers contract the service level which satisfies their needs for the majority of 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 so as to accommodate occasional load peaks. From the business standpoint, this approach can be advantageous both to customers who pay for extra capacity only when needed, and to providers who can offer more attractive service plans by operating with more flexibility.

In this work, the SLA performance requirements quantify the capacity of the provider's infrastructure to process transactions, given that per-request response time probabilistic guarantees are satisfied. Moreover, the proposed SLA contracts contain performance targets for each operation mode.

For the normal operation mode, the SLA defines a throughput X_i^{NSLA} for each class i, which the provider is expected to satisfy, given that the class arrival rate is high enough. In case of SLA violations, the provider agrees to refund part of the service charged to its customers. This penalty is proportional to the difference between X_i^{NSLA} and the actual *valid* throughput (see below). For the surge operation mode, the SLA defines $X_i^{SSLA} \geq X_i^{NSLA}$, the throughput up to which a customer is willing to pay a reward to the provider for serving requests in excess of X_i^{NSLA}. Rewards are also proportional to the extra *valid* throughput achieved. Penalties and rewards are calculated using SLA parameters c_i and r_i per unit of throughput below or above X_i^{NSLA}, respectively.

The *valid* throughput is composed of all requests that were served with a response time that satisfies the specified SLA. We consider a tail distribution response time requirement stating that the response time of requests from class i must not exceed a given threshold R_i^{SLA} for more than $\alpha_i \times 100\%$ of the time. In other words, $P(R_i > R_i^{SLA}) \leq \alpha_i$, where R_i is the response time of a class i request.

Note that the proposed approach can be extended to more than two SLA levels, specifying multiple performance targets so as to account for different levels of demands from applications and customers. Given this pricing model, the provider's business objective is defined as the provision of capacity to VMs that execute the applications so as to maximize the net revenues from penalties and rewards.

IV. Capacity Management for Multi-Tier Services

This section describes our self-adaptive capacity management model for multi-tier Web services. Section IV-A presents the self-adaptive framework, which is built from our previous single-resource model [8]. The most significant new contributions lie on the more accurate performance model and on the extended optimization model designed to use it, presented in Sections IV-B and IV-C, respectively.

A. Self-Adaptive Framework

Our self-adaptive framework proposes a system operation model based on feed-forward control, shown in Fig. 2. Its core entity is the *capacity manager*, which is called periodically to allocate the capacity available on each tier among the hosted

TABLE I
SLA, SYSTEM AND WORKLOAD FORECASTING PARAMETERS

SYMBOL	DESCRIPTION
X_i^{NSLA}	Valid throughput required for class i in normal mode (req/s).
X_i^{SSLA}	Maximum valid throughput for class i in surge mode (req/s).
R_i^{SLA}	Response time requirement for class i (sec).
α_i	Upper-bound on the probability of response time exceeding R_i^{SLA} for a class i request.
c_i	Penalty cost for a unit of class i throughput below X_i^{NSLA}.
r_i	Reward for a unit of class i throughput in excess of X_i^{NSLA}.
N	Number of hosted classes (and, thus, of VMs on each tier).
K	Number of tiers.
$\nu_{i,j}$	Maximum utilization planned for a VM on tier j for class i.
$d^*_{i,j}$	Average service time of class i requests at tier j physical infrastructure running on its full capacity (sec).
$p_{i,j}$	Prob. of class i requests leaving system after visiting tier j.
λ^*_i	Predicted class i arrival rate (in req/s) for the next interval.

application classes, aiming at maximizing the provider's business objective. We refer to the interval between consecutive interventions as the *controller interval*.

At the end of each *controller interval*, the capacity manager receives estimates of the workload expected for each class in the next interval as well as the SLA requirements, the average service time (i.e., service demand) of requests from each class on each tier, and the routing probabilities for each class (i.e., probabilities that requests leave the system after visiting each tier). These parameters are used to compute the fraction of the capacity available at each tier that should be given to the local VM (i.e., on the tier) assigned to each class. They are also used to estimate the fraction of the expected request rate for each class that can be accepted into the system without violating capacity limitations. The new capacity allocation is then sent to the virtualization layer, which updates the virtual resource mappings accordingly.

Note that SLA requirements and system configuration parameters may change whenever contracts change (i.e., application classes are added or removed, SLA requirements change). Note also that the controller intervals could have fixed or variable durations, depending on the characteristics of the system and stability of the workloads of hosted classes. Regardless, its minimum duration is constrained by the time the capacity manager requires to reconfigure the system. Finally, we assume future workload estimates are provided by a workload forecaster module, which implements one of the existing forecasting methods [20], and that an admission control mechanism (such as those in [21]) is used to enforce the per-interval accepted request rates. The design and evaluation of these modules is outside the present scope.

The parameters used by the capacity manager, describing the pricing model and SLA requirements, system configuration and workload characteristics, are defined in Table I. We assume all requests from a class are statistically indistinguishable, thus having the same average service time on each tier infrastructure (i.e., running at full capacity), given by $d^*_{i,j}$. Parameter $\nu_{i,j}$, an upper-bound on the utilization planned for the VM assigned to class i on tier j ($0 \leq \nu_{i,j} < 1$), is

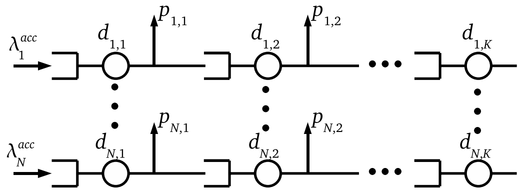


Fig. 3. Multi-Tier System Model.

introduced to minimize long-term response time degradation due to overload, keeping a certain level of stability in the VMs.

The capacity manager is built from an optimization model that links an analytical performance model with the two-level SLA-driven pricing model presented in Section III-B. The performance and optimization models are presented next.

B. Performance Model

This section presents an analytical queuing model to estimate the performance metrics used by the capacity manager, namely, per-tier resource utilization, system throughput and the probability of response time SLA violations for each class. Our model assumes that request arrivals from each class follows a Poisson process, as observed in real systems [10], [11], [22]. Class i requests accepted into the system arrive at the first tier with rate λ_i^{acc}, and leave the system after visiting tier j with probability $p_{i,j}$ ($p_{i,K} = 1$). We assume that classes have exponentially distributed service times at each tier, leaving the study of other, application-specific, patterns for future work.

Under these assumptions, the multi-tier virtualized system is modeled as N tandem queue networks, one for each class, as shown in Figure 3. The VM assigned to each class on each tier is represented as a M/M/1 queue with FCFS scheduling [23]. This queue has been often used as a reasonable model for transactional service centers [10], [11], [12], [18]. A class i request has a system response time equal to the sum of its residence times at each queue, which are assumed to be independent of each other. This assumption trades solving time and deployability over accuracy. Nevertheless, we claim that our solution captures the system primary performance aspects and trade-offs, improving on previous models, while still solving a complex optimization model efficiently for practical scenarios. Capturing interdependencies is left for future work.

Since each application is guaranteed to have access to at least the amount of resources assigned to it, we estimate the average service time of a request from class i at its assigned VM on tier j, $d_{i,j}$, by the average service time at the tier with full capacity, inflated by the fraction of its capacity currently assigned to class i, given by $f_{i,j}$. In other words, $d_{i,j} = \frac{d^*_{i,j}}{f_{i,j}}$. Moreover, given the routing probabilities $p_{i,j}$, the effective request arrival rate from class i at tier j is given by $\lambda^e_{i,j} = \lambda_i^{acc} \prod_{k=1}^{j-1} (1 - p_{i,k})$. Note that $\lambda^e_{i,1} = \lambda_i^{acc}$. Furthermore, $\lambda^e_{i,j} = \lambda_i^{acc}$ if $p_{i,j} = 0, \forall i \in N$ and $j = 1..K-1$. Parameters $d^*_{i,j}$ and $p_{i,j}$ can be estimated in a pre-production execution of each class, as discussed in [18].

We are now ready to present our performance model. Note that, given the job flow balance condition [23], which states that all accepted requests are actually processed by each VM, class i system throughput is given by the accepted request rate λ_i^{acc}. Moreover, the utilization of tier j by class i, $\rho_{i,j}$, can be estimated by the product of class i average service time at its assigned VM on tier j by the rate at which its requests arrive at the tier [23]. That is, $\rho_{i,j} = \lambda_{i,j}^{e} d_{i,j}$.

The most challenging component of our performance model is the estimate of the probability that a request from class i violates its response time SLA. Given the residence time of a class i request at tier j, $R_{i,j}$, and its system response time $R_i = \sum_{j=1}^{K} R_{i,j}$, our goal is to estimate $P(R_i \geq R_i^{SLA})$.

Note that $R_{i,j}$ is the response time of a M/M/1 queue, which is exponentially distributed with parameter $\gamma_{i,j} = \frac{1}{d_{i,j}} - \lambda_{i,j}^{e}$ [23]. Moreover, recall that the sum of K independent exponential variables with rates $\gamma_{i,j}$ ($j = 1..K$) follows a hypoexponential distribution [24]. Thus, the probability of a response time SLA violation by a class i request is equal to the complement of the cumulative distribution of the hypoexponential variable with parameters $\gamma_{i,j}$. In other words,

$$P(R_i \geq R_i^{SLA}) = \sum_{j=1}^{K} \left(\prod_{k=1, k \neq j}^{K} \frac{\gamma_{i,k}}{\gamma_{i,k} - \gamma_{i,j}} \right) e^{-\gamma_{i,j} R_i^{SLA}} \tag{1}$$

Because our queuing-based performance model captures the parallelism that can exist in a multi-tier environment, we expect Equation 1 to be more precise than applying any of the approximations presented in [8] to our target environment. Based on a single-resource model, the most cost-effective approximation is derived from the response time of a single M/M/1 queue, which is exponentially distributed with parameter $\frac{1}{\sum_{j=1}^{K} d_{i,j}} - \lambda_i^{acc}$ [8]. However, Equation 1 is also more complex, which may compromise the model solution time. Thus, we also consider an approximation derived from the Chebyshev's Inequality [23], which provides the following upper-bound on the probability of a violation for class i:

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

$E[R_i]$ and $Var[R_i]$ are the mean and variance of the system response time for class i, and are equal to the sum of the corresponding measures of each exponential component of the hypoexponential distribution. In other words, $E[R_i] = \sum_{i=1}^{K} \frac{1}{\gamma_{i,j}}$ and $Var[R_i] = \sum_{i=1}^{K} (\frac{1}{\gamma_{i,j}})^2$.

We do not use the simpler Markov's Inequality [23], which depends only on the mean response time, as this upper-bound is typically very loose, leading to more conservative and less cost-effective allocation decisions [8].

C. Optimization Model

The core component of our capacity manager is the optimization model shown in Figure 4. Its main decision variable is vector $f_{i,j}$, the fraction of tier j's capacity assigned to the local VM responsible for class i. The objective function expresses the provider's business objective, given by the sum, over all classes, of the net revenues from individual penalties and rewards. The penalty (or reward) for class i, g_i, depends on its SLA requirements and accepted request rate λ_i^{acc}, which, in turn, is constrained by the values of $f_{i,j}$. Before elaborating on the formulation of g_i, we describe the model constraints.

$$\begin{array}{llll}
max & \sum_{i=1}^{N} g_i(\lambda_i^{acc}) & & \\
s.t. & 0 \leq \lambda_i^{acc} \leq min(\lambda_i^{*}, X_i^{SSLA}) & \forall i \in N & (a) \\
 & d_{i,j} = \frac{d_{i,j}^{*}}{f_{i,j}} & \forall i \in N, j \in K & (b) \\
 & \rho_{i,j} = \lambda_{i,j}^{e} d_{i,j} \leq \nu_{i,j} & \forall i \in N, j \in K & (c) \\
 & \sum_{i=1}^{N} f_{i,j} \leq 1 & \forall j \in K & (d) \\
 & f_{i,j} \geq 0 & \forall i \in N, j \in K & (e) \\
 & \lambda_{i,j}^{e} = \lambda_i^{acc} \prod_{k=1}^{j-1} (1 - p_{i,k}) & \forall i \in N, j \in K & (f) \\
 & P(R_i \geq R_i^{SLA}) \leq \alpha_i & \forall i \in N & (g)
\end{array}$$

Fig. 4. Capacity Manager Optimization Model

Constraint (a) states that the accepted request rate for each class is limited by its predicted arrival rate and by the maximum throughput the provider can capitalize upon when the class is on surge mode. Constraint (b) defines the average service time of class i at its assigned VM on tier j. Constraint (c) states that the utilization of tier j by class i, $\rho_{i,j}$, which ultimately depends on λ_i^{acc}, is limited by the maximum planned utilization for the corresponding VM. Constraints (d) and (e) impose obvious limits on vector $f_{i,j}$. Constraint (f) defines the effective arrival rate at each tier.

Finally, constraint (g) expresses the SLA response time requirement. Two variants of the model are created by using the expressions in Equations 1 and 2. Note that this constraint expresses the trade-off between throughput and quality of service. A smaller value of α_i assures that most class i requests are served with short response times. However, fewer requests are accepted into the system, and throughput is lower. Larger values of α_i allow more requests into the system and thus higher throughput. However, accepted requests observe longer response times more frequently.

We now turn to the formulation of g_i, the provider's revenue from class i. Recall that rewards are given to the provider whenever the accepted request rate from class i exceeds its throughput SLA requirement for normal operation mode, i.e., whenever $\lambda_i^{acc} > X_i^{NSLA}$. Constraint (a) guarantees the upper-bound on such rewards based on X_i^{SSLA}. On the other hand, penalties are incurred if λ_i^{acc} is lower than X_i^{NSLA}, as long as this is due to capacity limitations and not to lower request arrival rates. In other words, a penalty is incurred whenever $\lambda_i^{acc} < min(\lambda_i^*, X_i^{NSLA})$. Thus, the net revenue obtained from class i, g_i, is given by:

$$g_i = \begin{cases} -c_i\left(min(\lambda_i^*, X_i^{NSLA}) - \lambda_i^{acc}\right) & \lambda_i^{acc} \le X_i^{NSLA} \\ r_i\left(\lambda_i^{acc} - X_i^{NSLA}\right) & \lambda_i^{acc} > X_i^{NSLA} \end{cases} \tag{3}$$

Note that g_i, and thus the objective function, increases with λ_i^{acc}. However, λ_i^{acc} is constrained by the workload and SLA contract (constraint (a)), by limitations on resource utilization (constraint (c)), and, above all, by the response time SLA requirement (constraint (g)). The last two constraints indirectly link the values of λ_i^{acc} to the decision variables $f_{i,j}$.

The optimization model shown in Figure 4 is an extension, for multi-tier environments, of the one proposed in [8]. As in [8], the main challenge, from the optimality and solution time perspectives, lies in the piecewise linear objective function and in the response time constraint (g). If the two-step piecewise linear objective, computed from Equation 3, is concave (i.e., $c_i \ge r_i$, $\forall i \in N$), it can be expressed as a set of linear constraints which can be easily solved. Here, we have focused on this scenario. Otherwise, a binary variable δ_i can be used to combine penalties and rewards into a single expression for g_i, as in [8]. Approximating the objective function by a polynomial is an alternative solution.

The probability of a response time SLA violation derived from the Chebyshev's inequality (Equation 2) as well as the approximations proposed in [8], though convex and non-linear functions in the valid range of the decision variables, are reasonably simple. Thus, the optimization models can be easily solved. This is true even for the most precise approximation proposed in [8], which is based on the exponential distribution of a M/M/1 queue response time [8].

The expression derived from the hypoexponential distribution (Equation 1), on the other hand, is much more complex, yielding a much more challenging optimization model. In particular, the distribution is undefined whenever two of its parameters $\gamma_{i,j}$ have equal values, making the problem unsolvable. Several strategies can be used to remedy this problem. First, the hypoexponential function can be approximated by a polynomial with a compromise in optimality. Second, one can solve different instances of the model, covering complementary regions of the solution space where the function is clearly defined, and then take the maximum solution from all instances as the global optimum. We implemented this strategy and successfully tested it with a small number of tiers (2) and application classes (up to 4). However, it does not scale well for larger numbers of tiers and classes. Finally, one can approximate terms of the hypoexponential distribution with equal parameter values by an Erlang distribution. This approximation is asymptotically exact, as the sum of identically distributed exponential variables has an Erlang distribution [24]. We have selected this approach due to its better scalability, discussed in Section V-A.

Our optimization model was implemented and tested in AMPL [25], a modeling language for mathematical programming. We ran a number of different solvers, and all of them converged to the same solution for all tested inputs.

V. Experimentation

In this section, we evaluate the cost-effectiveness of our self-adaptive capacity management framework for multi-tier environments, comparing it with our previous self-adaptive strategy based on a single-resource model [8] and with a static capacity allocation, in various scenarios. The main metric for comparison is the provider's net revenue obtained with each strategy. In our experiments, we consider a *two-tier* environment (i.e., $K = 2$), applicable to a service platform with a front-end server (e.g., an HTTP server) and a back-end resource (e.g., a storage area network or a database server).

We evaluate the two variants of our capacity manager that estimate the probability of response time SLA violations using the hypoexponential distribution and Chebyshev's inequality. In the single-resource model approach, this probability is estimated from the exponential distribution of response time of a single M/M/1 queue [8], with per class average service time equal to the sum of the average service times at both tiers. Finally, the static allocation assumes the best capacity allocation at each tier for the given workloads, assigning a fixed fraction of tier j's total capacity to class i that is proportional to its average utilization over class i's entire workload. It also uses the system response time distribution for the two M/M/1 queue network (Equation 1) to estimate the maximum request rate from each class that can be admitted into the system while still meeting the response time SLAs. Thus, like our new approaches, it is based on a more accurate representation of the system. These strategies are referred to, in this section, as the *hypoexponential*, *Chebyshev*, *single-resource* and *static* approaches.

We built an event-driven simulator that models the system as a tandem queue with two centers, and is fed with workload traces from N application classes. For the self-adaptive strategies, the simulator is coupled to an optimization model solver, which is called at the end of each controller interval to calculate the capacity allocation vector $f_{i,j}$ and the accepted request rate λ_i^{acc} for each class i and tier j, for the next interval. During each interval, per-request response time as well as per-class throughput and tier utilization are collected and used to compute the provider's revenue.

Our simulator employs a fair admission control mechanism, which accepts a class i request with probability $\frac{\lambda_i^{acc}}{\lambda_i^*}$. Thus, the assumption of Poisson arrivals holds for the accepted requests. This is a conservative approach compared to other mechanisms that aims at minimizing the inter-arrival time variance [23]. Moreover, we assume every accepted request visits both tiers, and that the maximum planned utilization for all VMs is 95% (i.e., $p_{i,1} = 0$ and $\nu_i = 0.95$, for $i = 1..N$). Finally, since our current focus is on the cost-effectiveness of our new approaches, we compare them in a best-case scenario to understand trade-offs: we assume there

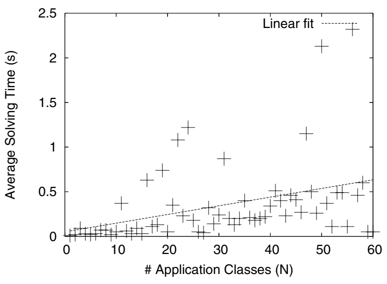


Fig. 5. Scalability of Self-Adaptive Multi-Tier Capacity Manager.

is no time limitation for adapting the system, and that an ideal workload forecasting, where future arrival rates are known *a priori*, is used. The selection and evaluation of a practical workload forecasting method, among several existing time series techniques [20] with varying degrees of accuracy, is left for future work.

In Section V-A, we briefly discuss the scalability of our capacity manager. Simulation results for synthetic workloads and for more realistic workload profiles are given in Sections V-B and V-C. Finally, Section V-D discusses a scenario where both workload and available capacity dynamically change. Our simulator was validated comparing the system response time measured with the analytical model (Equation 1) with errors under 1%. All results presented are averages of 5 runs (20 in Section V-A), with standard deviation under 2% of the means.

A. Capacity Manager Scalability

We evaluate the scalability of our multi-tier framework for configurations with up to 60 classes. We focus on the hypoexponential approach, due to its higher complexity and longer average solving times. Our experiments are conducted using the SNOPT nonlinear solver [26], on a computer with a 2 GHz AMD Sempron 2400 CPU and 512 MB of RAM.

Figure 5 shows the average solving time as the number of classes increases. The linear fitting of the data indicates that average solving time, typically under 1 second, increases with a small factor of the number of classes. Thus, our new capacity manager scales well to practical scenarios.

B. Synthetic Workloads

This section presents experimental results for synthetic workloads and two application classes. Two scenarios illustrate the main trade-offs and benefits of our solution. The controller interval is set to 1000 seconds in both scenarios.

In *scenario 1*, requests from each class arrive according to the periodic step-like non-homogeneous Poisson processes shown in Figure 6-a). Arrival rates vary from 0 to 1000 requests per second, with steps and periods of 1000 and 10000 seconds, respectively. Both workloads have identical profiles with a shift in their periods. This is an interesting scenario for the self-adaptive approaches, which are able to reassign the idle capacity from the underloaded VMs to the overloaded ones to satisfy the SLA requirements. The self-adaptive capacity manager is called at the end of each controller interval, which coincides with instants when per-class request rates change.

TABLE II

PER-TIER AVERAGE SERVICE TIMES IN SCENARIOS 1 AND 2

	Scenario 1		Scenario 2	
Class i	$d^*_{i,1}(ms)$	$d^*_{i,2}(ms)$	$d^*_{i,1}(ms)$	$d^*_{i,2}(ms)$
1	0.6	0.4	1	0.7
2	0.4	0.6	0.7	1

TABLE III

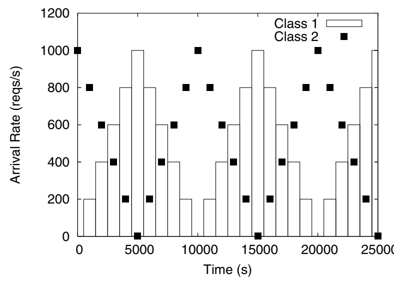
BUSINESS MODEL PARAMETER VALUES ($i = 1..N$)

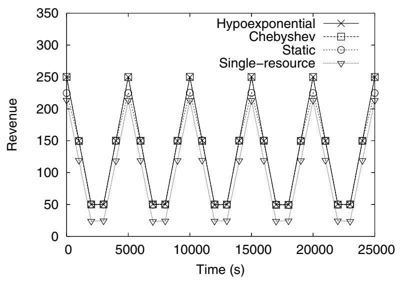
Scenario	R_i^{SLA}	X_i^{NSLA}	X_i^{SSLA}	c_i	r_i	α_i
1 and 2	0.1 s	500 req/s	1200 req/s	1.0	0.5	0.1
3	210 s	0.08 req/s	10 req/s	3500	1750	0.1
4	105 s	0.08 req/s	10 req/s	1750	875	0.1

Average service times for each class at each tier as well as business model parameter values are given in Tables II and III, respectively. Note that classes 1 and 2 have bottlenecks at tiers 1 and 2, respectively. In this case, our self-adaptive multi-tier approaches are able to dynamically assign, for each application, more resources to the tier it needs the most. Nevertheless, note that, for each class, the service time unbalance is not very significant. Moreover, both classes have equal pricing model parameter values, as our interest is on the relative cost-effectiveness of the approaches analyzed.

Figure 6-b) shows the provider's net revenue achieved by each approach throughout the simulation. The repeating pattern of the curve is produced by the periodic behavior of the workloads. The hypoexponential and Chebyshev approaches yield quantitatively similar revenues throughout the simulation. Interestingly, both approaches provide only marginal gains (11%) over the static approach when classes have complementary loads, even though this is the best scenario for self-adaptive approaches, as they can reassign idle capacity from the underloaded VMs to the overloaded ones. This is because the load imposed at each tier is very light throughout simulation. Thus, the ability of self-adapting does not play a significant role, and most requests are admitted into the system by all three strategies. Revenues are mostly dictated by the opportunities for capitalizing from an application class running on surge mode, which are the same for all approaches.

On the other hand, with the single-resource approach, the per-class average service time is equal to 1 ms, making the infrastructure slightly underprovisioned for the aggregate request rates. The revenue gains provided by our self-adaptive multi-tier approaches over the single-resource approach varies from 17% (during peaks) to 103% (during valleys). In fact, when both classes have similar load, the single-resource approach, based on a simplified system model, is significantly outperformed even by the static approach. Figure 6-c) summarizes these results, showing the revenue cumulative distributions over all intervals. The self-adaptive multi-tier approaches yield an overall revenue gain of 28% over the single-resource approach.


(a) Workload Profiles


(b) Provider's Revenue over Time

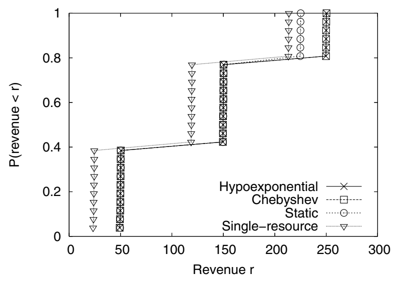
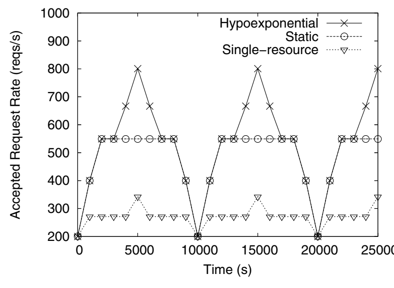
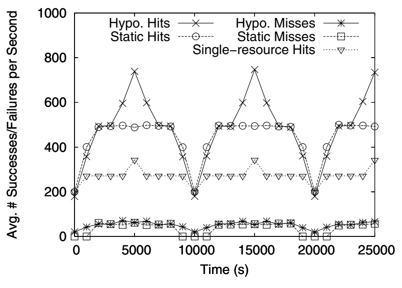

(c) Provider's Revenue Distribution

Fig. 6. Experimental Results for Synthetic Workloads: Scenario 1


(a) λ_1^{acc}


(b) Successes and Failures

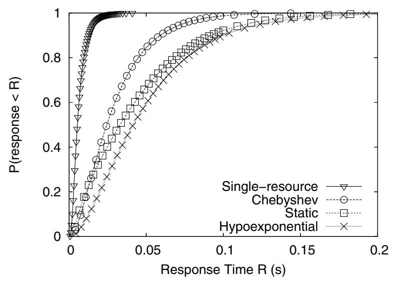
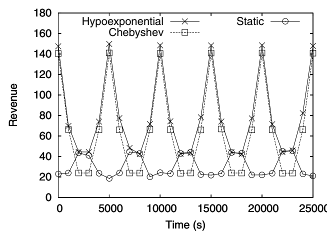

(c) Response Time Distribution

Fig. 8. Application Class 1 Performance Metrics for Synthetic Workloads: Scenario 2


(a) Revenue over Time

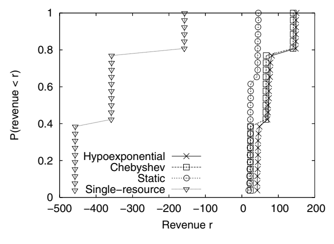

(b) Revenue Distribution

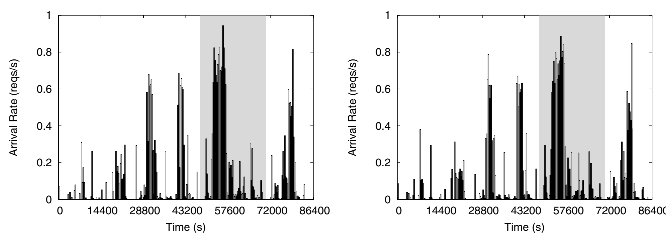
Fig. 7. Provider's Revenue for Synthetic Workloads: Scenario 2.

We now turn our evaluation to *scenario 2*, characterized by heavier workloads with more unbalanced tier average service times. Workload profiles are identical to those in Figure 6-a), but rates vary from 200 to 1200 arrivals per second. System parameter values are shown in Tables II and III.

Figures 7-a) and 7-b) show the provider's revenue obtained with each approach. The hypoexponential approach leads to higher revenues than Chebyshev, when per-class request rates are balanced. Overall revenue gains are around 20%. As one might expect, the approximation error becomes more significant for heavier loads. Unlike the previous scenario, both approaches significantly outperforms the static strategy (up to 580%) in intervals when classes have complementary request rates. Moreover, the single-resource strategy, with revenues fluctuating from -458 to -157, is outperformed by the three other strategies, by orders of magnitude.

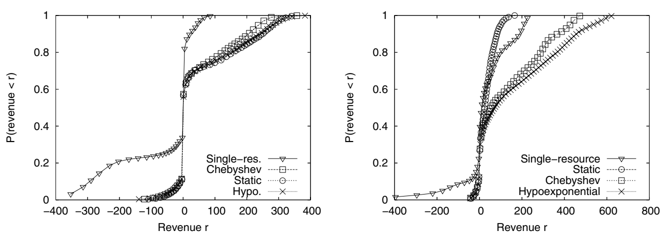
We note that two primary factors that impact the cost-effectiveness of the capacity management strategies are the ability to adapt to workload changes and the performance model accuracy, which impacts both capacity allocation and admission control decisions. The fixed capacity allocation significantly penalizes the static approach for heavy and heterogeneous workloads (i.e., scenario 2). Furthermore, in both analyzed scenarios (and in an omitted scenario with fully balanced and homogeneous applications), the single-resource approach is significantly penalized by its simpler, and thus more inaccurate, performance model, where response times are exponentially distributed with mean given by the sum of the average service times at each tier. On the other hand, the hypoexponential and static approaches use the hypoexponential distribution of response time for two M/M/1 queues. It can be easily shown that, for fixed average service times, the mean of the exponential distribution is always larger [24]. Thus, in order to meet the response time constraint, the single-resource approach is forced to make more conservative allocation and admission control decisions, incurring in lower revenues.

This conclusion is illustrated in Figures 8-a)- c), which show the accepted request rate, the rates of response time SLA hits and misses (i.e., violations) for the accepted requests, and the response time distribution for one application class. Rates for the Chebyshev approach, omitted, are between those of the hypoexponential and static approaches. Note that the more aggressive allocation and admission control decisions made by the hypoexponential and static approaches lead to a larger number of SLA misses. Nevertheless, Figure 8-c) shows that the target SLA constraint ($P(R > 0.1) < 0.1)$) is met by all



(a) Application 1 (b) Application 3

Fig. 9. Request Traces from Two Real E-Business Applications.



(a) Scenario 3 (b) Scenario 4

Fig. 10. Revenue Cumulative Distribution for Realistic Workloads.

TABLE IV
PER-TIER AVERAGE SERVICE TIMES IN SCENARIOS 3 AND 4.

Scenario / Avg Serv. Times (s)		Application Class i							
		1	2	3	4	5	6	7	8
3	$d^*_{i,1}$	0.5	2.0	1.5	3.0	-	-	-	-
	$d^*_{i,2}$	3.0	1.5	2.0	0.5	-	-	-	-
4	$d^*_{i,1}$	0.25	1.25	0.75	0.9	0.85	1.0	0.5	1.5
	$d^*_{i,2}$	1.5	0.5	1.0	0.85	0.9	0.75	1.25	0.25

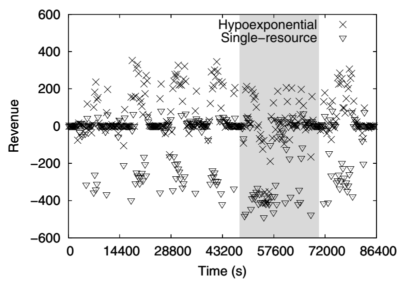


Fig. 11. Provider's Revenue Over Time for Dynamic Capacity (Scenario 5).

approaches. Note also that, even accepting a larger number of requests, the Chebyshev approach has a more skewed response time distribution than the static approach. The static allocation allied to more aggressive admission control decisions result in longer queues at each tier, thus increasing response time.

Finally, to verify the impact of mispredicting the incoming workload, we ran simulations using different controller intervals for *scenario 2*. We chose interval lengths that do not coincide with the instants when the workload changes. For interval lengths of 300 and 600 seconds, the overall loss in revenue compared to the results shown in Figure 7-a is only 5% (8%), and 11% (15%), respectively, for the hypoexponential (Chebyshev) model. Thus, our solution is reasonably robust to workload mispredictions.

C. Realistic Workload Profiles

In this section, we evaluate the capacity management approaches for more realistic workload profiles. New workloads are built from traces containing the request rate, at each 5-minute interval, to 4 different real e-business applications, over a period of 3 months (from 11/23/2004 to 2/23/2005). Confidentiality agreements prevent us from informing the source of our workloads. All four traces have similar load profiles, with peaks around the same time. Request rates vary widely, with a peak of 17 requests per second and an average of 0.078 requests per second. Figures 9-a) and 9-b) show the request rate variation for two applications, on a typical day. Realistic workloads are built by assuming request arrivals follow non-homogeneous Poisson processes, with rates changing at each 5 minute interval, and given by the traces.

Two new configuration scenarios are considered in our analysis. In *scenario 3*, we simulate 4 application classes with the workloads built from our traces. *Scenario 4* uses a larger number (i.e., 8) of application classes, whose workloads are built by duplicating each of the 4 baseline request traces, and shifting the requests in each replica by 6 hours to the future. Average service times at each tier are selected so as to make them underprovisioned for the aggregated workload. Moreover, the aggregated demand on each tier as well as the throughput SLA requirements are fixed in both scenarios. The parameter values are given in Tables III and IV. The controller interval is set to 5 minutes.

The revenue cumulative distributions for both scenarios are shown in Figures 10-a) and 10-b). The hypoexponential approach yields the highest revenues in both cases. In scenario 3, the static strategy is as good as the hypoexponential approach. The very similar profiles of all four workloads leave little room for improvements from dynamic management. Note, however, its significant degradation in scenario 4, which has more opportunities for dynamic capacity allocation among the 8 classes. In this case, the hypoexponential approach provides average per-day revenue gain of 429%.

As in scenario 2, the hypoexponential approach is more cost-effective than the Chebyshev approach, yielding average per-day revenue gains of 20% and 26% in scenarios 3 and 4, respectively. Again, the single-resource approach is outperformed by orders of magnitude.

D. Varying the Available Capacity

In our last scenario, we evaluate the applicability of the self-adaptive strategies when the total capacity available at one of the tiers (i.e., tier 1) drops suddenly. This would be the case, for instance, when the first tier (e.g., an HTTP server) is target of a malicious attack (e.g., a Denial-of-Service attack) and has to dedicate some of its capacity for recovery and other management tasks. During this period, the local capacity available for serving legitimate requests from the hosted classes decreases. The other tier (e.g., an application or a database server) can still dedicate its full capacity to them.

We ran simulations with the same workload and system parameters used in scenario 3. However, we limit the simulation to the day shown in Figure 9. Tier 1 capacity is dynamically reduced by 25% during the highlighted 6-hour period. Average service times are scaled up accordingly. Figure 11 shows the revenues for the hypoexponential and single-resource approaches. Clearly, the multi-tier hypoexponential approach is much more robust, yielding significantly higher revenues even when tier 1 capacity is reduced.

Scenario 5 was devised to allow a preliminary evaluation of the applicability of our self-adaptive multi-tier framework for virtualized environments under security attacks. More sophisticated testing scenarios as well as workload and system modeling strategies are directions we intend to pursue next.

VI. Conclusions and Future Work

In this paper, we presented a new self-adaptive capacity management framework for multi-tier virtualized systems. Built from a previous single-resource model, our extended framework shares with its origin the two-level SLA-driven pricing model. However, it implements a much more accurate multi-queue performance model, which captures application specific bottlenecks and the parallelism inherent to multi-tier architectures, as well as an extended and much more challenging optimization model.

We ran simulation experiments for five configuration scenarios, which highlighted different trade-offs of our new solutions, compared to the single-resource model and a multi-tier static allocation strategy. Our main conclusions are threefold. First, our multi-tier self-adaptive solutions scale well and are significantly more cost-effective than the static allocation for heavy and unbalanced workloads. Second, the simplified and inaccurate single-resource performance model leads to very conservative allocation decisions, which ultimately, compromise its cost-effectiveness, compared to the multi-tier self-adaptive approaches and, commonly, to the multi-tier *static* approach. This was true even for homogeneous and balanced workloads. Finally, our multi-tier approaches are robust and can be applied for capacity management of virtualized environments subjected to capacity variations due to the local execution of management and security-related tasks.

Possible directions for future work include: (a) extending our models for alternative application traffic patterns and to capture each tier specific resources individually; (b) including operational costs (e.g., energy) in our framework; (c) designing richer business models; (d) further evaluating the solution for environments under stress (i.e., attacks); and (e) prototyping the capacity manager in a real system.

Acknowledgment

This work was developed in collaboration with HP Brazil R&D.

References

[1] J. Ross and G. Westerman, “Preparing for Utility Computing: The Role of IT Architecture and Relationship Management,” *IBM Systems Journal*, vol. 43, no. 1, pp. 5–19, 2004.

[2] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, “Managing Energy and Server Resources in Hosting Centers,” in *18th ACM SOSP*, Banff, Canada, 2001.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization,” in *19th ACM SOSP*, Bolton Landing, NY, 2003.

[4] A. Whitaker, M. Shaw, and S. Gribble, “Scale and Performance in the Denali Isolation Kernel,” in *5th OSDI*, Boston, MA, 2002.

[5] G. Banga, P. Druschel, and J. Mogul, “Resource Containers: a New Facility for Resource Management in Server Systems,” in *3rd OSDI*, New Orleans, LO, 1999.

[6] J. Wilkes, J. Mogul, and J. Suermondt, “Utilification,” in *11th ACM SIGOPS European Workshop: Beyond the PC*, Leuven, Belgium, 2004.

[7] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, “Adaptive Entitlement Control to Resource Containers on Shared Servers,” in *9th IFIP/IEEE IM*, Nice, France, 2005.

[8] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, and F. Safai, “Self-Adaptive SLA-Driven Capacity Management for Internet Services,” in *10th IEEE/IFIP NOMS*, Vancouver, Canada, 2006.

[9] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, “Resource Management in the Autonomic Service-Oriented Architecture,” in *3rd IEEE ICAC*, Dublin, Ireland, 2006.

[10] Z. Liu, M. S. Squillante, and J. L. Wolf, “On Maximizing Service-Level-Agreement Profits,” in *3rd ACM Conference on Electronic Commerce*, Tampa, Florida, 2001.

[11] D. Villela, P. Pradhan, and D. Rubenstein, “Provisioning Servers in the Application Tier for e-Commerce Systems,” in *12th IEEE International Workshop on Quality of Service*, Passau, Germany, 2004.

[12] D. Menascé and M. Bennani, “Autonomic Virtualized Environments,” in *IEEE International Conference on Autonomic and Autonomous Systems*, Silicon Valley, CA, 2006.

[13] L. Cherkasova and P. Phaal, “Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites,” *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 669–685, 2002.

[14] D. Menascé, V. Almeida, R. Fonseca, and M. Mendes, “Business-Oriented Resource Management Policies for e-Commerce Servers,” *Performance Evaluation*, vol. 42, no. 2-3, pp. 223–239, 2000.

[15] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, “Statistical Service Assurances for Applications in Utility Grid Environments,” *Performance Evaluation*, vol. 58, no. 2+3, pp. 319–339, 2004.

[16] F. Baskett, M. Chandy, R. Muntz, and F. Palacios, “Open, Closed, and Mixed Networks of Queues with Different Classes of Customers,” *Journal of the ACM*, vol. 22, no. 2, pp. 248–260, 1975.

[17] J. Rolia and K. Sevcik, “The Method of Layers,” *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689–700, 1995.

[18] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, “An Analytical Model for Multi-Tier Internet Services and its Applications,” in *ACM SIGMETRICS*, Banff, Canada, 2005.

[19] Y. Diao, J. Hellerstein, S. Parekh, H. Shaikh, M. Surendra, and A. Tantawi, “Modeling Differentiated Services of Multi-Tier Web Applications,” in *14th IEEE MASCOTS*, Washington, DC, 2006.

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

[21] H. Perros and K. Elsayed, “Call Admission Control Schemes: A Review,” *IEEE Communications Magazine*, vol. 34, no. 11, pp. 82–91, 2003.

[22] 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.

[23] L. Kleinrock, *Queueing Systems*. John Wiley and Sons, 1975.

[24] K. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons, 2002.

[25] R. Fourer, D. Gay, and B. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Boyd and Fraser, 1993.

[26] P. Gill, W. Murray, and M. Saunders, “SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization,” *SIAM Journal on Optimization*, vol. 12, no. 4, pp. 979–1006, 2002.