aMOSS: Automated Multi-Objective Server Provisioning with Stress-Strain Curving

Palden Lama and Xiaobo Zhou
Department of Computer Science
University of Colorado at Colorado Springs, CO 80918, USA
{plama, xzhou}@uccs.edu

***Abstract*—A modern data center built upon virtualized server clusters for hosting Internet applications has multiple correlated and conflicting objectives. Utility-based approaches are often used for optimizing multiple objectives. However, it is difficult to define a local utility function to suitably represent one objective and to apply different weights on multiple local utility functions. Furthermore, choosing weights statically may not be effective in the face of highly dynamic workloads. In this paper, we propose an automated multi-objective server provisioning with stress-strain curving approach (aMOSS). First, we formulate a multi-objective optimization problem that is to minimize the number of physical machines used, the average response time and the total number of virtual servers allocated for multi-tier applications. Second, we propose a novel stress-strain curving method to automatically select the most efficient solution from a Pareto-optimal set that is obtained as the result of a non-dominated sorting based optimization technique. Third, we enhance the method to reduce server switching cost and improve the utilization of physical machines. Simulation results demonstrate that compared to utility-based approaches, aMOSS automatically achieves the most efficient tradeoff between performance and resource allocation efficiency. We implement aMOSS in a testbed of virtualized blade servers and demonstrate that it outperforms a representative dynamic server provisioning approach in achieving the average response time guarantee and in resource allocation efficiency for a multi-tier Internet service. aMOSS provides a unique perspective to tackle the challenging autonomic server provisioning problem.**

I. Introduction

A modern data center is built upon virtualized server clusters for hosting multiple Internet applications. Given the limited amount of resources, it needs to allocate the shared resources dynamically and efficiently among competing applications. There are multiple correlated and conflicting objectives. A data center wants to maximize its profitability by freeing up as many physical machines as possible and by improving the utilization efficiency of the shared resources [10]. It needs to minimize the total number of virtual servers allocated for all applications to reduce virtualization overhead [11]. And, it aims to minimize the average response time for each application for quality-of-service (QoS) provisioning [3].

Autonomic resource provisioning in data center servers has been an active and important research area [1], [5], [6], [16], [17], [18], [19], [21], [23]. Many of previous efforts applied the utility computing paradigm to achieve multiple objectives of a data center. In a data center hosting multiple applications, a local utility function is chosen for one objective of an application. It represents various degrees of desirability for different QoS levels [21]. Then, a global utility function is formed by combining multiple local utility functions using pre-determined weights. Such a global utility function is optimized often using analytic queuing network models and combinatorial search techniques.

Utility based server provisioning has made significant contributions to the self-optimization capability of autonomic computing systems. However, the complexity of this kind of approaches increases significantly in modern complex multi-tier services with diverse QoS needs. As pointed out by Huebscher and McCann, the major problem with utility functions is that they can be extremely hard to define, as every aspect that influences the decision by the utility function must be quantified [9]. It is also difficult to choose different weights on multiple local utility functions. The fact is, there is no specific guideline on choosing the weights for local utility functions. Furthermore, choosing functions and weights statically may not be efficient in the face of dynamic workloads.

We address the issue of autonomic server provisioning in a virtualized multi-tier cluster environment from a novel perspective. Unlike a traditional utility approach, we treat each objective as a separate entity in the optimization without applying any pre-determined weights. We formulate a multi-objective optimization problem that is to minimize the number of physical machines used, the total number of virtual servers allocated and the average end-to-end response time of Internet applications. System response time, a major performance metric of multi-tier applications, is the response time of a request that flows through a multi-tier computer system [11], [12], [20]. We apply an analytic queuing network model to correlate the average end-to-end response time with the number of servers allocated.

Importantly, we propose a stress-strain curving method to automatically select the most efficient Pareto-optimal solution considering tradeoffs between multiple objectives. In the server provisioning problem, promoting performance and improving resource allocation efficiency are essentially conflicting objectives. We first obtain a set of Pareto-optimal solutions that offer various tradeoffs between the objectives. We apply a non-dominated sorting based optimization algorithm for the purpose. Members of a Pareto-optimal set are non-dominated. That is, any solution in the set does not dominate another solution with regard to all optimization objectives. Then, we apply the stress-strain curving method to find the yield-point solution from the Pareto-optimal set, which essentially provides the most efficient tradeoff between conflicting objectives. The integration of multi-objective optimization and stress-strain curving leads to an automated multi-objective server provisioning with stress-strain curving approach (aMOSS).

Furthermore, we propose two enhancement techniques to reduce the cost due to server-switching in the face of dynamic workloads and improve the utilization of physical machines.

A significant merit of aMOSS is that it provides an automated method for finding the most efficient trade-off between multiple conflicting objectives. It does so by applying stress-strain curving method on a Pareto-optimal set of solutions. The name of this method comes from applied physics where stress-strain curve is used as a graphical representation of a material's mechanical properties [2]. When a material is elongated by applying controlled force upon a given area (called stress), the elongation (called strain) is proportional to the stress at the beginning. The graph of stress vs. strain continues to be linear up to certain point, after which it begins to curve as the material reaches its elastic limit. The yield point of a material is the stress at which a material begins to deform plastically. We find an interesting and natural analogy between the average response time vs. server allocation in a multi-tier cluster and the stress vs. strain of a material. By analogy with the physical process, the yield-point solution of the Pareto-optimal set corresponds to the knee region of the performance curve plotted against server allocation in a virtualized server cluster beyond which the benefit gained by additional server allocation is relatively negligible.

To apply the stress-strain curving method to the multi-objective server provisioning problem, we formulate stress and strain as functions of end-to-end response time, the number of physical machines and the number of virtual servers allocated in a multi-tier cluster. The yield-point solution is selected from the Pareto-optimal set by finding the yield point of the stress-strain curve.

For performance evaluation, we build a simulation model for a typical three-tier virtualized server cluster that runs multiple Internet applications. We conduct extensive simulations to evaluate aMOSS, using a synthetic workload [11], [27]. First, we compare aMOSS with three utility function based optimization approaches in the face of a highly dynamic workload. Experimental results demonstrate that aMOSS provides the solution with the most efficient trade-off between conflicting objectives. Then, we illustrate the merit of aMOSS in achieving automated trade-off between performance and server allocation. We use absolute end-to-end response time as a performance metric. We observe that applying a utility-based approach for finding a solution comparable to that of aMOSS requires many more iterations of utility optimization using various weights on local utility functions. Furthermore, there is no standard method for choosing a solution even when the data corresponding to various iterations of utility optimization is available. We also demonstrate the improvement in the utilization of physical machines and reduction in server switching cost due to the enhancement techniques of aMOSS.

We demonstrate the feasibility and performance of aMOSS with a testbed implementation in virtualized blade servers hosting RUBiS application [17], [20], [24], a multi-tier online auction Web site benchmark. We compare aMOSS with a representative dynamic server provisioning scheme proposed in [20]. Experimental results show that aMOSS significantly improves performance in achieving the end-to-end response time guarantee and in resource allocation efficiency due to its automated multi-objective server provisioning optimization.

In the following, Section II reviews related work in autonomic resource provisioning for performance assurance. Section III presents the problem formulation and analytic model. Section IV describes the aMOSS approach. Section V gives two enhancement techniques. Section VI presents experimental results and performance evaluation. Section VII presents the case study based on a testbed implementation. Concluding remarks are given in Section VIII.

II. Related Work

Significant research has been conducted on utility-based autonomic resource management in recent years [1], [16], [19], [21], [23]. Welsh and Culler proposed the use of utility functions expressed in high-level service-level attributes to dynamically allocate resources in realistic autonomic computing system [21]. A table-driven approach was used to store response time values obtained from experiments for different values of the workload intensity and different number of servers. The work demonstrated the effectiveness of using the utility function scheme for dealing with Web-based transactional workloads on a Linux cluster. Bennani and Menascé replaced the table-driven approach with predictive multi-class queueing network models to provide a utility based resource allocation mechanism [1]. It is scalable with respect to the number of transaction classes, applications and resources in a data center. Lumezanu et al. examined the problem of optimal resource allocation for event-driven distributed infrastructures and proposed a scalable distributed algorithm to maximize the total system utility [15]. The work in [5] presented a fully decentralized resource selection algorithm by which resources autonomously select themselves in large-scale utility computing infrastructures. In this paper, we propose a combination of multi-objective optimization with a stress-strain curving method for automated and efficient server provisioning. aMOSS approach automatically selects the *yield-point* solution from the Pareto-optimal set and provides the most efficient tradeoff between performance and resource allocation efficiency. It provides a novel perspective to tackle the challenging autonomic server provisioning problem.

Resource management for performance assurance in multi-tier Internet applications is a very active research topic [12], [14], [24]. A few studies focused on the modeling and analysis of multi-tier servers with queueing and control foundations. Diao et al. [8] described a performance model for differentiated services of multi-tier applications. Urgaonkar et al. proposed an important dynamic server provisioning scheme [20].

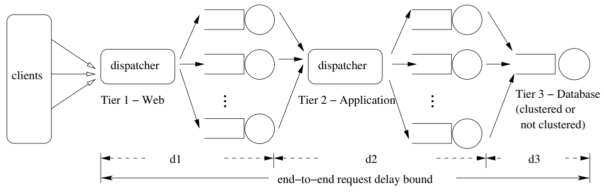


Fig. 1. A multi-tier server cluster architecture.

An end-to-end response time guarantee target is decomposed into per-tier targets and per-tier server provisioning is conducted at once for all tiers so as to guarantee the end-to-end target. The work demonstrated that adding servers to one tier does not necessarily increase the effective system performance due to cross-tier performance dependencies. Wang et al. proposed an approach to identify performance bottlenecks caused by cross-tier delay in a multi-tier application [22]. Padala et al. [17] proposed a combination of an online model estimator and a multi-input multi-output controller to achieve the average response time based service level objective in a shared server infrastructure. Leite et. al [14] applied a stochastic approximation technique to estimate the tardiness quantile of response time distribution, and coupled it with a PID feedback controller to obtain the CPU frequency of single-tier servers for performance assurance.

Virtualization technologies facilitate dynamic server allocation in data centers. Menascé and Bennani considered dynamic priority scheduling and allocation of CPU shares to virtual servers [16]. Wang et al. proposed a virtual-appliance-based autonomic resource provisioning framework for large virtualized data centers [23]. Weng et al. designed a management framework for a virtualized cluster system, and presented an automatic performance tuning strategy to balance the workload [25]. Watson et al. modeled the probability distributions of performance metrics, in terms of percentiles, based on variables that can be readily measured and controlled in a virtualized environment [24]. Our work incorporates switching delay caused by addition and removal of virtual servers in the analytic model for highly responsive server allocation.

Power management in server clusters is an important issue [6], [13], [14]. For instance, Leite et al. [14] applied stochastic optimization to minimize power consumption while maintaining tardiness. We minimize the total number of physical machines used for hosting multi-tier applications with a motivation for power saving.

Statistical machine learning techniques are used for capacity planning and resource provisioning in complex Internet systems. For instance, Bu et al. [3] proposed a reinforcement learning approach for autonomic configuration and reconfiguration of multi-tier web systems. It will be interesting to explore the integration of analytic approaches and statistical learning techniques for autonomic resource management.

III. Modeling and Analysis

Popular Internet applications employ a multi-tier architecture, with each tier provisioning a certain functionality to its preceding tier and making use of the functionality provided by its successor to carry out its part of the overall request processing [4], [8], [11], [20], [26]. For load sharing, a tier is often replicated and clustered. A typical e-commerce application usually consists of three tiers; a front-end Web tier that is responsible for HTTP request processing, a middle application tier that implements core application functionality say based on Java Enterprise platform, and a backend database that stores product catalogs and user orders. In this context, an incoming user request undergoes HTTP processing and application server processing, and triggers queries or transactions at the database. Figure 1 shows a three-tier server cluster serving one application with an end-to-end response time guarantee.

A. Problem Definition

A modern data center built upon virtualized server clusters for hosting multi-tier applications has multiple objectives. Each application competes for shared resources for QoS provisioning to its users. From the perspective of the data center, virtual servers need to be allocated efficiently. Under-utilized resources in physical machines become a liability issue to the data center because of inefficient power consumption, space utilization, and excessive cost of ownership. We thus consider three important objectives:

1) Minimize the total number of physical machines used for all applications.
2) Minimize the average system end-to-end response time, a key QoS metric, for all applications.
3) Minimize the total number of virtual servers allocated to all applications to free up more physical machines and also to reduce virtualization overhead.

There are constraints due to limited resources and QoS needs. We consider four important constraints as follows:

1) The average end-to-end response time of each application must be below a given bound according to the service level agreement [11], [20].
2) The total number of virtual servers running for all applications on one physical machine must not exceed a specified limit due to the concurrency limit [8].
3) The utilization of a virtual server cannot exceed its resource capacity limit.
4) The number of physical machines available is limited.

We consider a virtualized data center that has N physical machines virtualized and shared by M applications with K tiers. Let a_{ijk} be the number of virtual servers allocated to tier k of application j and placed in physical machine i. Consider requests of application j arrive at a K-tier server cluster in a rate λ_j. A request in different tiers usually demands different processing resources [8], [11], [20]. Let r_{jk} be the resource demand of a request of application j at tier k normalized with the virtual server capacity. With a load balancer, the workload at a tier is shared by the allocated virtual servers. Thus, the utilization at a tier is given by $\rho_{jk} = \lambda_j r_{jk} / \sum_{i=1}^{N} a_{ijk}$.

Let d_{jk} be the average response time of requests served at tier k of application j. The average end-to-end response time,

U_j, experienced by requests flowing through multiple tiers of an application j is the sum of average response time at each tier [11], [12], [20]. Let W be the virtual machine allocation limit of a physical server.

We formulate dynamic server provisioning as a multi-objective optimization problem as follows:

$$\text{Minimize} \quad m \tag{1}$$

$$\text{Minimize} \sum_{j=1}^{M}\sum_{k=1}^{K} d_{jk} \tag{2}$$

$$\text{Minimize} \quad \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{K} a_{ijk} \tag{3}$$

$$\text{Subject to Constraints:} \tag{4}$$

$$\forall j \in [1, M], \quad \sum_{k=1}^{K} d_{jk} \leq U_j \tag{5}$$

$$\forall i \in [1, N], \quad 0 \leq \sum_{j=1}^{M}\sum_{k=1}^{K} a_{ijk} \leq W \tag{6}$$

$$\forall j \in [1, M], \forall k \in [1, K], \quad 0 \leq \rho_{jk} < 1 \tag{7}$$

$$m \leq N. \tag{8}$$

Eqs. (1), (2) and (3) give the optimization objectives. Eqs. (5), (6), (7) and (8) define the four constraints. The number of physical machines used is given by $m = \sum_{i=1}^{N} f(a_{ijk})$ where

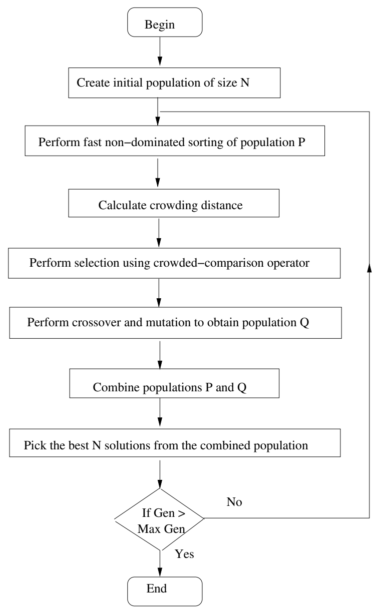
$$f(a_{ijk}) = \begin{cases} 1 & if \ (\sum_{j=1}^{M}\sum_{k=1}^{K} a_{ijk}) \geq 1 \\ 0 & if \ (\sum_{j=1}^{M}\sum_{k=1}^{K} a_{ijk}) < 1 \end{cases}$$

B. An Analytic Model

We apply a queuing theoretical model for performance analysis of multi-tier applications. One application is represented by a network of queues and each queue represents a virtual server at a particular tier. The queues from a tier feed into the next tier, as shown in Figure 1. We model a virtual server at a tier as a $G/G/1$ system to capture arbitrary arrival distributions and service time distributions [20]. The requests traverse multiple tiers and are serviced in a FCFS order [11], [20]. Let X_{jk} and T_{jk} be the service time distribution and arrival distribution of requests of application j at tier k, respectively. According to the queuing foundations, we have

$$d_{jk} = E[W_{jk}] + E[X_{jk}] = \frac{\lambda_j(E[T_{jk}^2] + E[X_{jk}^2])}{2\sum_{i=1}^{N} a_{ijk}(1-\rho_{jk})} + E[X_{jk}],$$

where $E[W_{jk}]$ is the expected queueing delay on an approximate basis, $E[X_{jk}]$ and $E[X_{jk}^2]$ are the first moment and second moment of the service time distribution X_{jk}, $E[T_{jk}]$ and $E[T_{jk}^2]$ are the first moment and second moment of the inter-arrival time distribution T_{jk}, respectively.



Begin
Create initial population of size N
Perform fast non−dominated sorting of population P
Calculate crowding distance
Perform selection using crowded−comparison operator
Perform crossover and mutation to obtain population Q
Combine populations P and Q
Pick the best N solutions from the combined population
If Gen > Max Gen
No
Yes
End

Fig. 2. The flow chart and major steps of NSGA-II algorithm.

IV. The aMOSS Approach

A. Obtaining a Pareto-Optimal Set

Classical optimization methods suggest converting a multi-objective optimization problem to a single-objective problem which aims to optimize the weighted sum of multiple objectives. Such methods assume that the weights to be assigned are well known in advance. However, in practice, it is difficult to choose proper weights that will result in an efficient solution to the optimization problem. More important and challenging is that choosing weights statically may not be efficient in the face of dynamic workloads. We want to achieve an automated server provisioning scheme that provides a solution with the most efficient trade-off between multiple objectives. To this end, we first solve the multi-objective server provisioning problem by applying a non-dominated sorting based optimization algorithm. This results in a Pareto-optimal set of solutions, offering various tradeoffs between performance and server allocation.

We apply a computationally fast multi-objective genetic algorithm (NSGA-II) [7] to obtain multiple Pareto-optimal solutions in one single run. In order to apply the genetic algorithm, we represent a solution to the optimization problem by a "chromosome". It is a string of numbers, coding information about the decision variables. The decision variables in our multi-objective optimization problem are the elements of a 3-dimensional matrix of size $N \times M \times K$. An element is denoted as a_{ijk}. For encoding the decision variables in a chromosome, we convert the 3-dimensional matrix to a chromosome vector of length $N \times M \times K$. It is denoted as $A = (a_1, a_2, ..., a_{N\times M\times K})$. As a result, the element a_{ijk} of the original decision matrix is equal to the $(i \times M \times K + j \times K + k)^{th}$ element of vector A. It is known as a gene in the chromosome. Figure 2 shows the major steps of the NSGA-II algorithm.

B. The Stress-Strain Curving Method

The Pareto-optimal set presents an opportunity to choose a solution by considering various trade-offs between conflicting objectives, i.e., the average end-to-end response time, the number of virtual servers allocated and the number of physical machines used. For autonomic server provisioning, we need a method for automatically selecting the solution from a broad range of the Pareto-optimal set. Note that each solution in the Pareto-optimal set is non-dominated and it provides the end-to-end response time guarantee. We aim to have the average end-to-end response time as low as possible. Meanwhile, we need to make sure that per-tier resource utilization is efficient.

We propose a stress-strain curving method for the automatic solution selection. Note that the average delay of a web application decreases significantly on addition of resources, when existing resources are highly utilized. At some point, as the resource utilization decreases further with addition of more resources, there is little impact on the average delay. We formulate the stress as a function of the average response time normalized by the number of physical machines allocated. The normalization ensures that the selected solution will favor using less number of physical machines, although it may result in a non-smooth stress-strain curve. We formulate the strain as a function of the number of virtual servers allocated. Each solution in the Pareto-optimal set is mapped to the stress-strain curve.

Let P be the population size of the Pareto-optimal set. We calculate the stress and strain corresponding to each solution x in the set. Let D_x be the sum of average end-to-end response time of all applications, PS_x be the total number of physical machines used, and VS_x be the total number of virtual servers allocated according to solution x. Note that the solutions in the Pareto-optimal set are in decreasing order of the average end-to-end response time and increasing order of the number of virtual servers. Hence, D_{x-1} is greater than D_x, and, VS_{x-1} is smaller than $VS_x$for all solutions x in the Pareto-optimal set. We formulate the stress and the strain as follows:

$$stress_x = (D_1 - D_x)/PS_x, \forall x \in [1, P]. \tag{9}$$

$$strain_x = (VS_x - VS_1)/VS_1, \forall x \in [1, P]. \tag{10}$$

The stress and the strain depend, respectively, on the change in the average response time and the number of virtual servers allocated compared to the first solution in the Pareto-optimal set. Figure 3 depicts a stress-strain curve obtained from a Pareto-optimal set of solutions. The stress graph is linear up to certain value of strain, after which it begins to curve. This is because a high value of strain corresponds to a large number of virtual server allocation, resulting in low per-tier resource utilization. Thus, further increase in the virtual server allocation (strain) will have little impact on the average response time (stress) reduction. The fluctuation in the stress-strain curve after a strain of 0.84 is due to variations in the number of physical machines used.

We consider the yield point of the stress-strain curve as the preferred solution. The rationale is that it corresponds

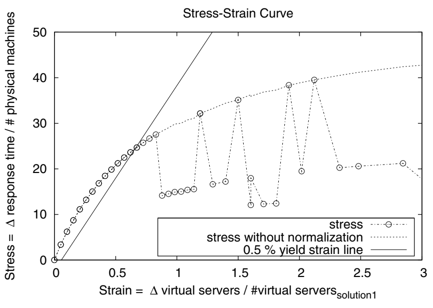


Fig. 3. A stress-strain curve.

to the average response time of an application due to one particular server allocation, after which further increase in the server allocation has little impact on the average response time decrease. To find the yield point, we apply the widely used *offset yield point method* in applied physics. The offset value of strain commonly used is 0.5% of the maximum strain calculated from the Pareto-optimal set. In the method, a line originating from 0.5% offset value on the strain axis is drawn parallel to the initial linear part of the curve. It is called *yield strain line*. The intersection of this line and the stress-strain curve gives the yield point. It is the point corresponding to the reduction in the average response time at which the required increase in server allocation exceeds the projected value by 0.5%. The projected value is the increase in allocation that would be required if the stress-strain curve was consistently linear. Smaller the offset value, the closer will be the selected solution to the linear region of the stress-strain curve.

Depending on the Pareto-optimal set under consideration, the 0.5% yield strain line may intersect the fluctuating region of the stress-strain curve. In such cases, we choose the yield point as the intersection point that is farthest from the origin of the stress-strain curve. Thus, the yield point chosen will be closer to the upward spike in the curve. This results in selecting an efficient solution that uses a smaller number of physical machines compared to other solutions neighboring the yield point. Figure 3 also shows a stress-strain curve obtained without normalizing stress by the number of physical machines. Such a curve would have yielded a solution with a larger number of physical machines for achieving similar average response time.

C. Algorithm Complexity Analysis

The time complexity of aMOSS approach is dominated by the chosen multi-objective optimization algorithm because the stress-strain curving technique has a linear complexity of $O(n)$ in the worst case, where n is the population size of the Pareto-optimal set. In this work, we apply NSGA-II algorithm for multi-objective optimization that has a complexity of $O(gmn^2)$, where g is the number of generations used for optimization iteration and m is the number of objectives. There are three objectives in this server provisioning problem. We have observed that 1000 generations are sufficient to obtain a Pareto-optimal set with diverse solutions. Thus, the time complexity of the aMOSS is $O(n^2)$. The automation overhead of aMOSS is evaluated in section VI.

V. Enhancements on the aMOSS Optimization Algorithm

We enhance the aMOSS multi-objective optimization algorithm used for obtaining a Pareto-optimal set of solutions, utilizing the system knowledge about practical server switching cost and behavior of a virtualized multi-tier system. Note that the enhancement techniques are applicable to any heuristic search technique such as simulated annealing, genetic algorithm, etc.

A. Improving usage of physical machines

One main advantage of server virtualization for resource allocation is the improvement in the utilization efficiency of physical resources [23], [25]. We emphasize the advantage by applying a threshold-based enhancement on the optimization algorithm. The enhancement feeds incremental search space to the genetic algorithm for finding the Pareto-optimal set. It initiates the genetic algorithm for multi-objective optimization using a small fraction of the total number of physical machines available. The physical machines are chosen randomly for this purpose, assuming that available resources are homogenous due to virtualization. The algorithm searches for the Pareto-optimal set within this confined search space. After a certain number of evaluations, it increases the number of physical machines in the search space by one if the percentage of evaluations that violated the resource constraint defined in Eq. (6) exceeds a certain threshold.

Based on simulation, we consider a cluster of 20 physical machines shared by an application with an end-to-end response time target of 300 ms. Let the total number of virtual servers allocable to one application on a physical machine be up to 60. For a particular workload, we compare the Pareto-optimal set obtained by the optimization with the enhancement and without the enhancement. Figure 4 shows that compared to the optimization without the enhancement, the Pareto-optimal set obtained by the optimization with the enhancement uses less than half of the number of physical machines for achieving the same range of average end-to-end response time [247.89 ms $\sim$ 299.04 ms].

Figure 5 illustrates that the solutions obtained with the enhancement have higher average utilization of physical machines. Experimental results indicate that the developed enhancement improves utilization of physical machines, which in turn frees up more of the available physical machine pool.

B. Reducing server switching cost

Server switching by addition and removal of a virtual server at a tier introduces non-negligible latency to a multi-tier service. It affects the perceived average response time of users. A newly added server spends time adapting to the existing system. For example, an addition of database replica goes through a data migration and system stabilization phase [4], during which the delay will be higher than expected. A removal of a server does not happen instantaneously, since it has to process residual requests of an active session. We enhance the optimization algorithm by reducing the undesired effects of server-switching cost. The enhancement incorporates the time required for the reconfiguration of server allocation scheme into calculation of the average response time, while evaluating a group of candidate solutions. A candidate solution provides a potential server configuration scheme. Consider that the addition of a virtual server at tier k to an application needs time T_k^a and the removal of virtual server needs time T_k^r. Given N physical machines and M applications, the server switching cost in terms of delay is calculated by:

$$T_c = \max_{i\epsilon[1,N]} (\sum_{j=1}^{M} \sum_{k=1}^{K} ((b_{ijk} - a_{ijk})^* \cdot T_k^a + (a_{ijk} - b_{ijk})^* \cdot T_k^r)),$$

where a_{ijk} is the current virtual server configuration and b_{ijk} is a candidate solution. $(b_{ijk} - a_{ijk})^*$ represents $max(0, b_{ijk} - a_{ijk})$. For application j, let D_j^a and D_j^b be the average end-to-end response time of the current virtual server configuration and the candidate solution respectively. Let T_s be the "control interval" at which reconfiguration decision is made periodically. Considering the time required for reconfiguration, the enhancement calculates the expected average end-to-end response time of the application as follows:

$$D_j = \frac{T_c \cdot D_j^a + (T_s - T_c) \cdot D_j^b}{T_s}, \forall j \in [1, M].$$

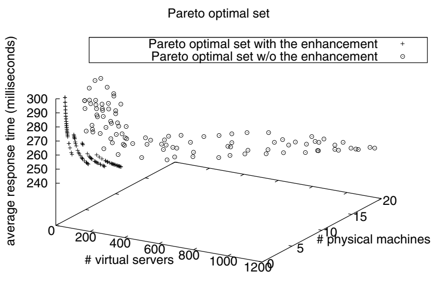


Fig. 4. Impact of the enhancement on the Pareto-optimal set.

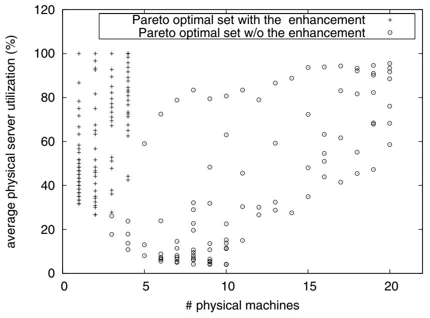


Fig. 5. Impact of the enhancement on utilization of physical machines.

TABLE I
WORKLOAD CHARACTERISTICS.

	WebTier	AppTier	DBTier
$E[X_{jk}]$	64.679 ms	94.78 ms	84.75 ms
$E[X_{jk}^2]$	4191.695	8991.681	7191.683

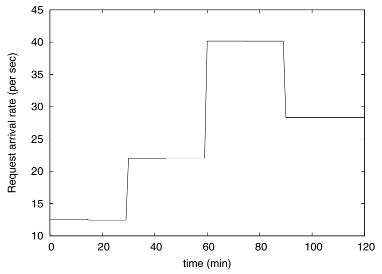


Fig. 6. A dynamic workload variation with time.

This enhancement essentially penalizes too frequent server switching. It plays an important role in avoiding any potential system thrashing that could result from dynamic workload fluctuations. Experimental results in Section VI show that the enhancement reduces unnecessary virtual server switching while minimizing the average end-to-end response time in the face of dynamic workloads.

VI. PERFORMANCE EVALUATION

For performance evaluation, we first build a simulation model. In the simulations, each physical machine has a limit of 60 virtual server allocation per application. We generate a synthetic workload using bounded-Pareto distribution of request inter-arrival time and service time [11], [27]. Table I gives the workload characteristics.

The workload is measured periodically on "control interval" of 5 minutes and aMOSS is executed when a significant change in workload is observed. aMOSS avoids potential system thrashing in case of fluctuating workloads, since it is invoked only at certain control intervals and also due to its awareness of server switching cost. Each result reported is an average of 100 runs. In the experiments, we choose a population size of 100 and limit the number of generations to 1000. These parameters are chosen to obtain a good diversity of solutions in the Pareto-optimal set.

A. Efficiency Comparison with Utility based Approaches

We compare aMOSS with utility-function based approaches in obtaining the most efficient tradeoff among multiple correlated yet conflicting objectives. We use a highly dynamic workload with varying step-change intensity similar to workload scenarios used in [20], as shown in Figure 6. First, we consider a single application with an average response time target of 300 ms. We consider three utility-based approaches, in which the global utility functions are defined as:

$$U_1 = 0.2 \cdot U_{VS} + 0.2 \cdot U_{PS} + 0.6 \cdot U_D.$$

$$U_2 = 0.34 \cdot U_{VS} + 0.33 \cdot U_{PS} + 0.33 \cdot U_D.$$

$$U_3 = 0.8 \cdot U_{VS} + 0.1 \cdot U_{PS} + 0.1 \cdot U_D.$$

where U_{VS}, U_{PS}, U_D are local utility functions corresponding to the number of virtual servers allocated, the number of physical machines used and the average response time respectively. We use a sigmoid function for U_D as the work in [1], [21], [23] and linear functions for U_{PS} and U_{VS}.

Figure 7 (a) shows that the average response time due to aMOSS is just slightly higher than the utility based approaches due to global utility functions U_1 and U_2. However, it is compensated by a significantly large improvement in terms of the virtual server allocation and the number of physical machines used, as shown in Figures 7 (b) and 7 (c). aMOSS reduces the number of virtual servers allocated by 63% at best compared to utility function U_1 and by 38% compared to utility function U_2. It reduces the number of physical machines used by 80% and 67% compared to the two utility-function based approaches respectively. Compared to utility function U_3, the number of virtual servers allocated by aMOSS is slightly higher because function U_3 over-emphasizes the utility of the virtual server allocation. However, utility function U_3 resulted in significantly higher average end-to-end response time than aMOSS. Furthermore, as shown in Figure 7 (a), it actually fails to achieve the target end-to-end response time when the workload increases at time 60th minute. aMOSS is consistently able to satisfy the end-to-end response time target throughout the experiment.

The main reason behind the efficiency of aMOSS is that it performs multi-objective optimization by considering the trade-off between performance and server allocation in response to a highly dynamic workload. Whereas, a utility based approach applies static weights on the local utility functions for each objective and aims to optimize a global utility function. Results illustrate that applying static weights is not efficient in the face of a highly dynamic workload. The stress-strain curving method of aMOSS automatically chooses the yield-point solution of the Pareto-optimal set, which essentially guarantees the most efficient tradeoff among multiple conflicting objectives.

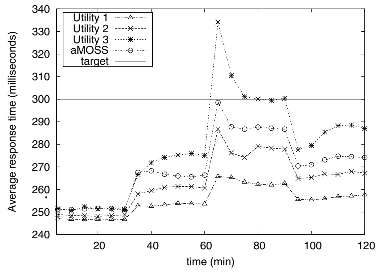
We obtained similar results in case of multiple applications hosted in the virtualized server clusters.

B. Automation Performance and Overhead

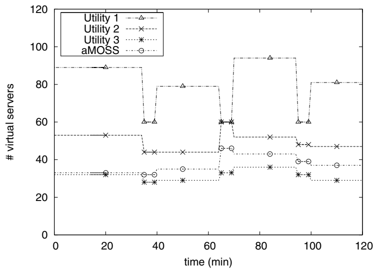
Next, we highlight aMOSS's agility to automatically select an efficient server allocation scheme. We compare its overhead with that of a utility optimization approach.

Efficient server allocation occurs in the knee region of the delay vs. server allocation curve. The benefit gained by adding additional servers is negligible beyond the knee region. A utility function based server allocation approach often requires many iterations of utility optimization with various weights on delay and server allocation in order to find the knee operating region. Consider a global utility function defined by Eq. (11). Here U_{VS}, U_{PS}, U_D are local utility functions weighted by W_{VS}, W_{PS} and W_D corresponding to the number of virtual servers allocated, the number of physical machines used and the average response time respectively.

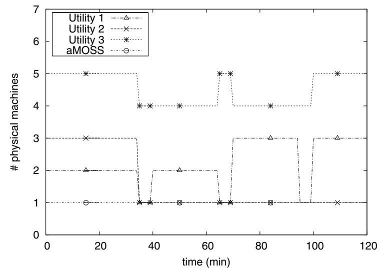
$$U_g = W_{VS} \cdot U_{VS} + W_{PS} \cdot U_{PS} + W_D \cdot U_D. \tag{11}$$



(a) Average response time.

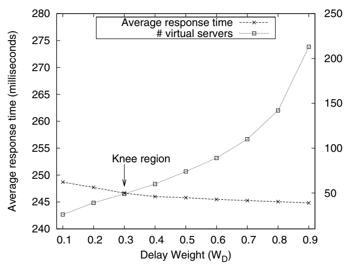


(b) Virtual server allocation.

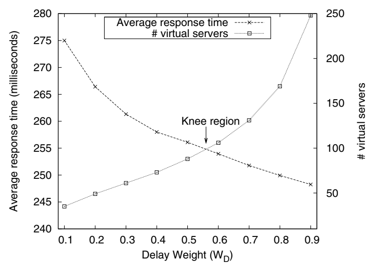


(c) Physical machines used.

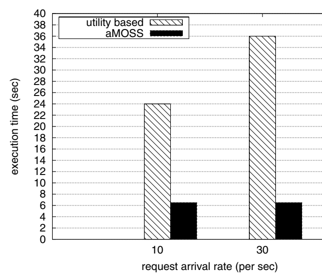
Fig. 7. Comparison of aMOSS with utility based approaches (single application).



(a) at 10 requests/sec



(b) at 30 requests/sec).



(c) Overhead comparison with aMOSS.

Fig. 8. Automated average response time and virtual server allocation tradeoffs at various workloads.

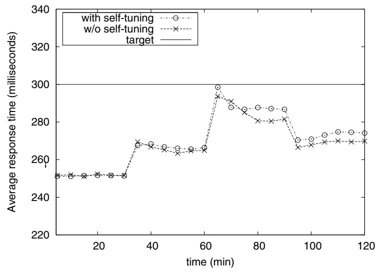
Figure 8 (a) shows the average response time achieved and the number of virtual servers allocated when the utility function in Eq. (11) is optimized iteratively for different values of delay weight W_D that varies from 0.1 to 0.9. The values of W_{VS} and W_{PS} are chosen to be equal and the sum of three weights is equal to one. The workload arrival rate is average 10 requests per second. We observe that a low value of W_D results in a small number of server allocation and a correspondingly high value of the average response time. As W_D increases, the number of allocated virtual servers increases and the average response time decreases. Thus, only after repeatedly performing utility optimization using various weights, a utility function based approach can measure the trade-offs between performance and server allocation. We have performed the experiment with delay weight increasing in steps of 0.1. For more accurate estimation of trade-off between performance and server allocation, the step size can be reduced further. However, it will result in even more number of optimization iterations. Furthermore, there is no standard method for selecting an efficient solution even when the data is available. In this experiment, we choose the intersection of the average response time curve and the server allocation curve as the knee region, which provides an efficient solution. The knee region corresponds to a delay weight of 0.3 for the given workload. In this case, it requires four iterations of utility optimization to find the knee region.

aMOSS performs a multi-objective optimization only once for a measured workload, which results in a pareto-optimal set of solutions. It applies stress-strain curving method to automatically determine an efficient trade-off between performance and server allocation. aMOSS took an average of 6.5 seconds, running on a linux machine with 2 GHz Intel Core 2 Duo processor and 2 GB RAM, to find a yield-point solution for the multi-objective optimization problem. The utility based approach took about 23 seconds to find a similar solution. Figure 8 (c) shows that aMOSS reduces the computational overhead in terms of the number of optimization iterations and algorithm execution time by about 75% compared to utility based provisioning approach.

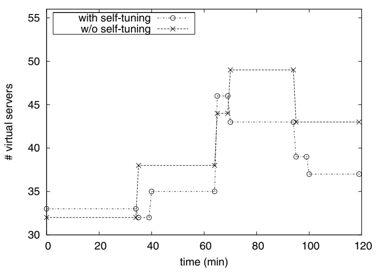
Next, we change the workload intensity to average 30 requests per second. Figure 8 (b) shows the experimental results. We note two important observations. First, the knee region of delay and server allocation graph is found at different values of delay weight. Hence, a utility based provisioning approach with static weights may not provide an efficient tradeoff between average end-to-end response time and server allocation in the face of dynamic workloads. On the other hand, aMOSS is able to find the yield point solution in spite of workload variations. Second, the computational overhead of utility based approach for finding an efficient trade-off between delay and server allocation may vary for different workloads. In this case, it requires six iterations of utility optimization to find the knee region. Figure 8 (c) show the improvement in computational overhead by aMOSS is 83%.

C. Impact of Reducing Server Switching Cost Enhancement

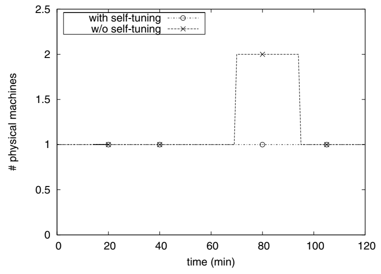
To evaluate the impact of the aMOSS enhancement technique for reducing the server switching cost, we assume the times taken by addition and removal of a virtual server of an application are five and two seconds respectively. We compare



(a) Average response time.

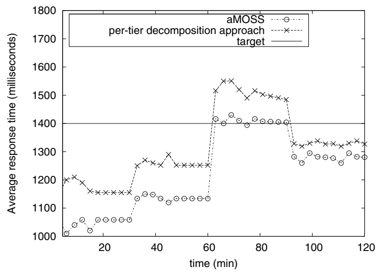


(b) Virtual server allocation.

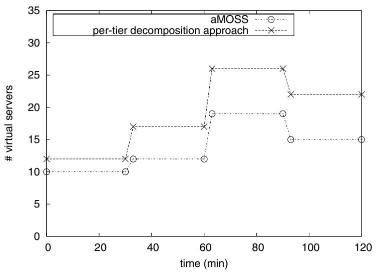


(c) Physical machines used.

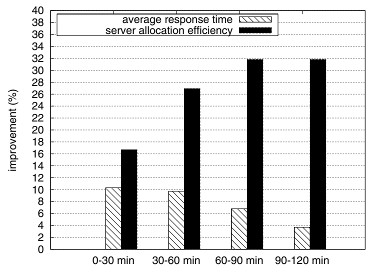
Fig. 9. Impact of the aMOSS server switching cost reducing enhancement technique.



(a) Average response time.



(b) Virtual server allocation.



(c) Performance improvement.

Fig. 10. Comparison of aMOSS with a per-tier decomposition approach based on a testbed implementation.

the performance of aMOSS with and without the enhancement for a single application that has the average response time target of 300 ms. Figure 9 (a) shows that the average response time due to the enhancement overall is slightly higher than the average response time due to aMOSS without the enhancement. This is because the enhancement tries to reduce the number of server addition or removal between consecutive control intervals. Figure 9 (b) shows the significant benefit due to the enhancement. It shows that the number of virtual servers allocated is much less with the enhancement of aMOSS. Figure 9 (c) shows that the enhancement also makes aMOSS more conservative in changing the usage of physical machines, freeing up more physical machines. This is beneficial to a data center due to cost reduction and efficient space utilization. It also avoids unnecessary physical machine usage in the face of temporary spike in the dynamic workload during the interval of minutes 60-85.

VII. Testbed Implementation with a Case Study

We implement aMOSS in a testbed that consists of HP ProLiant BL460C G6 blade server modules and a HP EVA storage area network. A blade server is equipped with Intel Xeon E5530 2.4 GHz quad-core processor and 32 GB PC3 memory. We implement a virtualized multi-tier server cluster assuming that the database tier is not replicated. Virtualization of the cluster is enabled by an enterprise-level virtualization product, VMware ESX 4.1. aMOSS uses vSphere API to dynamically instantiate or de-instantiate VMs on the hosts. Each tier of an application is hosted inside a virtual machine with 1 VCPU, 4 GB RAM and 15 GB hard disk space.

Like others in [17], [20], [24], we use open-source multi-tier application RUBiS in our experimental study. RUBiS implements the core functionality of an eBay like auction site: selling, browsing and bidding. We configure the RUBiS clients to generate workloads of step-change time varying intensity. The number of concurrent users is 300 in the first 30 minutes, and is changed to 500, 900, and 700 in the consecutive 30-minute intervals.

We compare aMOSS with a representative dynamic server provisioning approach proposed in [20]. In that per-tier decomposition based approach, an average end-to-end response time target is decomposed into per-tier targets and per-tier server provisioning is conducted based on a $G/G/1$ queueing model to meet the per-tier targets. We experimented with the approach that sets the average response time target of the three tiers to be 10%, 50% and 40% of the end-to-end response time target [20]. We estimate the parameters for our analytic performance model by examining the server logs, similar to the approach applied in [20].

Figure 10 (a) shows that aMOSS achieves a lower average response time than the per-tier decomposition based approach does. It is due to the fact that the per-tier decomposition based approach only aims to achieve the per-tier delay target. Whereas, aMOSS aims to minimize the average end-to-end response time in a multi-tier architecture while maintaining efficient resource utilization. It also shows that the per-tier decomposition based approach violates the average response time target by a big margin during the "control interval" of minutes 60-65. However, aMOSS avoids the violation because it effectively allocates servers to multiple tiers and achieves a lower response time due to the automated multi-objective server provisioning optimization.

Figures 10 (b) shows that aMOSS is more efficient in the virtual server allocation. Both approaches use only one physical machine in this experiment. We observe up to 32% improvement in virtual server allocation efficiency and 10% improvement in the average response time reduction as shown in Figure 10 (c). Certainly, the improvement margin of aMOSS depends on how the per-tier decomposition based approach decomposes the end-to-end response time target to per-tier targets and how per-tier delay factor associates with the resource allocation and workload characteristics. The case study illustrates the merits of aMOSS for automated multi-objective optimization.

VIII. Conclusions and Future Work

In this paper, we proposed to explore the promise of automated multi-objective optimization for autonomic server provisioning using a novel stress-strain curving method (aMOSS). Extensive experimental results based on the simulation as well as implementation have demonstrated its significant merits in trading off the resource allocation and the performance of multi-tier applications. aMOSS automatically chooses the yield-point solution of the Pareto-optimal set, which essentially guarantees the most efficient tradeoff among multiple conflicting objectives. aMOSS also significantly outperforms a representative dynamic server provisioning approach for the end-to-end response time guarantee in a multi-tier Internet service.

aMOSS provides a novel perspective to tackle the challenging autonomic server provisioning problem. The aMOSS approach is generalizable to any objective in a datacenter that can be modeled with sufficient accuracy. Furthermore, multiple correlated objectives can be combined together as stress and the objectives conflicting to stress can be formulated as strain. In our future work, we will address aMOSS application for joint power and performance control in data centers. We will also address service differentiation among competing applications in case of resource saturation.

Acknowledgement

This research was supported in part by U.S. National Science Foundation CAREER Award CNS-0844983 and research grants CNS-08244448 and CNS-0720524. The authors thank the NISSC for providing blade server equipments for conducting the experiments.

References

[1] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2005.

[2] H. Boyer. *Atlas of Stress-Strain Curves*. ASM International, 1987.

[3] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system auto-configuration. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 670–673, 2009.

[4] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content Web servers. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2006.

[5] P. Costa, J. Napper, G. Pierre, and M. Steen. Autonomous resource selection for decentralized utility computing. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2009.

[6] R. Das, J. O. Kephart, J. Lenchner, and H. Hamann. Utility-function-driven energy-efficient cooling in data centers. In *Proc. IEEE Int'l Conf. on Autonomic computing (ICAC)*, 2010.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6:182–197, 2002.

[8] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, and M. Surendra. Controlling quality of service in multi-tier Web applications. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2006.

[9] M. C. Huebscher and J. A. McCann. A survey of autonomic computing: Degrees, models, and applications. *ACM Computing Surveys*, 40(3), 2008.

[10] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2010.

[11] P. Lama and X. Zhou. Efficient server provisioning for end-to-end delay guarantee on multi-tier clusters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2009.

[12] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010.

[13] P. Lama and X. Zhou. Perfume: Power and performance guarantee with fuzzy mimo control in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2011.

[14] J. C. Leite, D. M. Kusic, and D. Mossé. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *Proc. IEEE Int'l Conf. on Autonomic computing (ICAC)*, 2010.

[15] C. Lumezanu, S. Bhola, and M. Astley. Utility optimization for event-driven distributed infrastructures. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2006.

[16] D. A. Menascé and M. N. Bennani. Autonomic virtualized environments. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2006.

[17] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of the EuroSys Conference (EuroSys)*, pages 13–26, 2009.

[18] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, pages 21–30, 2010.

[19] G. Tesauro, R. Das, W. E. Walsh, and J. O. Kephart. Utility-function-driven resource allocation in autonomic systems. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2005.

[20] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1):1–39, 2008.

[21] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2004.

[22] H. Wang, Q. Teng, X. Zhong, and P. Sweeney. Using the middle tier to understand cross-tier delay in a multi-tier application. In *Proc. IEEE Int'l Parallel Distributed Processing Symp.(IPDPS)*, 2010.

[23] X. Wang, D. Lan, G. Wang, X. Fang, Y. Meng, Y. Chen, and Q. Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2007.

[24] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. Probabilistic performance modeling of virtualized resource allocation. In *Proc. IEEE Int'l Conf. on Autonomic computing (ICAC)*, 2010.

[25] C. Weng, M. Li, Z. Wang, and X. Lu. Automatic performance tuning for the virtualized cluster system. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2009.

[26] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier Internet applications. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2007.

[27] X. Zhou, J. Wei, and C.-Z. Xu. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, pages 88–97, 2004.