Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments

Qian Zhu, *Student Member*, *IEEE*, and Gagan Agrawal, *Senior Member*, *IEEE*

Abstract—The recent emergence of clouds is making the vision of utility computing realizable, i.e., computing resources and services can be delivered, utilized, and paid for as utilities such as water or electricity. This, however, creates new resource provisioning problems. Because of the *pay-as-you-go* model, resource provisioning should be performed in a way to keep resource costs to a minimum, while meeting an application's needs. In this work, we focus on the use of cloud resources for a class of *adaptive applications*, where there could be application-specific flexibility in the computation that may be desired. Furthermore, there may be a fixed time-limit as well as a *resource budget*. Within these constraints, such adaptive applications need to maximize their Quality of Service (QoS), more precisely, the value of an application-specific benefit function, by dynamically changing *adaptive parameters*. We present the design, implementation, and evaluation of a framework that can support such dynamic adaptation for applications in a cloud computing environment. The key component of our framework is a *multi-input-multi-output* feedback control model-based dynamic resource provisioning algorithm which adopts reinforcement learning to adjust adaptive parameters to guarantee the optimal application benefit within the time constraint. Then a trained resource model changes resource allocation accordingly to satisfy the budget. We have evaluated our framework with two real-world adaptive applications, and have demonstrated that our approach is effective and causes a very low overhead.

Index Terms—Cloud computing, adaptive applications, control theory

1 Introduction

Utility computing was a vision stated more than 40 years ago [20]. It refers to the idea that computing resources and services can be delivered, utilized, and paid for as utilities such as water or electricity. The recent emergence of *cloud computing* is making this vision realizable. Examples of efforts in this area include Infrastructure as a Service (IaaS) providers like Amazon EC2 [2] and Software as a Service (SaaS) providers like Google AppEngine [5] and Microsoft Azure [3]. More recently, there is a growing interest in the use of cloud computing for scientific computing, as evidenced by a large-scale funded project like Magellan,[1] and new workshops.[2,3]

Dynamic provisioning of computing and storage resources is possible in cloud computing, and in fact, is often the key attraction for its users. Because of the *pay-as-you-go* model, the goal of resource provisioning should be to keep the resource budget to a minimum, while meeting an application's needs. Current cloud service providers have taken some steps toward supporting a true *pay-as-you-go* or a utility-like pricing model. For example, in Amazon EC2, users pay based on the number or type of the instances they use, where an instance is characterized (and priced) on basis of parameters like CPU family/cores, memory, and disk capacity. The ongoing research in this area is pointing toward the likelihood of supporting more fine-grained allocation and pricing of resources [26], [32]. Thus, we can expect cloud environments where CPU and/or memory allocation for a virtual instance can be changed on-the-fly, with an associated change in price for every unit of time.

While current cloud systems are beginning to offer the utility-like provisioning of services, provisioning of resources has to be controlled by the end users. This, however, is a new and unfamiliar paradigm for computer application developers and users, who are accustomed to working with a fixed set of resources they own. We feel that it is desirable that resource allocation in a cloud environment can be performed *automatically* and *dynamically*, based on users' high-level needs, i.e., based on their resource budget, time constraint, and/or desired quality of results.

The resource provisioning problem becomes particularly challenging for the emerging class of applications, which we refer to as the *adaptive applications* [11], [14], [23], [36], [38]. In these applications, there is typically an application-specific flexibility in the computation that may be desired. For example, input data can be sampled at different rates or models can be run at different spatial and temporal granularities. Such adaptive applications are clearly suitable for cloud computing. This is because cloud environments are *elastic*, i.e., they can provide resources on-demand to meet the computation, memory, and storage requirements of these applications. In fact, several recent efforts have

1. Please see http://www.cloudbook.net/doe-gov.
2. For example, http://dsl.cs.uchicago.edu/ScienceCloud2010/.
3. http://www.usenix.org/event/hotcloud10/.

- *Q. Zhu is with Accenture Technology Labs, 50 W San Fernando St. Suite 1200, San Jose, CA 95113. E-mail: qian.zhu@accenture.com.*
- *G. Agrawal is with the Department of Computer Science and Engineering, The Ohio State University, 2015 Neil Ave. #395, Columbus, OH 43210. E-mail: agrawal@cse.ohio-state.edu.*

Manuscript received 24 Nov. 2010; revised 18 Oct 2011; accepted 5 Dec. 2011; published online 27 Dec. 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2010-11-0145. Digital Object Identifier no. 10.1109/TSC.2011.61.

specifically focused on exploiting the elasticity of Clouds [16], [17], [31] for various application classes.

When such adaptive applications are executed in a cloud setting, there is a tradeoff between resource costs incurred and the QoS obtained. It can be extremely difficult for an application developer or user to manually control the value(s) of adaptive parameter(s), and/or control the allocation of resources, so as to achieve the desired tradeoff. This paper addresses the automated and dynamic resource allocation problem to support the execution of adaptive applications in a cloud environment. We particular consider applications that comprise several *services*, each of which is mapped to a virtual machine (VM) in a cloud environment. We assume that the allocation of resources (CPU cycles and memory) to each VM can be dynamically controlled, and the resource costs incurred depend upon the resources allocated. The problem considered in this paper involves a prespecified *time-limit*, as well as a *resource budget* for a particular task. Within these constraints, an adaptive application needs to maximize the Quality of Service (QoS) metric, more precisely, the value of an application-specific *benefit function*, which captures what is most desirable to compute within the specified constraints.

The key conceptual component of our framework is a dynamic resource provisioning algorithm, which is based on control theory. We have applied a reinforcement learning guided control policy to adjust the adaptive parameters so that application benefit is maximized within the time constraint using modest overhead. Such a control model can be trained fast and accurately. Furthermore, a *resource model* is proposed to map any given combination of values of adaptive parameters to resource requirements in order to guarantee that the resource cost stays under the budget. Our framework is not specific to any particular cloud environment, and in the future, it can easily be integrated into existing cloud infrastructures such as EUCALYPTUS [19] and Amazon EC2 [2].

Since we dynamically assign CPU cycles and memory to multiple virtual machines, this work is somewhat related to the existing work on virtualized resource scheduling [26], [37], [18]. However, our work is distinct in considering a pricing model and a fixed resource budget. Our work is also related to the research on budget constrained scheduling in utility and grid computing [29], [22], [40]. The work we presented is distinct in considering adaptive applications and parameter adaptation.

We have evaluated our framework with two real adaptive applications. The main observations from our experiments are as follows:

- The CPU cycle/memory allocation generated through the use of our resource model is within 5 percent of the actual CPU/memory utilization. Furthermore, the model can be trained on one system and then applied on a different system effectively.
- Second, our dynamic resource provisioning algorithm achieves a benefit of up to 200 percent of what is possible through a static provisioning scheme. At the same time, the scheme could perform parameter adaptation to meet a number of different time and budget constraints for the two applications.
- The overhead caused by the dynamic resource provisioning algorithm is less than 10 percent, comparing to an optimal execution.

The rest of the paper is organized as follows: We motivate our work by describing adaptive applications we are targeting in Section 2. We also describe the cloud computing environment and pricing models used in this work in this section. In Section 3, we present the control theory-based model and our dynamic resource provisioning algorithm. Section 4 describes the design of our framework, which has been developed with the goal of easy integration with any cloud system. Results from experimental evaluation are reported in Section 5. We compare our work with related research efforts in Section 6 and conclude in Section 7.

2 Adaptive Applications, Target Environment, and Pricing Models

We first discuss the adaptive applications that motivates this work. Next, the cloud environment we target is described. Finally, we present two pricing models that have been used for our current evaluation.

Adaptive applications. Our work is driven by a class of applications that we can refer to as *adaptive applications*. In these applications, there is some flexibility with respect to the computing performed. For example, input data can be sampled at different rates, models can be run at different spatial and temporal granularities, or different values of an *error rate* or a similar accuracy parameter can be chosen.

These applications have been studied in a number of different contexts. For example, projects have focused on operating systems, middleware, and networking support for adapting applications to meet quality of service goals. Bhargavan et al. [14] focus on adapting frame rate over wireless networks. SWiFT is a software feedback toolkit to support program adaptation [45]. Adve et al. have focused on compiler support for adaptive applications [11]. The particular adaptive applications they have considered include a stochastic optimization solver and video tracking. Schwan and his group take into account the runtime resource management issues when supporting adaptable Applications [36]. Web applications in data centers could be another good candidate as there are many parameters associated with web server and/or data server such as number of threads per session, maximum number of database connections, can be adapted and are impactive on system throughput and response time [15].

Cloud environment. Our work has been conducted on a *synthetic* cloud environment, which has been emulated using clusters we had access to. The cloud environment we use is similar to the existing and emerging cloud environments in many respects. Our cloud environment allows *on-demand* access to resources. Like the existing cloud environments, applications are charged for their resource usage according to a *pricing model*. We also use virtualization technologies, which enable applications to set up and deploy a customized virtual environment suitable for their execution. Secure sharing of resources between different applications and users is allowed.

Our work is based on the assumption that fine-grained allocation and pricing of resources is possible for the virtual

environment. This is not true for the existing cloud environments, but is consistent with the utility vision of computing, and recent research points to the progression of clouds in such a direction [26], [32]. Thus, we assume that CPU cycles and memory can be shared in a fine-grained fashion between different instances. The user of an instance can request for a different CPU cycle percentage or memory allocation at any point during its execution. Two pricing models we have used target a cloud environment with such a fine-grained sharing of resources. These models are described later in this section.

In our implementation and experiments, we use virtual machines (VMs) that run on top of the Xen hypervisor [35]. Since the adaptive applications we target in our work are compute-intensive, we have considered virtual CPU and memory usage, as elaborated below.

CPU usage: Xen provides a Simple Earliest Deadline First (SEDF) scheduler that implements *weighted fair sharing* of the CPU capacity among all the VMs. The share of CPU cycles for a particular VM can be changed at runtime. The SEDF scheduler can operate in two modes: *capped* and *noncapped*. In the capped (or *non-work-conserving*) mode, a VM cannot use more than its share of the total CPU time in any time-interval, even if there are idle CPU cycles available. In contrast, in the noncapped (or *work-conserving*) mode, a VM can use extra CPU time beyond its share, if other VMs do not need them. We particularly focus on the *non-work-conserving* use of VMs, which allows a better performance isolation among multiple VMs, and supports a fair pricing model.

Memory usage: We use *balloon driver* in Xen to dynamically change the memory allocations for the virtual machines. Each VM is configured with a maximum entitled memory (*maxmem*). The VM starts with an initial memory allocation, which can be later increased up to the specified *maxmem* value. If one VM does not need all of its entitled memory, the unused memory can be transferred to another VM that needs it. This can be done by *inflating* the balloon (i.e., reducing allocation) in the first VM and *deflating* the balloon (i.e., increasing allocation) in the second VM. This mechanism allows dynamic memory repartitioning among multiple VMs as needed.

Pricing model. Our work assumes a fine-grained pricing model where a higher allocation of CPU cycle percentage or memory is associated with a higher cost for each time unit. Beyond this basic assumption, our resource allocation framework is independent of the details of the pricing model. For our experiments, we have evaluated our framework using two different pricing models, which we describe below. For simplicity, we only focus on costs associated with computing cycle allocation and memory allocation. Depending upon the application, additional costs may be associated with storage and data transfers.

We have used a *linear pricing model* and an *exponential pricing model*. In the *linear pricing model*, the resource cost charged to the users is linearly scaled with the amount of resources that have been assigned to the application. Let us first consider CPU cycles. Assume there are m phases with distinct CPU cycle percentage during the execution of an application, i.e., allocation percentage is changed $m-1$ times. R_i^{cpu} is the CPU cycle percentage at the ith allocation, t_i be the duration we stay at the ith allocation. If CPU_{base} is the base price charged for the smallest amount of CPU cycle allocation and CPU_{trans} denotes the *transfer fee* charged each time we change the CPU allocation. With a total of m changes, the cost with regard to CPU usage can be calculated as following:

$$CPU_{cost}(m) = \sum_{i}^{m} CPU_{base} \times R_i^{cpu} \times t_i + (m-1) \times CPU_{trans}.$$

We can calculate the cost of memory usage, Mem_{cost}, in a similar fashion. Note that R_i^{mem} is the memory assigned at the ith allocation, Mem_{base} is the base price charged for allocating 1 GB of memory, and Mem_{trans} represents the transfer fee of memory change

$$Mem_{cost}(m) = \sum_{i}^{m} Mem_{base} \times R_i^{mem} \times t_i + (m-1) \times Mem_{trans}.$$

In our second pricing model, which is the *exponential pricing model*, the intuition is that when a high percentage of CPU cycles and/or a large fraction of the available total memory is dedicated to a single VM, other VMs may have to be suspended or the applications running on them may have to be rejected. Thus, we assume an exponential pricing based on the reserved CPU cycle percentage. So, the cost of CPU percentage allocation in this model will be calculated as follows:

$$CPU_{cost}(m) = \sum_{i}^{m} \left(CPU_{base} \times \left(1 - exp^{-0.01R_i^{cpu}}\right) \times t_i\right) + (m-1) \times CPU_{trans.}$$

Similarly, the cost of memory usage in this model can be calculated as

$$Mem_{cost}(m) = \sum_{i}^{m} \left(Mem_{base} \times \left(1 - exp^{-0.01R_i^{mem}}\right) \times t_i\right) + (m-1) \times Mem_{trans.}$$

As noted above, other resources such as disk I/O and network I/O bandwidths are not considered in our current model, though they can be incorporated in the future. Our adaptation framework currently targets compute-intensive applications, where key factors in adapting the resource costs and performance of the application are the CPU cycle allocation and memory size. Furthermore, dynamic (on-the-fly) changes to disk I/O and network I/O bandwidth settings are not available in Xen yet.

3 Dynamic Resource Provisioning Algorithm

In this section, we present our dynamic resource provisioning algorithm. In context of the overall framework design, which is presented in the following section, this algorithm is implemented in the resource provisioning controller.

We first define the resource provisioning problem we want to address in this work. Then, we give an overview of our approach. Next, the details of our approach, based on an adaptive feedback-loop, are described. Finally, we describe how resource models are generated with the goal

of converting changes in values of an adaptive parameter into CPU cycles and memory allocation requests.

3.1 Problem Formulation

An adaptive application we target comprises a series of interacting service components, which we denote as $S_1, S_2, \ldots, S_n$. An application has a benefit function, denoted as B. This function takes certain application parameters as the input, and outputs a Quality of Service indicator, called the *benefit*. Note that such benefit function is application-specific and user-defined. Every processing of a time-critical event we perform is associated with a prespecified time constraint, denoted as T_c. We also have a specified budget for each application invocation, which is denoted as C_0.

Our goal is to perform the processing, as to maximize the application benefit value, within the time limit T_c and the budget C_0. Impacting the benefit value obtained by an execution are the *adaptive parameters*. Each service could have one or more such adaptive parameters. Certain choices of values of these adaptive service parameters are likely to either speedup the processing, or require fewer CPU/memory resources. However, the resulting value of the benefit function will also be lower. Other choices of values would increase the budget, but would lead to an increase of the benefit. Thus, the goal of our framework is to choose the right tradeoff between the benefit and the resource cost, by adapting the values of the service parameters.

There can be two possible ways of approaching this problem. The first approach involves treating application's adaptive parameters as the *controllable parameters*, and CPU cycles and memory as the *dependent parameters*. Given any change in the value of adaptive parameter(s), we adjust the resource allocation in a way to ensure that the rate of progress is maintained. While adjusting the adaptive parameters, we can examine the modification to the resource allocation that would be required, and estimate the impact on the resource budget. The second approach would consider *resources* as the controllable parameters, and the adaptive parameters as the dependent parameters. This approach will require a mapping from changes in the CPU/memory assignments to the changes in the value(s) of the adaptive parameter(s), to determine whether a better application benefit can be achieved at the current rate of progress, as the benefit function is dependent on the values of adaptive parameters. Since it is more straightforward to relate the adaptive parameter values to the execution time and the application benefit, rather than relating resource usage to these metrics, we have taken the first of the above two approaches.

Note that the problem we solve in this paper involves a fixed time limit as well as a budget limit, within which the goal is to maximize the application benefit. Depending upon the application, users, and/or the organization, different formulations may be appropriate. For example, one formulation could involve a time constraint and a certain benefit level, subject to which the goal is to minimize the total resource costs. Yet another formulation could be as a *biobjective* problem [24], where we try to achieve a tradeoff between resource costs and the benefit. These variations to the problem can be easily solved by small changes to our framework. However, for our presentation, we only focus on the original problem we have formulated, where benefit is maximized subject to a resource budget and a time limit.

In any of the above formulations, users can take advantage of a feedback mechanism, which makes them aware of possible tradeoffs. For example, they can be informed that a significantly better benefit value can be obtained by only a small increase in their budget, or that a significant reduction in budget is possible by reducing their benefit target only marginally. Such feedback is currently possible within our framework, though the details are omitted due to the space constraints.

3.2 Overview of the Approach

Our approach is based on the observation that the problem of resource (CPU/memory) provisioning corresponds to a *feedback control model*. Feedback control model has been applied for dynamic virtual resource provisioning [32], [37], [44]. Unlike the previous work which optimizes a single performance metric (response time or throughput) by directly controlling the resources allocated to the application, we consider a more unique and complex problem where the application benefit depends on the values of the adaptive parameters thus making it hard to maximize the benefit by controlling the resource allocations. We used the feedback control model to guide the parameter adaptation in order to maximize the application benefit while satisfying the time constraint and resource budget. Then virtual resources are dynamically provisioned according to the change in the adaptive parameters.

In control theory, an object to be controlled is typically represented as an *input-output* system, where the inputs are the *control knobs* and the outputs are the metrics being controlled. Typically, a *controller* manipulates the inputs to the system under the guidance of a *performance objective*. In viewing our problem as a feedback control problem, the inputs are a set of adaptive parameters, $u_i, \ldots, u_n$. There are three outputs we consider, which are the application benefit, execution time, and the resource cost. The performance objective here is to maximize the application benefit, i.e., values of the function B, subject to constraints on the execution time (T_c) and the resource costs incurred (C_0). The objective of the control problem we are considering is to find the sequence $u_i(1), \ldots, u_i(N)$ for each adaptive parameter i, given N time steps.

Note that the resource allocation at any point during the execution is an important parameter, but is not considered as either an input or the output in our formulation. This is because any change in the values of the adaptive parameters results in a corresponding change in resource requirements for maintaining the same *progress rate*, and therefore, resource allocation is changed based on any change in the value of an adaptive parameter. This is formally captured through our *resource model*, which is described toward the end of this section.

3.3 Detailed Control Model Formulation

In order to effectively utilize control theory, we need to define a *system model*. The goal of the system model is to capture the relationship between the application performance and the input parameters, to be able to guide the adaptation process.

TABLE 1
Main Terms Used in Formulation

Variable	Description
$u_i(k)$	i^{th} Adaptive parameter at time step k
$S(k)$	State at time step k
$D(k)$	Action taken at time step k
$W_0(S(k), D(k))$	Benefit value at time step k
$W_1(S(k), D(k))$	Total time elapsed at time step k
$W_2(S(k), D(k))$	Resource costs at time step k
$\mathbf{Perf}(k)$	Observed performance at time step k
$\hat{\mathbf{Perf}}(k)$	Estimated performance at time step k

The first step in developing such a system model is to simplify our performance objective. This is done by introducing the following two components of the performance of the application.

- *Processing progress*: It is defined as the ratio between the currently obtained application benefit and the elapsed execution time. This metric measures the rate at which the application processing is gaining the benefit.
- *Performance/cost ratio*: It is defined as the ratio between the currently obtained application benefit and the cost of the resources that have been assigned to the application. This metric measures the rate of gaining the benefit for every unit of resource budget consumption.

The main symbols used in our problem description and formulation are summarized in Table 1. During the execution, $\hat{\mathbf{Perf}}(k)$ is the performance vector that is observed at time step k.

The outputs in our case are not likely to be a linear function of the set of inputs. Based on the Padala et al. work [37], we assume the behavior of the system at any point in time can be approximated locally as a linear model. Specifically, by plotting two performance terms in $\mathbf{Perf}$, we determined that the performance at the time step k depends on performance in the previous time step(s), and the *exogenous inputs* at the current time step, which are the adaptive parameter values. Therefore, we have chosen an *auto-regressive-moving-average with exogenous inputs* (ARMAX) model to represent the system behavior. Furthermore, we applied different ARMAX models in our experiments and found out the *second-order* model can predict the application performance with adequate accuracy. Formally,

$$\hat{\mathbf{Perf}}(k) = a_1(k) \times \mathbf{Perf}(k-1) + a_2(k) \times \mathbf{Perf}(k-2) + \sum_i b_0^i(k) \times u_i(k) + \sum_i b_1^i(k) \times u_i(k-1). \quad (1)$$

In the equation above, the *model parameters* $a_1(k)$ and $a_2(k)$ capture the correlation between the application's past and present performance. $b_0^i(k)$ and $b_1^i(k)$ are the weights given to the current and previous values of the ith adaptive parameters in measuring the performance. Recall that changing such values will impact both application benefit and execution time. The model parameters a_1, a_2, b_0^i, and b_1^i are also functions of the control interval k. These parameters are updated at the end of every interval. This is done using SVM regression [43], after the measurement for the performance $\mathbf{Perf}(k)$ is available. We validated the linear model by calculating the mean square error (MSE) between $\mathbf{Perf}(k)$ and $\hat{\mathbf{Perf}}(k)$ and observed the MSE value is below 0.3, demonstrating linearity of such a relationship. Alternative system models, like Neural Networks, could have also been applied to model the nonlinear time series.

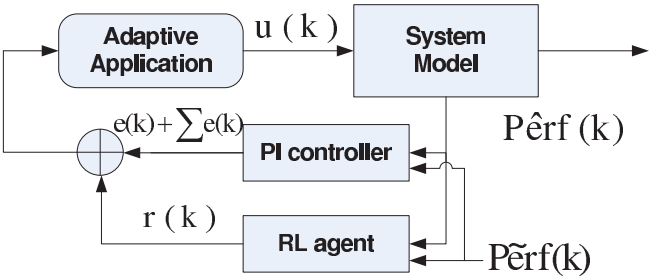

Fig. 1. A PI controller combined with the reinforcement learning agent.

Our controller uses the above system model to achieve our performance objective, which is to maximize benefit, subject to time and budget constraints. We have used the Proportional-Integral (PI) control, combining with a reinforcement learning component. In the absence of knowledge of the underlying process, a PID (Proportional-Integral-Derivative) controller is the best controller where P depends on the present error, I on the accumulation of past errors, and D is a prediction of the system future errors [13]. Instead of using the derivative term which is highly sensitive to the noise in the error thus causing unstability to the controller, we have considered a reinforcement learning (RL) agent combined with the PI controller. With the *trial-and-error* training, the RL agent learns to minimize the sum of the squared error of the control variables (i.e., the adaptive parameters) without violating the time and budget constraints over time. Furthermore, the trained RL agent augments the output of the PI controller when it results in improved application benefit, thus achieving the maximum value. The proposed controller design for adaptive applications is illustrated in Fig. 1. We now discuss the detailed PI controller design and the reinforcement learning algorithm that has been applied.

PI controller. Before the execution of the algorithm, we applied a biobjective optimization algorithm [41] to decide the CPU cycle assignment and memory size, respectively, where the application benefit was maximized and resource cost was minimized. We refer this approach to as `Static Scheduling`. The maximum benefit achieved from the `Static Scheduling` is the baseline benefit, denoted as B_0. The baseline benefit represents the minimum value we want to achieve. Thus, we denote the setpoints for the PI controller as $\tilde{\mathbf{Perf}}(k) = <B_0/T_c, B_0/C_0>$ to guarantee that our proposed algorithm can always achieve better benefit than the static counterpart does within the time and resource cost constraints. The following formula calculates the ith control variable

$$u_i(k) = K_P e(k) + K_I \sum_{\tau=0}^{k} e(\tau) + K_R r_i(k), \quad (2)$$

where

$$e(k) = \tilde{\mathbf{Perf}}(k) - \hat{\mathbf{Perf}}(k),$$

and $r_i(k)$ is the action generated from the RL agent, which will be discussed below. All weights K_P, K_I, and K_R are tuned using the Ziegler heuristic [46].

Reinforcement learning. The reinforcement learning component in the controller is to maximize the application benefit. Specifically, we have used Q-learning. Now, we formally state our performance objective. From Table 1, the current state of the system, which is the set of values of the adaptive parameters, is captured as $S(k)$. The controller can change one or more of the values of adaptive parameters, which is reflected as an action in this state. We refer it to as $D(k)$. Note that $r_i(k) = S(k) + D(k)$. We have three reward functions, as defined in Table 1. W_0 is the reward function to be maximized while the other two are used to specify constraints. Each of them is calculated as follows. The relationship between the values of the adaptive parameters and the benefit (captured as a function f_B), and the relationship between the values of the adaptive parameters and execution time (captured as the function f_T), are both learned online using *recursive least square* [38]. For each processing round, the obtained application benefit (f_B) and measured execution time (f_T) are regressed based on the adaptive parameters. We considered linear and nonlinear terms in both functions. More details are in our previous work [38].

Recall that a change in a value of an adaptive parameter results in a change in resource allocation, and our pricing models assume that there is an extra cost associated with each change. In view of this, and to achieve better stability, we want to minimize changes to adaptive parameters. Therefore, $W_0(S(k), D(k))$ is a combination of the application benefit and a *control cost*. The benefit at the current time step is the benefit relationship function that takes parameter values at the next time step, i.e., $f_B(S(k+1))$, and we use $\sum_i (u_i(k) - u_i(k-1))^2$ to denote the control overhead. Thus, we have,

$$W_0(S(k), D(k)) = f_B(S(k+1)) - \sum_i (u_i(k) - u_i(k-1))^2.$$

Similarly, $W_1(S(k), D(k))$ is the execution time relationship function with $S(k)$ as the input

$$W_1(S(k), D(k)) = f_T(S(k)).$$

The resource cost, $W_2(S(k), D(k))$, is only indirectly related to adaptive parameters. As we had mentioned before, and as we elaborate in the next section. We use a resource model to map the change of such parameter values to CPU cycle/memory requests. Thus, the resource cost reward function is defined using the resource requests and pricing model presented earlier in Section 2

$$W_2(S(k), D(k)) = CPU\ request \times CPU_{cost}(k) + mem\ request \times Mem_{cost}(k).$$

With the above, formally, our controller solves the following:

$$\max \sum_{k=0}^{N} t_i \times W_0(S(k), D(k)), \tag{3}$$

subject to the constraints,

$$\sum_{k=1}^{N} W_1(S(k), D(k)) \le T_c \quad \text{and} \quad \sum_{k=1}^{N} \mathrm{W_2(S(k), D(k))} \le C_0. \tag{4}$$

3.4 Resource Model

Whenever a change in the value of an adaptive parameter is made, a corresponding change is needed in the CPU cycle and/or memory allocation, if we want to maintain the same rate of progress. In our approach, this is done by developing a *resource model* for each service component. Unlike the previous efforts where control theory has been used to guide resource allocation [37], [18], resource requests are not directly modeled in our control formulation. Instead, we control the adaptation of service parameters, which, in turn, requires that the resource allocation changes to maintain the same rate of progress. We develop the resource models with an *offline training* phase. Such offline training could be performed on a different type of hardware than the one that may be available in the cloud environment. Our experiments will demonstrate that models developed on a different type of hardware could still be very effective.

We now describe how these resource models are developed. A straightforward solution would be to regress the relationship between the adaptive parameter value and the CPU/memory requirements. However, one important factor we need to consider is that changing CPU cycle allocation or memory partition frequently is not desirable. We take the following approach. A tuple D_i collected from the offline training phase is of the type, $<CPU_i, mem_i, t_i, \mathbf{x}_i>$, where CPU_i represents a relative CPU percentage, mem_i is the memory usage, t_i is the execution time, which denotes the time to complete a task, and $\mathbf{x}_i$ is the current values of the adaptive parameters. Due to the heterogeneity in processor architectures, frequencies, size of L1/L2 cache across different platforms, an absolute CPU usage percentage in one environment may not be very useful while driving adaptation on a different environment. Thus, we consider a relative value, i.e., $CPU = CPU/r$, where r is used to account for the heterogeneity between two different types of hardware, with $r = 1$ if there is no difference in hardware. A reasonable initial value for r can be the ratio of the two CPU frequencies. However, as the model is applied, r can be changed based on the feedback of model prediction and real CPU usage.

Our resource modeling consists of the following three steps.

Step 1: We cluster these data points based on their execution time. Namely, data points with execution time within $t \pm \Delta t$ are grouped into one cluster. We applied the *k-means* clustering algorithm for this step [27]. As a result, data points are categorized into different clusters where within each cluster, the parameter adaptation does not necessarily require an increase or decrease in the CPU cycle percentage and/or memory allocation.

Step 2: We apply SVM regression with a cubic kernel [43] to learn the relationship between the values of the adaptive

TABLE 2
Model Optimization

Algorithm III.1: MODELOPT(S, CPU_{req}, mem_{req})

INPUT S: set of services
$< CPU_{req}, mem_{req} >$: CPU and memory requests
OUTPUT Actual CPU cycle and memory allocation
for each $S_i \in S$
// Calculate slow down factor
$d_i = CalculateSlowDownFactor(S_i)$;
if $d_i > \theta_s$
$S' = S' \cup S_i$;
// Generate resource contention sets
$S_c = ResourceContentionSet(S')$;
for each $S_c^j \in S_c$
$InitialResourceassignment(S_c^j)$;
while $(bottleneck)$
$AdjustResourceallocation(S_c^j)$;

parameters and the resource usage, i.e., CPU cycle percentage and memory, within each cluster. Using the SVM regressor, dependencies among parameters in terms of CPU and memory usage have been taken into account.

Step 3: We further study the relationship between $<CPUcycle, memory>$ and the value of adaptive parameter along the time axis.

The final resource model is the product of models from steps 2 and 3. Once the resource model has been trained, given a value of the adaptive parameter, the model will generate a CPU cycle/memory request for a service component.

Model optimizer. This module further optimizes the CPU cycle and memory allocation requests generated by the resource models for the individual services. There are two goals in performing these optimizations. First, we do not want to change CPU cycles with every change in adaptive parameter values. The reason is that, there is a cost associated with performing such a change. In addition, resource underprovisioning or overprovisioning from not performing such a change is likely to be small. Similarly, memory does not have to be repartitioned for each parameter adaptation. This is due to the observation that memory usage below a certain threshold of the available memory will not impact the application performance. Such threshold is denoted as $mem_{threshold}$ and it is set to be 0.9 in our implementation. The other goal for the optimizer is balancing global CPU cycle allocation among multiple services. Specifically, when there is a contention for shared CPU cycles and/or memory size, such resource requests should be satisfied based on the *priority* of the service. At the same time, because services could be interdependent on one another, we need to make sure that the global decisions do not cause a bottleneck among the services.

The outline of the model optimization strategy we use is shown in Table 2. The optimization procedure starts by checking a *slow-down* factor for each service, i.e., by how much is the execution time slowed if we remain at the current allocation. We mark the services for which the slow-down factor is above a certain threshold, θ_s, and include these services in the set S'. The significance of the set S' is that if we remain at the current allocation, a significant overall slow-down is likely for the entire application. We reject resource requests from the services in two cases, i.e., they are not in the set S', or, they only require memory change and the changed value is still below 90 percent of the total memory. Now, the services from the set S' are grouped into *resource contention sets*. These sets reflect services that are sharing the same resource, and thus, the only contention for CPU cycles and/or memory is from the services within the same set. Within such a set, service priority has to be considered. We proposed the following service priority assignment heuristic. For all services that are associated with adaptive parameters, we change 20 percent of the parameter values and observe the change in the benefit function (f_B). Service priorities are assigned based on the change of the benefit value, with a bigger change implying a higher priority. While the services without adaptive parameters are assigned with lower priorities than the first group. Within the set, we sort such services based on their execution time where the most time consuming service is assigned with the highest priority. A service with a higher priority should be more likely to obtain the resource it is requesting. However, resource allocation to a service with a lower priority may be needed to avoid the possibility of its becoming a bottleneck. We first assign the available CPU cycles and memory to the services within the same resource contention set based on their priority. Next, a *speed-up* factor is calculated for each service, given the newly assigned resources. If for a particular service, the value of this factor is much lower than that of other services, we claim that it could potentially become a bottleneck. The initial assignment is adjusted until we eliminate any such bottleneck. Note that the total limit on the resources allocated is determined by the performance/cost ratio we need, so as to not exceed the budget.

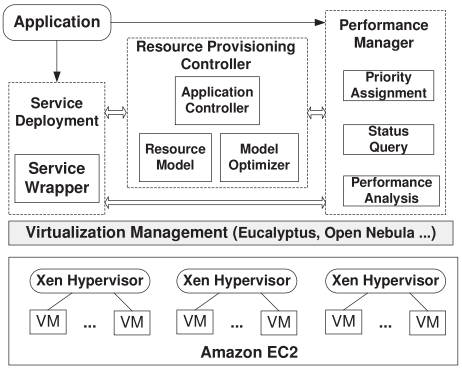

Fig. 2. Overall framework design.

4 Framework Design

The feedback control-based methodology described in the previous section has been implemented as a framework which can be easily incorporated as part of various cloud middleware systems. Our design adopts the SOA concepts and the underlying functionality is provided as services. In this section, we present the major design aspects of our framework.

4.1 System Architecture and Design

The overall system architecture is illustrated in Fig. 2. It is a flexible and modular design reflecting common characteristics of adaptive application and facilitating its execution in

clouds. An application is implemented as a set of loosely-coupled service components, with possible dependencies between them. Each service components is able to *self-describe* its interface and *self-optimize* its contribution to the overall processing. This is accomplished through *service wrapper* and the *performance manager* modules, respectively.

In order to prepare the adaptive application to be ready for deployment in the cloud computing environment, when it is submitted to the system, the *service wrapper* of our framework first wraps it into a set of service components. In our current implementation, this is done according to the Web Service Resource Framework (WSRF) standards, though we can support other web and cloud computing standards in the future. Since each service component has one or more adaptive parameters and their values need to be stored for each processing step, our service components are *stateful* and such adaptive parameters are exposed as *state resources*. In our design, the service wrapper is the entry-point into the clouds for users and administrators.

The *performance manager* takes into account multiple factors that relate to application QoS. It first analyzes the benefit function and assigns different priorities to service components based on their contribution to the overall application benefit. This is a critical issue, as tuning adaptive parameters from a service with a higher priority is more important for achieving a larger benefit.

The performance manager also plays an important role in addressing the resource provisioning challenge, for which it works in conjunction with the *resource provisioning controller*, which is the central component in our framework. During the application execution, the performance manager iteratively queries the processing progress and performance/cost ratio. As presented in Section 3, in the resource provisioning controller, feedback control theory is applied to help throttle up or down the CPU cycle and memory allocation for different VMs. Such controller is customized to the application needs, thus decoupled from the underlying cloud platform mechanism. A more detailed design of the resource provisioning controller is elaborated later in this section. The models and the dynamic resource provisioning algorithm have been described in the previous section.

Different service components of the adaptive application do not share physical resources directly, but rely on virtualization technologies to abstract them. The *virtualization management* module controls the VMs. Note that a unique functionality of our framework is the ability to dynamically change virtual CPU and memory assignment among multiple VMs, as requested by the application services they host. As stated previously, we believe such fine-grained control could be exported by the cloud providers in the near future. In our current implementation, we have a simple virtual machine interface for VM management. However, our framework can be easily ported to the existing cloud infrastructures, such as EUCALYPTUS [19] and Open Nebula [9] for better VM management. Furthermore, integration into such cloud systems could help control VMs hosted by a cloud provider such as Amazon EC2 [2].

4.2 Resource Provisioning Controller

We now describe the detailed design of the *resource provisioning controller*. There are two major technical aspects to the design of this module. First, the feedback control theory-based adaptation has been implemented in the *application controller*. This module relates the values of the adaptive parameters to the rate of application processing progress and the performance/cost ratio.

For each service component in the adaptive application, its resource model periodically polls the resource usage, the current values of adaptive parameters and time elapsed since the last time step. Each resource model will compare its output (CPU/memory requests) with the actual resource usage. Although the initial resource model for each service is trained offline, the model will be updated based on the comparison. These updates to the models are critical for achieving accuracy, because the original model could have been trained on a different type of hardware. At each time step, the resource model of the service component sends out a resource request to the application controller, which is due to the change of its adaptive parameters.

Based on the collective requests from all resource models of the various service components, the application controller determines whether it has enough resource of each type to satisfy all demands and generates the actual resource allocation using the model optimization method described in Section 3.4. The generated resource allocations are fed into the virtualization management layer in our framework for actuation. Our design of the resource provisioning controller allows the placement of resource models and application controllers in a distributed fashion. For example, each of them can be hosted in a physical node separately, assuming high-speed network connection is available to communicate between VMs and these physical nodes.

5 Experimental Evaluation

In this section, we present results from a number of experiments we conducted to evaluate our framework and the dynamic resource provisioning algorithm.

5.1 Schemes Compared, Metrics, and Goals

For comparison, we used two alternative resource allocation methods. `Work-conserving` approach is commonly used on consolidated infrastructures today. In the work-conserving mode, the services run in the default Xen settings, where a cap of zero is specified for the shared CPU on a node, indicating that the services can use any amount of CPU resources. In this mode, we further assume that each VM can request any amount of memory, within what is available on the server. This may not be a practical scheme to use in a cloud environment, where users have fixed resource budgets. We use this scheme to compare our method against the scenario where parameter adaptation can be performed to achieve high application benefit. This scheme is also used to evaluate the accuracy of our resource models.

The second approach is the `Static Scheduling` approach which has been described in Section 3. Recall that this is a static method, the decision is made before initiating the execution, and the CPU cycle and memory assignment will remain the same during the processing.

To evaluate the performance of our dynamic resource provisioning algorithm against `Work-conserving` and `Static Scheduling`, we use the following two metrics:

- *Benefit percentage*: This shows the benefit obtained from the resource provisioning algorithm, as the percentage of a baseline benefit. Recall that we take the maximum benefit achieved from the `Static Scheduling` approach as the baseline benefit.
- *Resource cost*: This is the price charged for the CPU cycles as well as memory used by the execution of the application.

Specifically, we have applied two pricing models that were previously defined in Section 2. Also, we used the following pricing rates from Amazon EC2: $CPU_{base} = \$0.0017$ per compute unit per hour and $Mem_{base} = \$0.05$ per GB per hour. Note that in our pricing model, a compute unit is 1 percent of one physical core. Additionally, we set $CPU_{trans} = \$0.005$ and $Mem_{trans} = \$0.50$, as they are required in our pricing models. These values are chosen to reflect a case where the cloud provider allows frequent changes in resource allocation, particularly, the CPU percentage allocation.

Using the three resource allocation approaches and the above two metrics, we designed the experiments with the following goals:

- Demonstrate that our proposed model is effective in CPU cycle and memory allocation with high resource utilization. Also, models trained on one type of hardware can still be effective on another type of hardware.
- Demonstrate that the maximum benefit achieved by our dynamic resource provisioning method is larger than that achieved by `Static Scheduling`, within the time constraint. At the same time, the resource cost always stays under the prespecified budget.
- Evaluate the overhead of our dynamic resource provisioning algorithm.

5.2 Experimental Setup

Our experimental cloud testbed is emulated using 2 Linux clusters, each of which consists of 64 computing nodes. One cluster has dual Opteron 250 (2.4 GHz) processors with 8 MB L2 cache and 8 GB main memory. While the other has Intel Xeon CPU E5345 (2.33 GHz), comprising two quad-core CPUs, with 8 MB L2 cache and 6 GB main memory. Computing nodes are interconnected with switched 1 Gb/s Ethernet within each cluster. The two clusters are located in different buildings, about 0.5 miles apart, within the Ohio State university campus and are connected using two 10 Gb/s optical fibers. We chose Xen as the virtualization technology and we used Xen-enabled 3.0 SMP Linux kernel in a stock Fedora 5 distribution. On a single consolidated server, hardware resources are shared between the virtual machines that host application service components and the management domain (`dom0` in Xen terminology). Throughout our experiments, we restrict the management domain to use one physical core, thus isolating it and avoiding performance interference. The virtual machines hosting application services share the remaining physical cores. The placement of VMs is decided at the start of application execution. Placements of VMs by taking performance interference into account could be an important factor for performance, but is beyond the scope of this paper. Similar to Amazon EC2 VM instances, the VMs are created and customized for each service component. However, as stated earlier, we allow fine-grained CPU cycle and memory allocation in our cloud environment.

The experiments we report were conducted using two applications, the Great Lake nowcasting and forecasting system (`GLFS`) and `Volume Rendering`. `GLFS` monitors the meterological information of lake Erie for nowcasting (for the next hour) and forecasting (for the next day). Normally, the Lake Erie is divided into multiple coarse grids, each of which is assigned available resources for model calculation and prediction. Moreover, these models can be executed for various other tasks of interest to local and state authorities, such as managing sewage disposal. Flexibility of this application includes the *grid resolution* of the computational grids, *interal time step* and *external time step*, which are related to the accuracy of the prediction as well as the application execution time. `Volume Rendering` interactively creates a 2D projection of a large time-varying 3D data set (volume data) [21]. Under normal circumstances, the system invokes services for processing and outputs images to the user at a certain *frame-rate*. In cases where a *notable event* is detected in a particular portion of the image, the user may want to obtain detailed information on that area as soon as possible. There is typically a strict time limit, because of the need for altering parameters of the simulation or the positioning of the instrument. In obtaining the detailed information, there is flexibility with respect to parameters such as the *error tolerance* (which is related to image resolution), the *image size*, and also the *number of angles* at which the new projections are done. The benefit function for each application has been described in our earlier work [38].

5.3 Model Validation

In this section, we validate the resource models we use. Specifically, we want to show that our model can effectively assign CPU cycles and memory to service components, in accordance to the parameter adaptation, so as to keep the resource costs low. Furthermore, we show that the model can be trained on one server, and then used on a server with a different type of hardware effectively. This is an important requirement in cloud computing, as a service provider may use different types of hardware, with a user not having any control on what is made available to them.

We first use the `GLFS` application to validate our model. Particularly, two service components, i.e., *POM 3D Model* service (`S1`) and *Grid Resolution* service (`S2`) are chosen. Recall that the first service has the *external time step* and *internal time step* as the two adaptive parameters. We triggered a 1 hour event to predict the meteorological information for an area of 200 square miles in Lake Erie. Throughout our experiments, the benefit and resource cost are reported for fixed size area, unless explicitly stated otherwise. The CPU and memory usage was measured for both services that are hosted on one single server, while supporting parameter adaptation at the time interval of 2 minutes. The results of CPU usage/allocation comparison are shown in Fig. 3a. In the charts, the legend `Our Approach *` refers to our approach without using

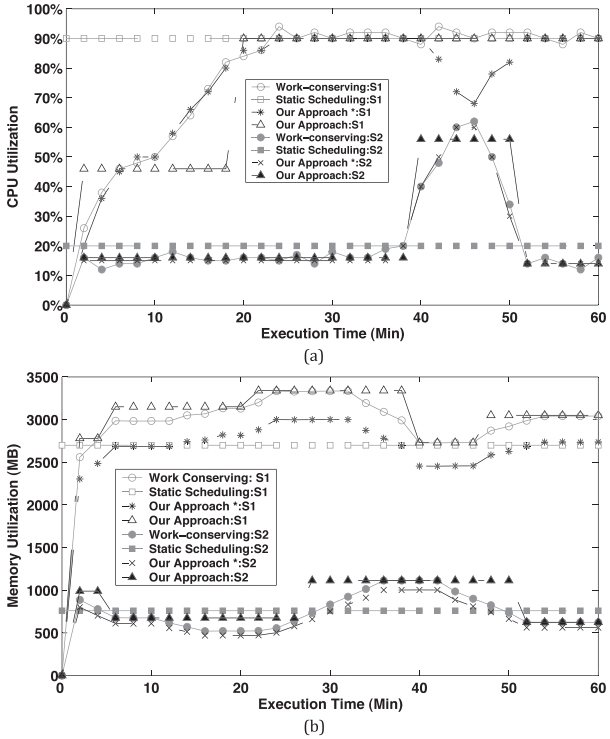

Fig. 3. CPU/memory usage and allocation comparison: *POM 3D Model* Service(S1) and *Grid Resolution* Service (S2) from `GLFS`: (a) CPU. (b) Memory.

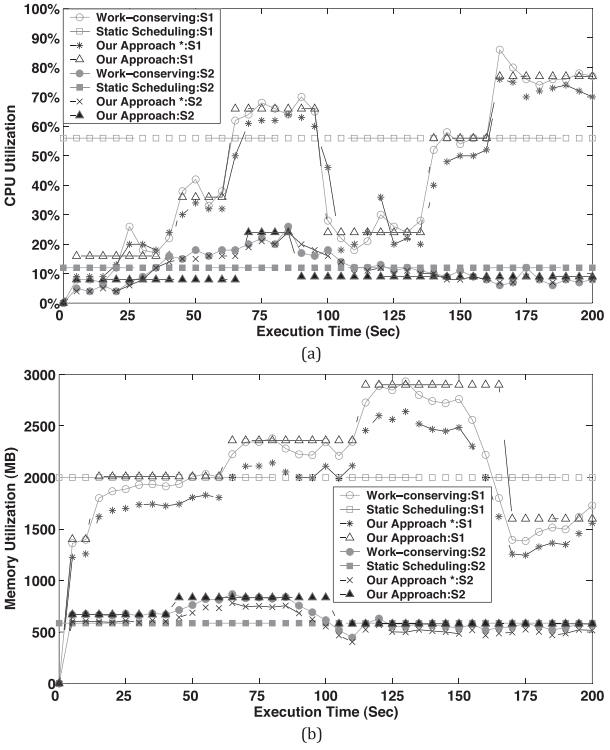

Fig. 4. CPU/memory usage and allocation comparison: *Unit image rendering* service (S1) and *temporal tree construction* service (S2) from `VolumeRendering`: (a) CPU. (b) Memory.

the model optimizer, whereas this optimization is enabled in the case of `Our Approach`. We compared the CPU cycle assignment generated from our model with the `Static Scheduling` and the `Work-conserving` approaches. The former statically assigns a certain amount of CPU cycles to the two services, whereas, the latter allows us to measure the actual usage of the two services. We observe that the CPU cycle requests generated from our proposed model are close to those from the `Work-conserving` approach, or the actual CPU usage, for both services. Specifically, the difference is less than 3 percent for the *POM 3D Model* service and 1 percent for the *Grid Resolution* service. These results indicate that the resource model we presented in Section 3.4 is effective in mapping the service parameter adaptation to CPU cycle requests. The `Static Scheduling` fixed the CPU assignment to be 90 and 20 percent for the two services, respectively. As we can see, such assignment results in underutilization or overutilization during the entire execution. Also, the `Work-conserving` is not suitable for other applications or services that may want to share the same CPU, while resource usage is being changed frequently. This becomes more severe if the time-limit is shorter. In comparison, `Our Approach` first accumulates certain CPU requests before actually changing the current CPU allocation. By doing this, we are able to reduce the number of CPU allocation changes to 2, while `Work Conserving` makes 30 changes. This is an important factor in saving resource costs with our pricing models, as a transfer fee is charged each time a reallocation is performed. More broadly, reducing frequency of resource reallocations is important for stability of a cloud environment. It is also possible that resource reallocation may take a significant amount of time in a real cloud environment, and thus a scheme involving frequent reallocations may incur large delays.

One interesting observation made from Fig. 3a is that, the CPU usage of these two services peak around the 48th time-step. In order to eliminate the bottleneck in the application, CPU allocation for the *POM 3D Model* service was reduced to give more share to the *Grid Resolution* service, as suggested by our model optimizer. However, since our approach decides the CPU allocation based on the service priority, CPU requests from the S1 should be satisfied first, as compared to those from S2. This helps us achieve better application benefit, as we will demonstrate it in the next section.

We compare memory usage and allocation next. The results are demonstrated in Fig. 3b. Two observations can be made from the figure. First, the actual memory usage always stays below 90 percent of the allocated memory to each VM, which is generated from our resource model, in spite of the varying memory demand from both services due to parameter adaptation. Second, we compared the ideal memory size (i.e., actual memory usage divided by 0.9) and the prediction from `Our Approach`. The difference is below 0.5 percent. Furthermore, by applying the model optimizer, we are able to reduce the frequency of repartitioning the memory to one sixth of that of the `Work-Conserving` scheme.

We repeated the experiment using two services, again denoted as `S1` and `S2`, from the `Volume Rendering` application. The results are shown in Figs. 4a and 4b. Similar observations can be made. The prediction accuracy in terms of CPU is 5 percent and 1 percent for S1 and S2, respectively.

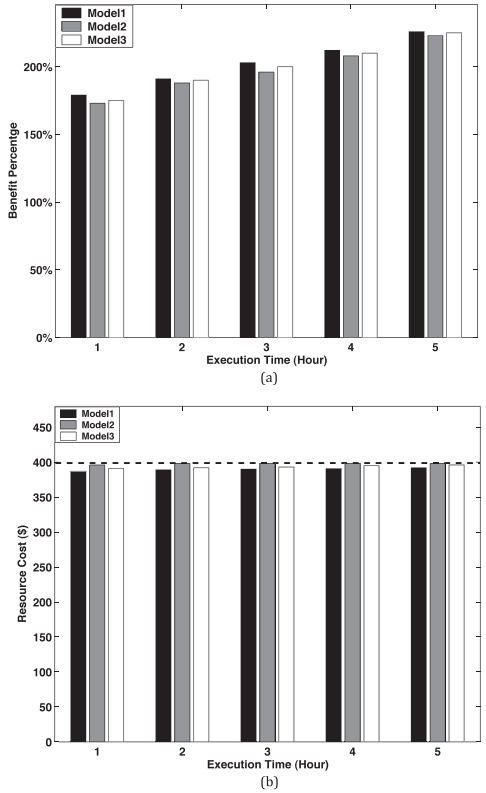

Fig. 5. Performance of GLFS with model trained on homogeneous hardware (M1) and heterogeneous hardware (M2 and M3). (a) Benefit percentage. (b) Resource cost.

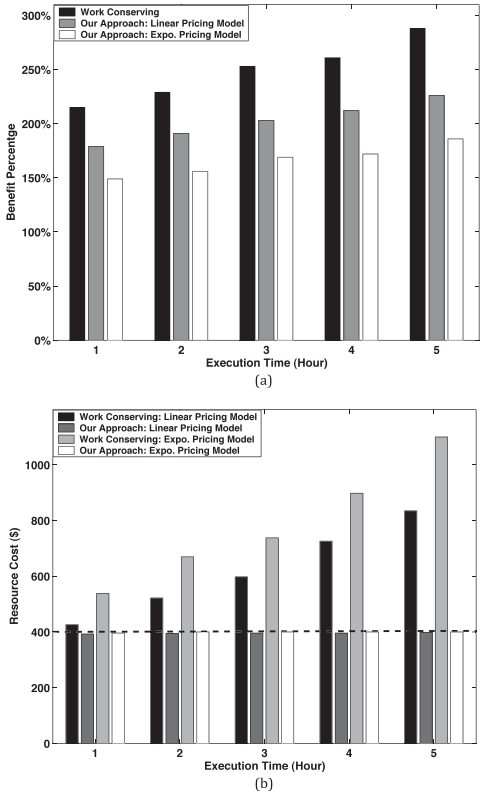

Fig. 6. Comparing different approaches for GLFS using two pricing models: (a) Benefit percentage. (b) Resource cost.

The memory prediction error is below 0.05 percent. Furthermore, by applying the model optimizer, we are able to reduce the CPU and memory reallocation frequencies from 40 to 6 and 5, respectively, compared to Work Conserving.

Next, we demonstrate that the actual hardware type on which we train our model does not restrict the effectiveness of the model, i.e., the model can still be used on a different type of server. We used three different servers, which were Intel Xeon quad-core CPU-2.33 GHz, AMD Opteron model 252-2.6 GHz, and Intel Xeon dual core CPU-1.6 GHz. We trained and obtained models on these three machines independently, and we refer to these three models as Model1, Model2, and Model3, respectively. We conducted this set of experiments using the first machine, i.e., Intel Xeon quad-core CPU-2.33 GHz. Our goal was to compare Model2 and Model3 against the Model1, to see how effective a model trained on a different type of hardware is.

Events with 1, 2, 3, 4, and 5 hours as time constraint were triggered from the GLFS application. We applied the linear pricing model presented in Section 2. The resource budget can be used to process this event is $400. The obtained benefit percentage and resource costs using the three models are shown in Figs. 5a and 5b, respectively. As we can see, although Model1 achieved slightly better benefit with less resource costs. The average difference of benefit percentage among three models is 1.2 percent and their corresponding resource cost difference is less than 3 percent. This demonstrates that models trained on a different hardware are still very effective.

5.4 Performance of the Dynamic Resource Provisioning Algorithm

In this section, we evaluate the performance of our approach against Static Scheduling and Work-conserving approaches with respect to the five metrics, i.e., benefit percentage and resource cost from the application point of view, as well as number of satisfied users, revenue, and average benefit-cost ratio from the provider's perspective. We present results obtained with the use of both the linear pricing model and the exponential pricing model, which were presented earlier in Section 2. Our goal is to demonstrate that regardless of the pricing model used, our proposed dynamic resource provisioning algorithm is able to optimize the benefit, while meeting the time constraint and the resource budget. Furthermore, the proposed algorithm can also be applied in nonadaptive applications. We demonstrate the effectiveness of our approach toward the end of this section. The overhead of different approaches is compared in the next section.

Adaptive applications: First, we show how the GLFS application could benefit from our dynamic resource provisioning. We fix the resource budget as $400. During the processing, we invoked multiple time-critical events, with the time constraints being 1, 2, 3, 4, and 5 hours, respectively. For each event, we executed 10 runs and report the average values. Benefit percentage, calculated as the ratio of obtained application benefit over the baseline benefit, is shown in Fig. 6a. Our approach can always perform better than Static Scheduling. Recall that Work-conserving is a scheme that favors benefit maximization but is unrealistic for a cloud environment. Every step in parameter adaptation leads to a CPU and/or

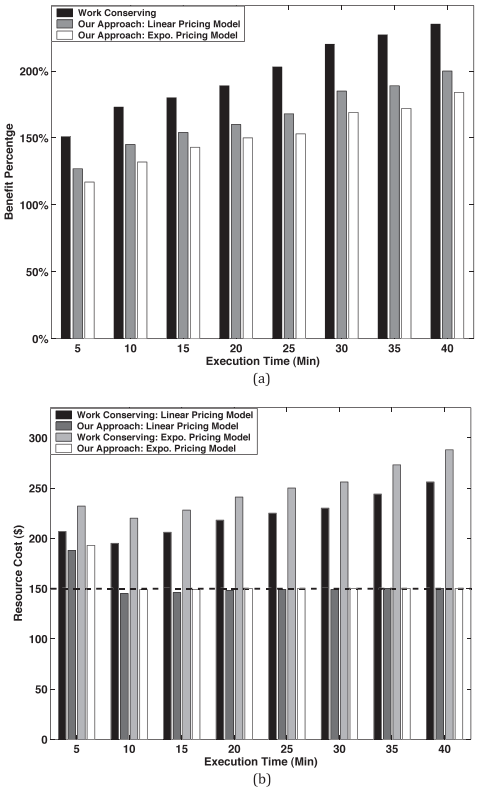

Fig. 7. Comparing different approaches for `Volume Rendering` using two pricing models: (a) Benefit percentage. (b) Resource cost.

memory resource request with this scheme. Since each resource request is always granted, the best benefit percentage was achieved. By using the linear pricing model, our approach is 24 percent worse, as compared to `Work-conserving`. However, the benefit achieved by `Work-conserving` comes with a significant resource cost, as shown in Fig. 6b. It costs 66 percent higher on the average. The following factors contribute to such a high resource cost. First, repartitioning memory is expensive, especially if it is done for each step in parameter adaptation. Second, even though CPU change overhead is small in our models, it can add up to be a significant factor in cost for a longer execution. Finally, service prioritization is not enabled for `Work-conserving`. Thus, low priority services could request more resources for processing.

We also calculate the resource cost using the exponential pricing model. Similar trends are observed. In this model, a high CPU percentage or a large memory size costs much more, as compared to the linear model. As a result, we achieve less benefit than the linear case. However, we still managed to complete the task within the given $400 resource budget. In comparison, `Work-conserving` costs are even higher, as expected. While the achieved benefit is lower in this case, our approach can quantitatively estimate the tradeoff between the benefit and resource cost, and through such a feedback, a user can increase the budget if they are not satisfied with the current benefit.

We also used the `Volume Rendering` application for the evaluation. The resource budget for this application is $150. The results are shown in Fig. 7. Similar to the previous results, we observe that the benefit percentage achieved by our approach is 16 percent lower than what could be achieved by `Work-conserving` approach (under the linear pricing model). With respect to the resource costs, our approach incurs 40 percent lower costs than that of the `Work-conserving` approach.

TABLE 3
Comparing Different Approaches: Provider Metrics

Approach	Static	W.C. (linear)	Proposed (linear)	W.C. (expo.)	Proposed (expo.)
#User	2	10	20	6	20
$ Revenue	800	2200	4000	1100	4000
$\frac{Benefit\%}{cost}$	0.25	0.36	0.48	0.32	0.40

Now we evaluate the three metrics from the provider's point of view. Given a fixed number of resources, *number of satisfied users* refers to the number of cases where the application is able to achieve the baseline benefit within the time constraint using the prespecified resource budget. *Revenue* is the amount of money collected from satisfied users. While *average benefit-cost ratio* is defined as the ratio of relative benefit and resource cost charged for successfully executed applications. This metric evaluates whether the proposed solution is able to maximize the application benefit given a fixed amount of resource budget.

The results from `GLFS` are presented in Table 3. We assume an environment where users submit jobs with time and cost constraints. If the time constraint is not met, they do not pay anything (i.e., the revenue of the cloud provider is 0). Moreover, the user is assumed to be dissatisfied, and is not likely to return the provider. 20 events from this application were submitted to the system, each with different time constraints and resource budgets. As we can see, our approach was able to successfully handle all submitted events. While `Static Scheduling` only accommodated 2 out of 20, and `Work Conserving` (`W.C.`) hosted less than 50 percent of submitted events. This is due to the over/under provisioning issues, i.e., either resources are wasted to prevent other events or there were not enough resources to support the current event processing. Accordingly, the revenue achieved from these two approaches is much less than that of our proposed solution. Finally, the benefit gain given per monetary unit is compared. Our approach outperformed the static case by 92 percent and 60 percent, and 33 percent and 25 percent for the linear and exponential pricing model, respectively. We have observed similar trends from the `Volume Rendering` application and the results are not reported here.

Nonadaptive cases: To evaluate how the proposed algorithm performs when applied to nonadaptive applications, we conducted the experiment using both `Volume Rendering` and `GLFS` applications. Instead of exposing the adaptive parameters, the values of such parameters are fixed at different levels for variant cases. To be specific, each case is associated with prespecified baseline benefit and time constraint, and the adaptive parameters remain at certain values throughout the execution. We compared our proposed solution against `Static Scheduling`, where the baseline benefit is always achieved.

For each of the applications, we considered 300 cases, i.e., the adaptive parameters are set at 300 different combinations

of values. An application execution fails when the baseline benefit is achieved but using excessive time and/or resource budget. The number of failures without using our proposed solution is 116 and 82 (or 38.67 percent and 27.33 percent) for `GLFS` and `Volume Rendering`, respectively. When our algorithm is enabled, such numbers both dropped to 0. In other words, the dynamic resource provisioning algorithm proposed in this paper works effectively in nonadaptive situations as well. The reason is that although we do not change the parameter values during the application execution, such values are *virtually* adjusted to trigger resource reallocation. By doing so, our proposed solution guarantees the baseline benefit using the prespecified resource budget within the prespecified time frame.

5.5 Overhead of Dynamic Scheduling

We have already demonstrated that our approach is much better than a static approach we compared against. Still, one of the potential problems with a dynamic approach is a high runtime overhead. We now show that this not the case with our approach, i.e., the overhead caused by our dynamic resource provisioning is quite small.

For demonstrating this, we compared two different executions as follows. In the first case, we started from random initial values for adaptive parameters and applied our model and the dynamic resource provisioning algorithm. This version is referred to as the `Adaptive Execution`. In the other case, the parameters were set to the ideal or converged values we obtained from the first case. Similarly, both the CPU cycle and memory allocation is set according to the ideal resource configuration from the first case. We refer to it as the `Optimal Execution`. This is clearly unrealistic, as such parameter values or the ideal CPU/memory allocation configuration cannot be known in advance. Thus, this version is only used as a baseline to measure the overhead in the execution of our algorithm.

We first conducted this experiment with the `GLFS` application, and the results are shown in Fig. 8a. The overhead is around 4 percent, 2 percent, 2 percent, 1 percent, and 0.8 percent for the 5 cases, which correspond to time limits of between 1 and 5 hours. The main part of the overhead is because of the system calls to Xen hypervisor to change the maximum entitled memory (*maxmem*) for each VM, as well as the cap value for the Xen scheduler. Furthermore, the extra computation and communication with nonoptimal values for the adaptive parameters also create an overhead. But, overall, this overhead is quite small, and furthermore, the overhead percentage decreases as the time constraint increases. This is because the actual CPU cycle allocation requested by our approach does not change frequently, and with increasing execution time, its relative cost becomes smaller. This is also true with memory allocation.

This experiment was repeated with the `Volume Rendering` application. The results we observed are very similar. The overhead compared to the `Optimal Execution` is within 9 percent for all 8 cases that we have considered.

6 Related Work

We now discuss the relevant research efforts from the areas of middleware for cloud computing, virtualized resource scheduling, and scheduling with budget constraints.

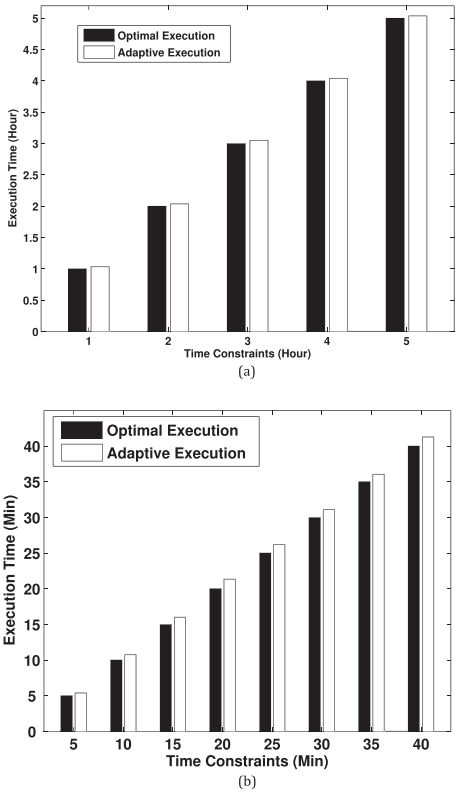

Fig. 8. Scheduling overhead (a) `GLFS`. (b) `Volume Rendering`.

Cloud computing systems. Cloud computing has received much attention recently in both academia and industry. Besides the efforts at popular service providers like, Amazon [2], Google [5], Microsoft [3], and many others, Cloud computing research testbeds and/or research projects have been initiated at Georgia Tech [4], HP/Intel/Yahoo! [8], CMU [1], UCSB [19], and U.C.Berkeley [33], The *Eucalyptus* middleware from UCSB [19] supports Infrastructure as a Service (IaaS), with the goal being to give users the ability to run and control the entire virtual machine instances. More recently, researchers in the scientific community are actively examining cloud computing for scientific applications, with Nimbus [7] and Magellan [6] two representatives efforts. In comparison, our focus has been on optimization of compute-intensive adaptive applications when there is a fixed resource budget. To the best of our knowledge, this has not been addressed by any of the existing projects. Overall, our work can be viewed as an optimization module that can be incorporated with any cloud infrastructures.

Virtualized resource scheduling. There has been much research on the topic of virtualized resource allocation [30], [42], [32], [37], [44], [10], [12], [34], [28]. Guitart et al. presented a comprehensive survey of the work in the literature [25]. To support multi-tier web application in virtualized environments, the resource management problem has been formulated as an optimization problem to achieve the maximum satisfaction of application SLA's [10], [12]. Our resource provisioning problem can also be solved using optimization techniques. However, given the potential large number of adaptive parameters, the optimization problem may become too complicated to solve, or even untrackable. Previous work has applied control theory in

virtualized resource management [28], [44], [37], [30]. Padala et al. considered both CPU and disk as resources for web applications, taking application throughput and response time as the metrics [37]. Xu et al. proposed multi-layer controller with fuzzy logic [28]. The key distinctive aspect of our work is to control the application adaptive parameters, instead of the virtualized resources, as the value of such parameters directly impact both the application benefit and execution time while the relationship between resource usage and those metrics are hard to model. Also, we considered the resource cost as another constraint, making the control formulation more challenging. Norris et al. considered handling workload spikes by allowing applications to trade computing capacity on the market using automated market policies [34]. By doing so, the system is able to handle temporary workload spikes for certain applications while maintaining service to the others. Handling dramatic changes in workloads is our future work. However, we target at adaptive applications which are more complex and different from web applications from previous work, given the data/control dependencies and adaptive parameters. Moreover, we used two real-world applications for our experiments.

Scheduling with budget constraints. In the context of grid and utility computing, many efforts have contributed to the problem of resource scheduling with a budget constraint [39], [29], [22], [40]. Yu et al. proposed a generic algorithm for scheduling workflows onto the utility grids, with the goal of minimizing the execution time, within a specific resource budget. Sakellariou et al. developed two scheduling approaches, *LOSS* and *GAIN*, to adjust schedules that are generated by time-optimized and cost-optimized heuristics, to meet users' budget constraints. Although the resource provisioning problem we consider in this work is also budget constrained, parameter adaptation with the goal of maximizing application benefit within a time deadline distinguishes our work. As opposed to static resource scheduling performed in most existing work, we considered two pricing models for the cloud resources and applied a dynamic approach where CPU cycle and memory assignments are changed with parameter adaptation at application runtime. Furthermore, we also consider the problem of resource contention among multiple virtual machines. Shared resources are assigned to VMs based on the priority of services running on the VM.

7 Conclusion

The realization of the vision of utility computing through the emergence of clouds is creating new resource provisioning problems. The work presented in this paper has been driven by the challenge of effectively supporting a class of *adaptive applications* on these systems. We have developed a feedback control-based approach for maximizing application QoS, while meeting both a time constraint and a resource budget limit.

We have evaluated our framework with two real adaptive applications. The main observations from our experiments are as follows. First, the CPU cycle/memory allocation generated through the use of our resource model is within 5 percent of the actual CPU/memory utilization. Furthermore, the model can be trained on one type of hardware and then applied on another type of hardware effectively. Second, our dynamic resource provisioning algorithm achieves a benefit of up to 200 percent of what is possible through a static provisioning scheme. At the same time, the scheme could perform parameter adaptation to meet a number of different time and budget constraints for the two applications. Finally, the overhead caused by the dynamic resource provisioning algorithm is less than 10 percent.

References

[1] Cloud Computing at SCS, http://www.cmu.edu/news/blog/2009/Spring/cloud-computing-at-scs.shtml, 2012.

[2] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2, 2012.

[3] Azure Services Platform, http://www.microsoft.com/azure/default.mspx, 2012.

[4] Cloud Computing Testbed, http://www.ajc.com/metro/content/business/stories/2008/03/25/autonomics_0326.html, 2012.

[5] Google App Engine, http://code.google.com/appengine, 2012.

[6] Magellan, http://www.lbl.gov/cs/Archive/news101409.html, 2012.

[7] Nimbus, http://www.nimbusproject.org, 2012.

[8] Open Cirrus: The HP/Intel/Yahoo! Open Cloud Computing Research Testbed, https://opencirrus.org, 2012.

[9] Open Nebula, http://www.opennebula.org, 2012.

[10] B. Abrahao, V. Almeida, J.A. J, A. Zhang, D. Beyer, and F. Safai, "Self-Adaptive SLA-Driven Capacity Management for Internet Services," *Proc. 10th IEEE/IFIP Network Operations and Management Symp. (NOMS '06)*, pp. 557-568, Apr. 2006.

[11] V. Adve, V.V. Lam, and B. Ensink, "Support for Adaptive Distributed Applications," *Proc. SIGPLAN Workshop Optimization of Middleware (OM) and Distributed Systems*, June 2001.

[12] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource Management in the Autonomic Service-Oriented Architecture," *Proc. Third Int'l Conf. Autonomic Computing (ICAC '06)*, pp. 84-92, June 2006.

[13] S. Bennett, *A History and Control Engineering 1930-1955.* Peter Peregrinus Ltd., 1993.

[14] V. Bhargavan, K.-W. Lee, S. Lu, S. Ha, J.R. Li, and D. Dwyer, "The Timely Adaptive Resource Management Architecture," *IEEE Personal Comm. Magazine*, vol. 5, no. 4, pp. 20-31, Aug. 1998.

[15] H.F. Chen, W.X. Zhang, and G.F. Jiang, "Experience Transfer for the Configuration Tuning in Large-Scale Computing Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 3, pp. 388-401, Mar. 2011.

[16] D. Chiu, A. Shetty, and G. Agrawal, "Elastic Cloud Caches for Accelerating Service-Oriented Computations," *Proc. 24th Int'l Conf. High Performance Computing and Networking (SC '10)*, Nov. 2010.

[17] S. Das, D. Agrawal, and A.E. Abbadi, "Elastras: An Elastic Transactional Data Store in the Cloud," *Proc. Workshop Hot Topics in Cloud (HotCloud)*, 2009.

[18] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO Control of an Apache Web Server: Modeling and Controller Design," *Proc. Am. Control Conf. (ACC '02)*, pp. 4922-4927, May 2002.

[19] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," *Proc. Cloud Computing and Its Applications*, Oct. 2008.

[20] D. Parkhill, *The Challenge of Computer Utility.* Addison-Wessley, 1966.

[21] R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering," *Proc. 15th Ann. Conf. Computer Graphics and Interactive Techniques*, pp. 65-74, 1988.

[22] S.K. Garg, R. Buyya, and H.J. Siegel, "Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-Off Management," *Proc. 32nd Australasian Computer Science Conf. (ACSC '09)*, pp. 139-147, Jan. 2009.

[23] A. Gounaris, N.W. Paton, A.A.A. Fernandes, and R. Sakellariou, "Adaptive Query Processing: A Survey," *Proc. 19th British Nat'l Conf. Databases: Advances in Databases (BNCOD)*, 2002.

[24] G. Singh, C. Kesselman, and E. Deelman, "A Provisioning Model and Its Comparison with Best-Effort for Performance-Cost Optimization in Grids," *Proc. 16th IEEE Int'l Symp. High Performance Distributed Computing (HPDC '07),* June 2007.
[25] J. Guitart, J. Torres, and E. Ayguad, "A Survey on Performance Management for Internet Applications," *Concurrency and Computation: Practice and Experience,* vol. 22, pp. 68-106, 2010.
[26] J. Heo, X. Zhu, P. Padala, and Z. Wang, "Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management (IM '09),* pp. 630-637, June 2009.
[27] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data.* Prentice Hall, 1988.
[28] J. Xu, M. Zhao, J.B. Fortes, R. Carpenter, and M. Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management," *Proc. Fourth Int'l Conf. Autonomic Computing (ICAC '07),* June 2007.
[29] J. Yu and R. Buyya, "A Budget Constrained Scheduling of Workflow Applications on Utility Grids Using Genetic Algorithms," *Proc. Workshop Workflows in Support of Large-Scale Science,* June 2006.
[30] Q. Li, Q. Hao, L. Xiao, and Z. Li, "An Integrated Approach to Automatic Management of Virtualized Resources in Cloud Environments," *The Computer J.,* vol. 54, pp. 905-919, 2011.
[31] H. Lim, S. Babu, and J. Chase, "Automated Control for Elastic Storage," *Proc. Int'l Conf. Autonomic Computing (ICAC),* June 2010.
[32] H.C. Lim, S. Babu, J.S. Chase, and S.S. Parekh, "Automated Control in Cloud Computing: Challenges and Opportunities," *Proc. First Workshop Automated Control for Datacenters and Clouds (ACDC '09),* pp. 13-18, June 2009.
[33] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Technical Report TR-2007-169, Dept. of Electrical Eng. and Computer Science, Univ. of California, Berkeley, Feb. 2009.
[34] J. Norris, K. Coleman, A. Fox, and G. Candea, "OnCall: Defeating Spikes with a Free-Market Application Cluster," *Proc. First Int'l Conf. Autonomic Computing (ICAC '04),* pp. 198-205, May 2004.
[35] 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," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* pp. 64-177, 2003.
[36] C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative Run-Time Management of Adaptive Applications and Distributed Resources," *Proc. 10th ACM Int'l Conf. Multimedia,* 2002.
[37] 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," *Proc. Fourth ACM SIGOPS/EuroSys European Conf. Computer Systems (Eurosys '09),* pp. 13-26, 2009.
[38] Q. Zhu and G. Agrawal, "An Adaptive Middleware for Supporting Time-Critical Event Response," *Proc. Fifth Int'l Conf. Autonomic Computing (ICAC '08),* June 2008.
[39] G. Reig, J. Alonso, and J. Guitart, "Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud," *Proc. IEEE Ninth Int'l Symp. Network Computing and Appl. (NCA '10),* pp. 162-167, July 2010.
[40] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M.D. Dikaiakos, "Scheduling Workflows with Budget Constraints," *Integrated Research in GRID Computing,* 2007.
[41] S. Agrawal, Y. Dashora, M.K. Tiwari, and Y. Son, "Interactive Particle Swarm: A Pareto-Adaptive Metaheuristic to Multiobjective Optimization," *IEEE Trans. System, Man and Cybernetics,* vol. 38, no. 2, pp. 258-277, Mar. 2008.
[42] A. Sangpetch, A. Turner, and H. Kim, "How to Tame Your Vms: An Automated Control System for Virtualized Services," *Proc. 24th Large Installation System Administration Conf.,* Nov. 2010.
[43] A.J. Smola and B. Schoolkopf, "A Tutorial on Support Vector Regression," *J. Statistics and Computing,* vol. 14, no. 3, pp. 199-222, 2004.
[44] S.M. Park and M. Humphrey, "Feedback-Controlled Resource Sharing for Predictable Escience," *Proc. 22nd Int'l Conf. High Performance Computing and Networking (SC '08),* Nov. 2008.
[45] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," *Proc. Symp. Operating Systems Design and Implementation (OSDI),* Dec. 1999.
[46] J.G. Ziegler and N.B. Nichols, "Optimum Settings for Automatic Controllers," *Trans. ASME,* vol. 64, pp. 759-768, 1942.

Qian Zhu received the PhD and MS degrees from The Ohio State University in 2010 and 2008, respectively, and the BE degree from the Beijing Information Technology Institute, China, in 2004, all majoring in computer science and engineering. She is a researcher at Accenture Technology Labs in San Jose, California. Her research interests include cloud/grid computing, autonomic computing, and adaptive middleware, with a focus on applying machine learning techniques for autonomic system management in distributed environments. She is a student member of the IEEE.

Gagan Agrawal received the BS degree from IIT Kanpur in 1991 and the MS and PhD degrees from the University of Maryland, College Park, in 1994 and 1996, respectively. He is a professor of computer science at the Ohio State University. His research interests include parallel and distributed computing, compiler and runtime systems, and data mining/integration. He has published more than 175 papers on these topics. He is a senior member of the IEEE.