Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments

Qian Zhu
Department of Computer Science and Engineering
Ohio State University
Columbus, OH, 43210
zhuq@cse.ohio-state.edu

Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University
Columbus, OH, 43210
agrawal@cse.ohio-state.edu

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 carefully, keeping 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 feedback control based dynamic resource provisioning algorithm. 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.

1. INTRODUCTION

Utility computing was a vision stated more than 40 years ago [24]. 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, resource provisioning should be performed carefully. The goal should be to keep the resource budget to a minimum, while meeting an application's needs. Current cloud service providers have taken some steps towards 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 towards the likelihood of supporting more fine-grained allocation and pricing of resources [19, 22]. 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. Resource cost is an important consideration while using cloud resources for scientific computing as well [12]. 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* [10, 11, 17, 26, 28]. 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, 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. Such adaptive applications are clearly suitable for cloud computing. This is because a cloud environment can provide resources on-demand for computation, memory, and storage requirements posed by these applications, particularly when there is a time-constraint for completing a task. When such adaptive applications are executed in a cloud setting, there is a trade-off between resource costs incurred and the QoS obtained.

This paper focuses on an automated and dynamic resource allocation problem associated with the execution of adaptive applications in a cloud environment. We particularly consider the problem where there is a fixed *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 time-limit.

We present the design, implementation, and evaluation of a framework that can support adaptive applications in a cloud computing environment. The key component of the framework is a dynamic resource provisioning algorithm, which is based on control theory. We have considered dynamic CPU and memory allocation based on a con-

[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/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$10.00.

trol model. Furthermore, a *resource model* is proposed to map any given combination of values of adaptive parameters to resource requirements. 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 [14] 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 [19, 27, 13]. 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 [21, 16, 29]. 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 allocation generated through the use of our resource model is within 5% of the actual CPU 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% 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%, 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. The resource provisioning problem is defined in Section 3. In Section 4, we present the control theory based model and our dynamic resource provisioning algorithm. Section 5 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 6. We compare our work with related research efforts in Section 7 and conclude in Section 8.

2. MOTIVATING 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.* [11] focus on adapting frame rate over wireless networks. SWiFT is a software feedback toolkit to support program adaptation [33]. Adve *et al.* have focused on compiler support for adaptive applications [10]. 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 [26].

2.1 Great Lake Forecasting System (GLFS)

We now describe one such adaptive application in details. Particularly, we first show the Quality of Service (QoS) achieved can be quantified through a *benefit function*. Then, we explain why `GLFS` application is suitable for the cloud computing environment, but has an associated time constraint, and likely a budget constraint as well.

This application monitors meteorological conditions of the Lake Erie for nowcasting (for the next hour) and forecasting (for the next day). Every second, data comes into the system from the sensors planted along the coastal line, or from the satellites supervising this particular coastal district. 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.

`GLFS` is a compute-intensive application. On a standard PC (Dual Opteron 254 (2.4GHz) processors, with 8 GB of main memory and 500 GB local disk space), predicting meteorological information for the next two days for an area of 1 square miles took nearly 2 hours. The execution time could grow rapidly with need for increasing spatial coverage, (for example, the total area of Lake Erie is $9,940$ square miles), and/or better spatial granularity. While invoking these models for nowcasting, forecasting, or other requirements (such as sewage management), there is clearly a time constraint or time limit, since the results from a particular invocation are likely not useful beyond a certain time.

We believe that such an application is very well suited for Cloud computing, as CPUs, memory, and storage can be requested on demand to meet time constraints. Further, virtualization in the clouds can enable easy application deployment and execution. However, every additional resources comes with an associated cost. Local and state authorities operate under a difficult budget environment, and will like to impose a resource cost limit while executing these models. This is particularly true for information that only has a non-critical or recreational use, and there is no method for state and local authorities to recover the costs associated with execution of the models.

The resource and time constraints can be met by exploiting the adaptivity in the application. While it is desirable to run the models with high spatial and temporal granularity, clearly there is some flexibility in this regard. For example, such flexibility could be in the resolution of grids assigned to the model from a spatial view, or the internal and external time steps deciding the temporal granularity of model prediction. Furthermore, if computing resources available are limited, running new models in certain areas may be more critical than running those at other areas. Thus, we can formalize a *benefit function*, capturing a user's priority while executing this application. The benefit function clearly depends upon the specific user, but one example function can be as follows:

$$Ben_{GLFS} = (w \times R + N_w \times \frac{1}{4}R) \times \sum_{i=1}^{M} \frac{P(i)}{C(i)} \qquad (1)$$

$$w = \begin{cases} 1 & \text{if } \textit{water level} \text{ is predicted} \\ 0 & \text{otherwise} \end{cases}$$

Equation 1 specifies that the *water level* has to be predicted by the model within the time constraint, since it is the most important meteorological information. R is a constant value for reward if this criterion is satisfied. It also gives credits to other outputs, by stating that the number of outputs N_w has to be maximized. Besides outputting useful results, the user also wants the resources to be allocated to the models with high priority. This is captured by getting the ratio of the model priority $P(i)$ and its cost $C(i)$.

Given a benefit function, a user will like to maximize the value of the benefit function, operating within a fixed resource budget and a time constraint. We formulate this problem and present our framework in the subsequent sections.

3. TARGET ENVIRONMENT AND PROBLEM FORMULATION

We first describe the cloud environment we target. Next, we de-

scribe two pricing models that have been used for our current evaluation.

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 each 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 [19, 22]. 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 [25]. 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 *non-capped*. 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 non-capped (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 $1GB$ 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:

$$\begin{aligned} CPU_{cost}(m) &= \sum_i^m (CPU_{base} \times (1 - exp^{-0.01 R_i^{cpu}}) \times t_i) + \\ &\quad (m-1) \times CPU_{trans} \end{aligned}$$

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

$$\begin{aligned} Mem_{cost}(m) &= \sum_i^m (Mem_{base} \times (1 - exp^{-0.01 R_i^{mem}}) \times t_i) + \\ &\quad (m-1) \times Mem_{trans} \end{aligned}$$

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.

4. DYNAMIC RESOURCE PROVISIONING ALGORITHM

In this section, we present our dynamic resource provisioning algorithm. In context of the overall framework design, which is illustrated in Figure 2, 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, which is 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.

4.1 Problem Formulation

An adaptive application we target comprises a series of interacting service components, which we denote as $S_1, S_2, ..., S_n$. An application has a benefit function, denoted as B. This function takes certain application parameters as the input, and outputs a Quality of Service (QoS) indicator, called the *benefit*. An example benefit function was shown earlier in Equation 1. Note that such benefit function is application-specific and user-defined. Every processing of a time-critical event we perform is associated with a pre-specified 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. When the values of service parameters are modified, we may need to change the CPU/memory allocation as well, to maintain the desired rate of progress. We may also need to provide a certain level of QoS differentiation to give higher priority to more critical services.

Note that the problem we solve in this paper involves a fixed time limit as well as a budget limit. Within these constraints, our 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 *bi-objective* problem [18], where we try to achieve a trade-off 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 not explained here.

4.2 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 [22, 27, 32]. 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. 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 the 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.

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 towards the end of this section.

Variable	Description
C_0	Resource Budget
T_c	Time Constraint
$u_i(k)$	i^{th} Adaptive Service Parameter at time step k
$S(k)$	State at the time step k
$D(k)$	Action taken at the time step k
$W_0(S(k), D(k))$	Benefit value at the time step k
$W_1(S(k), D(k))$	Total time elapsed at the time step k
$W_2(S(k), D(k))$	Resource costs at the time step k
$Perf$	Observed Application Performance

Table 1: Main Terms Used in Formulation

4.3 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.

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, $\mathbf{Perf}(k) = < B(k)/T(k), B(k)/C(k) >$ 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. We assume, however, that the behavior of the system at any point in time can be approximated locally as a linear model [27]. Specifically, by plotting two performance terms in $\mathbf{Perf}$, we determined that both of them form a time-series, where 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,

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

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 [31], after the measurement for the performance $\mathbf{Perf}(k)$ is available. Alternative system models, like Neural Networks, could have also been be applied to model the non-linear time series.

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. 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 $D(k)$.

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), can both be learned online using the approach from our previous work [28].

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 subsection. 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 3.

$$\begin{aligned} W_2(S(k), D(k)) \;\; &= \;\; CPU\ request \times CPU_{cost}(k) + \\ &\quad\;\; mem\ request \times Mem_{cost}(k) \end{aligned}$$

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,

$$\begin{aligned} \sum_{k=1}^{N} W_1(S(k), D(k)) \;\; &\leq \;\; T_c \\ \text{and } W_2(S(k), D(k)) \;\; &\leq \;\; C_0 \end{aligned} \tag{4}$$

4.4 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 [27, 13], 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 [20]. 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 [31] to learn the relationship between the values of the adaptive 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 <CPU cycle, memory> and the value of adaptive parameter along the time axis. The final resource model is the product of models from step 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 adaptation parameter values. The reason is that, there is a cost associated with performing such a change. In addition, resource under-provisioning or over-provisioning 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

Algorithm 4.1: 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 ($bottleneck$)
$AdjustResourceallocation(S_c^j)$;

Figure 1: Model Optimization

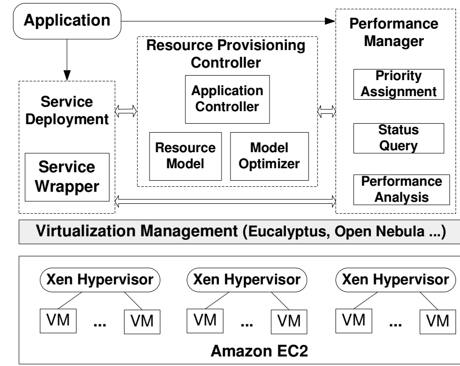


Figure 2: Overall Framework Design

be satisfied based on the *priority* of the service. At the same time, because services could be inter-dependent 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 Figure 1. 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 $mem_{threshold}$. 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, and 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.

5. 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.

5.1 System Architecture and Design

The overall system architecture is illustrated in Figure 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 4, 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 shown in Figure 3, and 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 [14] 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].

5.2 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.

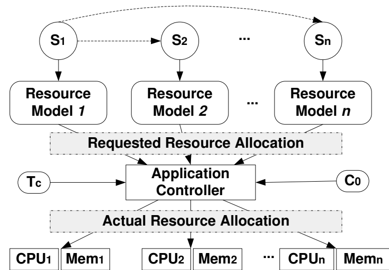


Figure 3: Detailed Architecture of Resource Provisioning Controller

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 module 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 4.4. 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.

6. 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.

6.1 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. Before the execution of the algorithm, we applied a bi-objective optimization algorithm [30] to decide the CPU cycle assignment and memory size, respectively, where the application benefit was maximized and resource cost was minimized. However, as 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. 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 3. 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% 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 pre-specified budget.

- Evaluate the overhead of our dynamic resource provisioning algorithm.

6.2 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 (2.4GHz) processors with 8MB 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. One 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 are 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

Application	Services		Dataset
`GLFS`	POM Model (2-D mode) Grid Resolution	POM Model (3-D mode) Linear Interpolation	21.0 GB
`Volume Rendering`	*Preprocessing* WSTP Tree Construction Temporal Tree Construction Compression	*Rendering* Unit Image Rendering Decompression Image Composition	7.5GB (30 time steps)

Table 2: Details of `GLFS` and `VolumeRendering`

earlier, we allow fine-grained CPU cycle and memory allocation in our cloud environment.

The experiments we report were conducted using two applications, `GLFS` and `Volume Rendering`. The first of these applications was described earlier in Section 2. `Volume Rendering` interactively creates a 2D projection of a large time-varying 3D data set (volume data) [15]. 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. A benefit function capturing this aspect has been described in our earlier work [28].

Some of the key details of both applications are listed in Table 2. The adaptable parameters for the first application are the *number of internal time steps*, *number of external time steps*, from POM Model service and *grid resolution*, from Grid Resolution service. Similarly, there are three adaptive parameters in the second application, *wavelet coefficient* from the Compression service, and both *error tolerance* and *image size* from the Unit Image Rendering service.

6.3 Model Validation

In this subsection, 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 Figure 4(a). In the charts, the legend `Our Approach *` refers to our approach without using 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

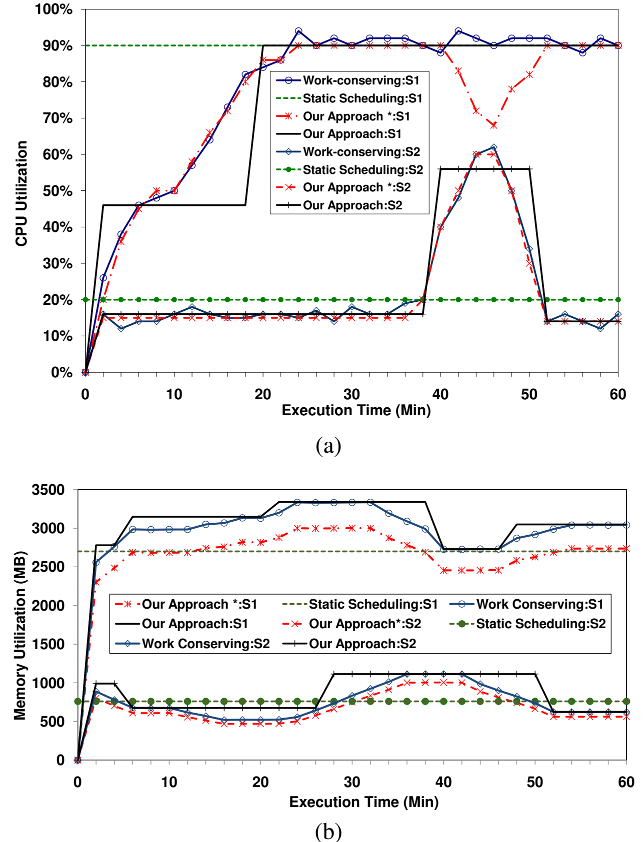
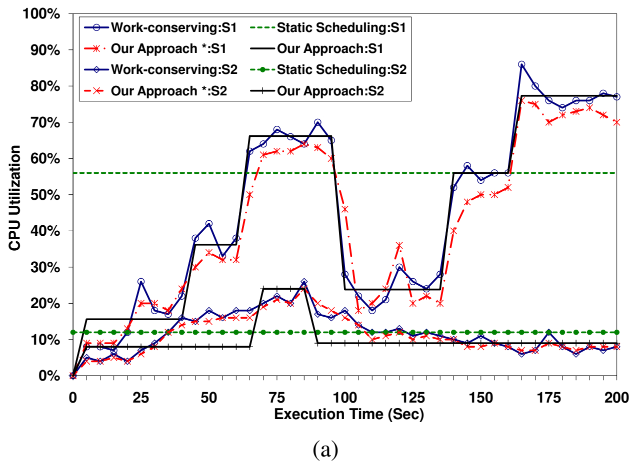
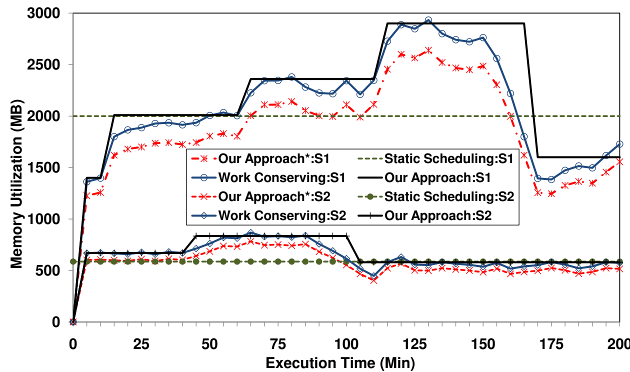


Figure 4: CPU/Memory Usage and Allocation Comparison: ***POM 3D Model*** **Service(S1) and** ***Grid Resolution*** **Service (S2) from `GLFS`: (a) CPU (b) Memory**

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% for the *POM 3D Model* service and 1% for the *Grid Resolution* service. These results indicate that the resource model we presented in Section 4.4 is effective in mapping the service parameter adaptation to CPU cycle requests. The `Static Scheduling` fixed the CPU assignment to be 90% and 20% for the two services, respectively. As we can see, such assignment results in under-utilization or over-utilization 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 trans-

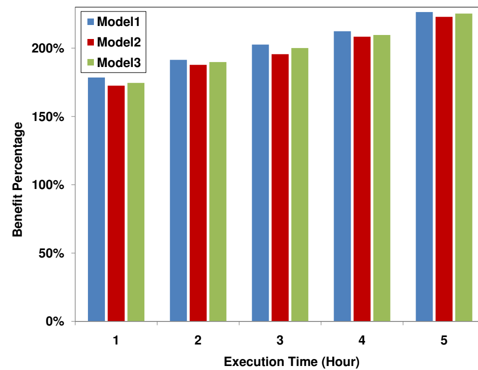


(a)

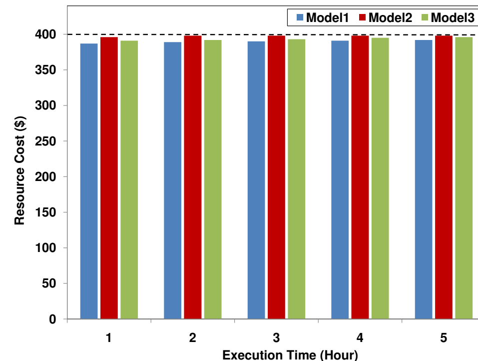


(b)

Figure 5: CPU/Memory Usage and Allocation Comparison: *Unit Image Rendering* **Service (S1) and** *Temporal Tree Construction* **Service (S2) from** `VolumeRendering`**: (a) CPU (b) Memory**



(a)



(b)

Figure 6: Performance of `GLFS` **with Model Trained on Homogeneous Hardware (M1) and Heterogeneous Hardware (M2 and M3): (a) Benefit Percentage (b) Resource Cost**

fer 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.

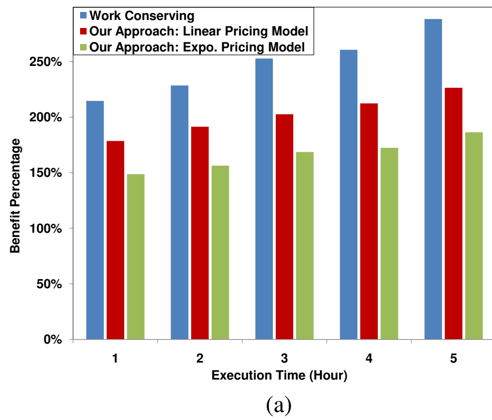
One interesting observation made from Figure 4(a) is that, the CPU usage of these two services peak around the 48th time-step. In order to eliminate the bottleneck in the application, CPU allocation for the *POM 3D Model* service was reduced to give more share to the *Grid Resolution* service, as suggested 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 subsection.

We compare memory usage and allocation next. The results are demonstrated in Figure 4(b). Two observations can be made from the figure. First, the actual memory usage always stays below 90% 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%. 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 Figure 5 (a) and (b). Similar observations can be made. The prediction accuracy in terms of CPU is 5% and 1% for S1 and S2, respectively. The memory prediction error is below 0.05%. 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.

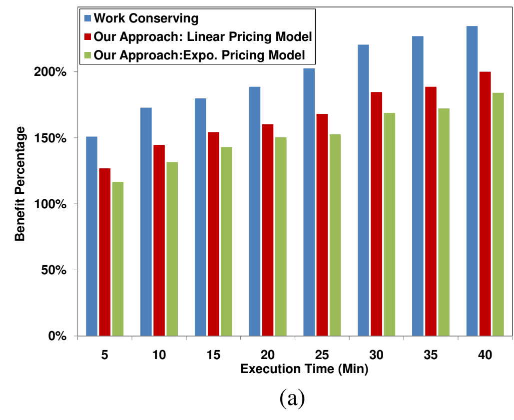
Events with 1, 2, 3, 4 and 5 hours as time constraint were triggered from the `GLFS` application. We applied the linear pricing model presented in Section 3. The resource budget can be used to process this event is $400. The obtained benefit percentage and resource costs using the three models are shown in Figure 6(a) and (b), 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% and their corresponding resource cost difference is less than 3%. This demonstrates that models trained on a different hardware are still very effective.



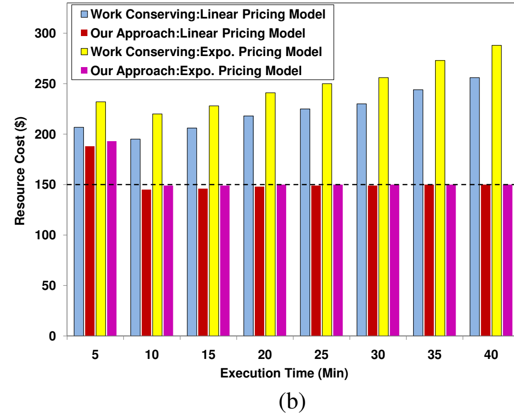
(a)

(b)

Figure 7: Comparing Different Approaches for `GLFS` using Two Pricing Models: (a) Benefit Percentage (b) Resource Cost



(a)



(b)

Figure 8: Comparing Different Approaches for `Volume Rendering` using Two Pricing Models: (a) Benefit Percentage (b) Resource Cost

6.4 Performance of the Dynamic Resource Provisioning Algorithm

In this subsection, we evaluate the performance of our approach against `Static Scheduling` and `Work-conserving` approaches with respect to the two metrics, i.e., benefit percentage and resource cost. We present results obtained with the use of both the linear pricing model and the exponential pricing model, which were presented earlier in Section 3. 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. The overhead of different approaches is compared in the next subsection.

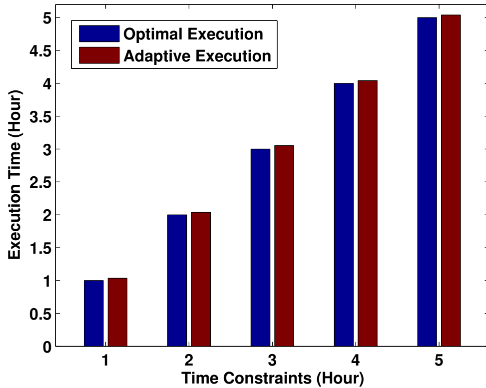
First, we show how the `GLFS` application could benefit from our dynamic resource provisioning. We fix the resource budget as $400. During the processing, we invoked multiple time-critical events, with the time constraints being 1, 2, 3, 4 and 5 hours, respectively. For each event, we executed 10 runs and report the average values. Benefit percentage, calculated as the ratio of obtained application benefit over the baseline benefit, is shown in Figure 7(a). 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 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% worse, as compared to `Work-conserving`. However, the benefit achieved by `Work-conserving` comes with a significant resource cost, as shown in Figure 7(b). It costs 66% 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.

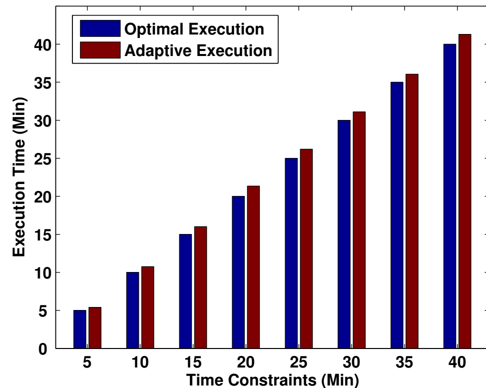
We also used the `Volume Rendering` application for the evaluation. The resource budget for this application is $150. The results are shown in Figure 8. Similar to the previous results, we observe that the benefit percentage achieved by our approach is 16% lower than what could be achieved by `Work-conserving` approach (under the the linear pricing model). With respect to the resource costs, our approach incurs 40% lower costs than that of the `Work-conserving` approach.

6.5 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



(a)



(b)

Figure 9: Scheduling Overhead (a) `GLFS` **(b)** `Volume Rendering`

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 Figure 9(a). The overhead is around 4%, 2%, 2%, 1%, and 0.8% 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 non-optimal 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% for all 8 cases that we have considered.

7. RELATED WORK

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

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 [14], and U.C.Berkeley [23], The *Eucalyptus* middleware from UCSB [14] 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 [22, 19, 27, 32, 13]. Diao [13] *et al.* applied MIMO control to adjust two configuration parameters within Apache to regulate CPU and memory utilization of the Web server. However, they used static linear models. In our work, a dynamic approach has been taken, as static models cannot perform parameter adaptation for maintaining application processing progress. A similar control model was used by Padala [27] for virtualized resources management, targeting web applications. They considered both CPU and disk as resources, taking application throughput and response time as the metrics. The distinctive aspects of our work are adaptive parameter adaptation, a fixed resource budget, and a time-constraint, which make our control formulation more challenging. In addition, we develop resource models through offline learning.

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 [21, 16, 29]. 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 model 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.

8. 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 allocation generated through the use of our resource model is within 5% of the actual CPU 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% 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%.

9. REFERENCES

[1] In *http://www.cmu.edu/news/blog/2009/Spring/cloudcomputing-atscs.shtml*.

[2] Amazon elastic compute cloud. In *http://aws.amazon.com/ec2/*.

[3] Azure services platform. In *http://www.microsoft.com/azure/default.mspx*.

[4] Cloud computing testbed. In *http://www.ajc.com/metro/content/business/stories /2008/03/25/autonomics_0326.html*.

[5] Google app engine. In *http://code.google.com/appengine/*.

[6] Magellan. In *http://www.lbl.gov/cs/Archive/news101409.html*.

[7] Nimbus. In *http://www.nimbusproject.org/*.

[8] Open cirrus: the hp/intel/yahoo! open cloud computing research testbed. In *https://opencirrus.org/*.

[9] Open nebula. In *http://www.opennebula.org/*.

[10] Vikram Adve, Vinh Vi Lam, and Brian Ensink. Language and Compiler Support for Adaptive Distributed Applications. In *Proceedings of the SIGPLAN workshop on Optimization of Middleware (OM) and Distributed Systems*, June 2001.

[11] V. Bhargavan, K.-W. Lee, S. Lu, S. Ha, J. R. Li, and D. Dwyer. The TIMELY Adaptive Resource Management Architecture. *IEEE Personal Communications Magazine*, 5(4), August 1998.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 22nd International Conference on High Performance Computing and Networking (SC08)*, pages 1–12, Nov. 2008.

[13] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury. Mimo control of an apache web server: Modeling and controller design. In *Proceedings of the 2002 American Control Conference (ACC02)*, pages 4922–4927, May 2002.

[14] D.Nurmi, R.Wolski, C.Grzegorczyk, G.Obertelli, S.Soman, L.Youseff, and D.Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of Cloud Computing and Its Applications*, Oct. 2008.

[15] R.A Drebin, L.Carpenter, and P.Hanrahan. Volume rendering. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 65–74, 1988.

[16] S.K. Garg, R. Buyya, and H.J. Siegel. Scheduling parallel applications on utility grids: Time and cost trade-off management. In *Proceedings of the 32nd Australasian Computer Science Conference (ACSC09)*, pages 139–147, Jan. 2009.

[17] Anastasios Gounaris, Norman W. Paton, Alvaro A.A. Fernandes, and Rizos Sakellariou. Adaptive query processing: A survey. In *In 19th BNCOD*, pages 11–25. Springer, 2002.

[18] G.Singh, C.Kesselman, and E.Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing(HPDC07)*, June 2007.

[19] J. Heo, X. Zhu, P. Padala, and Z. Wang. Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM09)*, pages 630–637, June 2009.

[20] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[21] J.Yu and R.Buyya. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Proceedings of the Workshop on Workflows in Support of Large-Scale Science (works06), in conjunction with HPDC06*, June 2006.

[22] H.C. Lim, S. Babu, J.S. Chase, and S.S. Parekh. Automated control in cloud computing: Challenges and opportunities. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC09)*, pages 13–18, June 2009.

[23] 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. In *Tech. Rep. TR-2007-169, Dept. of Electrical Engineering and Computer Science, U.C.Berkeley*, Feb. 2009.

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

[25] P.Barham, B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, R.Neugebauer, I.Pratt, and A.Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP03)*, pages 64–177, 2003.

[26] Christian Poellabauer, Hasan Abbasi, and Karsten Schwan. Cooperative run-time management of adaptive applications and distributed resources. In *Proceedings of the tenth ACM international conference on Multimedia*, 2002.

[27] 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 *Proceedings of the 4th ACM SIGOPS/EuroSys European Conference on Computer Systems (Eurosys09)*, pages 13–26, 2009.

[28] Q.Zhu and G.Agrawal. An adaptive middleware for supporting time-critical event response. In *Proceedings of the 5th International Conference on Autonomic Computing (ICAC08)*, June 2008.

[29] R.Sakellariou, H.Zhao, E.Tsiakkouri, and M.D.Dikaiakos. Scheduling workflows with budget constraints. *Integrated Research in GRID Computing*, 2007.

[30] S.Agrawal, Y.Dashora, M.K.Tiwari, and Y.Son. Interactive particle swarm: A pareto-adaptive metaheuristic to multiobjective optimization. *IEEE Transaction on System, Man and Cybernetics*, 38(2), 2008.

[31] A.J. Smola and B. Scholkopf. A tutorial on support vector regression. *Journal of Statistics and Computing*, 14, 2004.

[32] S.M.Park and M.Humphrey. Feedback-controlled resource sharing for predictable escience. In *Proceedings of the 22nd International Conference on High Performance Computing and Networking (SC08)*, Nov. 2008.

[33] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole:. A Feedback-driven Proportion Allocator for Real-Rate Scheduling. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. ACM Press, December 1999.