Economical and Robust Provisioning of N-Tier Cloud Workloads: A Multi-level Control Approach

Pengcheng Xiong [†1], Zhikui Wang [‡2], Simon Malkowski [†3],Qingyang Wang [†4], Deepal Jayasinghe [†5], Calton Pu [†6]
[†]*College of Computing, Georgia Institute of Technology*
266 Ferst Dr., Atlanta, GA 30332, USA
[1,3,4,5,6]{pxiong3,zmon,qywang,deepal,calton}@cc.gatech.edu

[‡]*Hewlett Packard Laboratories*
Palo Alto, CA 94304 USA
[2]zhikui.wang@hp.com

***Abstract*—Resource provisioning for N-tier web applications in Clouds is non-trivial due to at least two reasons. First, there is an inherent optimization conflict between cost of resources and Service Level Agreement (SLA) compliance. Second, the resource demands of the multiple tiers can be different from each other, and varying along with the time. Resources have to be allocated to multiple (virtual) containers to minimize the total amount of resources while meeting the end-to-end performance requirements for the application. In this paper we address these two challenges through the combination of the resource controllers on both application and container levels. On the application level, a decision maker (i.e., an adaptive feedback controller) determines the total budget of the resources that are required for the application to meet SLA requirements as the workload varies. On the container level, a second controller partitions the total resource budget among the components of the applications to optimize the application performance (i.e., to minimize the round trip time). We evaluated our method with three different workload models—open, closed, and semi-open—that were implemented in the RUBiS web application benchmark. Our evaluation indicates two major advantages of our method in comparison to previous approaches. First, fewer resources are provisioned to the applications to achieve the same performance. Second, our approach is robust enough to address various types of workloads with time-varying resource demand without reconfiguration.**

***Keywords*-N-tier web application; Cloud computing; service level agreement; adaptive control**

I. Introduction

Clouds describe a new supplement, consumption, and delivery model for IT web services by providing dynamically scalable and often virtualized resources as a service over the Internet [1]. The Cloud providers offer infrastructure resources and computing capabilities as commodities to the users. In this way, the Cloud computing paradigm provides a range of attractive features such as resource elasticity, cost efficiency, and ease of management. Moreover, it also compels the rethinking of economic relationships between the provider and the users based on cost and performance of the services.

Platform-as-a-Service (PaaS) is one of the most significant components of the Cloud computing paradigm. On one hand, the PaaS providers' revenues are determined through the delivery of client request service under Service-Level Agreements (SLAs). The actual revenue model depends on the chosen performance metrics (e.g., round trip time, availability, bandwidth, or throughput) and the statistics defined on them. On the other hand, the PaaS providers may rent their resources from Infrastructure-as-a-Service (IaaS) providers in order to offer their service. Computing resources are regarded as services and billed similarly to other commodities. [1] Hence, from a PaaS provider point of view, the profit is determined based on two factors: quality of service and resource expenditures.

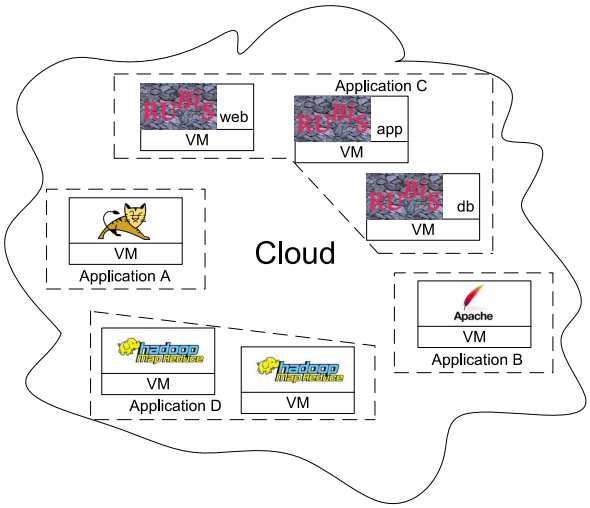

Figure 1. Applications in a typical Cloud environment

Consider the illustration in Figure 1 for example. Several client applications are deployed in the Cloud. Applications A and B are single-tiered. Application A only contains a

[1]For instance, the cost of a "small instance" with 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of local instance storage and 32-bit platform, on Amazon's Elastic Compute Cloud (EC2) [2] is $0.085 per hour.

Tomcat server and Application B only contains an Apache web server. Applications C and D are multi-tiered (e.g., a typical RUBiS benchmark application configuration includes at least three server tiers). As the multi-tier architecture shows the fastest adoption rate in the Cloud environment, we focus on economical and robust resource provisioning for N-tier Cloud workloads in this study.

For a single-tiered application, a feedback controller can be used to allocate resources based on the current system status and the time-varying workload in order to meet SLA [3]. For example, if SLA is a function of round trip time and resources are rented based on CPU time, then a feedback controller can be built to achieve minimal-cost rental (e.g., from an IaaS providers) of CPU resource while maintaining a sufficiently low round trip time level under the time-varying workload.

Compared with a single-tiered application, the decision to allocate resources is not straightforward for an N-tier application as each tier can have different levels of resource demand and contribute different amount from each other to the end-to-end response time. For instance, the application tier usually has much higher CPU demand than the other tiers, while the database tier can need much more memory and disk I/O bandwidth. If an inappropriate resource-partitioning scheme is adopted, e.g., to allocate too much CPU for those disk I/O intensive tiers, it will only waste the resource budget. Therefore, for a given total resource budget of an N-tier web application that can meet the end-to-end performance threshold, there exists degrees of freedom to partition the resource budget among the individual tiers. Well-designed partitioning scheme can achieve the best performance for the given total resource budget, or minimize the total resource cost without compromising the end-to-end performance.

We propose in this paper an SLA-based control method for both economical and robust resource provisioning for N-tier web applications.

First, we model an open N-tier web application as a tandem queue, which consists of several queuing systems in series. The Round Trip Time (RTT) for each tier is estimated through an M/G/1/PS queue. We show that under a given total CPU budget, the optimal partitioning which minimizes the RTT, can be calculated based on offline models for open workloads and online measurement. Such a partitioning scheme can be used by the PaaS provider to economically operate the service. We also test the optimal partitioning method against closed/semi-open N-tier web applications, which shows that the optimal partitioning scheme is robust enough w.r.t. the workload models. Although closed/semi-open N-tier web application cannot be modeled as M/G/1/PS queue, the optimal partitioning scheme still outperforms other approaches described in previous works, e.g., "Equal Shares" and "Equal Utilization".

Second, we propose a two-level control architecture for optimal resource allocation for N-tier web applications. In the application level, an adaptive feedback controller is applied to decide the total resource demand of the application in real time to maintain the RTT threshold upon varying workload. At the container level, an optimal controller partitions the total resource budget among the multiple tiers that can minimize the end-to-end response time. The two controllers work together to achieve the optimal resource allocation for the application.

Through experiments, we show that, the performance controller with the optimal partition scheme can achieve at least the same performance as a couple of other schemes, e.g., 'Equal Shares" and "Equal Utilization", while use up to 20% less resources. We also test the two level performance controller with both open and closed N-tier web applications to show the robustness of our approach.

The rest of the paper is organized as follows. Section II outlines our experimental setup and the system architecture. In Section III, we model the N-tier system as a tandem queue and propose an optimal resource partitioning method which is evaluated in Section IV. In Section V, we explore the relationship between the total resource and the mean round trip time and design an application controller. The performance controller which integrates the application controller and the resource partitioning method is evaluated in Section VI. Related works are summarized in Section VII before Section VIII which concludes the paper.

II. Background and system architecture

In this section, we first discuss the service level agreements (SLAs) that we use in the paper. Then we give a high-level description of the test bed and three types of workload generators for our experimental studies. Finally, we describe the control system architecture that we use throughout the paper.

A. Service Level Agreements

Service level agreements (SLAs) are contracts between a service provider and its clients. SLAs in general depend on certain chosen criteria, such as latency, reliability, availability, throughput and security. In this paper, we focus on end-to-end latency, or round trip time. Although SLA cost function may have various shapes, we believe that a staircase function is a natural choice used in the real-world contracts as it is easy to describe in natural language [4]. We use a single step function for SLA in our paper as a reasonable approximation. We assume that if the round trip time of the request is shorter than Ref (reference time), then the service provider will earn some revenue. Otherwise, the service provider will pay a penalty back to the client. As a result, in order to minimize the SLA penalty cost, the performance controller should keep the response time right below Ref.

B. Test Bed

We use RUBiS [5] as the benchmark application. It is an on-line auction benchmark comprised of a front-end Apache Web server, a Tomcat application server, and a back-end MySQL database server. There are 26 transaction types in RUBiS. The types of the next request generated by the workload generator are defined by a state transition matrix that specifies the probability to go from one transaction to another. In our experiments, we use "Browsing mix" workload that has 10 transaction types, e.g., Home, Browse, ViewItem. These transactions have different resource demands.

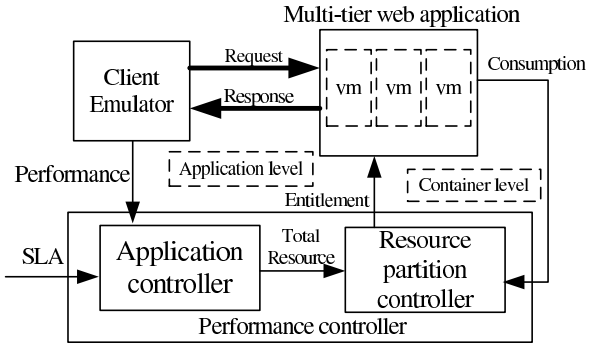

Figure 2. The architecture of our test bed.

We assume that each of the three tiers of the application is hosted in one Xen virtual machine [6]. Our test bed consists of three physical machines as shown in Figure 2. One of them is used for hosting the three VMs, one for the client emulator and the last one for running the controller. Each machine is an Intel Pentium 4 1.80GHz, 1 GB RAM PC with Gigabit Ethernet connected to the switch. All machines run Linux kernel 2.6.16.29. The hosting machine runs Xen 3.0.3. We use Apache v2.0.54 as the web server, Jakarta Tomcat v5.0.28 as the application server and MySQL v3.23.58 as the database server.

C. Closed, open and semi-open workload generators

A workload generator is needed to generate requests to emulate the behavior of clients. To evaluate the robustness of our approach, we create three types of workload generators that can represent different client behaviors [7].

A workload generator is called closed if new requests are triggered only after previous requests have been completed or timeout. The original RUBiS implementation is a standard closed-loop client emulator. The client generates new request(interaction) after it receives the response of the previous request and waits for an exponentially distributed "think time". Then each client has three possible statuses: (a) waiting in queue; (b) being served by server or (c) "thinking" for some amount of time. The action sequence of each session follows these steps: (a) to (b), (b) to (c) and (c) back to (a). Then the intensity of the workload depends on the number of the clients and the think time. The number of the clients is also called multiprogramming level (MPL). and the default think time between two requests is 3.5s. Therefore, different MPL represents different intensity of the workload, or request rate.

A workload generator is called open if new requests are generated independently of completion of previous requests. We modify the source code of original RUBiS workload generator to emulate open clients where the number of requests follows the Poisson distribution with a parameter of the arrival rate.

A workload generator is called semi-open if after a client receives a response for the previous requests, it will stay and make a follow up request with some probability p and will leave system with probability $1-p$. In the extreme cases with very small or large p, the semi-open workload generator resembles an open or a closed one respectively. We also modify the source code of original RUBiS workload generator to be a semi-open client emulator. The intensity of the workload is determined by the arrival rates as well as the probability p. In order to get a balance between closed and open models, we set the default probability $p = 0.5$ in our experiments.

It is worthwhile to note that, neither the open system model nor the closed system model is entirely realistic [7]. The client behavior in many N-tier web applications is best represented using an "in-between" system model, i.e., semi-open model. In the rest of the paper, we call the RUBiS web application using closed, open and semi-open workload generators as closed, open and semi-open RUBiS web applications respectively.

D. Control architecture for an N-tier web application

Table I
NOTATIONS

k	control interval for container level controller
K	control interval for application level controller
N	number of tiers (e.g., Web, App, DB)
Ω	number of transaction types (e.g., Browse, Bid)
T_{cpu}	average resident time on CPU resources
T_{others}	average resident time on non-CPU resources
λ_ω	average arrival rate of transactions type ω
λ	aggregate arrival rate of all transaction types
α_ω	average service time of non-CPU resources of transaction type ω
u_n	CPU entitlement that is allocated to the virtual server at tier n
c_n	CPU consumption of the virtual server at tier n
r_n	CPU utilization of the virtual server at tier n

To manage the resources for the application, we developed and implemented a performance controller as shown in Figure 2. In the application level, an application controller is used for end-to-end performance guarantee of the whole application through dynamic tuning of the total amount of the resources allocated to the application. The controller works in 90 seconds time interval. In the container level, there is one resource partition controller that is to allocate the total resource to the application tiers or the containers.

The controller works in 10 seconds time interval. We will use the notations in Table I in our paper. Specially, we use k and K to denote the control intervals of the container level and application level controllers respectively. We use "entitlement" (u) and "consumption or usage" (c) to refer to the CPU shares (in percentage of total CPU capacity) allocated to a virtual machine and the CPU share actually used by the virtual machine respectively. We use "utilization" (r) to refer to the ratio between consumption and entitlement, i.e, $r = \frac{c}{u}$.

III. Resource Partition Controller

For a given end-to-end performance target such as round trip time of request threshold, there is one optimal resource allocation to the tiers of an N-tier web application that can minimize the total resource allocation. On the other hand, given certain amount of resources available to the application, there exists also one optimal resource allocation to the tiers that can minimize the end-to-end performance, which is studied in this section based on queuing theory.

A. Modeling N-tier web application with open workload

1) N-tier web application: In our model, we consider a multi-tier application consisting of N tiers. We assume that each tier runs on a separate virtual machine. We consider a workload with Ω transaction types. If we define the intensity of the workload for the transaction type ω as λ_ω, then the intensity of the workload can be defined as a vector $(\lambda_1, ..., \lambda_\Omega)$. We also define the aggregate rate of the transaction as $\lambda = \sum_{\omega=1}^{\Omega} \lambda_\omega$.

2) End-to-end performance: The Round Trip Time (RTT) of an N-tier web application with open workload can be calculated by aggregating the resident times over all resources (e.g., CPU, disk) across all the tiers. The resident time on each tier is composed of two parts, i.e., the resident time on CPU resources and that on non-CPU resources. We assume that non-CPU resources are adequately provisioned and hence the effect of contention for these resources on the response time (i.e., the queuing delay) is negligible.

Since processor sharing (PS) approximates round-robin scheduling with small quantum size and negligible overhead, it is representative of scheduling policies in current commodity operating systems [8]. Moreover, a Poisson process is a good approximation of requests for open workload. We model CPU in each tier as an M/G/1/PS queue.

We use T_{cpu} to denote the total resident time on CPU across all the tiers. According to the queuing theory, for M/G/1/PS queue, the CPU resident time in the n-th tier is represented by $T_n = \frac{r_n}{\lambda(1-r_n)}$ where r_n is the CPU utilization of tier n. Note that CPU utilization in the above equation is the ratio between the virtual machine's CPU consumption and its effective CPU capacity, then we have

$$T_{cpu} = \sum_{n=1}^{N} T_n = \sum_{n=1}^{N} \frac{r_n}{\lambda(1-r_n)} = \sum_{n=1}^{N} \frac{c_n}{\lambda(u_n - c_n)}$$

We use T_{others} to denote the total resident time spent on all non-CPU resources. We use α_ω to represent service times of transaction type ω on all non-CPU resources of all tiers on the execution path of that transaction type. Then the mean resident time on non-CPU resources can be approximated by the weighted sum of each transaction type's service time.

$$T_{others} = \sum_{\omega=1}^{\Omega} \alpha_\omega \frac{\lambda_\omega}{\lambda}$$

Assume that there is an additive relationship between time spent in CPU and non-CPU resources, by combining T_{cpu} and T_{others}, we have

$$RTT = T_{cpu} + T_{others} = \frac{1}{\lambda}\left(\sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \sum_{\omega=1}^{\Omega} \alpha_\omega \lambda_\omega\right)$$

As stated in Section II, the types of the next request generated by the virtual clients are defined by a state transition matrix. Given a state transition matrix which describes the transition relationship among the 10 transaction types in "Browsing mix" of RUBiS, we can assume that the share of each transaction types is constant when the running time is long enough, i.e., $\frac{\lambda_\omega}{\lambda}$ is constant. For simplicity, we also assume that the average service time of non-CPU resources of each transaction type is constant since the effect of contention for these resources on the response time (i.e., the queuing delay) is negligible, i.e., α_ω is constant. If we can denote $\sum_{\omega=1}^{\Omega} \alpha_\omega \frac{\lambda_\omega}{\lambda} = \beta$ as a constant, then we have

$$RTT = T_{cpu} + T_{others} = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta$$

We can see that, the resource entitlement solution for an N-tier web application is not unique for given RTT target. Similarly, for given capacity available to the application, there is an optimal solution for the resource allocation to the multiple tiers that can minimize the RTT.

B. Optimal resource partition

Assume that the total CPU resource available for the application, or the total CPU shares that the N-tier web application provider rents, is fixed. Then we have an optimization problem defined as following.

$$\begin{aligned} \text{Minimize} \quad & RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \\ \text{s.t.} \quad & S = \sum_{n=1}^{N} u_n. \end{aligned}$$

There are N independent variables, i.e., u_n where $n = 1, ..., N$. The problem can be denoted as

$$\begin{aligned} \text{Minimize} \quad & f(u_1, u_2, ..., u_N) \\ \text{s.t.} \quad & g(u_1, u_2, ..., u_N) = 0 \end{aligned}$$

where

$$f(u_1, u_2, ..., u_N) = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta$$

and

$$g(u_1, u_2, ..., u_N) = \sum_{n=1}^{N} u_n - S = 0$$

Then we have the Lagrange function as

$$\begin{aligned}\Gamma(u_1, u_2, ..., u_N) &= f(u_1, u_2, ..., u_N) + \gamma g(u_1, u_2, ..., u_N) \\ &= \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta + \gamma(\sum_{n=1}^{N} u_n - S).\end{aligned}$$

where γ is a Lagrange multiplier and the partial derivative equations are

$$\frac{\partial \Gamma}{\partial u_n} = 0 = -\frac{1}{\lambda} \frac{c_n}{(u_n - c_n)^2} + \gamma = 0, n = 1, 2, ..., N$$

$$\frac{\partial \Gamma}{\partial \gamma} = 0 = \sum_{n=1}^{N} u_n - S$$

By solving the above equations, we can get the optimal solution

$$u_n = c_n + \frac{\sqrt{c_n}}{\sum_{n=1}^{N} \sqrt{c_n}} (S - \sum_{n=1}^{N} c_n)$$

Then the controller to implement the optimal solution in every control interval k is

$$u_n(k+1) = c_n(k) + \frac{\sqrt{c_n(k)}}{\sum_{n=1}^{N} \sqrt{c_n(k)}} (S(k) - \sum_{n=1}^{N} c_n(k))$$

From the solution we can see that, the optimal resource budget for each tier is composed of two parts: the first is equal to the actual resource consumption of that tier, and the second is a weighted share of the remaining budget (i.e., $S(k) - \sum_{n=1}^{N} c_n(k)$). It is worthwhile to note that, the optimal solution depends on the CPU consumption of the tiers in the last interval, but not on β, the effect of the non-CPU resources. Moreover, the optimal solution does not necessarily result in the equal utilization of the tiers if the resource consumptions are different from each other, as we will show in the next section.

IV. Evaluation of Resource Partition Controller

In this section, we evaluate the optimal partition controller through comparison with two other approaches: "Equal Utilization" and "Equal Shares".

A. Different resource partition schemes and experimental settings

Optimal : With the scheme "Optimal", the resource is allocated to the tiers according to optimal resource partition scheme as described in the previous section.

Equal Utilization : With the scheme "Equal Utilization", the resource is allocated to the tiers such that they have the same utilization, i.e., for $n = 1, 2, .., N$, r_n are the same.

Equal Shares : With the naive scheme of "Equal Shares", the resource is shared equally by all the tiers, i.e., for $n = 1, 2, .., N$, u_n are the same.

In our experiments, we fix the total CPU share for partition at 0.5 CPU. We then vary the workload rate from 15req/s to 50req/s. For each workload style, each workload rate and each partition scheme, we run one experiment for 900s. For open workload generator, we change the arrival rate between 15req/s and 50req/s. For closed one, we change the MPL between 52 and 175 such that the average request rate is varied between 15req/s and 50req/s [2] with the response time of the requests much less than the think time [9]. For semi-open one, we change the arrival rate between 7.5req/s and 25req/s. Since the default probability $p = 0.5$, the average request rate is 15req/s to 50req/s [3] since the response times of the requests are much less than the think time [9]. Although the optimal solution is derived based on an open queueing model, it would be interesting to evaluate how well it works for closed or semi-open workloads.

B. Experimental results

We denote "Optimal", "Equal Utilization" and "Equal Shares" as "Opt", "Util" and "Shares", respectively. Figures 3-5 show the mean round trip time resulted from the experiments for the three approaches, for the three workloads, "closed", "open", or "semi-open", when the rate can vary between 15 and 50req/sec.

We have several observations. Firstly, the "Opt" approach outperforms the other two approaches. On average, with the same total CPU shares, our method "Opt" can achieve about 20% shorter round trip time than the other two methods. Note that, this better performance can be achieved under different workload intensity (from 15 req/s to 50 req/s) as well as under different workload type (open, close, semi-open). This proves the robustness of our optimal partition scheme. Secondly, as we can see, the performance of Semi-open is in between that of Open and Closed. This validates the previous results that different workload type will demonstrate different performance as reported in [7] and [9]. Finally, Figures 3-5 show the relationship between the response time and the workload, or equivalently the relationship between the response time and the resource

[2] When the response time of the requests are much less than the think time and the system is not saturated, the request rate can be simply calculated as $\frac{MPL}{ThinkTime}$. For example, given $MPL = 52$ and $ThinkTime = 3.5s$, the request rate is around 15req/s according to queueing theory.

[3] When the response time of the requests are much less than the think time and the system is not saturated, the request rate can be simply calculated as $\frac{\lambda}{1-p}$. For example, given $\lambda = 7.5$ and $p = 0.5$, the request rate is around 15req/s according to queueing theory.

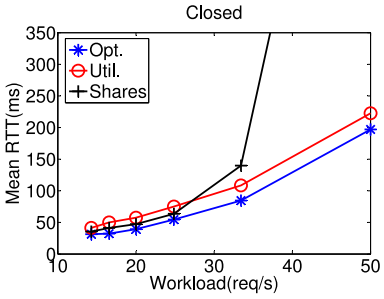

Figure 3. Mean RTT for Closed workload

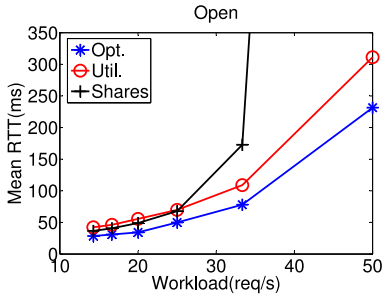

Figure 4. Mean RTT for Open workload

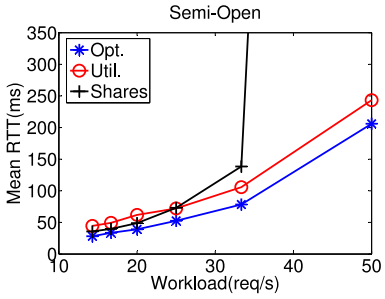

Figure 5. Mean RTT for Semi-Open workload

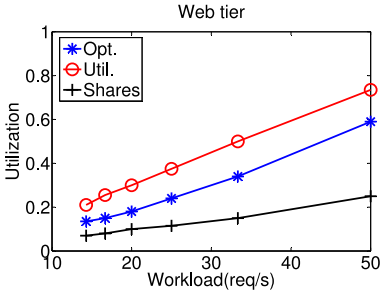

Figure 6. Web tier with Open workload

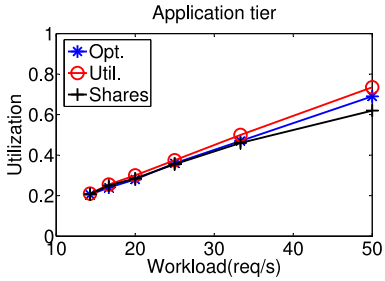

Figure 7. Application tier with Open workload

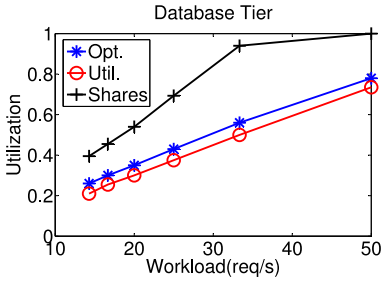

Figure 8. Database tier with Open workload

utilization as the total CPU allocation is the same, and the CPU consumption is almost the same with the different workload models. The response time is a nonlinear, but monotonically increasing function of CPU utilization. Note that the change of utilization can result from both workload variance and dynamic resource entitlement.

We zoom in the application with open workload generator. Figures 6-8 show the resource utilization levels of the three tiers of the application with open workload generator, when the resource is under control of the three approaches respectively. We also can have several observations. (1)As we expected, the utilization of the three tiers for the "Util" partition scheme is always the same when the workload varies. This is because the "Util" partition scheme tries to keep the same utilization level for all the three tiers. (2) When the application is under control of the "Opt" controller, the utilization of the web tier is overall lower than that resulted from "Util" controller, while that of the DB tier is overall higher. This implies that, compared with "Util" controller, relatively less CPU is allocated to the DB tier, while more CPU is provided to the Web tier. (3) The CPU resource is equally shared by the three tiers when the application is under control of "Shares" controller. It results at further lower utilization at Web tier, the higher utilization at the DB tier than those from the other two controllers. When workload is high, e.g., 50req/s, "Equal Shares" has very long round trip time, because of the extremely high utilization of the DB tier as shown in Figure 8.

C. Discussion

The solution that is proposed in the previous two sections can be generally used to address the optimal partition problem for the N-tier web application provider. It is worthwhile to note that the optimal solution does not depend (directly) on the workload parameters such as the workload intensity, the workload transaction types, and the service times that are usually challenging to be derived or measured. Instead, it depends on the CPU consumption of the individual tiers, which are readily available in standard systems with non-intrusive measurement. Moreover, the solution is independent of the service time on all non-CPU resources (with the general assumption that they are not bottleneck). In the future, we will use more complicated model, e.g., layered queuing model [10], to consider other system resources (e.g., disk and network).

From the experiments, although M/G/1/PS model is a good approximation for the average behavior of the open RUBiS web application, it is interesting to find that the "Opt" partition scheme still outperforms the other methods for closed and semi-open workload styles as well. As a piece of future work, we are to explore further on how well this approximation can be to improve the resource efficiency.

V. Application Controller Design

We describe how to design an application level feedback controller in this section. Note that, the aim of the controller is to keep the round trip time right below Ref in order to minimize the SLA penalty cost while minimizing the total resource allocated to the application. For the N-tier web application, this objective can be formalized as another constraint optimization problem, i.e., to minimize the total resource consumption with constraints on the reference response time Ref.

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{1}$$

$$\text{s.t.} \quad RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \leq Ref \tag{2}$$

However, there are at least three issues that make it challenging to solve the optimization problem directly. First, one more parameter β is to be identified. Second, there is always in-accuracy with the model. And third, the workload can vary along the time. In our approach, we solve the problem in two steps, or two layers as shown in Figure 2. In the first step, a feedback controller is applied to find the minimal total amount of CPU resources that can meet the end-to-end response time threshold. We call it the application controller. In the second step, the optimal resource partition controller, as described in previous section, is taken to allocate the "minimal total CPU resource" to the different tiers of the application.

A. System Identification

As stated in related works [11], the relationship between the resource entitlement and response time is nonlinear and depends on the variation of workloads. However, we can still assume that the relationship can be estimated by a linear function in the neighborhood of an operating point. We adopt autoregressive-moving-average (ARMA) model [12] to represent this relationship.

We run system identification experiment to determine the relationship between the mean RTT and the total CPU allocation to all the three tiers. In each experiment with workload rate w, the total CPU is randomly varied in [0.20CPU, 0.80CPU]. The mean RTT sampling interval is fixed at 90 seconds. Each experiment runs 100 intervals, i.e., 9000 seconds. The workload rate w varies from 10 to 25 requests per second. The experiment was repeated for each rate. We use the first 50 samples to train the model and then use the second 50 samples to evaluate the model.

We find that, if we take the mean RTT as output and CPU shares as input in the ARMA model, there is no good fit model. However, if we take the reverse of mean RTT as output and CPU share as input in the ARMA model, there exist good fit models. Assume the mean RTT at the K-th interval is $RTT(K)$, we define $y(K) = 1/RTT(K)$. We define the operating point of $y(K)$ as y_0 and the CPU share as S_0, define $\Delta y(K) = y(K) - y_0$, and $\Delta S(K) = S(K) - S_0$. We choose the following ARMA model to represent the dynamic relation between $\Delta y(K)$ and $\Delta S(K)$.

$$\Delta y(K) = \sum_{i=1}^{n} a_i \Delta y(K-i) + \sum_{j=1}^{m} b_j \Delta S(K-j).$$

where the parameters a_i, b_j, the orders n and m characterize the dynamic behavior of the system. For convenience, we refer to such a model as "ARXnm" in the following discussion. The model above was estimated offline using least-squares based methods in the Matlab System ID Toolbox [13] to fit the input-output data collected from the experiments. The models are evaluated using the r^2 metric defined in Matlab as a goodness-of-fit measure. In general, the r^2 value indicates the percentage of variation in the output captured by the model.

From the data in Table II, we can find that a simple model ARX01 can fit the input-output data well enough, although the ARX11 model has marginally better fitting numbers. This is reasonable, given that, the modeling and control interval is much longer than the queuing time, and the queuing process is the main resource of the dynamics in the system. Figure 9 demonstrates how the model works when rate is 20req/s. However, we also find that, the parameters of the models vary along the workload, which implies that a controller with fixed parameters may not work in the whole range of operation conditions.

Table II
r^2 VALUES FOR ARX MODELS

Rate(req/s)	10	15	20	25
r^2 (ARX01)	0.9370	0.9123	0.9015	0.8687
r^2 (ARX11)	0.9410	0.9282	0.9265	0.8935

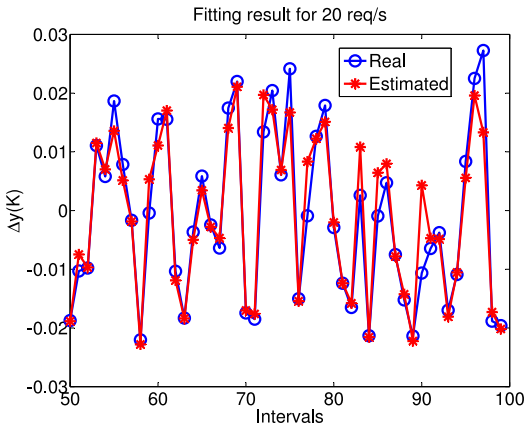

Figure 9. Fitting result for 20req/s

B. Controller Design

In our experiments, we choose ARX01 as our model, which implies that

$$\frac{\Delta y(z)}{\Delta S(z)} = az^{-1}$$

where $y(z)$ and $S(z)$ are the z-transform of y and S. We use an proportional-integral (PI) controller whose transfer function can be described as

$$\frac{\Delta S(z)}{e(z)} = k_p + \frac{k_i z}{z-1}$$

where k_p and k_i are the proportional and integral gains of the controller, respectively and $e(z)$ is the z-transform of the error. Then we have the closed model transfer function:

$$C(z) = \frac{G(z)}{1-G(z)} = \frac{(k_i + k_p)az - k_p a}{z^2 + ((k_i + k_p)a - 1)z - k_p a}$$

During our experiments, the parameter a of the model above is identified online through a recursive least square method to accommodate the nonlinear and time-varying behavior of the system. Then we use the Root Locus [14] method to

design the controller parameters k_p and k_i so that the setting time of the controller is within three steps and overshooting within 10% for a step response.

VI. Performance guarantee through adaptive PI control

We evaluate the two-layer performance controller that integrates application controller with different resource partition schemes in this section. We first show how the different resource partition schemes can be formulated as different constraint optimization problems. Then we describe the time-varying workload that we use for evaluation in detail. Finally, we present the evaluation results using different SLAs.

A. Comparison of performance controller based on different resource partition schemes

The application performance controller with "Optimal" partition scheme solves the constraint optimization problem as shown in (1) and (2).

According to the definitions, the application performance controller with "Equal Utilization" solves the following constraint optimization problem:

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{3}$$

$$\text{s.t.} \quad RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \leq Ref \tag{4}$$

$$r = \frac{c_1}{u_1} = \frac{c_2}{u_2} = ... = \frac{c_N}{u_N} \tag{5}$$

Similarly, the application performance controller with "Equal Shares" solves the following constraint optimization problem:

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{6}$$

$$\text{s.t.} \quad RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \leq Ref \tag{7}$$

$$u = u_1 = u_2 = ... = u_N \tag{8}$$

Comparing the different constraint optimization problems, we can see that, all the three problems share the same objective (1) and the constraint (2). However, "Equal Utilization" appends the constraint (5) while "Equal Shares" has the constraint (8). "Equal Utilization" has been used in [15] for the benefits of simple communication between an application controller and container controllers. Although it does differentiate the resource partition to the three tiers using the relative metrics, e.g., the resource utilization, it is still not optimal as there is no guarantee that constraint (5) will still hold in the optimal solution.

Table III
Steady-state Performance when setting RTT=35ms

	Response Time (ms)				Thr	CPU Resource	
	Mean (std)	50p	90p	95p		Ent	Con
Opt (close)	36 (52)	12	103	139	18.8	0.54	0.16
Util (close)	36 (56)	12	115	153	18.9	0.64	0.16
Opt (open)	37(53)	12	104	142	19.1	0.57	0.16
Util (open)	37 (58)	12	115	154	19.1	0.67	0.16

B. Time-varying workload for evaluation

We use a real workload trace as shown in Figure 10 to evaluate the application performance under the application controller with different resource partition schemes. The workload trace is generated based on the Web traces from the 1998 World Cup site [16]. We extract the "request rate" metric from the Web trace and scale it down to fit our experiment environment. For example, we use 10req/s to mimic 10k request rate and use 20req/s to mimic 20k request rate. The initial CPU shares were set to 50. Each experiment runs 9000 seconds. To evaluate the controllers, we run a set of experiments with different styles of workloads (open or closed), and different resource partition schemes ("Opt" or "Util") in the container level. Thus, there will be four cases in a set of experiments. In our experiments, we try two SLAs where the threshold for the mean round trip time is 35ms and 200ms, respectively.

C. Setting point for the mean round trip time is 35ms

Figures 11 and 12 show the experimental results when the setting point for the mean round trip time is 35ms. Table III shows more statistics of the four cases: the mean, the standard deviation, the 50/90/95 percentiles of the response times of the individual requests, the throughput (req/sec), the total CPU entitlement and the total CPU consumption. The samples for the statistics are between the 10th interval and the 70th interval, where there are no obvious overshoots of the response times and the mean RTT is maintained much closer to the reference values. The cases with "Opt" controller have lower percentile response times, compared with that with "Util" controller. Moreover, up to 20% less amount of CPU resource is provisioned to the application in the cases with "Opt" controller while the throughputs are the same and so are the consumptions.

D. Setting point for the mean round trip time is 200ms

The application with closed workload is a "self-tuning" system. According to Little's law, we can derive that the relation between the throughput and RTT for the application with closed workload can be approximated as

$$Throughput = \frac{MPL}{RTT + ThinkTime}$$

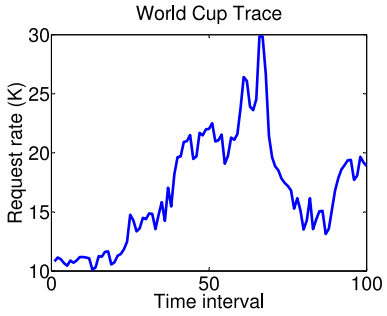

Figure 10. World Cup Trace

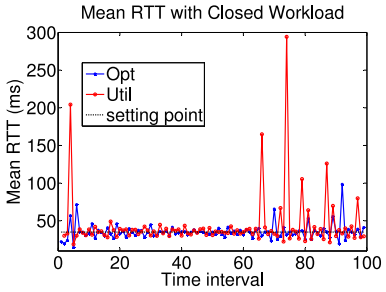

Figure 11. Mean RTT with Closed Workload

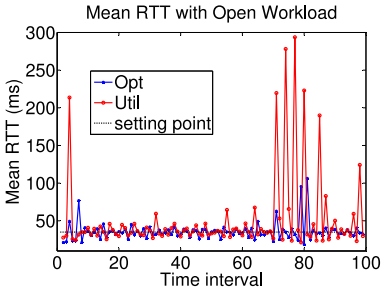

Figure 12. Mean RTT with Open Workload

The above model implies that the throughput should vary along with the RTT. However, as shown in Table III, the throughput of the applications with the open or closed workloads is almost the same. This is because that, in the above experiments, the default think time 3.5s and the setting point of RTT=35ms, that is, $RTT << ThinkTime$. So the throughput is almost not affected by the RTT, and the closed system works very similar as the open one.

To see the difference of the closed and open systems, we reduce the think time for the closed system to 350ms and increase the setting point of RTT to 200ms. We run the same set of experiments. The RTT during each interval and the steady state performance are shown in Figures 13-14 and Table IV. Table IV shows that the throughput can be affected by the response time for the closed system. From Figure 13, we can see that the controller works well with closed workload and the performance is well tracked as shown in Table IV. However, the performance seems out of control for the open system as shown in Figure 14. Upon sharp changes of the workload, there are very long transient processes before the response time converges. This can be due to the high utilization of the applications, when the response time is very sensitive to changes of the resources. It implies that, the parameters of the adaptive PI controllers have to be carefully tuned for the open system when the utilization is pushed high. As one more piece of work, we are working with a more robust adaptive controller to accommodate different types of workloads in a wider operation region.

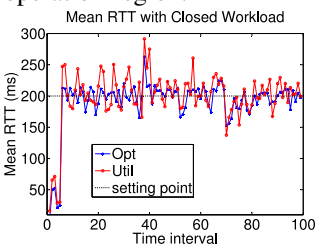

Figure 13. Mean RTT with Closed Workload

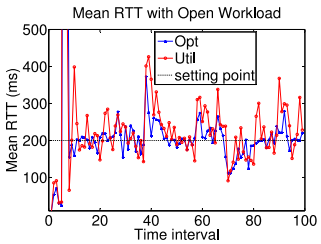

Figure 14. Mean RTT with Open Workload

VII. Related Works

Virtual resource management in Cloud environment has been studied with goals such as QoS awareness, performance isolation and differentiation as well as higher resource utilization [1]. Pradeep et al. develop an adaptive resource control system that dynamically adjusts the resource shares to applications in order to meet application-level QoS goals while achieving high resource utilization in the data center [12]. Lu et al. dynamically adjust the cache size for multiple request classes [17]. Krasic et al. [18] propose an approach called cooperative polling to ensure rigorous all applications fairly share resources. Lu et al. propose scheduling scheme to satisfy the requirements of different QoS requests for access to the networked storage system [19]. Most of the above works adopt either closed workload or open workload and pay little attention to whether a workload generator is closed or open. As illustrated in [7] and our paper, there is a vast difference in behavior between open and closed models in real-world settings. This motivates the use of partly open system models, whose behavior we show to lie between that of closed and open models. Compared with previous works, we specify how control policies are impacted by different models, and explain the differences in user level performance.

Table IV
Steady-state Performance when setting RTT=200ms

	Response Time (ms)				Thr	CPU Resource	
	Mean (std)	50p	90p	95p		Ent	Con
Opt (close)	205(214)	140	461	608	28	0.26	0.213
Util (close)	205(229)	142	472	617	28	0.29	0.214
Opt (open)	225(363)	136	475	661	42	0.43	0.301
Util (open)	230(399)	136	481	686	42	0.47	0.313

Besides the general system metrics that can be tuned for general systems, there are also lots of special intrinsic parameters for N-tier web applications that can be tuned to improve the performance. By carefully exploiting those potential parameters, we are able to use the resources more efficiently. For example, some previous works implement utilization controllers for each tier to maintain the utilization at the targeted level. A few workload management products like [15] maintain the utilization at a default utilization target, e.g., 75%. However, setting the same utilization for all the tiers is not the best option as shown in our experiments. Instead, after modeling the system with a tandem queue, we

formulate the problem as an optimization problem and also derive a solution for the problem. The experiment shows that our method can save up to 20% resource than "Equal Utilization" and "Equal Shares" schemes but achieve almost the same performance. Moreover, with different workload styles like closed and semi-open, our method still outperforms others.

VIII. Conclusion

Although several contemporary approaches employ control theory to manage resources in virtualized applications, not enough attention has been paid to the effects of workload styles and resource partition scheme. However, both of these problems are important in N-tier web applications deployed in Cloud environments.

In this paper, we address these two problems through a multi-level control approach. First, we showed that a round trip time-optimal resource partitioning scheme for the component servers could be found. We have benchmarked our method against state-of-the-art methods like "Equal Utilization" and "Equal Shares". Although we obtain our optimal partition scheme through open workload style, it still outperforms others for applications with closed/semi-open workload styles. Second, we developed an adaptive PI controller to control the mean round trip time for N-tier web application with resource partitioning schemes. Experimental results show that our solution outperforms other existing approaches in two aspects, (1) our multi-level control approach can achieve almost the same performance with the state-of-the-art methods but save up to 20% CPU resources. (2) our approach is robust to deal with different workload styles.

There are quite a few pieces of on-going and future work. (1) We use a single step function to model SLA cost function so that we can use performance controller to track the reference mean round trip time. However, we need to look into more general SLA cost function and explore how it can affect the application of the derived results. (2) We assume that there is an additive relationship between time spent in CPU time and non-CPU resources. However, if we look at networking as a non-CPU resource (e.g., data transfer between two tiers), then data transfer time can overlap with CPU time. Hence, we need a better model for the parameter β. (3) We design the performance controller by using the open workload. Although the controller also works for the close/semi-open workloads, it would be interesting to see what can we get if we design the performance controller by using the closed or semi-open workloads.

References

[1] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proc. of EuroSys*, 2010.

[2] "Amazon elastic compute cloud," http://aws.amazon.com/ec2/.

[3] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Proc. of NOMS*, 2002.

[4] S. Malkowski, M. Hedwig, D. Jayasinghe, C. Pu, and D. Neumann, "Cloudxplor: A tool for configuration planning in clouds based on empirical data," in *Proc. of SAC*, 2010.

[5] "Rubis," http://rubis.ow2.org/.

[6] 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 *Proc. of SOSP*, 2003.

[7] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: a cautionary tale," in *Proc. of NSDI*, 2006.

[8] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. NJ: Prentice-Hall, Inc., 1984.

[9] P. Xiong, Z. Wang, G. Jung, and C. Pu, "Study on performance management and application behavior in virtualized environment," in *Proc. of NOMS*, 2010.

[10] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu, "A cost-sensitive adaptation engine for server consolidation of multi-tier applications," in *Proc. of Middleware*, 2009.

[11] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser, "Appraise: Application-level performance management in virtualized server environments," *IEEE Trans. on network and service management*, vol. 6, no. 4, 2009.

[12] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Shin, "Automated control of multiple virtualized resources," in *Proc. of Eurosys*, 2009.

[13] "System identification," http://www.mathworks.com/products/sysid/.

[14] J. Hellerstein, Y. Diao, S. Parekh, and D.Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[15] "Hp global workload manager (gwlm)," http://mslweb.rsn.hp.com/gwlm/index.html.

[16] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup. web site," *HP Tech. Rep.*, 1999.

[17] Y. Lu, T. F. Abdelzaher, and A. Saxena, "Design, implementation, and evaluation of differentiated caching services," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, pp. 440–452, 2004.

[18] C. Krasic, M. Saubhasik, A. Sinha, and A. Goel, "Fair and timely scheduling via cooperative polling," in *Proc. of Eurosys*, 2009.

[19] Y. Lu, D. H.-C. Du, C. Liu, and X. Zhang, "Qos scheduling for networked storage system," in *Proc. of ICDCS*, 2008.