

# A Distributed Control Framework for Performance Management of Virtualized Computing Environments

Rui Wang  
ECE Department  
Drexel University  
Philadelphia, PA 19104, USA  
rui.wang@drexel.edu

Dara M. Kusic  
Department of CS  
University of Pittsburgh  
Pittsburgh, PA 15260, USA  
dmk64@pitt.edu

Nagarajan Kandasamy  
ECE Department  
Drexel University  
Philadelphia, PA 19104, USA  
kandasamy@drexel.edu

## ABSTRACT

This paper develops a distributed cooperative control framework to manage the performance of virtualized computing environments. We consider a server cluster hosting multiple enterprise applications on a set of virtual machines (VMs) in which the system must dynamically optimize the CPU capacity provided to each VM in response to incoming workload intensity such that desired response times are satisfied. We solve the overall control/optimization problem by decomposing it into a set of smaller subproblems that can be solved cooperatively by individual controllers. Model-predictive controllers, implemented locally within each server, independently decide the CPU capacity to allocate to VMs under their control such that the overall system's performance goals are satisfied. We experimentally validate the proposed framework on a server cluster supporting three on-line services, showing that our scheme is highly scalable, naturally tolerates server failures, and allows for the dynamic addition/removal of servers during system operation without requiring changes to the overall control architecture.

## Categories and Subject Descriptors

C.4 [Performance of systems]: Design studies, modeling techniques, fault tolerance

## General Terms

Algorithms, Performance, Management, Reliability

## Keywords

Performance management, virtualization, model-predictive control, distributed control

## 1. INTRODUCTION

Data centers host online services on distributed computing systems comprising heterogeneous networked servers. On-line services are enabled by *enterprise applications*, defined

broadly as any software which simultaneously provides services to a large number of users over a computer network. Virtualization technology is a promising solution to support multiple enterprise applications using fewer computing resources. This technology enables a single server to be shared among multiple performance-isolated platforms called virtual machines (VM), where each VM can serve one or more applications. Also, virtualization enables *on-demand computing* where resources such as CPU, memory, and disk space are allocated to applications as needed, based on the currently prevailing workload demand, rather than statically, based simply on the peak workload demand. By dynamically provisioning virtual machines and turning servers on and off properly, data center operators can maintain the desired quality of service (QoS) while achieving higher server utilization and lower power consumption.

A promising method for automating system management tasks is to formulate them as online control problems in terms of cost/performance metrics [4, 3]. We refer the reader to Section 6 for a discussion on related work in this area. Most proposed control techniques, however, are centralized designs, aimed at managing the performance of a stand-alone server or a small-scale system comprising a few servers. Significant challenges must still be addressed to achieve real-time control of a large-scale computing system with multiple interacting components. For an optimization scheme to be of practical value in a distributed setting, it must successfully tackle the so-called "curses" of modeling and dimensionality. The number of available tuning options is typically quite large in distributed systems and the corresponding search space grows exponentially with each new variable, making centralized controller designs intractable. Complex, non-linear, and possibly time-varying, component behavior as well as component interactions must be accurately modeled and carefully managed at run time to achieve system-wide performance goals. Finally, the system management task is further complicated when these components must communicate with each other to solve the overall problem.

Control theory provides techniques that can be used to reduce the computational burden of managing large-scale computing systems. For example, concepts from approximation or aggregation theory can be integrated within the control scheme to make relevant approximations when constructing dynamical models to predict system behavior, and when optimizing the control variables issued to the system [10]. Another method, problem permitting, is to structure controllers in a decentralized, hierarchical fashion wherein the overall problem is decomposed into a set of simpler sub-

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.

ICAC'10, June 7–11, 2010, Washington, DC, USA.

Copyright 2010 ACM 978-1-4503-0074-2/10/06 ...\$10.00.

problems and solved in cooperative fashion by multiple controllers [11].

Previously, we had proposed some general concepts to develop a decentralized control framework for resource provisioning in distributed computing systems, and had used a simulation-based case study to establish its feasibility [13]. Building on this work, we now develop and experimentally validate a distributed control scheme to manage the performance of virtualized computing environments. We consider a heterogeneous and virtualized server cluster hosting multiple enterprise applications on VMs, and processing a time-varying workload. The problem of interest is to optimize the CPU share provided to each VM to accommodate dynamic changes in workload intensity to meet desired response times. The overall control problem is decomposed into a set of corresponding subproblems and each subproblem is mapped to an underlying system component—in this case, a server. Model-predictive controllers, implemented locally within each server, work cooperatively to decide the CPU shares of the VMs under their control to satisfy the overall performance goals for the system.

The proposed framework has the following desirable characteristics.

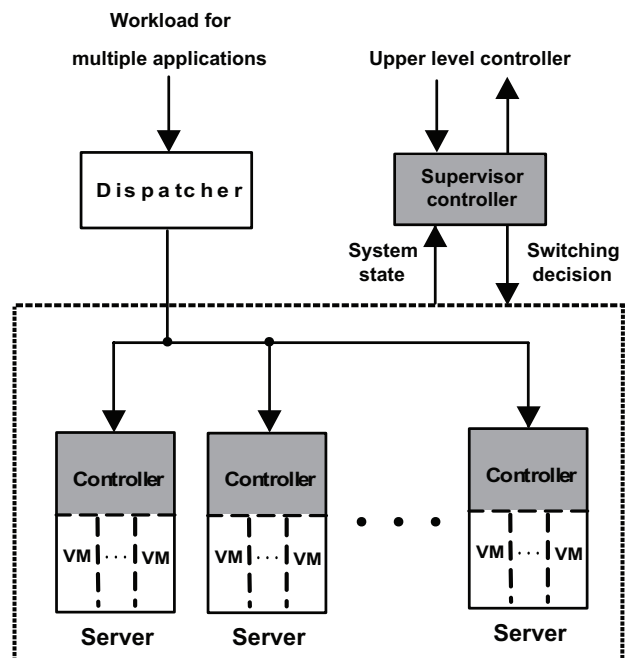
- Compared to a centralized controller implementation, each controller in our architecture incurs a much lower computational complexity since it only makes local resource provisioning decisions for its server. Thus, the framework is highly scalable as well as flexible, in the sense that servers can be added/removed anytime while maintaining the overall system performance.
- The framework can tolerate a certain number of server and VM failures. The surviving components, without being coordinated by a higher-level controller, can automatically adjust their control decisions accordingly and continue to meet system-level objectives.

We validate the control framework using a testbed of heterogeneous servers hosting Trade6, a stock-trading service that allows users to browse, buy, and sell stocks. The cluster processes a time-varying incoming workload, and results demonstrate that our scheme is highly scalable, tolerates server failures, and allows for the dynamic addition/removal of servers during system operation.

The paper is organized as follows. Section 2 outlines an architecture for managing the performance of large-scale computing systems. Section 3 introduces the cluster testbed used in our experiments. Section 4 constructs the dynamical system models, and formulates and solves the optimization problem, and Section 5 presents experimental results validating the control framework. Section 6 discusses some related work on managing the performance of virtualized computing systems and Section 7 concludes the paper with a discussion on future work.

## 2. CONTROL OF LARGE-SCALE SYSTEMS

When using control strategies to manage computing systems, it is readily apparent that centralized designs become quickly intractable for larger systems. Fortunately, *hierarchical* or *decentralized* control, where multiple controllers interact with each other to satisfy system-wide QoS goals, can be used to reduce the dimensionality of the overall problem. In a hierarchical structure, a controller is only responsible



**Figure 1: A hierarchical structure for managing large-scale systems, comprising a supervisory controller and a set of distributed controllers.**

for optimizing the behavior of the component(s) under its control while satisfying the constraints imposed on it by a higher-level controller.

We now briefly outline a hierarchical solution for controlling a large-scale computing system hosting multiple enterprise applications. Fig. 1 shows the structure consisting of a supervisory controller and distributed controllers local to each server. The incoming application workload is dispatched to the appropriate VMs hosted within the servers. The controllers within the hierarchy have the following responsibilities.

- The supervisory controller makes high-level switching decisions based on the system state and estimates of the incoming workload intensity that dictate which physical servers are turned on or off.
- Local controllers on each server dynamically optimize the CPU share provided to VMs under their control to match the workload dispatched to the server.

The supervisor and the distributed controllers can cooperate to manage the power consumed by the overall system while satisfying QoS requirements as follows. Since local controllers tune the CPU share of VMs to closely match the incoming workload, servers typically have spare processing capacity available during periods of light workload. The supervisory controller can use this knowledge to both increase server utilization and reduce power consumption by packing VMs into the fewest number of physical servers (by migrating live VMs between servers) and shutting down the extra machines. Another option is to simply shut down the lightly loaded machines. The workload is then re-distributed to VMs within the operational servers whose controllers will tune the CPU share accordingly. Conversely, the supervisor

Machine	CPU Speed	CPU Cores	Memory
Apollo	2.3 GHz	8	8 GB
Poseidon	2.3 GHz	8	8 GB
Eros	1.6 GHz	8	4 GB
Demeter	1.6 GHz	8	4 GB
Rada	2.3 GHz	2	4 GB
Megatron	2.3 GHz	2	4 GB
Starscream	2.3 GHz	2	4 GB
Chronos	1.6 GHz	8	4 GB

**Figure 2: Key system information for each of the eight heterogeneous servers in the testbed.**

can power up additional servers when the workload exceeds the cluster’s current processing capacity.

For its part, the supervisor may not require a detailed behavioral model of the lower-level components (both servers and local controllers) to make switching decisions. For instance, the supervisor can continuously learn the behavior of the distributed controllers in terms of how they tune the VMs’ processing capacity in response to the incoming workload. So, over time, the supervisor can build an approximate model that maps workload intensity and mix to a corresponding processing capacity, and use this model to make switching decisions that tune this processing capacity.

Hierarchical schemes have the potential to be highly scalable. First, treating the dashed box in Fig. 1 as a component, the number of servers within this component can be increased without affecting the overall control structure. Secondly, an upper-level controller with essentially the same logic as the supervisor can manage multiple such components by switching them on/off.

This paper focuses on developing and validating the distributed controllers that operate under the guidance of the supervisor. The design of the supervisory controller itself is planned for future work.

### 3. THE EXPERIMENTAL SETUP

This section describes our experimental setup, including the system architecture, the enterprise applications used for the online services, and workload generation.

#### 3.1 The Testbed

The computing cluster used in our experiments consists of the eight servers detailed in Fig. 2, networked via a gigabit switch. Virtualization of this cluster is enabled by VMware’s ESX Server 3.5 running a Linux RedHat 3.2 kernel. The operating system on each VM is SUSE Enterprise Linux (Server Edition 10). The ESX server controls the disk space, memory, and CPU share (in MHz) allotted to the VMs, and provides an application programming interface (API) to support the remote management of VMs. The controllers use this API to dynamically assign CPU shares to the virtual machines.

#### 3.2 The System Architecture

Fig. 3 shows the two-tier architecture supporting three web-based applications termed Gold, Silver, and Bronze, using front-end application servers and back-end databases, as

well as the schematic of a local controller (to be discussed later in Section 4).

The application tier comprises four servers, each hosting three VMs. Each VM within a server is dedicated to one of the Gold, Silver, or Bronze applications, and VMs residing on different servers but supporting the same application form a virtual computing cluster. The local controller on a server dynamically allocates the optimal CPU share to each of its VMs in response to the incoming workload intensity. Servers comprising the database tier are not virtualized and we do not perform dynamic resource provisioning at this tier. These servers run SUSE Enterprise Linux with DB2 as the database component and each machine supports a dedicated database servicing a single application.

Incoming requests to an application are dispatched to VMs within the corresponding virtual cluster in weighted round-robin fashion with the weights being proportional to CPU share. Thus, at the beginning of a control step, each local controller transmits its most recent CPU-share decision to the dispatcher. Since a VM’s CPU share reflects processing capacity, the larger the CPU share, the more requests that VM will receive.

To improve scalability of the distributed control architecture, local controllers are developed as non-communicating agents, wherein the aggregate processing capacity of the computing cluster is inferred by each controller without explicit exchange of messages between controllers. This is achieved by examining the dynamics of the global workload, which is a shared system variable forming an implicit coupling between the various local controllers. Each controller independently estimates the incoming workload to the applications as well as the aggregate computing capacity of other VMs in the virtual clusters, and then intelligently assigns CPU shares to VMs under its control<sup>1</sup>. Therefore, if the controllers cooperate well during normal system operation, then at any given time instant, the server cluster would possess just enough aggregate processing capacity to satisfy the response-time requirements of the time-varying incoming workload.

Finally, one key issue to consider when designing a distributed control system is whether the decentralized controllers operate synchronously in lockstep or asynchronously. Consider the case where controllers operate synchronously. They would observe the same external environment and system state, and make the same decisions, causing the system to oscillate. For example, suppose the incoming request rate for an application increases at some time instant. Each controller will observe this happening at exactly the same time (their sampling times are synchronized), and since controllers do not communicate with each other, each will increase its VMs’ CPU share appropriately to consume the increased workload. However, since all controllers take the same action, the total processing capacity of the cluster will be a lot higher than necessary. During the next time step, the controllers will compensate for this situation by decreasing CPU share to the VMs, and the cycle will repeat itself. This behavior may result in undesirable oscillations

<sup>1</sup>The problem formulation detailed in Section 4 can be extended in a straightforward way to also tune CPU operating frequency to manage the power consumed by a single server. The ESX virtualization layer does not support dynamic voltage scaling at this time, and so, we have not included CPU frequency as a tuning knob in this paper.

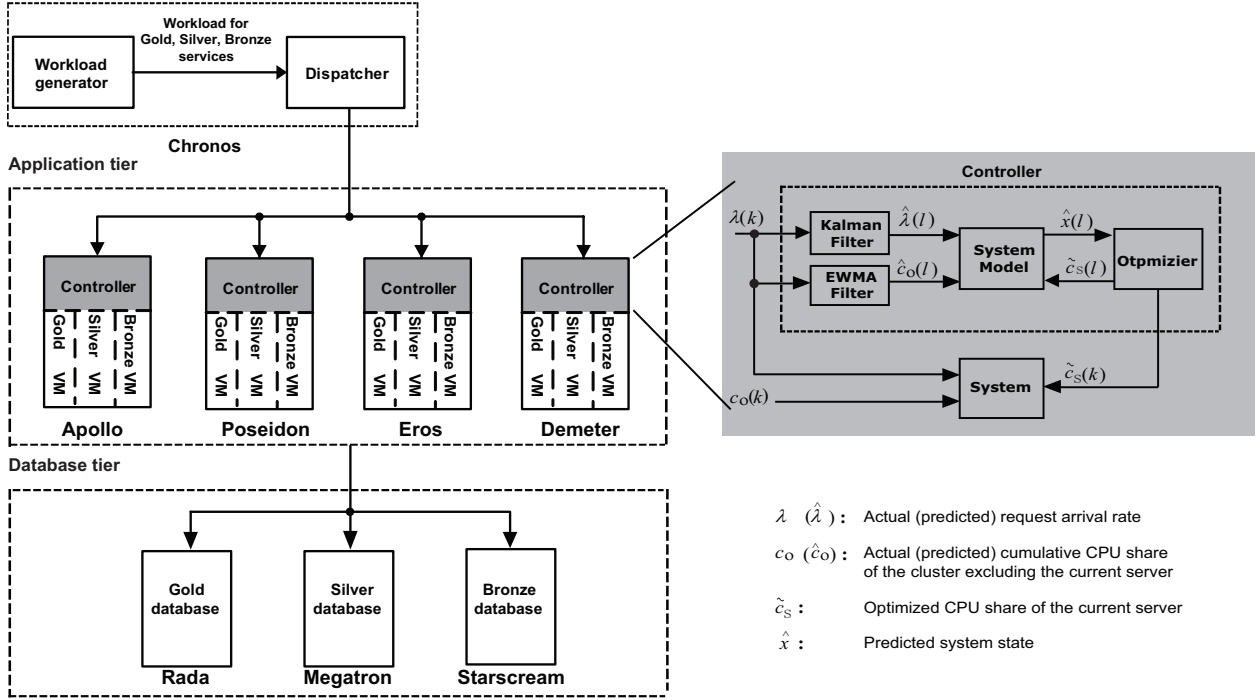


Figure 3: The system architecture hosting the online services. Local controllers on each server decide the CPU share to assign to VMs under their control within the application tier.

within the system. To avoid this problem, controllers in the proposed framework work asynchronously. By staggering the sampling times appropriately, we ensure that each controller observes slightly different environment conditions when making a control decision.

### 3.3 Applications and Workload Generation

The physical architecture shown in Fig. 3 hosts three web-based services. Since the emphasis in this paper is on validating the control framework, we simplified our system-development effort by using the Trade6 application as the basis for all three services. This simplification does not affect the validation results reported in the paper, and other enterprise applications such as DVDStore and RUBiS will be deployed on the testbed as part of future work.

Trade6 is a stock-trading application which allows users to browse, buy, and sell stocks. So, users can perform dynamic content retrieval as well as transaction commitments, requiring database reads and writes, respectively. The application logic for Trade6 resides within the IBM WebSphere Application Server, which in turn, is hosted by a VM in the application tier. The database component is DB2.

To simulate three separate online services, we elicit differing behavior from Trade6, in terms of response time, as follows. The *Gold* service uses Trade6 configured with a database size of 10,000 users and 20,000 stock quotes; the *Silver* service uses a database with 2,000 users and 4,000 quotes; and the *Bronze* service uses a small database with 100 users and 200 quotes. Trade6 is quite sensitive to database size—that is, for the same incoming workload intensity, the application exhibits different response times for different database sizes. The response time typically becomes longer with larger database sizes.

We use Httpperf [9], an open-loop workload generator, to

send browse/buy/sell requests to Trade6. Fig. 4 shows an example of a time-varying incoming workload to each of the three services. Request arrivals exhibit time-of-day variations typical of many enterprise workloads and the number of arrivals can change quite significantly within a very short time period. The workload used in our experiments was synthesized, in part, using log files from the Soccer World Cup 1998 Web site [2]. Also, the results presented in this paper assume a sessionless workload, meaning requests are assumed to be independent of each other and there is no

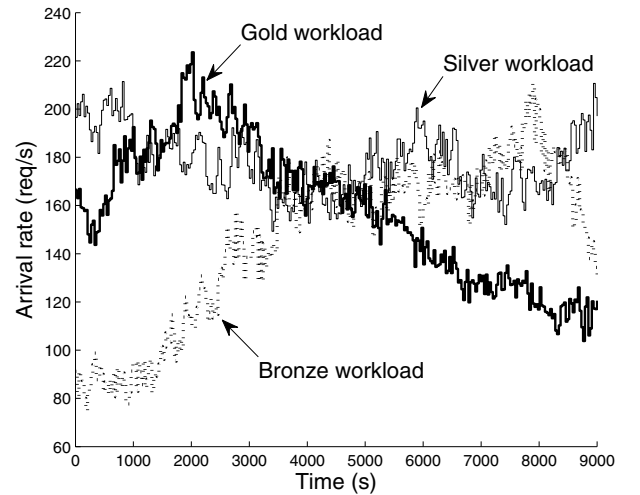


Figure 4: The incoming workload, in terms of the number of requests per second, to each of the three services.

need to maintain state information for multiple requests belonging to one user session.

#### 4. CONTROL-ARCHITECTURE DESIGN

The proposed control architecture aims to operate the computing cluster described in Section 3 most efficiently by tuning its aggregate processing capacity to closely track the incoming workload intensity. The QoS metric is stated in terms of a desired response time for the hosted services. This system-wide problem is decomposed into a set of simpler subproblems and each is assigned to a local controller. When these subproblems are solved independently, we approach the global solution.

Before describing our approach, we must note that not all control problems can be decomposed in a distributed fashion. However, it is well known that, given multiple subsystems whose dynamics and operating constraints are uncoupled and whose local cost functions are quadratic, simply having each subsystem independently optimize its local cost function can potentially achieve the global optimal. The performance management problem considered here falls in this category, and we can decompose it into subproblems for each server to solve such that the summation of the local costs recovers the centralized cost.

##### 4.1 The Local Control Scheme

Each local controller uses concepts from model-predictive control (MPC) to solve the subproblem assigned to it. The basic idea behind MPC is to solve a multi-objective optimization problem that minimizes the cost function over a given prediction horizon, and then periodically roll this horizon forward [8].

Fig. 3 shows the MPC scheme implemented on each local controller. Here,  $\lambda(k)$ ,  $c_o(k)$ ,  $c_s(k)$ , and  $x(k)$  denote the request arrival rate for applications hosted on the system, the cumulative CPU share of the cluster excluding the current server, the CPU share of the current server, and the system state, respectively. During control step  $k$ , a Kalman filter predicts the request rate  $\hat{\lambda}(l)$  along the prediction horizon and an exponential-weighted moving-average (EWMA) filter estimates  $\hat{c}_o(l)$ . Using these estimates and the system model, the optimizer finds a sequence of control actions  $\tilde{c}_s(l)$  along the prediction horizon minimizing a specified objective function. Then, the first action  $\tilde{c}_s(k)$  in the sequence is supplied to the system and the rest are discarded. The above process is repeated when updated system and environment information becomes available at time  $k+1$ .

##### 4.2 The Server Model

This section shows how to obtain a behavioral model capturing the dynamics of a server hosting multiple VMs (applications). For ease of notation, we use the subscript  $i$  to denote the  $i^{th}$  service class—that is,  $i = 1, 2$ , and  $3$  denote the Gold, Silver, and Bronze services, respectively. So, a VM hosted on server  $j$ , supporting service  $i$  is denoted as  $VM_{ij}$ .

From the local controller's viewpoint, the dynamics of the virtual cluster supporting service  $i$  at time step  $k$  is

$$x_i(k+1) = x_i(k) + T_s \cdot u_i(k), \quad (1)$$

where  $x_i(k)$  is the system state representing unprocessed client requests (queue size),  $T_s$  is the controller sampling

period, and  $u_i(k)$  is the control input representing the difference between the request arrival rate for service  $i$  and the corresponding processing rate. As shown in (2),  $u_i(k)$  contains three terms:  $\hat{\lambda}_i(k)$ , the estimated arrival rate for service  $i$ ;  $\hat{\mu}_{io}(k)$ , the estimated aggregate processing rate of other VMs in the virtual cluster (excluding the local VM on this server); and  $\mu_{is}(k)$ , the processing rate of the local VM.

$$u_i(k) = \hat{\lambda}_i(k) - \hat{\mu}_{io}(k) - \mu_{is}(k) \quad (2)$$

The arrival rate  $\hat{\lambda}_i(k)$  is predicted via a Kalman filter using previously observed values. The aggregate processing rate  $\hat{\mu}_{io}(k)$  of other VMs in the virtual cluster is obtained from their corresponding CPU share  $\hat{c}_{io}(k)$  (also estimated locally by the server). We conduct detailed profiling experiments to determine a mapping coefficient  $r_i$  which maps a given CPU share (in GHz) to a corresponding processing rate as

$$\hat{\mu}_{io}(k) = f(\hat{c}_{io}(k)) = r_i \cdot \hat{c}_{io}(k). \quad (3)$$

The profiling experiments are detailed in Section 5.

We can estimate  $\hat{c}_{io}(k)$  locally on a server without any explicit communication with other servers as follows. Recall that incoming requests are distributed to VMs based on their respective CPU share. Let  $\lambda_{io}(k-1)$  and  $\lambda_{is}(k-1)$  denote the arrival rate to the local VM, and to other VMs in the virtual cluster, respectively, and let  $c_{io}(k-1)$  and  $c_{is}(k-1)$  be the corresponding CPU shares. We then have

$$\frac{\lambda_{io}(k-1)}{c_{io}(k-1)} = \frac{\lambda_{is}(k-1)}{c_{is}(k-1)},$$

and so,

$$\begin{aligned} c_{io}(k-1) &= \frac{c_{is}(k-1)}{\lambda_{is}(k-1)} \cdot \lambda_{io}(k-1) \\ &= \frac{c_{is}(k-1)}{\lambda_{is}(k-1)} \cdot [\lambda_i(k-1) - \lambda_{is}(k-1)]. \end{aligned}$$

After obtaining  $c_{io}(k-1)$ , the local controller uses an EWMA filter to estimate  $\hat{c}_{io}(k)$  for the current control interval as

$$\hat{c}_{io}(k) = (1 - \eta) \cdot \hat{c}_{io}(k-1) + \eta \cdot c_{io}(k-1),$$

where  $0 \leq \eta \leq 1$  is a weighting factor.

Let us now revisit (2). As  $\hat{\lambda}_i(k)$  and  $\hat{\mu}_{io}(k)$  are uncontrollable from a local controller's viewpoint, they are considered as a control input  $\hat{u}_{io}(k)$  emanating from other VMs in the virtual cluster, where

$$\hat{u}_{io}(k) = \hat{\lambda}_i(k) - \hat{\mu}_{io}(k).$$

Since  $\mu_{is}(k)$  is controllable, it is the local control input  $u_{is}(k)$ , where

$$u_{is}(k) = \mu_{is}(k) = r_i \cdot c_{is}(k),$$

and  $c_{is}(k)$  is the CPU share of the local VM supporting service  $i$ . Then (2) becomes

$$u_i(k) = \hat{u}_{io}(k) - u_{is}(k) \quad (4)$$

and (1) can be rewritten in terms of  $u_{is}$  as

$$x_i(k+1) = x_i(k) + T_s \cdot [\hat{u}_{io}(k) - u_{is}(k)]. \quad (5)$$

Now, we can construct the dynamic model that includes all three services hosted on the server as

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \quad (6)$$

where

$$\mathbf{A} = \mathbf{I}, \quad \mathbf{x}(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix}, \quad \mathbf{B} = T_s [\mathbf{I} \quad -\mathbf{I}],$$

$$\mathbf{u}(k) = \begin{bmatrix} \hat{\mathbf{u}}_o(k) \\ \mathbf{u}_s(k) \end{bmatrix}, \quad \hat{\mathbf{u}}_o(k) = \begin{bmatrix} \hat{u}_{1o}(k) \\ \hat{u}_{2o}(k) \\ \hat{u}_{3o}(k) \end{bmatrix}, \quad \text{and} \quad \mathbf{u}_s(k) = \begin{bmatrix} u_{1s}(k) \\ u_{2s}(k) \\ u_{3s}(k) \end{bmatrix}.$$

So, (6) can be finally written as

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \mathbf{I} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} + T_s [\mathbf{I} \quad -\mathbf{I}] \begin{bmatrix} \hat{u}_{1o}(k) \\ \hat{u}_{2o}(k) \\ \hat{u}_{3o}(k) \\ u_{1s}(k) \\ u_{2s}(k) \\ u_{3s}(k) \end{bmatrix}. \quad (7)$$

### 4.3 The Optimization Problem

The QoS requirement for the computing cluster is specified in terms of a response time to be achieved by the services. To ensure that the response time satisfies QoS goals while simultaneously reducing the CPU share provided to the VMs, one must maintain the request-queue length near a certain set point and closely match the cluster's processing rate (which is controllable) to the request arrival rate (which is uncontrollable).

The objective function  $v(k)$  for a one step lookahead controller interprets the above problem as one of maintaining both  $\mathbf{x}$  and  $\mathbf{u}$  near their set points  $\bar{\mathbf{x}}(k)$ ,  $\bar{\mathbf{x}}(k+1)$ , and  $\bar{\mathbf{u}}(k)$  during each control interval. So,

$$\begin{aligned} v(k) = & [\mathbf{x}(k) - \bar{\mathbf{x}}(k)]' \mathbf{P} [\mathbf{x}(k) - \bar{\mathbf{x}}(k)] \\ & + [\mathbf{u}(k) - \bar{\mathbf{u}}(k)]' \mathbf{Q} [\mathbf{u}(k) - \bar{\mathbf{u}}(k)] \\ & + [\mathbf{x}(k+1) - \bar{\mathbf{x}}(k+1)]' \mathbf{P} [\mathbf{x}(k+1) - \bar{\mathbf{x}}(k+1)], \end{aligned} \quad (8)$$

where  $'$  denotes matrix transpose. Also, we have

$$\mathbf{P} = p \begin{bmatrix} p_1 & 0 & 0 \\ 0 & p_2 & 0 \\ 0 & 0 & p_3 \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = q \cdot \mathbf{I}.$$

Since (8) is a multi-objective optimization function,  $p$  and  $q$  are weights reflecting the tradeoff between queue length and CPU share, as specified by the system designer, and  $p_1$ ,  $p_2$ , and  $p_3$  are weights reflecting the relative priorities between the Gold, Silver, and Bronze services. Now, if

$$\bar{\mathbf{x}}_k^{k+1} = \begin{bmatrix} \bar{\mathbf{x}}(k+1) \\ \bar{\mathbf{x}}(k) \end{bmatrix}, \quad \tilde{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & 0 \\ 0 & \mathbf{P} \end{bmatrix}, \quad \tilde{\mathbf{Q}} = \mathbf{Q},$$

$$\Phi = \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}, \quad \text{and} \quad \mathbf{W} = \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}.$$

then, substituting the above equations into (8), we get the standard quadratic form of the objective function in terms of the control-input vector  $\mathbf{u}(k)$  as

$$v(k) = a + \mathbf{b}' \mathbf{u}(k) + \frac{1}{2} \mathbf{u}(k)' \mathbf{G} \mathbf{u}(k), \quad (9)$$

where

$$\begin{aligned} a &= \mathbf{e}(k)' \tilde{\mathbf{P}} \mathbf{e}(k) + \bar{\mathbf{u}}(k)' \tilde{\mathbf{Q}} \bar{\mathbf{u}}(k), \\ \mathbf{e}(k) &= \Phi \mathbf{x}(k) - \bar{\mathbf{x}}_k^{k+1}, \\ \mathbf{b} &= \beta(k) + \Gamma \mathbf{x}(k), \\ \beta(k) &= -2(\mathbf{W}' \tilde{\mathbf{P}} \bar{\mathbf{x}}_k^{k+1} + \tilde{\mathbf{Q}} \bar{\mathbf{u}}_k), \\ \Gamma &= 2\mathbf{W}' \tilde{\mathbf{P}} \Phi, \quad \text{and} \\ \mathbf{G} &= 2(\tilde{\mathbf{Q}} + \mathbf{W}' \tilde{\mathbf{P}} \mathbf{W}). \end{aligned}$$

As described before, the control input  $\hat{\mathbf{u}}_o(k)$  is based on the global request arrival rate and other servers' CPU share, therefore not tunable from the current local controller's perspective. Each local controller must also satisfy dynamic operating constraints when solving the above optimization problem, in terms of a lower and upper bound,  $c_{il}$  and  $c_{iu}$ , respectively, on each VM's CPU share. Moreover, the combined CPU share provided to the three VMs should not exceed  $c_{max}$ , the maximum CPU capacity available on the server. The dynamic operation constraints are

$$\begin{cases} \hat{\mathbf{u}}_o(k) = \hat{\mathbf{u}}_o(k) \\ c_{il} \leq c_{is}(k) \leq c_{iu} \\ \Sigma c_{is}(k) \leq c_{max} \end{cases} \quad (10)$$

Since processing rate and CPU share are related by the parameter  $r$ , and since the control decisions  $\mathbf{u}_s(\cdot)$  are in terms of a processing rate for each VM, we use

$$\mathbf{H}_b = \begin{bmatrix} \frac{1}{r_1} & 0 & 0 \\ 0 & \frac{1}{r_2} & 0 \\ 0 & 0 & \frac{1}{r_3} \end{bmatrix} \quad \text{and} \quad \mathbf{H}_m = \begin{bmatrix} \frac{1}{r_1} & \frac{1}{r_2} & \frac{1}{r_3} \end{bmatrix}$$

to convert the original constraints in (10) to those in terms of processing rate as

$$\begin{cases} \hat{\mathbf{u}}_o(k) = \hat{\mathbf{u}}_o(k) \\ \mathbf{c}_l \leq \mathbf{H}_b \cdot \mathbf{u}_s(k) \leq \mathbf{c}_u \\ \mathbf{H}_m \cdot \mathbf{u}_s(k) \leq c_{max} \end{cases} \quad (11)$$

Finally, combining (9) and (11), we formulate the MPC optimization problem as

$$\min_{\mathbf{u}(k)} \quad a + \mathbf{b}' \mathbf{u}(k) + \frac{1}{2} \mathbf{u}(k)' \mathbf{G} \mathbf{u}(k) \quad (12)$$

subject to

$$\begin{cases} \hat{\mathbf{u}}_o(k) = \hat{\mathbf{u}}_o(k) \\ \mathbf{c}_l \leq \mathbf{H}_b \cdot \mathbf{u}_s(k) \leq \mathbf{c}_u \\ \mathbf{H}_m \cdot \mathbf{u}_s(k) \leq c_{max} \end{cases}$$

This problem, when solved using the Matlab function *quadprog*, returns a vector  $\tilde{\mathbf{u}}(k)$  where

$$\tilde{\mathbf{u}}(k) = \begin{bmatrix} \hat{\mathbf{u}}_o(k) \\ \tilde{\mathbf{u}}_s(k) \end{bmatrix}.$$

Once  $\tilde{\mathbf{u}}(k)$  is obtained, we map it back to the corresponding CPU share  $\tilde{\mathbf{c}}_s(k)$  using the relationship

$$\tilde{\mathbf{c}}_s(k) = \mathbf{H}_b \cdot \tilde{\mathbf{u}}_s(k). \quad (13)$$

The controller then applies  $\tilde{\mathbf{c}}_s(k)$  to all the VMs residing on the server—in our case, three—and communicates this information to the dispatcher which then distributes the incoming requests appropriately based on each VM's CPU share.

## 5. EXPERIMENTAL RESULTS

The control framework has been validated using the testbed from Section 3 and we discuss some key results from the experiments. The distributed controllers are implemented as Java programs, invoking the necessary Matlab routines packaged within a stand-alone C executable. Controller implementations incur very low run-time overhead to solve their respective optimization problems: typically, under 3 sec. The starting times of local controllers on the four servers are staggered slightly so that they operate asynchronously. Here, the setting is  $t_{c1} = 30$  second,  $t_{c2} = 60$  second,  $t_{c3} = 90$  second,  $t_{c4} = 120$  second. The sampling period of each controller is set to  $T_s = 120$  seconds.

We set the incoming request rate to change every 30 seconds, and to ensure that the estimated rate adequately covers any variability in the actual arrival rate about 95% of the time, the original Kalman estimate is modified as  $\hat{\lambda} + 2\sigma$ , where  $\sigma$  is the standard deviation between the actual and predicted values. The weighting factor in the EWMA filter is tuned to  $\eta = 0.3$ . To maintain some minimal processing capacity on each VM, the lower bound on CPU share is set to 1.5 GHz, and since the ESX server restricts the maximum number of cores that a VM can use on a server to four, the upper bound is set to 8 GHz for VMs residing on Apollo and Poseidon, and 6 GHz for those on Eros and Demeter.

Finally, the weights  $p_1$ ,  $p_2$ , and  $p_3$  in the objective function are set to 3, 2, and 1, respectively, so that the controller prioritizes the Gold, Silver, and Bronze services correctly. The weights  $p$  and  $q$  are set to 2 and 1 respectively, giving greater priority to depleting the request queue over assigning lower CPU shares to the VMs.

### 5.1 Building System Models via Profiling

We now detail the profiling experiments performed to obtain the parameter  $r_i$  that maps a VM's CPU share to a corresponding processing rate for each service  $i$ . Also, since the system's QoS goal is specified in terms of a response time to be achieved by the services, we describe how this response time is chosen.

Fig. 5 shows the response times achieved by a VM hosting the Gold service as a function of CPU share, arrival rate, and workload mix<sup>2</sup>. Consider the case where the VM is provided a fixed CPU share of 3 GHz. Requests are then sent to the VM with an increasing arrival rate to generate the corresponding response times. As long as the arrival rate is below 50 req/s, the VM achieves a relatively steady response time below 200 ms. However, if the arrival rate exceeds 50 req/s, the response time increases dramatically from around 200 ms to thousands of ms. This jump indicates an unstable system in which the arrival rate has exceeded the VM's processing rate. So, we conclude that a 3 GHz VM can process approximately 50 req/s before queuing instability occurs. If the VM's CPU share is further constrained, say to 2 GHz, its maximum processing rate is constrained to about 30 req/s.

The above procedure is repeated, collecting data on the processing rates achieved by the VM for different CPU shares. Fig. 6 summarizes the processing rates achieved by a VM for each of the Gold, Silver, and Bronze services as a function of CPU share. Since a stable response time of around 200

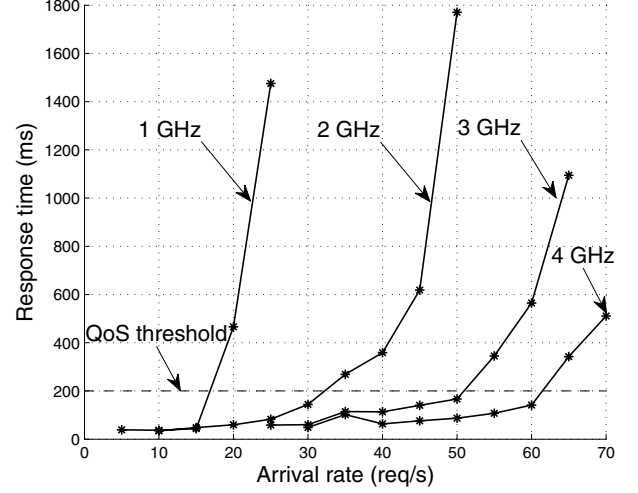


Figure 5: Average response times achieved by a VM hosting the Gold service as a function of request arrival rate and the CPU share provided to it.

App	1 GHz	2 GHz	3 GHz	4 GHz
Gold	15 req/s	30 req/s	50 req/s	60 req/s
Silver	20 req/s	30 req/s	55 req/s	70 req/s
Bronze	20 req/s	40 req/s	55 req/s	85 req/s

Figure 6: The processing rates achieved by VMs for each of the three services as function of CPU share.

ms seems achievable, it is set as the QoS metric for the services. With Matlab function *cftool*, the mapping factors in (3) for the services are obtained as  $r_1 = 15.5$ ,  $r_2 = 17.5$ , and  $r_3 = 20.17$ .

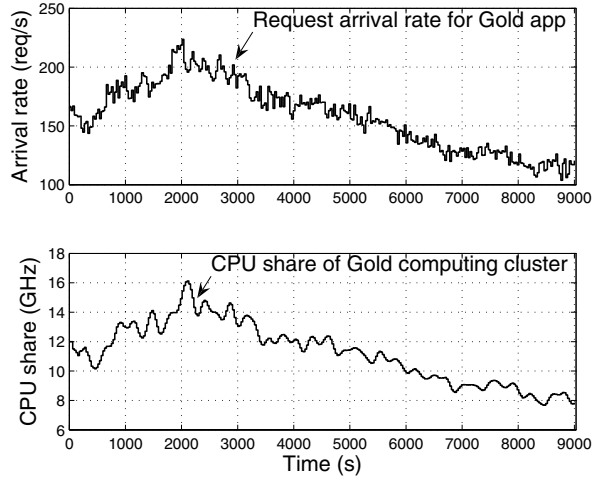
### 5.2 Normal System Operation

During normal system operation, decentralized controllers must cooperate to closely match the aggregate CPU share of each virtual cluster to the incoming request rate. Fig. 7 summarizes the performance of virtual clusters serving the Gold and Silver services, respectively. Focusing on Fig. 7(a), we see that the workload gradually increases before  $t = 2,100$  seconds and decreases thereafter, and the cumulative CPU share assigned to the VMs tracks this trend quite well.

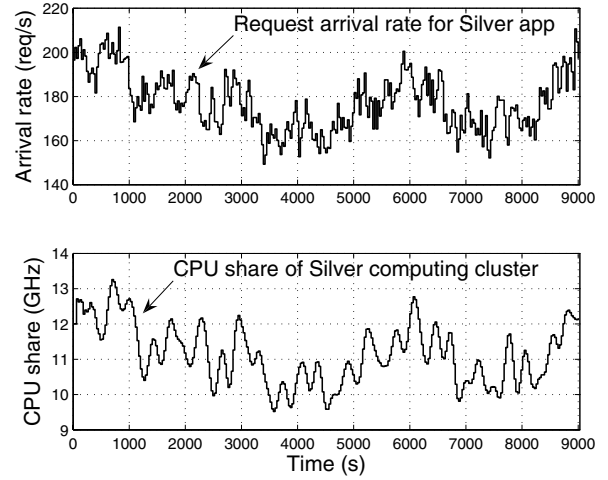
We now focus on the VMs within the Gold virtual cluster, specifically VM<sub>11</sub> and VM<sub>12</sub> whose operation is shown in Fig. 8. Before  $t = 2,100$  seconds, the controller on Server 1 estimates the arrival rate to increase and infers that the current cumulative CPU share will be insufficient to handle this increase. Therefore, it increases VM<sub>11</sub>'s CPU share accordingly to match the arrival and processing rates. After  $t = 2,100$  seconds, the workload begins to diminish, and the controller tunes the CPU share lower. The average response times for requests dispatched to VM<sub>11</sub> remain under 200 ms, satisfying the desired QoS. Other VMs in the virtual cluster, for example VM<sub>12</sub> on Server 2 behave similarly. However, as controllers operate asynchronously, they sample at different times, observing slightly different environment inputs such as the overall request arrival rate. As a result, controllers typically provide different inputs to VMs under their control.

Fig. 7(b) summarizes the performance of the virtual clus-

<sup>2</sup>The experiments reported in this paper assume a workload mix of 50% browse requests and 50% buy requests.

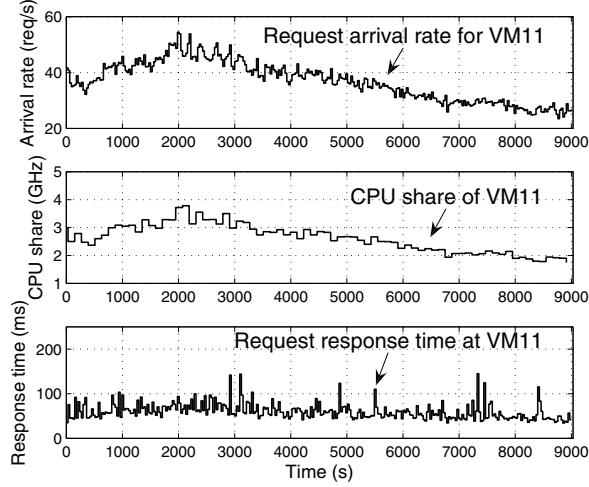


(a) Gold virtual cluster.

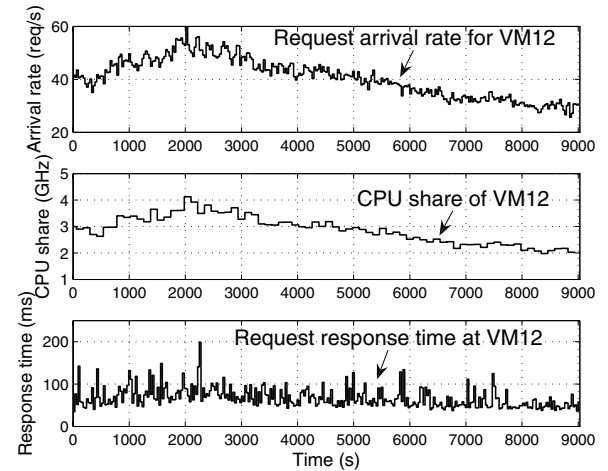


(b) Silver virtual cluster.

**Figure 7: CPU share provided to the virtual clusters serving the Gold and Silver services as a function of arrival rate.**



(a) Performance of VM<sub>11</sub> on Server 1



(b) Performance of VM<sub>12</sub> on Server 2

**Figure 8: The performance of selected VMs on different servers, both processing Gold requests.**

Response time	Gold	Silver	Bronze
Mean	72.05 ms	61.11 ms	57.98 ms
Standard deviation	25.84 ms	19.82 ms	29.01 ms
Num. violations	2	0	3
% violations	0.17%	0%	0.25%

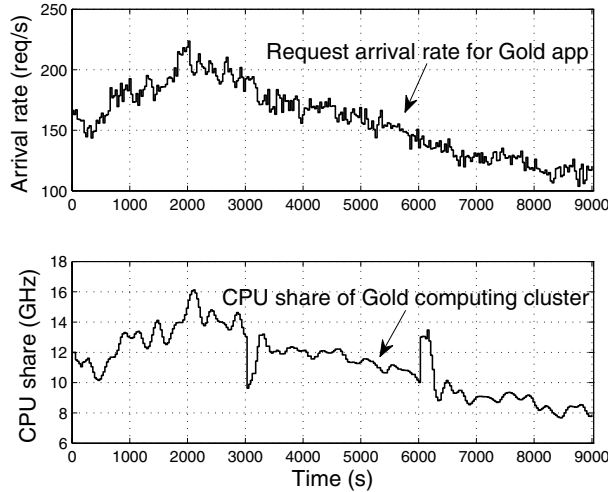
**Figure 9: Performance of the overall system during normal operation in terms of mean response times achieved and the number of QoS violations.**

ter serving Silver requests, showing that short bursts and fluctuations in the workload are tracked well. Finally, Fig. 9 summarizes the performance of the entire system during normal operation, assuming no server failures. The results indicate that the distributed controllers cooperate well to maintain the QoS metrics.

### 5.3 Removal/Addition of Servers

Experiments also indicate that the distributed framework is highly scalable and fault tolerant. In other words, the dynamic addition or removal (failure) of servers do not have a lasting affect on overall system performance. When a chunk of CPU share is added or removed from the cluster, other controllers detect this change within one or two control steps and tune CPU share accordingly.

Fig. 10 summarizes the performance impact on the Gold virtual cluster when server 4 is turned off to simulate a failure and then reintegrated at a later time. The server is removed from the system at  $t = 3,000$  seconds and reintegrated at  $t = 6,000$  seconds. At  $t = 3,000$  seconds, we see a big drop-off of about 3,500 MHz in the cluster's aggregate CPU share, which then rebounds back quickly in about 210 seconds, that is, in less than 2 control steps, since the



**Figure 10: Performance of the Gold virtual cluster when Server 4 is turned off at  $t = 3,000$  sec. and then reintegrated at  $t = 6,000$  sec.**

controller sampling period is 120 seconds. Fig. 11 focuses in on how VMs in the Gold cluster react to the failure affecting Server 4. Controllers on other servers, for instance, Server 1, infer the loss in computing capacity and increase their respective CPU shares to accept a bigger fraction of the Gold workload. The cumulative CPU share of the surviving VMs still tracks the incoming workload closely and the response time remains, for the most part, under 200 ms. At  $t = 6,000$  seconds, Server 4 is restored with an initial CPU share of 3,000 MHz, and the other controllers correspondingly reduce the CPU shares to their respective VMs.

## 6. RELATED WORK

During the past several years, feedback control theory has been applied to a variety of performance management problems in computing systems [4]. In particular, PID control has been successfully used for task scheduling [6], QoS adaptation in web servers [1], load balancing [7], and CPU power management [12]. Typically, a linear system with an unconstrained state space and a continuous input/output domain is assumed, and a closed-loop feedback controller is designed under stability and sensitivity requirements. More advanced state-space and multi-input multi-output (MIMO) methods accommodate multi-objective cost functions with constraints when optimizing performance. In [14], for example, a MIMO controller manages the power consumed by a cluster by shifting power between the various servers with respect to their performance needs, thereby manipulating the cluster's total power consumption to be lower than the specified power budget.

More recently, control-theoretic methods have been proposed to manage virtualized computing environments, including power management and CPU usage [15, 5]. In [5], the authors combine feedback control of response times with predictive control of environment disturbances (e.g., workload fluctuation) to control the CPU share of a single VM. We also see the use of hierarchical control strategies to manage multiple VMs in a coordinated fashion [17, 15, 16]. In [17], a two-level optimization scheme using fuzzy logic allo-

cates CPU shares to VMs processing two enterprise applications on a single host. A global controller arbitrates requests for CPU share from local controllers within VMs, aiming to maximize the profit generated by the server. The authors of [15] construct a two-layer control architecture for virtualized servers to reduce power consumption while achieving application-level QoS. They use a MIMO approach in the primary loop to balance the load among VMs so that every VM has generally the same performance. The second loop controls CPU frequency for power efficiency based on the performance level achieved by the primary loop. The authors of [16] develop a hierarchical scheme to adjust the CPU shares of VMs hosting an application whose components are spread over web, application, and database tiers, to achieve end-to-end response time targets.

To summarize, we note that a large majority of the control-theoretic methods aimed at performance management are centralized designs. Decentralized decision making and hierarchical control of computing systems is a recent phenomenon and significant challenges must still be addressed to achieve real-time control of large-scale systems. We believe that the distributed control framework described in this paper is an important step in this direction.

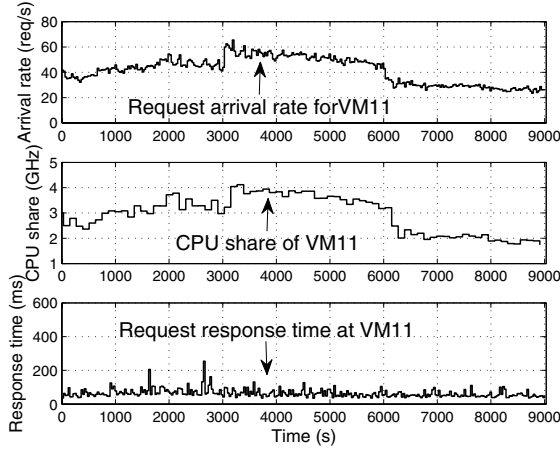
## 7. CONCLUSIONS

We have developed a distributed control framework to manage the performance of cluster hosting multiple enterprise applications on a set of virtual machines (VMs). The system must dynamically optimize the CPU capacity provided to each VM in response to incoming workload intensity such that desired response times are satisfied. The system-level problem of deciding the optimal CPU shares to VMs is decomposed into multiple identical sub-problems, and each sub-problem is solved in a cooperative fashion by controllers local to each server. The proposed method has been validated using a cluster of heterogeneous servers hosting the Trade6 enterprise application. Experimental results confirm that the control architecture is quite scalable, adapts quickly to workload fluctuations, and allows for the dynamic addition/removal of servers during system operation.

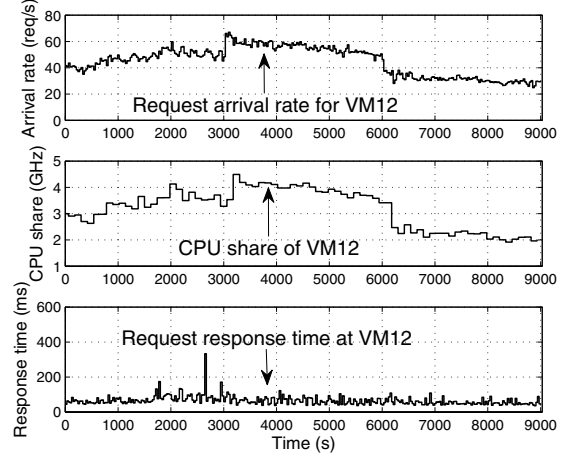
Going forward, we will develop and validate the hierarchical structure discussed in Section 2 by designing the higher-level supervisory controller to make appropriate switching decisions for the system.

## 8. REFERENCES

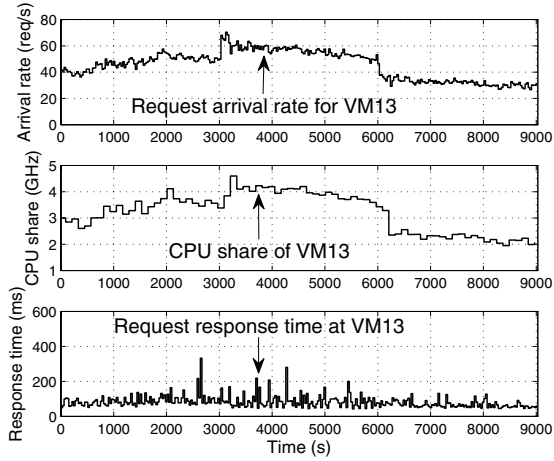
- [1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control theoretic approach. *IEEE Trans. Parallel & Distributed Syst.*, 13(1):80–96, January 2002.
- [2] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.
- [3] J. Hellerstein, S. Singhal, and Q. Wang. Research challenges in control engineering of computing systems. *IEEE Trans. Network & Service Mgmt.*, 6(4):206–211, Dec. 2009.
- [4] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.
- [5] M. A. Kjaer, M. Kihl, and A. Robertsson. Resource allocation and disturbance rejection in web servers



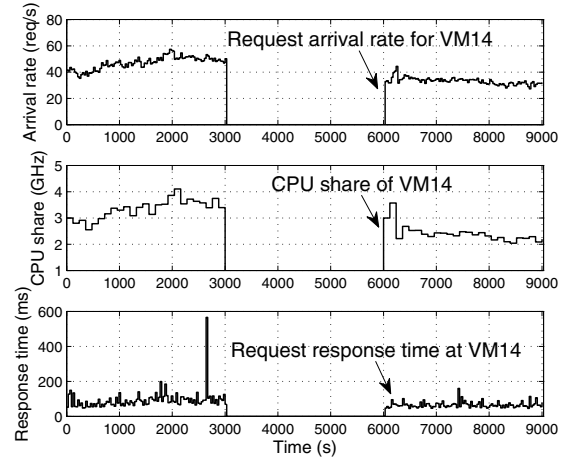
(a) Gold VM<sub>11</sub> on Server 1



(b) Gold VM<sub>12</sub> on Server 2



(c) Gold VM<sub>13</sub> on Server 3



(d) Gold VM<sub>14</sub> on Server 4

**Figure 11: Reaction of VMs within the Gold cluster in response to the failure of Server 4.**

using slas and virtualized servers. *IEEE Trans. Network & Service Mgmt.*, 6(4):226–239, Dec. 2009.

- [6] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *9th IFIP/IEEE Int'l Symp. Integrated Network Management (IM)*, pages 163–176, 2005.
- [7] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proc. USENIX Conf. File Storage Tech.*, pages 219–230, 2002.
- [8] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, London, 2002.
- [9] D. Mosberger and T. Jin. httpf: A tool for measuring web server performance. *Perf. Eval. Review*, 26:31–37, Dec. 1998.
- [10] K. Narendra and S. Mukhopadhyay. Adaptive control using neural networks and approximate models. *IEEE Trans. Neural Networks*, 8(3):475–485, May 1997.
- [11] N. Sandell, P. Varaiya, M. Athans, and M. Safonov. Survey of decentralized control methods for large scale systems. *IEEE Trans. Automatic Control*, 23(2):108–128, April 1978.
- [12] T. Simunic and S. Boyd. Managing power

consumption in networks on chips. In *Proc. Design, Automation & Test Europe*, pages 110–6, Mar. 2002.

- [13] M. Wang, N. Kandasamy, A. Guez, and M. Kam. Distributed cooperative control for adaptive performance management. *IEEE Internet Computing*, 11(1):31–40, January/February 2007.
- [14] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. pages 101–110, Feb. 2008.
- [15] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Proc. Real-Time Systems Symp.*, pages 303–312, Washington, DC, USA, 2008. IEEE Computer Society.
- [16] Z. Wang et al. Appraise: Application-level performance management in virtualized server environments. *IEEE Trans. Network & Service Mgmt.*, 6(4):240–254, Dec. 2009.
- [17] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. pages 25–35, Jun. 2007.