Power and Performance Management of Virtualized Computing Environments Via Lookahead Control[*]

Dara Kusic, Jeffrey O. Kephart, James E. Hanson[†] Nagarajan Kandasamy[‡] and Guofei Jiang [§]

September 21, 2008

Abstract

There is growing incentive to reduce the power consumed by large-scale data centers that host online services such as banking, retail commerce, and gaming. Virtualization is a promising approach to consolidating multiple online services onto a smaller number of computing resources. A virtualized server environment allows computing resources to be shared among multiple performance-isolated platforms called virtual machines. By dynamically provisioning virtual machines, consolidating the workload, and turning servers on and off as needed, data center operators can maintain the desired quality-of-service (QoS) while achieving higher server utilization and energy efficiency. We implement and validate a dynamic resource provisioning framework for virtualized server environments wherein the provisioning problem is posed as one of sequential optimization under uncertainty and solved using a lookahead control scheme. The proposed approach accounts for the switching costs incurred while provisioning virtual machines and explicitly encodes the corresponding risk in the optimization problem. Experiments using the Trade6 enterprise application show that a server cluster managed by the controller conserves, on average, 22% of the power required by a system without dynamic control while still maintaining QoS goals. Finally, we use trace-based simulations to analyze controller performance on server clusters larger than our testbed, and show how concepts from approximation theory can be used to further reduce the computational burden of controlling large systems.

Key words: Power management, resource provisioning, virtualization, predictive control

[*]A preliminary version of this paper appeared in the 2008 IEEE International Conference on Autonomic Computing.

[†]J.O. Kephart and J.E. Hanson are with the Agents and Emergent Phenomena Group, IBM T.J. Watson Research Center, Hawthorne, NY 10532. E-mail: kephart@us.ibm.com and jehanson@us.ibm.com

[‡]D. Kusic and N. Kandasamy are with the Electrical and Computer Engineering Department, Drexel University, Philadelphia, PA 19104. E-mail: dmk25@drexel.edu and kandasamy@ece.drexel.edu. D. Kusic is supported by NSF grant DGE-0538476 and N. Kandasamy acknowledges support from NSF grant CNS-0643888

[§]G. Jiang is with the Robust and Secure System Group, NEC Laboratories America, Princeton, NJ 08540. E-mail: gfj@nec-labs.com

1 Introduction

Web-based services such as online banking and shopping are enabled by enterprise applications. We broadly define an *enterprise application* as any software hosted on a server which simultaneously provides services to a large number of users over a computer network [1]. These applications are typically hosted on distributed computing systems comprising heterogeneous and networked servers housed in a physical facility called a data center. A typical data center serves a variety of companies and users, and the computing resources needed to support such a wide range of online services leaves server rooms in a state of "sprawl" with under-utilized resources. Moreover, each new service to be supported often results in the acquisition of new hardware, leading to server utilization levels at less than 20% by many estimates. With energy costs rising about 9% last year [2] and society's need to reduce energy consumption, it is imprudent to continue server sprawl at its current pace.

Virtualization provides a promising approach to consolidating multiple online services onto fewer computing resources within a data center. This technology allows a single server to be shared among multiple performance-isolated platforms called virtual machines (VMs), where each virtual machine can, in turn, host multiple enterprise applications. Virtualization also enables *on-demand* or *utility* computing—a just-in-time resource provisioning model in which computing resources such as CPU, memory, and disk space are made available to applications only as needed and not allocated statically based on the peak workload demand [3]. By dynamically provisioning virtual machines, consolidating the workload, and turning servers on and off as needed, data center operators can maintain the desired QoS while achieving higher server utilization and energy efficiency. These dynamic resource provisioning strategies complement the more traditional off-line capacity planning process [4].

This paper develops a dynamic resource provisioning framework for a virtualized computing environment and experimentally validates it on a small server cluster. The resource-provisioning problem of interest is posed as one of sequential optimization under uncertainty and solved using limited lookahead control (LLC). This approach allows for multi-objective optimization under explicit operating constraints and is applicable to computing systems with non-linear dynamics where control inputs must be chosen from a finite set.

Our experimental setup is a cluster of heterogenous Dell PowerEdge servers supporting two online services in which incoming requests for each service are dispatched to a dedicated cluster of VMs. The revenue generated by each service is specified via a pricing scheme or service-level agreement (SLA) that relates the achieved response time to a dollar value that the client is willing to pay. The control objective is to maximize the profit generated by this system by minimizing both the power consumption and SLA violations. To achieve this objective, the online controller decides the number of physical and virtual machines to allocate to each service where the VMs and their hosts are turned on or off according to workload demand, and the CPU share to allocate to each VM. This control problem may need to be re-solved periodically when the incoming workload is time varying.

This paper makes the following specific contributions.

- The LLC formulation models the cost of control, i.e., the switching costs associated with turning machines on or off. For example, profits may be lost while waiting for a VM and its host to be turned on, which is usually three to four minutes. Other switching costs include the power consumed while a machine is being powered up or down, and not performing any useful work.

- Excessive switching of VMs may occur in an uncertain operating environment where the incoming workload is highly variable. This may actually reduce profits, especially in the presence of the switching costs described above. Thus, each provisioning decision made by the controller is risky and we explicitly encode the notion of risk in the LLC problem formulation using preference functions.

- Since workload intensity can change quite quickly in enterprise systems [5], the controller must adapt to such variations and provision resources over short time scales, usually on the order of 10s of seconds to a few minutes. Therefore, to achieve fast operation, we develop a hierarchical LLC structure wherein the control problem is decomposed into a set of smaller sub-problems and solved in cooperative fashion by multiple controllers.

Experimental results using IBM's Trade6 application, driven by a time-varying workload, show that the cluster, when managed using the proposed LLC approach saves, on average, 22% in power-consumption costs over a twenty-four hour period when compared to a system operating without dynamic control. These power savings are achieved with very few SLA violations, 1.6% of the total number of service requests. The execution-time overhead of the controller is quite low, making it practical for online performance management. We also characterize the effects of different risk-preference functions on control performance, finding that a risk-aware controller reduces the number of SLA violations by an average of 35% compared to the baseline risk-neutral controller. A risk-aware controller also reduces both the VM and host switching activity—a beneficial result when excessive power-cycling of the host machines is a concern. The performance results also include a discussion on optimality issues.

The paper is organized as follows. Section 2 discusses related work on resource provisioning and Section 3 describes our experimental setup. Section 4 formulates the control problem and Section 5 describes the controller implementation. Section 6 presents experimental results evaluating the control performance. Section 7 addresses controller scalability using trace-based simulations. Section 8 concludes the paper.

2 Related Work

We now briefly review recent research on resource provisioning in virtualized computing environments. Our approach differs from the prior work in [6–11] in that it is a proactive control technique that encodes the risk involved in making provisioning decisions in a dynamic operating environment and accounts for the corresponding switching costs.

The authors of [12] propose an online method to select a VM configuration while minimizing the number of physical hosts needed to support this configuration. Their algorithm is reactive, and is triggered by events such as CPU utilization and memory availability to revise the placement of VMs. The authors consider VM migration costs in terms of the additional CPU cycles and memory needed to stop a VM, store its execution context, and restart it on another machine. In contrast, the cost of control in our work accounts for the time delays and opportunity costs incurred when switching hosts and VMs on/off. The placement algorithm in [12] also does not save power by switching off unneeded hosts.

The capacity planning technique proposed in [13] uses server consolidation to reduce the energy consumed by a computing cluster hosting web applications. The approach is similar to ours in that it estimates the CPU processing capacity needed to serve the incoming workload, but considers only a single application hosted on homogenous servers. The authors of [14] propose to dynamically reschedule/collocate VMs processing heterogeneous workloads. The problem is one of scheduling a combination of interactive and batch tasks across a cluster of physical hosts and VMs to meet deadlines. The VMs are migrated, as needed, between host machines as new tasks arrive. While the placement approach in [14] consolidates workloads, it does not save power by switching off unneeded machines, and does not consider the cost of the control incurred when migrating the VMs.

In [15], the authors propose a two-level optimization scheme to allocate CPU shares to VMs processing two enterprise applications on a single host. Controllers, local to each VM, use fuzzy-logic models to estimate CPU shares for the current workload intensity, and make CPU-share requests to a global controller. Acting as an arbitrator, the global controller affects a trade-off between the requests to maximize the profits generated by the host. The authors of [16] combine both power and performance management within a single framework, and apply it to a server environment without virtualization. Using dynamic voltage scaling on the operating hosts, they demonstrate a 10% savings in power consumption with a small sacrifice in performance.

Finally, reducing power consumption in server clusters has been a well-studied problem recently; for example, see [17–19]. The overall idea is to combine CPU-clock throttling and dynamic voltage scaling with switching entire servers on/off as needed, based on the incoming workload. In the presence of switching costs, however, two crucial issues must be addressed. First, turning servers off in a dynamic environment is somewhat risky in QoS terms—what if a server were just powered off in anticipation of a lighter workload, and the workload increases? Also, excessive power cycling of a server could reduce its reliability. The risk-aware controller presented here is a step towards addressing these issues.

3 Experimental Setup

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

3.1 The Testbed

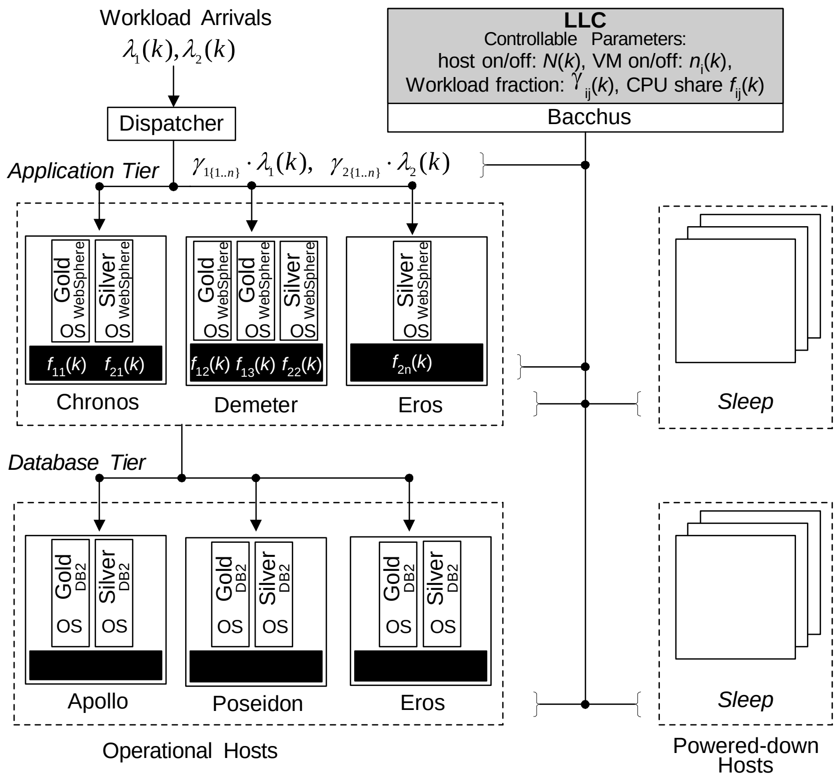
The computing cluster in Fig. 1(a) consists of the six servers detailed in Fig. 1(b), networked via a gigabit switch. Virtualization of this cluster is enabled by VMWare's ESX Server 3.0 Enterprise Edition running a Linux RedHat 2.4 kernel. The operating system on each VM is the SUSE Enterprise Linux Server Edition 10. The ESX server controls the disk space, memory, and CPU share (in MHz) allotted to the VMs, and also provides an application programming interface (API) to support the remote management of virtual machines using the simple object access protocol (SOAP). Our controller uses this API to dynamically instantiate or de-instantiate VMs on the hosts and to assign CPU shares to the virtual machines.

To turn off a virtual machine, we use a standard Linux shutdown command, and to physically turn off the host machine, we follow the shutdown command with an Intelligent Platform Management Interface (IPMI) command to power down the chassis. Hosts are remotely powered on using the wake-on-LAN protocol.

3.2 The Enterprise Applications and Workload Generation

The testbed hosts two web-based services, labeled *Gold* and *Silver*, comprising front-end application servers and back-end database servers. The applications perform dynamic content retrieval (customer browsing) as well as transaction commitments (customer purchases) requiring database reads and writes.

We use IBM's Trade6 benchmark—a multi-threaded stock-trading application that allows users to browse, buy, and sell stocks—to enable the Silver service. Trade6 is a transaction-based application integrated within the IBM WebSphere Application Server V6.0 and uses DB2 Enterprise Server Edition V9.2 as the database component. This execution environment is then distributed across multiple servers comprising the application and database tiers, as shown in Fig. 1(a). Virtual machines processing Silver requests are loaded with identical copies of the Trade6 execution environment, and a fraction of the incoming workload is distributed to each VM.



(a)

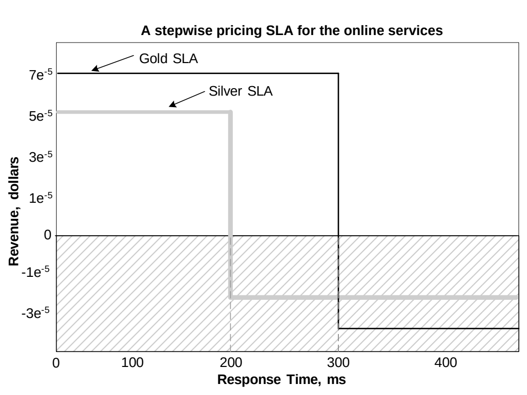
Host Name	CPU Speed	CPU Cores	Memory
Apollo	2.3 GHz	8	8 GB
Bacchus	2.3 GHz	2	8 GB
Chronos	1.6 GHz	8	4 GB
Demeter	1.6 GHz	8	4 GB
Eros	1.6 GHz	8	4 GB
Poseidon	2.3 GHz	8	8 GB

(b)

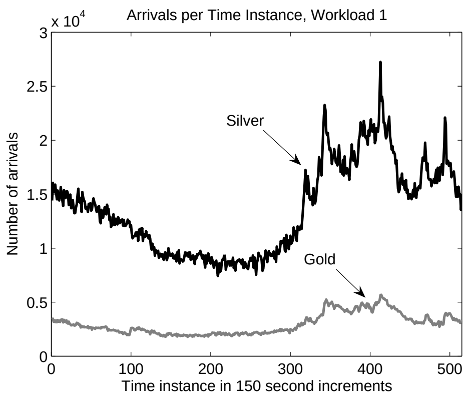
Fig. 1: (a) The system architecture supporting the Gold and Silver services. The controller sets $N(k)$, the number of active hosts, $n_i(k)$, the number of VMs to serve the i^{th} application, and $f_{ij}(k)$ and $\gamma_{ij}(k)$, the CPU share and the fraction of workload to distribute to the j^{th} VM, respectively. A *Sleep* cluster holds machines in a powered-off state. (b) The host machines comprising the testbed.

The Gold service is also enabled by Trade6, but the application is modified to perform some additional processing on the CPU for each Gold request. The amount of additional processing occurs with some variation, distributed about a mean value that is passed as a parameter by the request.

The Gold and Silver applications generate revenue as per the non-linear pricing graph shown



(a)



(b)

Fig. 2: (a) A pricing strategy that differentiates the Gold and Silver services. (b) Transaction requests to the Gold and Silver applications, plotted at 150-second intervals.

in Fig. 2(a) that relates the average response time achieved per transaction to a dollar value that clients are willing to pay. Response times below a threshold value result in a reward paid to the service provider, while response times violating the SLA result in the provider paying a penalty to the client.

We use Httperf [20], an open-loop workload generator, to send requests to browse, buy, or sell shares to the Gold and Silver applications. As shown by an example trace in Fig. 2(b), request-arrivals exhibit time-of-day variations typical of many enterprise workloads, and the number of arrivals changes 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 [21].

Finally, the results presented in this paper assume a session-less workload, that is, there is no state information to be maintained for multiple requests belonging to one user session, and requests are assumed to be independent of each other.

3.3 The Two-Tier System Architecture

Fig. 1(a) shows the virtualized server environment hosting the Gold and Silver services that are, in turn, distributed over the application and database tiers. The application (or database) and its operating system (OS) are encapsulated within a VM, and one or more VMs are supported on each physical host. A dispatcher balances the incoming workload, with arrival rates λ_1 and λ_2 for the Gold and Silver services, respectively, across those VMs running the designated application. Hosts not needed during periods of slow workload arrivals are powered down and placed in the *Sleep* cluster to reduce power consumption. Also, note that some machines, such as Eros, are shared between the two tiers.

The controller, executed on the host Bacchus, aims to meet the SLA requirements of the Gold and Silver services while minimizing the corresponding use of computing resources—the number of hosts and VMs, and the CPU share per VM—at the application tier. A VM's CPU share is specified in terms of an operating frequency. For example, Chronos, with eight CPU cores, each operating at 1.6 GHz, has $8 \times 1.6 = 12.8$ GHz of processing capacity that can be dynamically distributed among its VMs. The ESX server limits the maximum number of cores that a VM can use on a host to four, setting an upper bound of 6 GHz for a VM's CPU share, and reserves a total of 800 MHz of CPU share for system management processes. So, on a machine with eight CPU cores, we can, for example, host a 6 GHz Gold VM that uses four cores, a 3 GHz Gold VM that uses two cores, and a 3 GHz Silver VM that also uses two cores.

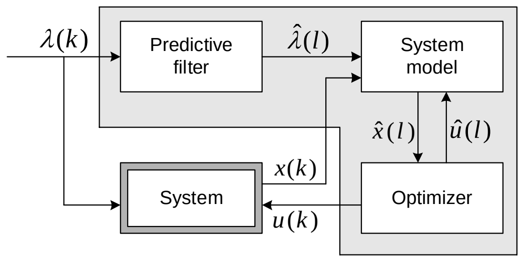
At the database tier, the DB2 databases for the Gold (Silver) service are executed on two 6 GHz VMs, hosted on Apollo and Poseidon, and one 3 GHz VM hosted on Eros. A limited form of resource provisioning is performed at this tier in that physical machines are switched off during periods of light workload. However, the CPU share of an executing VM is not tuned dynamically.

Given the configuration in Fig. 1(a), we can determine the worst-case workload intensity in terms of Gold and Silver request arrivals that can be handled by our system without SLA violations, and therefore, establish an admission policy to cap the maximum request arrival rate. This policy ensures that the system is able to meet target QoS goals given an initial cluster configuration under peak workload. It also prevents against monetary losses and ensures a fair comparison between the controlled and uncontrolled systems. The bottleneck is the database tier, and experiments using Trade6 indicate that a Silver database can process a workload of 190 requests/sec. before becoming the bottleneck resource. This can be inferred from Fig. 5(b), showing the timing behavior of Trade6 when using a 6 GHz VM for the application as well as the database. We see that approximately 190 Silver requests can be processed per second before queueing instability occurs. A simple analysis indicates that the maximum Silver arrival rate tolerated by two 6 GHz VMs and a 3 GHz VM is $190 + 190 + 90 = 470$ requests per second. A similar calculation for the Gold service indicates that the maximum arrival rate tolerated by two 6 GHz VMs and a 3 GHz VM is $33 + 33 + 24 = 90$ requests per second.

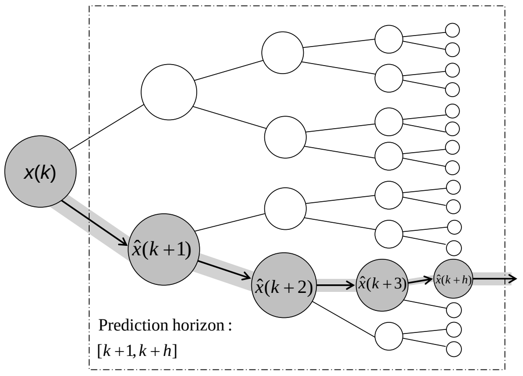
4 Problem Formulation

Given the system model in Fig. 1(a) and the SLA functions in Fig. 2(a), the control objective is to maximize the profit generated by the Gold and Silver services under a time-varying workload by dynamically tuning the following parameters: (1) the number of virtual machines to provision to each application; (2) the number of hosts on which to collocate the virtual machines; (3) the CPU share to be given to each VM; and (4) the number of host machines to power on.

We solve the above problem using limited lookahead control (LLC), a predictive control approach previously introduced in [22]. This method is quite useful when control actions have dead



(a)



(b)

Fig. 3: (a) The schematic of a limited lookahead controller. (b) The state-space trajectory explored by a limited lookahead controller within a horizon of length h. The shaded area shows the best predicted state trajectory $\{\hat{x}(l)|l \in [k+1, k+h]\}$ for a feasible control sequence.

times, such as switching on a server and waiting for the bootstrap routine, and for control actions that must be chosen from a discrete set, such as the number of hosts and VMs to switch on. Fig. 3(a) shows the basic concept where the environment input λ is estimated over the prediction horizon h and used by the system model to forecast future system states $\hat{x}$.

At each time step k, the controller finds a feasible sequence $\{u^*(l)|l \in [k+1, k+h]\}$ of control actions to guide the predicted system state $\{\hat{x}(l)|l \in [k+1, k+h]\}$ within the prediction horizon. The controller identifies the actions that maximize the profit generated by the cluster, as shown by the shaded trajectory in Fig. 3(b). Then, only the first control action in the chosen sequence, $u(k+1)$, is applied to the system and the rest are discarded. The entire process is repeated at time $k+1$ when the controller can adjust the trajectory, given new state information and an updated workload forecast.

The LLC method accommodates control problems posed as set-point regulation or utility optimization under dynamic operating constraints. In *set-point regulation*, key operating parameters must be maintained at a specified level (e.g., an average response time in web servers), and in *utility optimization*, the system aims to maximize its utility (e.g., the profit-maximization problem considered in this paper). The LLC method is conceptually similar to model predictive control (MPC) [23], with some key differences. MPC usually deals with systems operating in a continuous input and output domain whereas LLC can work in a discrete domain. Also, MPC problems are usually computationally expensive and suited for slow-changing processes (e.g., chemical reactions), whereas LLC problems must be solved quickly, given the dynamics of an enterprise workload.

4.1 System Dynamics

A virtual computing cluster is a group of VMs distributed across one or more physical machines, cooperating to host one online service, and the dynamics of a virtual computing cluster for the Gold and Silver applications is described by the discrete-time state-space equation[1]

$$x_i(k+1) = \phi\big(x_i(k), u_i(k), \lambda_i(k)\big) \tag{1}$$

where $x_i(k)$ is the state of the cluster, $\lambda_i(k)$ denotes the environment input, and $u_i(k)$ is the control input. The behavioral model ϕ captures the relationship between the system state, the control inputs that adjust the state parameters, and the environment input.

The operating state of the i^{th} virtual cluster is denoted as $x_i(k) = \big(r_i(k), q_i(k)\big)$ where $r_i(k)$ is the average response time achieved by the cluster and $q_i(k)$ is the number of queued requests. The control input to the i^{th} virtual cluster is denoted as $u_i(k) = \big(N(k), n_i(k), \{f_{ij}(k)\}, \{\gamma_{ij}(k)\}\big)$ where $N(k)$ is the system-wide control variable indicating the number of active host machines, $n_i(k)$ is the number of VMs for the i^{th} service, $f_{ij}(k)$ is the CPU share, and $\gamma_{ij}(k)$ is workload fraction directed to the j^{th} virtual machine. The environmental input $\lambda_i(k)$ is the workload arrival-rate.

An estimate for the environment input λ_i is required for each step along the prediction horizon. We use a Kalman filter [24] to estimate the number of future arrivals because the time-varying nature of the workload makes it impossible to assume an *a priori* distribution.

Since the actual values for the environment input cannot be measured until the next sampling instant, the corresponding system state for time $k+1$ can only be estimated as

$$\hat{x}_i(k+1) = \phi\big(x_i(k), u_i(k), \hat{\lambda}_i(k)\big) \tag{2}$$

We develop ϕ as a difference model for each virtual cluster i using the symbols in Fig. 4 in the following equations.

$$\hat{q}_i(k) = \max\{q_i(k) + \big(\lambda_i(k) - \mu_i(k)\big) \cdot T_s, 0\} \tag{3}$$

$$\hat{\lambda}_i(k) = \hat{\lambda}_i^K(k) + \frac{\hat{q}_i(k)}{T_s} \tag{4}$$

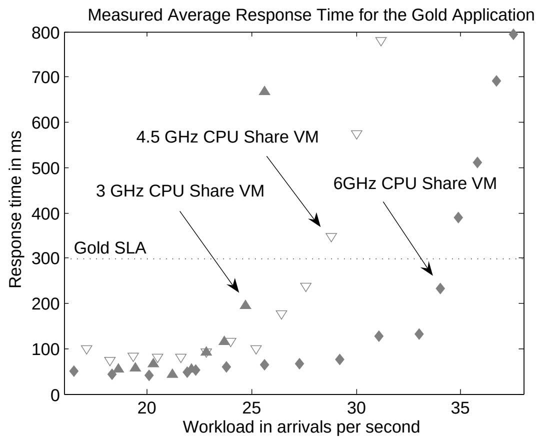
$$\mu_i(k) = \sum_{j=1}^{n_i(k)} \big(\mu_{ij}(k)\big), \quad \mu_{ij}(k) = p(f_{ij}(k)) \tag{5}$$

$$\hat{r}_i(k) = g\big(\mu_i(k), \hat{\lambda}_i(k)\big) \tag{6}$$

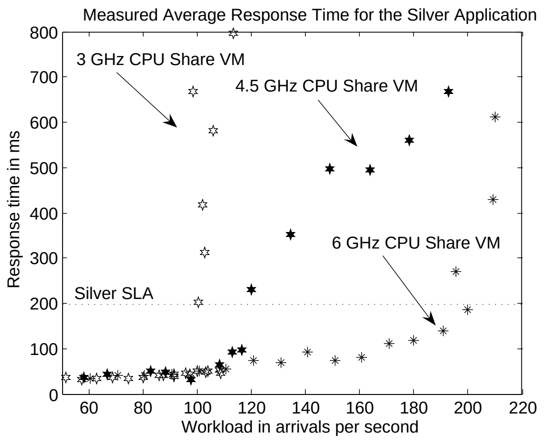
[1]We use the subscript i to denote the i^{th} service class; $i \in \{1, 2\}$ denotes the Gold and Silver services, respectively.

Symbol	Description
	Observable variables
$q_i(k)$	Queue length of the i^{th} virtual cluster
$\lambda_i(k)$	Arrival rate to the i^{th} virtual cluster
$\mu_i(k)$	Processing rate of the i^{th} virtual cluster
$r_i(k)$	Average response time of the i^{th} virtual cluster
T_s	Controller sampling time
	Control variables
$n_i(k)$	Size of the i^{th} virtual cluster
$N(k)$	Number of operational host machines
$f_{ij}(k)$	CPU share of the j^{th} VM in the i^{th} virtual cluster
$\gamma_{ij}(k)$	Fraction of the i^{th} workload to the j^{th} VM

Fig. 4: Explanation of the symbols used in Equations (3)-(6).



(a)

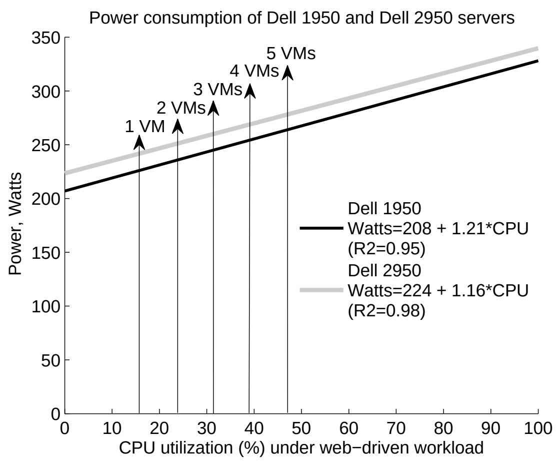


(b)

Fig. 5: The measured average response times for the Gold and Silver applications as a function of a VM's CPU share. Each VM is allocated 1 GB of memory.

Equations (3)-(6) capture the system dynamics over T_s, the controller sampling time. The estimated queue length $\hat{q}_i(k) \geq 0$ is obtained using the current queue length, the incoming workload $\lambda_i(k)$ dispatched to the cluster, and the processing rate $\mu_i(k)$. The estimated workload $\hat{\lambda}_i(k)$ to be processed by a VM is now given by the Kalman estimate $\hat{\lambda}_i^K(k)$ plus the estimated queue length (converted to a rate value).

The processing rate $\mu_i(k)$ of the cluster in (5) is determined by the number of VMs and the CPU share given to each VM. Each VM is given a share of the host machine's CPU, memory, and network I/O, and (5) uses the function $p(\cdot)$ to map the CPU share of the j^{th} VM in the cluster to a corresponding processing rate. This function is obtained via simulation-based learning, specifically by measuring the average response times achieved by a VM, when provided with different CPU shares. The estimated response time, $\hat{r}_i(k)$, output by the function $g(\cdot)$ in (6), maps request processing and arrival rates to an average response time as shown in Fig. 5.



(a)

Host State	Power (Watts) Dell 1950	Power (Watts) Dell 2950
Standby	18	20
Shutdown	213	228
Boot	288	299
1 VM	226	299
2 VMs	237	250
3 VMs	246	260
4 VMs	255	269

(b)

Fig. 6: (a) The power consumed by two models of Dell servers when loaded with VMs hosting an application server; the line is fit from experimentally collected data. (b) The power model used by the controller.

The function $p(\cdot)$ in (5) is obtained by measuring the average response time achieved by a VM, provided with different CPU shares, for $2,000$ requests having roughly a 50/50 mix of browsing and purchasing. Fig. 5 shows the average response times achieved by VMs processing Gold and Silver requests[2]. The response time increases slowly up to a particular arrival rate, and then suddenly increases exponentially. The arrival rate at the "knee" of the graph then determines the VM's maximum processing rate, given that CPU share and memory. For example, Fig. 5(b) shows that a 6 GHz VM with 1 GB of memory can process approximately 190 Silver requests per second before queueing instability occurs. Therefore, we conclude that the VM for the Silver service achieves a maximum processing rate of 190 requests per second when provided a CPU share of 6 GHz. If the CPU share is further constrained, say to 3 GHz, the VM's maximum processing rate decreases and the knee occurs much earlier, at about 90 requests per second.

The power consumption of the host machine is also profiled off-line by placing it in the different operating states shown in Fig. 6(a). Using a clamp-style ammeter, we measured the current drawn by the servers as we instantiated VMs, and loaded each one with an increasing workload intensity before booting the next one. We then multiplied the measured current by the rated wall-supply voltage to compute the power consumption in Watts.

We also computed the power consumed by our servers when booting up, powering down, and in a standby state, during which only the network card is powered on; the power consumption during these two states is included in Fig. 6(b). The Dell PowerEdge 1950 and 2950 servers consume 218 and 228 Watts, respectively, when booting up, and 213 and 228 Watts, respectively, when

[2]Recall that VMs in the database tier are always provided with the maximum CPU share.

powering down. The same machines consume 18 and 20 Watts in a standby state. To determine the cost of operating the host machine during each controller sampling interval T_s, the server's power consumption is multiplied by a dollar cost per kilo-Watt hour over T_s.

Inspecting the data in Fig. 6, we make the following observations regarding the power consumption of the Dell servers.

- An idle machine consumes 70% or more of the power consumed by a machine running at full CPU utilization. Therefore, to achieve maximum power savings on a lightly loaded machine, it is best to redirect the incoming workload and power down the machine. Other power-saving techniques, such as dynamic voltage scaling (DVS), that allow processors to vary their supply voltages and operating frequencies from within a limited set of values [25], have been shown to achieve only about a 10-20% reduction in energy consumption [16, 17].

- The intensity of the workload directed at the VMs does not affect the power consumption and CPU utilization of the host machine. The host machine draws the same amount of current regardless of the arrival rate experienced by the VMs; only the number of VMs running on the host machine affects its power consumption.

- The power consumed by a server is simply a function of the number of VMs instantiated on it at any time instance. Fig. 6(b) shows the simplified power model used by the controller, derived from Fig. 6, which shows power consumption as a function of the number of VMs running on the Dell 1950 and 2950 servers.

The vector $u_i(k)$ to be decided by the controller at sampling time k for each virtual cluster includes $n_i(k) \in Z^+$, the number of VMs to provision, $f_{ij}(k) \in \{3, 4, 5, 6\}$ GHz, the CPU share, and $\gamma_{ij}(k) \in \Re$, the workload fraction to give to the j^{th} VM of the cluster, and $N(k) \in Z^+$, the number of active hosts.

4.2 The Profit Maximization Problem

If $x_i(k)$ denotes the operating state of the i^{th} cluster and $u_i(k) = \big(N(k), n_i(k), \{f_{ij}(k)\}, \{\gamma_{ij}(k)\}\big)$ is the decision vector, the profit generated at time k is given by

$$R\big(x(k), u(k)\big) = \\ \Big(\sum_{i=1}^{2} H_i\big(r_i(k)\big)\Big) - O\big(u(k)\big) - S(\Delta N(k), \Delta n(k)) \quad (7)$$

where the revenue $H_i(r_i(k))$ is obtained from the corresponding SLA function H_i that classifies the average response time achieved per transaction into one of two categories, "satisfies SLA" or "violates SLA", and maps it to a reward or refund, respectively. The power-consumption cost incurred in operating $N(k)$ machines is given by $O(k) = \sum_{j=1}^{N(k)} \big(O\big(N_j(k)\big)\big)$ that sums the power-consumption costs incurred by the host machines in their current operational states, $O(N_j))$. $S(\Delta N(k), \Delta n(k))$ denotes the switching cost incurred by the system due to the provisioning decision. This cost accounts for transient power-consumption costs incurred when powering up/down VMs and their hosts, estimated via the power model shown in Fig. 6, as well as for the opportunity cost that accumulates during the time a server is being turned on but is unavailable to perform any useful service.

Due to the energy and opportunity costs incurred when switching hosts and VMs on/off, excessive switching caused by workload variability may actually reduce profits. Therefore, we convert the profit generation function in (7) to a risk-aware utility function that quantifies a controller's preference between different provisioning decisions. Using such utility functions to aid decision making under uncertainty has been well studied in the context of investment and portfolio management [26].

We augment the estimated environment input $\hat{\lambda}(k)$ with an *uncertainty band* $\hat{\lambda}(k) \pm \varepsilon(k)$, in which $\varepsilon(k)$ denotes the past observed error between the actual and forecasted arrival rates, averaged over a window. For each control input, the next state equation in (2) must now consider three possible arrival-rate estimates, $\hat{\lambda}(k) - \varepsilon(k)$, $\hat{\lambda}(k)$, and, $\hat{\lambda}(k) + \varepsilon(k)$ to form a set of possible future states $\mathbf{X}(k)$ that the system may enter. Given $\mathbf{X}(k)$, we obtain the corresponding set of profits generated by these states as $\mathbf{R}\big(\mathbf{X}(k), u(k)\big)$ and define the quadratic utility function

$$U\big(\mathbf{R}(\cdot)\big) = A \cdot \bar{u}\big(\mathbf{R}(\cdot)\big) - \beta \cdot \Big(\nu\big(\mathbf{R}(\cdot)\big) + \bar{u}\big(\mathbf{R}(\cdot)\big)^2\Big) \tag{8}$$

where $\bar{u}\big(\mathbf{R}(\cdot)\big)$ is the algebraic mean of the estimated profits, $\bar{u}\big(\mathbf{R}(\cdot)\big)$ is the corresponding variance, $A > 2 \cdot \big|\bar{u}\big(\mathbf{R}(\cdot)\big)\big|$ and $\beta \in \Re$ is a *risk preference* factor that can be tuned by the data center operator to achieve the desired controller behavior, from being risk averse ($\beta > 0$), to risk neutral ($\beta = 0$), to risk seeking ($\beta < 0$). Given a choice between two operating states with equal mean profits but with different variances, a risk-averse controller will choose to transition to the state having the smaller variance. The magnitude of β indicates the degree of risk preference.

Given the utility function in (8), we formulate the resource provisioning problem as one of utility maximization.

$$\begin{aligned}
\textbf{Compute:} \quad & \max_{u} \sum_{l=k+1}^{k+h} U\big(\mathbf{R}(\mathbf{X}(l), u(l)), u(l)\big) \qquad (9)\\
\textbf{Subject to:} \quad & N(l) \leq 5, \quad n_i(l) \geq K_{\texttt{min}}, \quad i = 1, 2\\
& \sum_{j=1}^{n_i(l)} \gamma_{ij}(l) = 1, \quad i = 1, 2 \quad \texttt{and}\\
& \sum_{i=1}^{2} \sum_{j=1}^{n_i(l)} e_{ijz}(l) \cdot f_{ij}(l) \leq F^{z}_{\max}, \quad z = 1...5
\end{aligned}$$

where h denotes the prediction-horizon length. As an operating constraint, $N(l) \leq 5$ ensures that the number of operating servers never exceed the total number of servers in the testbed, and $n_i(l) \geq K_{\texttt{min}}$ forces the controller to conservatively operate at least $K_{\texttt{min}}$ VMs at all times in the cluster to accommodate a sudden spike in request arrivals. In our experiments, $K_{\texttt{min}}$ is set to 1. We also introduce a decision variable $e_{ijz}(l) \in \{0, 1\}$ to indicate whether the j^{th} VM of the i^{th} application is allocated to host $z \in [1, 5]$, and the final constraint ensures that the cumulative CPU share given to the VMs does not exceed $F^{z}_{\max}$, the maximum capacity available on host z.

5 Controller Design

The optimization problem in (9) will show an exponential increase in worst-case complexity with an increasing number of control options and longer prediction horizons—the so-called "curse of dimensionality". To tackle this problem, we decompose (9) into smaller sub-problems that can be solved using a distributed control hierarchy. We have developed a two-level control hierarchy, and individual controllers within the hierarchy have the following responsibilities:

- *The L0 Control Layer.* At the lowest level, an L0 controller for each service class decides the CPU share $f_{ij}(\cdot)$ to assign to VMs in the cluster. The small set of discretized choices for possible CPU shares and the small number of components under its control allows each L0 controller to have low execution-time overhead, and therefore, operate frequently, on the order of seconds with a lookahead horizon of $h = 1$.

- *The L1 Control Layer:* An L1 controller with a global view of the system decides $\{n_i(\cdot)\}$, the size of the virtual cluster for the i^{th} application, and $N(\cdot)$, the number of hosts over which to collocate the virtual machines. The L1 controller also determines $\gamma_{ij}(\cdot)$, the workload-distribution factor for each new configuration. The control cost at this level includes the time needed to switch hosts/VMs on or off, and the transient power consumption costs. The lookahead horizon for the L1 controller is determined by the maximum time needed to bring a machine online—two control steps to turn on a host, and one control step to turn on a virtual machine. So the lookahead horizon h is greater than two control steps.

We further reduce the computational overhead of the L1 controller using local-search techniques. For example, to decide $n_i(k)$, the L1 controller searches a bounded neighborhood around an initial "seed" value $\tilde{n}_i(k)$. To obtain this value, the L1 controller divides the estimated arrival rate by the processing rate achieved by a virtual machine when given the maximum CPU share on a host, thereby providing a lower bound on the number of VMs needed to process the incoming workload. The controller then evaluates possible values between $[\tilde{n}_i(k), \tilde{n}_i(k) + b]$, where b is a user-specified value, for the best decision.

Finally, controllers at different levels in the hierarchy can operate at different time scales. Since the L1 controller has larger execution-time overhead, it operates on a longer time scale with sampling times on the order of few minutes. The L0 controllers operate on smaller time scales, on the order of seconds, while reacting quickly to short-term fluctuations in the environment inputs.

6 Experimental Results

Fig. 7 shows the system and controller parameters used in our experiments. The prediction horizons are set to the minimum lengths possible at each level of the control hierarchy. The sampling period for the L0 controller is set to 30 seconds, while the L1 controller sampling period of 2.5 minutes is determined by the maximum time needed to boot a VM as well as the execution-time overhead of the controller itself, which is about 10 seconds. The L1 controller must look ahead at least three time steps to account for the total time to boot a host machine and VM, while the L0 controller can suffice with one step.

The Kalman filter used to estimate the number of request arrivals is first trained using a small portion of the workload (the first 40 time steps) and then used to forecast the remainder of the load during controller execution, as shown in Fig. 8. Once properly trained, the filter provides effective estimates—the absolute error between the predicted and actual values is about 5% for the workloads used in the paper.

Fig. 10 summarizes the performance of a non-risk aware controller with a 3-step lookahead horizon, in terms of energy savings over an uncontrolled system where all available host machines remain powered on, and the number of SLA violations as a percent of the total workload, over a 24-hour period for five different workloads[3]. The energy costs for the controlled and uncontrolled system are estimated at each sampling instance using the model shown in Fig. 6, converting units

[3]All workloads have similar characteristics to that shown in Fig. 2(b), generated by superimposing two 24-hour traces of the WC'98 workload to serve as the Gold and Silver workloads.

Parameter	Value
Cost per KWatt hour	$ 0.3
Time delay to power on a VM	1 min. 45 sec
Time delay to power on a host	2 min. 55 sec
Prediction horizon	L1: 3 step, L0: 1 step
Control sampling period, non-risk aware	L1: 150 sec., L0: 30 sec.
Control sampling period, risk-aware	L1: 180 sec., L0: 30 sec.
Initial configuration for Gold service	3 VMs
Initial configuration for Silver service	3 VMs

Fig. 7: The simulation parameters.

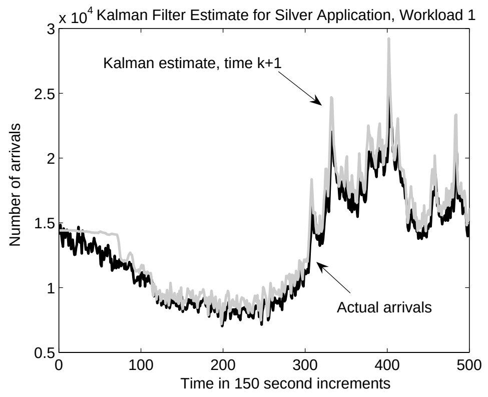
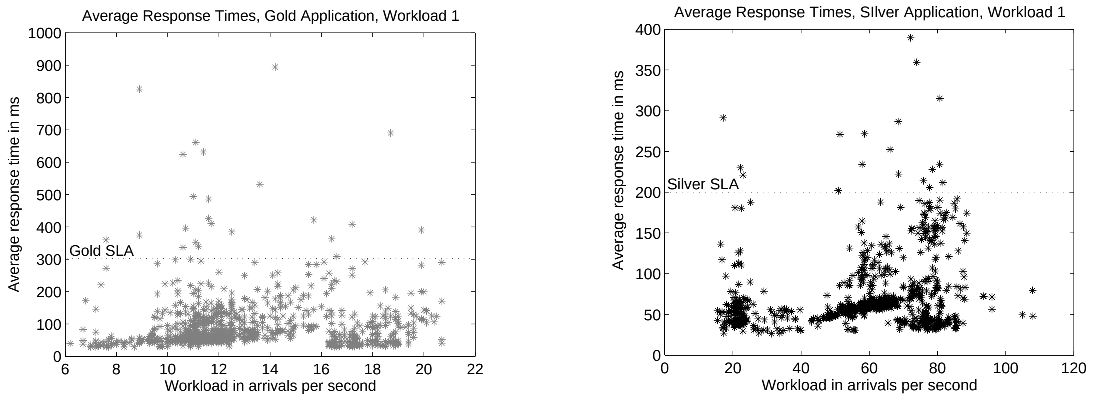


Fig. 8: Workload 1 for the Silver application and the corresponding predictions provided by the Kalman filter.



(a) (b)

Fig. 9: The measured average response times for the Gold and Silver applications for Workload 1.

of power to units of energy over each sampling period. An uncontrolled system incurs some SLA violations due to normal variability in application performance–about 1% to 5% of the total requests made to the system.

The uncontrolled system allocates a fixed number of VMs as noted in Fig. 7 to each application

Workload	Total Energy Savings	% SLA Violations (Gold)	% SLA Violations (Silver)
Workload 1	19%	3.1%	2.5%
Workload 2	19%	1.3%	1.2%
Workload 3	14%	1.9%	0.5%
Workload 4	26%	1.2%	0.3%
Workload 5	30%	1.4%	0.6%

Fig. 10: Control performance of a non-risk aware controller, in terms of the average energy savings and SLA violations, for five different workloads.

Workload	#VMs (Gold)	#VMs (Silver)	Total CPU (Gold)	Total CPU (Silver)
Workload 1	1.9	2.0	8.1 GHz	8.2 GHz
Workload 2	2.0	2.0	8.4 GHz	8.6 GHz
Workload 3	1.5	2.3	5.7 GHz	9.7 GHz
Workload 4	1.8	1.3	6.0 GHz	6.3 GHz
Workload 5	1.5	1.8	5.6 GHz	6.7 GHz

Fig. 11: Average resource allocations (number of virtual machines and total CPU cycles per second) per time instance for five different workloads.

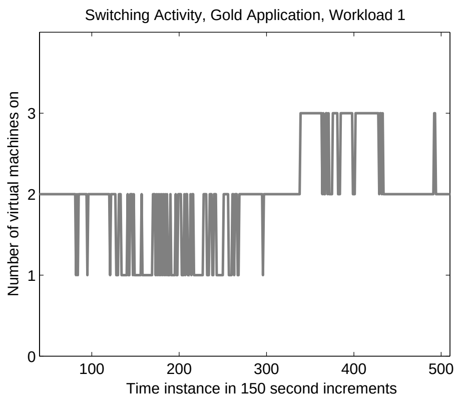
such that SLA goals can be met under the worst-case workload arrival rate. Fig. 9 shows the average measured response times for the Gold and Silver applications under a sample workload. Figs. 13 and 12 show the average CPU share and number of VMs, respectively, allocated by the controller at each time instance for the same workload. Average resource allocations for five workloads are summarized in Fig. 11.

6.1 Effects of Risk-aware Parameter Tuning

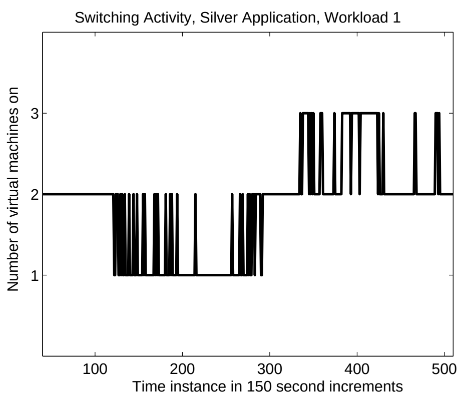
The execution-time overhead of a risk-aware controller is slightly higher than the non-risk aware controller due to the increased complexity of the control problem. Therefore, the sampling period of the L1 controller increases to 180 seconds while that of the L0 controller stays at 30 seconds. Note that only the L1 controller is risk aware in our design due to the switching costs incurred at that level.

Fig. 14 summarizes the performance of risk-averse ($\beta = 2$) and risk-neutral ($\beta = 0$) controllers, in terms of the energy savings achieved over an uncontrolled system, over a 24-hour period[4]. Fig. 15 summarizes the SLA violations, both the total number and as a percentage of the Gold and Silver requests. Although the energy savings decrease slightly from the risk-neutral to the risk-averse controller, averaging 30% and 27%, respectively, the number of SLA violations drops

[4]Risk-averse values of $\beta = 1$ to $\beta = 5$ were tested during experimentation; a value of $\beta = 2$ produced the best results under all workloads in terms of energy savings and SLA violations.

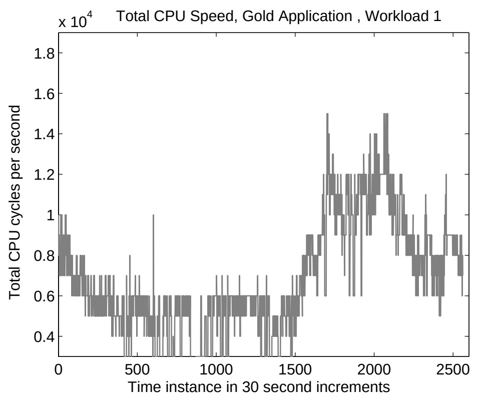


(a)

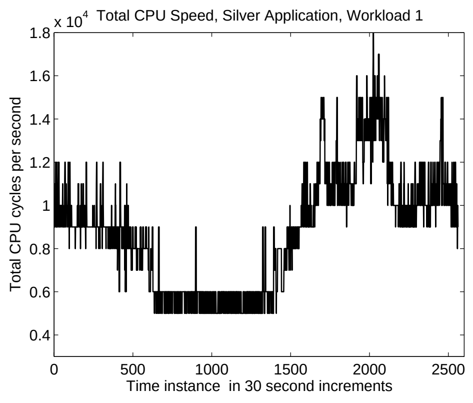


(b)

Fig. 12: The number of virtual machines assigned to the Gold and Silver applications processing Workload 1.



(a)



(b)

Fig. 13: CPU cycles assigned to the Gold and Silver applications under Workload 1.

significantly in the case of the risk-averse controller, reducing violations by an average of 52% when compared to the risk-neutral case. This is due to the conservative manner in which the risk-averse controller switches machines. Fig. 16 shows that the risk-averse L1 controller reduces the switching of hosts by an average of 65% over its risk-neutral counterpart. This reduced switching activity has the following benefits to system performance. First, conservative switching will typically result in extra capacity, should the actual number of request arrivals exceed the Kalman filter estimates. This effect is evident from the reduced number of SLA violations incurred by the risk-averse controller. Second, reduced switching activity mitigates the detrimental effects of repeated power cycling that reduce the reliability of the host machine in the form of disk failures [27, 28].

After testing risk-averse controllers having β values of integers between 1 and 5, we conclude that a value of $\beta = 2$ results in the best control performance in terms of energy savings and SLA violations. Energy savings and SLA violations improve from $\beta = 1$ to $\beta = 2$. Increasing β above

Workload	Total Energy Savings (Risk Neutral)	Total Energy Savings (Risk Averse)
Workload 6	41%	38%
Workload 7	19%	16%

Fig. 14: Energy savings achieved by risk-neutral and risk-averse controllers for two different workloads

Workload	Total SLA Violations (Risk Neutral)	Total SLA Violations (Risk Averse)	% Reduction in Violations
Workload 6	17,753 (1.4%)	8,879 (0.7%)	50%
Workload 7	36,906 (2.8%)	17,135 (1.3%)	54%

Fig. 15: SLA violations incurred by risk-neutral and risk-averse controllers for two different workloads

Workload	Num. Times Hosts Switched (Risk Neutral)	Num. Times Hosts Switched (Risk Averse)	% Reduction in Switching Activity
Workload 6	40	16	60%
Workload 7	39	12	69%

Fig. 16: Host machine switching activity induced by risk-neutral and risk-averse controllers for two different workloads

2 simply maintains or even slightly reduces the energy savings while resulting in a greater number of SLA violations.

6.2 Optimality Considerations

In an uncertain operating environment, control decisions cannot be shown to be optimal since the controller does not have perfect knowledge of future environment inputs. Furthermore, control decisions are made from a discrete set of inputs chosen from a localized search area explored within a limited prediction horizon. The final series of tests compare our sub-optimal controller against an "oracle" controller with perfect knowledge of future environment disturbances, representing the most feasible approximation of an optimal controller.

Fig. 17 compares the performance of an oracle controller against a risk-neutral controller ($\beta = 0$) and our best-performing risk-averse controller with $\beta = 2$. The oracle consumes about 3% more power than a risk-neutral controller, but incurs 64% fewer SLA violations. It equals the power consumption of the risk-neutral controller, but incurs 23% fewer SLA violations. The oracle also reduces the switching of hosts by an average of 19% from the risk-averse controller. The results indicate that the performance of the lookahead controller depends on the accuracy of the workload predictions, but a properly tuned risk-aware controller can reduce the number of SLA violations

Controller	Total Energy Savings	Total SLA Violations	Num. Times Hosts Switched
Risk-neutral	19%	36,906 (2.8%)	40
Risk-averse	16%	17,135 (1.3%)	16
Oracle	16%	13,181(1.0%)	13

Fig. 17: Energy savings and SLA violations of the oracle, risk-neutral, and risk-averse controllers

and excessive power cycling within the virtualized system.

7 Controller Scalability

When control inputs must be chosen from discrete values, the LLC problem in (9) will show an exponential increase in worst-case complexity with increasing system size, in terms of hosts and VMs, and longer prediction horizons. Our current testbed with 1,024 possible control options for VM assignments to servers requires about 10 seconds of controller execution time. The number of options increases exponentially with each server added to the system[5]. Fig. 18 summarizes the number of control options as the system grows from 5 to 15 servers, and the execution time of a risk-aware controller managing the system.

For larger clusters, we use a neural network (NN) to learn the tendencies of the controller, in terms of its decision making, via offline simulations. At run time, given the current state and environment inputs, the NN provides an approximate solution, which is used as a starting point around which to perform a local search of $\pm$ 1 VM per host to obtain the final control decision. Such a method of improving performance obtained from a neural network can be found in previous work [29]. Initial results using a back-propagation, feed-forward NN, implemented using the Matlab Neural Net Toolbox, reduce the execution time of a 4-step lookahead controller for 10 host machines to about 30 seconds, as shown in Fig. 18[6]. The network is trained using a set of 1,000 training points collected over two workloads similar to that shown in Fig. 2(b). While the energy savings of the neural network controller decrease slightly, from 29% to 27%, as compared to the

[5]The size of the search space can be estimated via v^N, where $v = 4$ is the maximum number of VMs per host, and $N = 5$ is the number of controllable host machines.

[6]Neural net results are pending for a system size of 15 hosts.

Controllable System Size	Control Options	Avg. LLC Exe. Time	Avg. NN Exe. Time
5 Hosts	1x10^3	$<$ 10 sec.	–
10 Hosts	1x10^6	2 min. 30 sec.	30 sec.
15 Hosts	1x10^9	30 min.	–

Fig. 18: Execution times incurred by the controller for larger clusters.

exploratory LLC controller, the number of SLA violations improves, from 5,116 to 4,502.

8 Conclusion

We have implemented and validated a LLC framework for dynamic resource provisioning in a virtualized computing environment. The problem formulation includes switching costs and explicitly encodes the notion of risk in the optimization problem. Experiments using time-varying workload traces and the Trade6 enterprise application show that a system managed using LLC saves, on average, 22% in power consumption costs over a 24-hour period, when compared to a system without dynamic control while still maintaining QoS goals. When the incoming workload is noisy, we conclude that a risk-aware controller with $\beta = 2$ provides superior performance compared to a risk-neutral controller by reducing both SLA violations and host switching activity. Finally, we use concepts from approximation theory to further reduce the computational burden of controlling large systems, and show that a NN-based scheme achieves energy savings comparable to the baseline controller while incurring significantly less computational overhead.

References

[1] Q. Li and M. Bauer, "Understanding the performance of enterprise applications," in *Proc. of IEEE Conference on Systems, Man and Cybernetics*, June 2005, pp. 2825–29.

[2] R. Smith, "Power companies consider options for energy sources," *The Wall Street J.*, p. A.10, Oct. 29 2007.

[3] F. Darema, "Grid computing and beyond: The context of dynamic data driven applications systems," *Proc. of the IEEE*, vol. 93, no. 3, pp. 692–97, March 2005.

[4] D. A. Menasc$\acute{e}$ and V. A. F. Almeida, *Capacity Planning for Web Services*. Upper Saddle River, NJ: Prentice Hall, 2002.

[5] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *Proc. of USENIX Sym. on Internet Technologies and Systems (USITS)*, March 2003.

[6] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration," in *Proc. of the IEEE Wkshp. on Virtualization Technology in Dist. Sys.*, Nov. 2006, p. 7.

[7] P. Garbacki and V. Naik, "Efficient resource virtualization and sharing strategies for heterogeneous grid environments," in *Proc. of the IEEE Sym. on Integrated Network Management*, May 2007, pp. 40–49.

[8] R. Nathuji, C. Isci, and E. Gorbatov, “Exploiting platform heterogeneity for power efficient data centers,” in *Proc. IEEE Intl. Conf. on Autonomic Computing (ICAC)*, Jun. 2007, p. 5.

[9] B. Lin and P. Dinda, “Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling,” in *Proc. of the IEEE/ACM Conf. on Supercomputing*, Nov. 2005, p. 8.

[10] R. Nathuji and K. Schwan, “Virtualpower: coordinated power management in virtualized enterprise systems,” in *Proc. of the ACM SIGOPS Sym. on Op. Sys. principles*, Oct. 2005, pp. 265–278.

[11] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, “I/o scheduling and xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms,” in *Proc. of the ACM SIGOPS Sym. on Op. Sys. principles*, Jun. 2007, pp. 126–136.

[12] G. Khanna, K. Beaty, G. Kar, and A. Kochut, “Application performance management in virtualized server environments,” in *Proc. of the IEEE Network Ops. and Mgmt. Sym.*, Apr. 2006, pp. 373–381.

[13] C. Tsai, K. Shin, J. Reumann, and S. Singhal, “Online web cluster capacity estimation and its application to energy conservation,” *IEEE Trans. on Parallel and Dist. Sys.*, vol. 18, no. 7, pp. 932–945, Jul. 2007.

[14] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess, “Server virtualization in autonomic management of heterogeneous workloads,” in *Proc. of the IEEE Sym. on Integrated Network Management*, May 2007, pp. 139–148.

[15] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, “On the use of fuzzy modeling in virtualized data center management,” in *Proc. IEEE Intl. Conf. on Autonomic Computing (ICAC)*, Jun. 2007, pp. 25–35.

[16] J. Kephart, H. Chan, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, “Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs,” in *Proc. IEEE Intl. Conf. on Autonomic Computing (ICAC)*, Jun. 2007, pp. 145–154.

[17] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, “Ensemble-level power management for dense blade servers,” in *Proc. of the IEEE Sym. on Computer Architecture*, Jun. 2006, pp. 66–77.

[18] C. Lefurgy, X. Wang, and M. Ware, “Server-level power control,” in *Proc. IEEE Conf. on Autonomic Computing*, Jun. 2007, p. 4.

[19] E. Pinheiro, R. Bianchini, and T. Heath, *Dynamic Cluster Reconfiguration for Power and Performance*. Kluwer Academic Publishers, 2003.

[20] D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance," *Perf. Eval. Review*, vol. 26, pp. 31–37, Dec. 1998.

[21] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," Hewlett-Packard Labs, Technical Report HPL-99-35R1, Tech. Rep., Sept. 1999.

[22] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online control for self-management in computing systems," in *Proc. IEEE Real-Time & Embedded Technology & Application Symp. (RTAS)*, 2004, pp. 368–376.

[23] J. M. Maciejowski, *Predictive Control with Constraints*. London: Prentice Hall, 2002.

[24] A. C. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge, UK: Cambridge University Press, 2001.

[25] *Enhanced Intel SpeedStep Tecnology for the Intel Pentium M Processor*, Intel Corp., 2004.

[26] T. Copeland and J. Weston, *Financial Theory and Corporate Policy, 3rd, ed.* Addison-Wesley, 1988.

[27] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning, "Paraid: A gear-shifting power-aware raid," *ACM Trans. on Storage*, vol. 3, p. 13, Nov. 2007.

[28] G. Hughes and J. Murray, "Reliability and security of raid storage systems and d2d archives using sata disk drives," *ACM Trans. on Storage*, vol. 1, pp. 95–107, Feb. 2005.

[29] D. Kusic and N. Kandasamy, "Approximation modeling for the online performance management of distributed computing systems," in *Proc. of IEEE Intl. Conf. on Autonomic Computing (ICAC)*, June 2007, p. 23.