Risk-Aware Limited Lookahead Control for Dynamic Resource Provisioning in Enterprise Computing Systems

Dara Kusic and Nagarajan Kandasamy
Department of Electrical and Computer Engineering
Drexel University
Philadelphia, PA, 19104, USA
kusic@drexel.edu, kandasamy@ece.drexel.edu

***Abstract*— Utility or on-demand computing, a provisioning model where a service provider makes computing infrastructure available to customers as needed, is becoming increasingly common in enterprise computing systems. Realizing this model requires making dynamic, and sometimes risky, resource provisioning and allocation decisions in an uncertain operating environment to maximize revenue while reducing operating cost. This paper develops an optimization framework wherein the resource provisioning problem is posed as one of sequential decision making under uncertainty and solved using a limited lookahead control scheme. The proposed approach accounts for the switching costs incurred during resource provisioning and explicitly encodes risk in the optimization problem. Simulations using workload traces from the Soccer World Cup 1998 web site show that a computing system managed by our controller generates up to 20% more revenue than a system without dynamic control while incurring low control overhead.**

***Key words*: Utility computing, resource provisioning, sequential optimization, limited lookahead control**

I. Introduction

Utility computing is an emerging provisioning model where a service provider makes computing resources available to the customer as needed, and charges them for specific usage rather than a flat rate. It is becoming increasingly common in enterprise computing, and is sometimes used for the consumer market as well as Internet services, Web site access and file sharing. Realizing the utility computing model requires making resource provisioning and allocation decisions in a dynamic operating environment to maximize revenue while reducing operating costs [1] and it is highly desirable for such systems to manage themselves, given only high-level objectives by administrators. Such *autonomic computing systems* aim to achieve quality-of-service (QoS) objectives by adaptively tuning key operating parameters with minimal human intervention [2] [3]. Also, as these applications continue to grow in complexity, ad hoc and heuristic-based approaches to performance management will quickly become insufficient. Recent research efforts have therefore focused on using concepts from control theory and dynamic programming as the theoretical basis for achieving self-managing behavior in computing applications and systems [4] [5].

The provisioning problem of interest is to decide an optimal allocation of computing resources to multiple client QoS classes under a dynamic workload. This discrete optimization problem may need continuous re-solving with observed environmental events such as time-varying client workload patterns and computer failures. Since the underlying control set is discrete, traditional optimal control techniques [6] cannot be applied directly and a closed-form expression for a feedback-control map cannot be established.

This paper develops an optimization framework to enable self-managing behavior in an enterprise computing system supporting multiple QoS or client classes wherein the resource provisioning and management problem is posed as one of sequential optimization under uncertainty and solved using a *limited lookahead control (LLC)* approach, a control and optimization technique developed in [5] [7]. The control actions governing system operation are obtained by optimizing its forecast behavior, described by a mathematical model, for the specified QoS criteria over a limited prediction horizon. The LLC concept is adopted from *model predictive control* [8], sometimes used to solve optimal control problems for which classical feedback solutions are extremely hard or impossible to obtain.

The LLC approach is a practical option for enforcing self-managing behavior in resource provisioning applications for the following reasons: (1) *Systematic use of predictions*: Future environmental inputs as well as the future implications of current control actions on application performance are taken into account during optimization. Also, actions such as dynamic provisioning of computing resources often incur substantial dead time (the delay between a control input and the corresponding response), requiring *proactive control* where control inputs must be provided in anticipation of future changes in operating conditions. (2) *Robust operation in uncertain environments*: LLC is robust with respect to dynamic environmental disturbances (e.g., time-varying workload patterns, and hardware and software failures). (3) *Optimization in the discrete domain*: LLC is applicable to applications where control or tuning options must be chosen from a finite set, and also accommodates multi-variable optimization problems. (4) *Explicit constraint handling*: LLC allows for optimization problems to be solved under explicit and dynamic operating constraints.

The resource provisioning problem addressed in this paper assumes a computing system supporting three QoS classes using dedicated server clusters for each, where incoming traffic (client requests) is dispatched to the appropriate cluster. To maximize the revenue generated by this system under a time-varying workload, the controller must solve a discrete and dynamic optimization problem to decide: (1) the number of computers to provision per service cluster, (2) the operating frequency at which to uniformly operate the servers in a cluster and (3) the number of computers to power down to reduce energy consumption. This paper builds on the LLC framework developed in [5] to solve the above provisioning problem in an uncertain and dynamic operating environment, and makes the following innovative contributions:

- The revenue maximization problem is solved for multiple client classes whose service-level agreements (SLA) follow a non-linear pricing strategy.
- Workload forecasting errors and inaccuracies are explicitly discounted along the lookahead horizon to diminish their effect on control performance.
- The LLC problem formulation models the various switching costs associated with provisioning decisions. Revenue may be lost while a computer is being switched between clients, if, for example, the corresponding computer will be unavailable for some time duration while a different operating system and/or application is loaded to service the new client. Other switching costs include the time delay incurred when powering up an idle computer and the excess energy consumed during this transient phase.
- In an operating environment where the incoming workload is noisy and highly variable, switching computers excessively between clusters may actually reduce the revenue generated, especially in the presence of the switching costs described above. Thus, each provisioning decision made by the controller is risky and we explicitly encode risk in the problem formulation using preference functions to order possible controller decisions.

Simulations using workload traces from the France World Cup 1998 (WC98) web site [9] show that a computing system, managed using the proposed LLC method, generates up to 20% more revenue per day when compared to a system operating without dynamic control, and with very low control overhead. We also characterize the effects of varying key controller parameters such as the prediction horizon and the risk preference function on its performance.

The paper is organized as follows. Section II discusses related work while Section III describes system modeling assumptions and the basic LLC concepts. Section IV formulates the resource provisioning problem and Section V describes the controller design and Section VI presents experimental results evaluating controller performance. We conclude the paper in Section VII with a discussion on future work.

II. Related Work

We now briefly review prior research addressing resource provisioning problems in utility computing models. In [10], a homogeneous computing cluster is operated energy efficiently using both predictive and reactive resource management techniques. Processors are provisioned using predicted workload patterns, and a feedback controller sets their aggregate operating frequency, reacting to short-term workload variations. The system model includes switching costs incurred when powering computers on and off. Resource provisioning in a multi-tier web environment is addressed in [11] while [12] develops a controller to allocate a finite number of resources among multiple applications to maximize a user-defined utility for the entire system. A reactive technique is proposed in [13] to allocate resources among multiple client classes, balance the load across servers, and handle dynamic fluctuations in service demand while satisfying client SLAs and producing differentiated service.

While [10] uses a simple SLA to optimize system performance around a desired average response time without any service differentiation, we use a more complex SLA — a stepwise non-linear pricing strategy [14] that affords a service provider greater returns for response times approaching zero, and diminishing returns for slower response times. Our controller design also accounts for forecasting errors and explicitly encodes risk during decision making. The work reported in [11] and [12] does not consider switching costs incurred during resource provisioning or the effect of forecasting errors on controller performance.

The authors of [15] develop a framework for making provisioning decisions using techniques from inventory control and supply-chain management. They argue the case for developing good workload forecasting algorithms while carefully analyzing the effects of forecasting errors on provisioning algorithms.

Re-distributing incoming workload between busy and idle clusters has been studied in [16], similar to the classic problems of load balancing [17] [18]. In [16], clusters lending resources are called donors, while those in need of servers are classified as beneficiaries. This work does not address switching costs or anticipate future workload demands while making allocation decisions.

Power consumption costs in a distributed system form a significant portion of the overall operating cost. Much work has been devoted to power-efficient data processing in computing systems, ranging from small mobile devices to large server farms [19] [20] [21]. Typical techniques include dynamic voltage scaling [20] and/or methods that take advantage of idle periods and parallelism within the workload [21].

III. Preliminaries

This section presents the system model and the pricing strategy used to differentiate the client classes of interest. We also discuss basic LLC concepts.

A. System Model

Fig. 1 shows the architecture of a computing system comprising *homogeneous* servers with identical processing capacities. The system supports three independent client classes, termed Gold, Silver, and Bronze, using a dedicated cluster for each class.

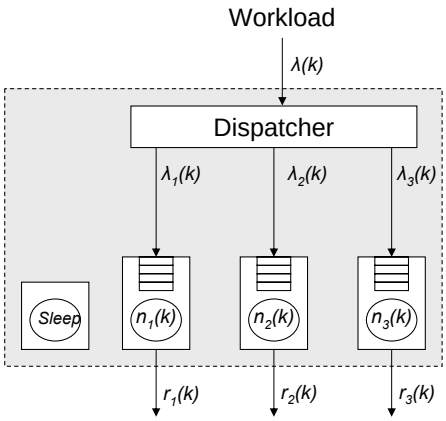

Fig. 1. The system model where $n_1(k)$, $n_2(k)$, $n_3(k)$, denote the number of servers within the Gold, Silver, Bronze, and clusters, respectively. A *sleep* cluster holds servers in a powered-down state

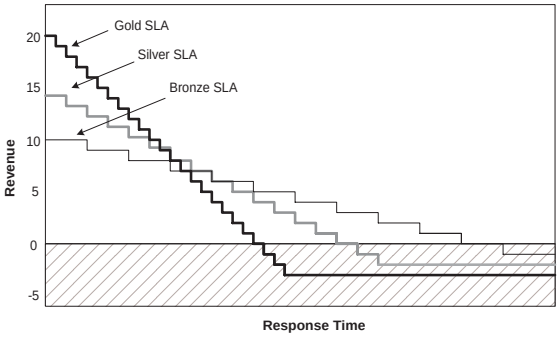

Fig. 2. A stepwise non-linear pricing strategy describing service level agreements for three client groups

Requests from the Gold, Silver, and Bronze clients arrive with time-varying rates $\lambda_1(k)$, $\lambda_2(k)$, and $\lambda_3(k)$ respectively, and are routed to the appropriate cluster. We do not assume an *a priori* stochastic distribution for the request arrival rate but estimate the arrival patterns using online forecasting techniques (see Section III-C). Within each cluster, requests are dispatched to individual servers using a simple round-robin scheme for load balancing and processed in first-come first-serve fashion. The system also maintains a *Sleep* cluster for computers that have been powered down to reduce power consumption.

Each server is assumed to support *dynamic voltage scaling* by varying both its supply voltage and operating frequency from a limited set of values [22]. The overall power consumption of the cluster at any given time instant includes a constant base cost for each operating computer (due to the energy requirements of its power supply, hard disk, etc.) and the dynamic power consumed to process the workload. So, if the time required to process a request while operating at the maximum frequency f_{max} is γ, then the corresponding processing time while operating at some frequency $f(k)$ is $\gamma \cdot \big(f(k)/f_{max}\big)$ where $f(k)/f_{max}$ is the scaling factor. This simple relationship between the operating frequency and the corresponding processing time has been widely used in previous work on dynamic voltage scaling; see, for example [23].

B. The Pricing Strategy

The various client classes are differentiated by a stepwise non-linear pricing graph, similar to the one shown in Fig. 2. This exemplifies a commonly used pricing strategy [14] where Gold customers expect to receive the best service, in terms of the shortest response time, and pay premium prices for the best service. Silver clients pay less for slightly downgraded service while Bronze clients pay the least for basic service. Clients are billed for the actual response time delivered by the provider and credited for response times violating the SLA.

The controller also enforces a policy to drop requests arriving above the maximum rate specified by the SLA, thereby preventing against losses due to inadequate capacity and providing for a fair comparison between controlled and uncontrolled systems. Given an initial cluster configuration in terms of the number of servers, the maximum arrival rate that can be handled by this cluster is obtained as follows. We assume the worst-case scenario — the maximum service time per request — and determine the number of such requests that can be processed by the cluster while achieving a zero (or break even) revenue return for the provider.

C. System Dynamics

Let the Gold, Silver, and Bronze service clusters be numbered 1, 2, and 3, respectively. The following discrete-time state-space equation describes the continuous dynamics of a cluster $i \in \{1, 2, 3\}$.

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

where $x_i(k) \in \Re$ is the state of a cluster at time step k, and $u_i(k) = \big(n_i(k), f_i(k)\big)$ is the control (or decision) vector, where $n_i(k) \in \jmath^+$ is a positive integer denoting the number of servers belonging to cluster i and $f_i(k)$ is the operating frequency of that cluster. The environment input $\omega_i(k) = \big(\lambda_i(k), \gamma_i(k)\big)$ is a vector comprising the request arrival rate $\lambda_i(k) \in \Re$ and the average per-request processing time $\gamma_i(k) \in \Re$. The system state $x_i(k) = (r_i(k), O_i(k))$ is defined by the achieved response time $r_i(k)$ and the corresponding operating cost $O_i(k)$.

The system model ϕ captures the relationship between the observed system parameters and the control inputs that adjust these parameters. We obtain ϕ from first principles using the following queuing model.

$$q_i(k+1) = q_i(k) + \big(\lambda_i(k) - p_i(k)\big) \cdot t_s \tag{2}$$

$$p_i(k) = \frac{n_i(k)}{\gamma_i(k)} \cdot \frac{f_i(k)}{f_{max}} \tag{3}$$

$$r_i(k+1) = \gamma_i(k) + \frac{q_i(k+1)}{p_i(k)} \tag{4}$$

$$O_i(k) = n_i(k) \cdot \big(c_0 + c_1 \cdot \ f_i(k)^3\big) \tag{5}$$

The queue length at time step $(k+1)$ is given by the current queue length $q_i(k)$ and the arrival and processing rates $\lambda_i(k)$

and $p_i(k)$, respectively. The processing rate $p_i(k)$ is given by the cluster capacity, in terms of the number of servers $n_i(k)$, and the average service time $\gamma_i(k)$. The response time, $r_i(k+1)$ is a sum of the average service time $\gamma_i(k)$ plus waiting time in the queue $\big(q_i(k+1)/p_i(k)\big)$. The operating cost $O_i(k)$ of the cluster is obtained in terms of its overall power consumption. Each server incurs a base (or idle) power consumption cost c_0 and a dynamic cost dependent upon the cubic operating frequency $f_i(k)^3$ and a scaling constant, c_1, as described in [24]. Table I in Section VI lists the specific parameters for c_0 and c_1.

The above equations adequately model the system dynamics of a cluster when the workload is mostly CPU intensive, i.e., the processor on each server is the bottleneck resource. Typically, this is true for web and e-commerce servers where both the application and data can be fully cached in memory, thereby minimizing (or eliminating) hard disk accesses.

D. Control Concepts

The overall control objective is to maximize revenue generation over all service classes while minimizing the operating costs related to power consumption. Thus, the variables to be decided by the controller at each sampling interval k are the number of servers $n_i(k)$ and the overall operating frequency $f_i(k)$ of these servers for cluster $i \in 1, 2, 3$. This control problem is posed as one of sequential optimization under uncertainty. Relevant parameters $\hat{\lambda}_i(k)$ and $\hat{\gamma}_i(k)$ of the operating environment are estimated, and used by the system model discussed in Section III-C, to forecast future behavior over the specified prediction horizon, h. At each time step j, the controller finds a feasible sequence $\{(n_i^*(j), f_i^*(j))|j \in [k+1, k+h]\}$ of resource provisioning decisions within the prediction horizon for each cluster, i. Then, only the first move is applied to the system and the whole optimization procedure is repeated at time $k+1$ when the new system state is available.

The lookahead nature of LLC requires that the environment inputs $\lambda_i(k)$ and $\gamma_i(k)$ be estimated for each step within the prediction horizon. Since the current values of the environment inputs cannot be measured until the next sampling instant, the corresponding system state can only be estimated as follows:

$$\hat{x}_i(k+1) = \phi\big(x_i(k), u_i(k), \hat{\omega}_i(k)\big) \tag{6}$$

Here, $\hat{\omega}_i(k) = \big(\hat{\lambda}_i(k), \hat{\gamma}_i(k)\big)$, where $\hat{\lambda}_i(k)$ and $\hat{\gamma}_i(k)$ denote the estimated request arrival rate and processing time, respectively. We use an ARIMA model [25], implemented via a Kalman filter [26], to estimate $\hat{\lambda}_i(k)$. The service time for requests is estimated using an exponentially-weighted moving-average (EWMA) filter specified by the expression $\hat{\gamma}_i(k+1) = \pi \cdot \gamma_i(k) + (1-\pi) \cdot \hat{\gamma}_i(k-1)$ where π is a smoothing constant.

IV. Problem Formulation

This section formulates the revenue maximization problem under uncertain operating conditions. When workload parameters are highly dynamic and variable, the corresponding forecast values typically have errors or inaccuracies. Therefore, we address how our design tackles such forecasting errors using two complementary techniques: (1) encoding the risk inherent to control decisions into the problem formulation itself; and (2) including a discounting factor during lookahead optimization to mitigate forecasting errors on control decisions.

A. The Revenue Maximization Problem

The revenue $R_i(x_i(k), n_i(k), f_i(k))$ generated by a computing cluster i at time k, given the current state $x_i(k)$, cluster capacity $n_i(k)$, and operating frequency $f_i(k)$ is:

$$R_i\big(x_i(k), n_i(k), f_i(k)\big) = l_i\big(r_i(k)\big) - O\big(n_i(k), f_i(k)\big) - S\big(\Delta n_i(k)\big) \tag{7}$$

where $l_i(r_i(k))$ is the money generated by cluster i, as per the SLA function shown in Fig. 2, for the achieved average response time $r_i(k)$ and $O\big(n_i(k), f_i(k)\big)$ denotes the operating cost. The switching cost incurred by the cluster due to the provisioning decision is denoted by $S(\Delta n_i(k))$. This cost is a function of the number of servers moved between different client clusters, including the sleep cluster, and accounts for power consumption costs incurred while starting up computers, plus the revenue lost when the computer is unavailable to perform any useful service for some duration (e.g., a different operating system and/or application must be loaded to service the client).

The revenue maximization problem for the three service classes $i \in \{1, 2, 3\}$ is then posed as follows:

Compute:

$$\max_{N,F} \sum_{j=k+1}^{k+h} \sum_{i=1}^{3} \hat{R}_i\big(\hat{x}_i(j), n_i(j), f_i(j)\big) \tag{8}$$

Subject to:

$$n_i(j) \geq K_{min} \ \forall j$$

The formulation in (8) is solved over all possible options in the finite control set N (number of servers) and F (set of operating frequencies) under explicit and dynamic constraints, reflecting current operating conditions such as the number of available computers and the overall energy budget for the system. Human operators may also enforce specific controller behavior using constraints. For example in (8), the controller is instructed to maintain a minimum cluster size K_{min} at all times to accommodate sudden (and rare) bursts of high traffic caused by flash crowds [27], thereby ensuring conservative performance.

B. Encoding Risk in the Cost Function

Many real workload traces, including those from the WC98 web site and others [28] [29] show high variability within short time periods, where characteristics such as request arrival rates change quite significantly and quickly - usually in the order of a few minutes. Therefore, forecasts of such environmental parameters typically incur errors or inaccuracies where the observed values differ from the corresponding predictions.

When arrivals are noisy, the controller may constantly switch computers between clients in an effort to maximize

revenue generation. However, due to switching costs, excessive switching is risky and may actually reduce the revenue generated. Therefore, in an uncertain operating environment where control decisions are risky, the optimization problem formulated in (8) may not be the most appropriate way to measure a controller's preference between different provisioning decisions. Therefore, we explicitly encode risk in the problem formulation using *preference functions* to order possible controller decisions. Using such functions to aid decision making under uncertainty has been previously studied in the context of Investment Analysis [30] as well as dynamic programming [31].

In noisy conditions, the estimated workload parameter $\hat{\lambda}_i(k)$ typically has an *uncertainty band* $\hat{\lambda}_i(k) \pm \varepsilon_i(k)$ around it, where $\varepsilon_i(k) = \|\lambda_i(k) - \hat{\lambda}_i(k)\|$ denotes the (average) observed error between the actual and forecasted values. Then, given a current state $x_i(k)$ and possible control inputs $n_i(k)$ and $f_i(k)$, we can generate a set of estimated future states, each corresponding to some $\hat{\lambda}_i(k)$ value in the uncertainty band. We can now augment the state-generation equation in (6) with the following new expression that considers three possible arrival-rate estimates, $\hat{\lambda}_i(j) - \varepsilon_i(j)$, $\hat{\lambda}_i(j)$, and, $\hat{\lambda}_i(j) + \varepsilon_i(j)$, for each step j within the prediction horizon h to form a corresponding set of possible future states.

$$\hat{X}_i(j) = \left\{ f(\hat{x}_i(j-1), u_i(j), ([\hat{\lambda}_i(j) + \varepsilon_i], \hat{\gamma}_i(j))) \right\} \quad (9)$$
$$\varepsilon_i \in \{-\varepsilon_i(j), 0, \varepsilon_i(j)\}$$

Given this collection of states obtained within the uncertainty band $\hat{\lambda}_i(k) \pm \varepsilon_i(k)$, we define $\mu(\hat{X}_i(j))$ as the algebraic mean of the revenue generated by the states $\forall \hat{x}_i(j) \in \hat{X}_i(j)$. We now associate the following *quadratic preference* or *utility function* with the set $\hat{X}_i(j)$:

$$U_i(\hat{X}_i(j)) = A \cdot \mu(\hat{X}_i(j)) - \beta \cdot \left(\nu(\hat{X}_i(j)) + \mu(\hat{X}_i(j))^2\right) \quad (10)$$

where $A > 2 \cdot \mu(\hat{x}_i(j))$ is a constant and β is the *risk preference* factor. Here, μ denotes the mean revenue generated by the states in $\hat{X}_i(j)$ and ν denotes the variance between the different revenue values. Equation (10) is a mean-variance model commonly used for stock portfolio management under risky conditions [30]. The risk preference factor can be tuned to induce desired controller behavior including being risk averse ($\beta > 0$), risk neutral ($\beta = 0$), or risk seeking ($\beta < 0$). (Intuitively, a risk-averse controller, when given a choice between two provisioning decisions generating nearly equal mean revenue but with markedly different variance, will prefer the decision with smaller variance.) We now modify the revenue maximization problem in (8) to one of *utility maximization*.

Compute:

$$\max_{N,F} \sum_{j=k+1}^{k+h} \sum_{i=1}^{3} U_i\left(\hat{X}_i(j), n_i(j), f_i(j)\right) \quad (11)$$

Subject to:

$$n_i(j) \geq K_{min} \ \forall j$$

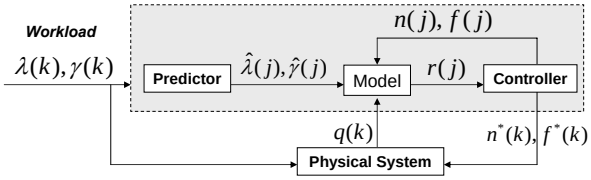

Fig. 3. Block diagram of the controller

C. Adding a Discounting Factor

In an uncertain operating environment, we expect the workload and environmental parameter estimations, and thus, the estimated system states to become increasingly inaccurate as we go deeper into the prediction horizon, degrading control performance. One can mitigate the effects of forecasting inaccuracies on control performance by associating a discounting factor with the cost function. Returning to the formulation in (8), let us associate a factor α^j such that $0 < \alpha < 1$, with the cost function R. If $\alpha < 1$, then future costs matter less than the same costs incurred at the present time, i.e., states further out in the prediction horizon have less impact on the current control action. Equation (12) shows the discounting factor, α, applied to the original formulation in (8).

Compute:

$$\max_{N,F} \sum_{j=k+1}^{k+h} \sum_{i=1}^{3} \alpha^j \hat{R}_i\left(\hat{X}_i(j), n_i(j), f_i(j)\right) \quad (12)$$

Subject to:

$$n_i(j) \geq K_{min} \ \forall j$$

V. Controller Design

The functional components of the controller are shown in Fig. 3. The predictor forecasts the request arrival rate and average processing time $\hat{\lambda}_i(j)$ and $\hat{\gamma}_i(j)$, respectively, within the prediction horizon $k+1 \leq j \leq k+h$. The system model computes the response time $r_i(j)$ for each cluster, when provided options $n_i(j)$ and $f_i(j)$ from the control set N and

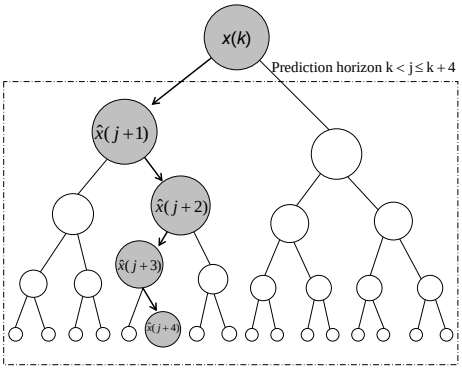

Fig. 4. A possible control trajectory selected within a 4-step prediction horizon

F, where N is the number of servers and F is the set of six frequencies from which the controller can choose to apply uniformly to servers within a cluster. The controller applies the control inputs $n_i^*(k)$ to each cluster, dictating its size, and $f_i^*(k)$, specifying the operating frequency for the servers in the cluster. The control input $n_i^*(k)$ causes servers to be shifted between client clusters, or to and from the power-saving Sleep cluster.

When control inputs must be chosen from a set of discrete values, the various LLC formulations in (8), (11), and (12), 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." Since the execution time available for the controller is often limited by hard bounds, it is necessary to consider the possibility that we may have to deal with suboptimal solutions. For adaptation purposes, however, it is not critical to find the global optimum; a feasible suboptimal solution will suffice. We would still like to use the available time exploring the most promising solutions leading to optimality. At each sampling instant, the proposed controller uses the current state information and the estimated environmental parameters to perform a *localized search* of possible state trajectories within the prediction horizon. A possible control trajectory within a 4-step predictive search space is shown in Fig. 4.

The controller searches within a localized neighborhood where the number of states is bounded by the constraint shown in (13) to limit the control overhead. The constraint specifies that up to two machines may be moved between work clusters per control input. The only relaxation of the constraint targets the Sleep cluster, which may lend or receive more than two machines over all service clusters per controller sampling time.

$$\left|n_i(k) - n_i(k-1)\right| \leq 2 \tag{13}$$

VI. Experimental Results

The controller, simulated for a computing system of 30 homogeneous servers, delivers promising initial results, generating up to 20% more revenue over a 24-hour period of operation when compared to an uncontrolled system. The results presented in this section were obtained via simulations written in Matlab 7.0.4 and executed on a 3 GHz Intel Pentium 4 processor with 1 GB RAM.

TABLE I

Simulation parameters

Parameter	*Value*
Operating frequency	1 GHz
Power consumed at max. operating frequency	130 Watts
Power consumed by idle server, c_0	50 Watts
Scaling constant for dynamic power consumed, c_1	80 Watts
Cost per kilo-Watt hour	$ 0.17
Reboot cost, in energy consumed	$ 0.00024
Time to reboot	30 sec.
Control sampling period	30 sec.
Initial configuration, num. servers	10/10/10
Kalman training length	50 samples
Total number of samples	2782 samples

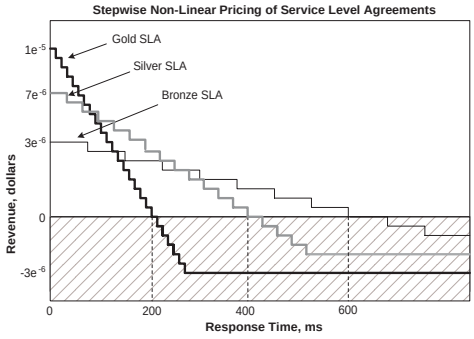

Fig. 5. The pricing strategy for the Gold, Silver, and Bronze clients used in the simulations

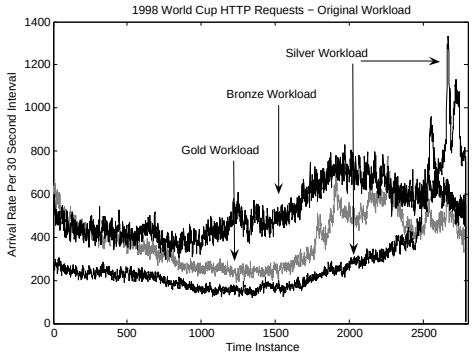

Fig. 6. The synthesized workload corresponding to the Gold, Silver, and Bronze clients, derived from WC'98 traces

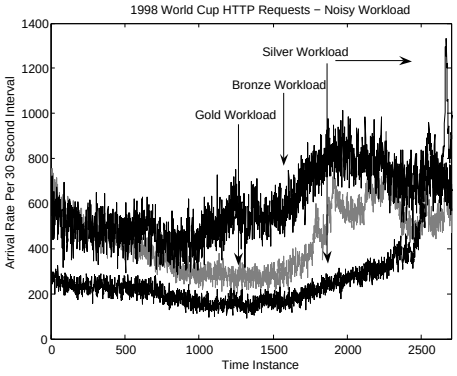

Fig. 7. A somewhat noisier workload generated by adding Gaussian noise to the workload in Fig. 6

A. Simulation Setup and Workload Generation

The pricing structure in Fig. 5 shows the revenue generated per client request in terms of the response time achieved by the system. The revenue generated per request is in the order of micro-dollars.

We simulated the simultaneous workload generated by the Gold, Silver, and Bronze clients in an Enterprise Computing system using three HTTP-request traces selected from different 24-hour periods during the 1998 Soccer World Cup. Fig. 6 shows the resulting synthesized client workload. Note that this workload displays an appreciable amount of inherent noise and variability.

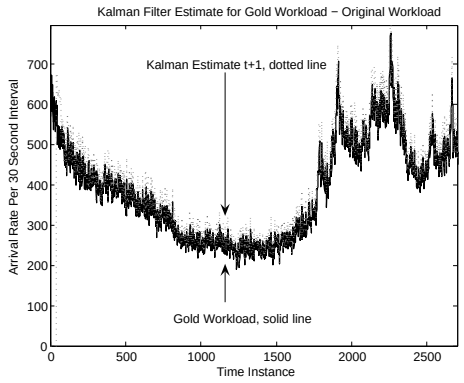

Fig. 8. The actual and predicted workload for the Gold service cluster, no noise added

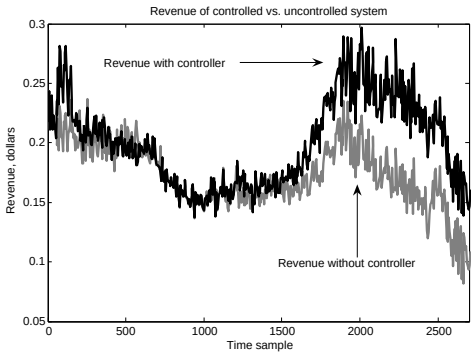

Fig. 9. Revenue comparison between the controlled and uncontrolled system

Parameters of the Kalman filter were first tuned using an initial portion of the workload in Fig. 6, and then used to forecast the remainder of the load during controller execution. The Kalman filter produces good estimates of the arrival rate with a small amount of error, as illustrated for the Gold cluster in Fig. 8. As shown in Fig. 7, we also generated a somewhat noisier workload by adding random Gaussian noise to the original workload in Fig. 6. The Kalman filter also produces reasonable estimates on this workload.

To generate the processing times for individual requests within the arrival sequence in Fig. 7, we assumed processing times within a range that reflects both static and dynamic page requests [32]. We generated a virtual store comprising 10,000 objects, and the time needed to process an object request was randomly chosen from a uniform distribution between [1.0, 43.0] ms. The distribution of individual requests within the arrival sequence was determined using two key characteristics of most web workloads:

- *Popularity*: It has been observed that a few files are extremely popular while many others are rarely requested, and that the popularity distribution commonly follows Zipf's law [33]. Therefore, we partitioned the virtual store in two — a "popular" set with 1000 objects receiving 90% of all requests, and a "rare" set containing the remaining objects in the store receiving only 10% of requests.
- *Temporal locality*: This is the likelihood that once an object is requested, it will be requested again in the near future. In many web workloads, temporal locality follows a lognormal distribution [34].

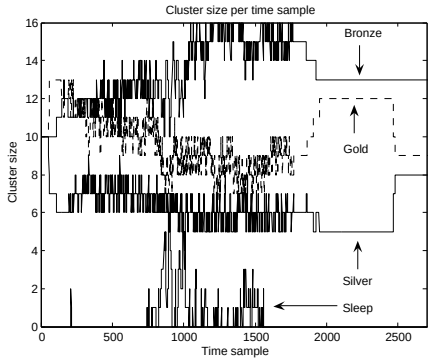

Fig. 10. Time-varying cluster sizes for the Gold, Silver, Bronze, and Sleep clusters as decided by the controller

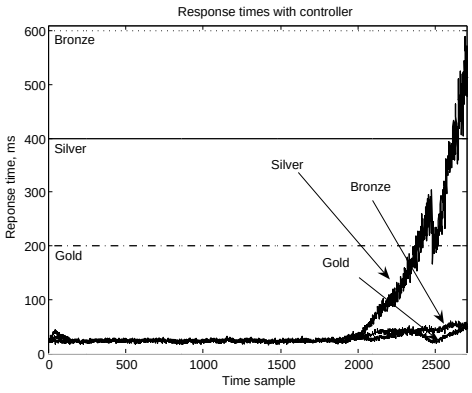

Fig. 11. The average response time achieved by a system with LLC

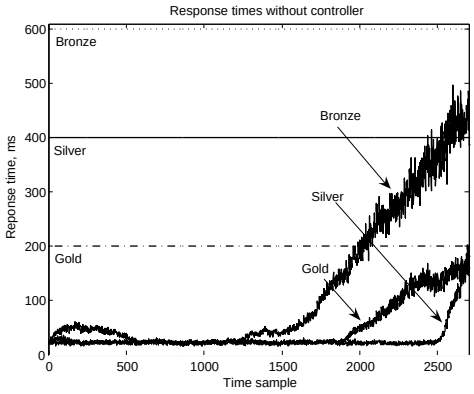

Fig. 12. The average response time achieved by a system without LLC

B. Controller Performance

Fig. 9 summarizes the performance of a 3-step controller over the workload shown in Fig. 7 for a 24-hour period. Each time instant on the X-axis represents a 30-second sampling period, and the revenue is generated in terms of micro-dollars per request, generating about $500 in revenue over a day. The switching activity is shown in Fig. 10 where computers are

dynamically provisioned between the various clusters. Fig. 11 shows the average response times achieved by the controlled system. Note that at time $k > 2500$, the controller attempts to correct for the sudden surge in Silver work requests (see Figs. 6 and 10). Note that the effects of bursty workloads can be mitigated by further relaxing the constraint in (13) (at the expense of increased control overhead).

The average response times achieved by an uncontrolled system is shown in Fig. 12. Comparing this figure to Fig. 11, it is clear that the controller attempts to maximize revenue by moving computers from the Bronze cluster to the Gold and Silver clusters at the expense of increased response times within the Bronze cluster.

C. Effects of Parameter Tuning

We now compare the revenue earned by various configurations of the online controller. Again, our base case is a computing system without online control. We assume an average request processing time of 23 ms for each of the three client classes, and show experimental results for the following controller types:

- A controller that only moves servers between clusters without tuning the operating frequency, i.e., the servers operate at frequency f_{max}. Both 2- and 3-step prediction horizons are considered for this controller.
- A controller that moves servers between clusters while simultaneously deciding the operating frequency of each cluster. Again, both 2- and 3-step prediction horizons are considered.

Table I shows the base controller parameters used within our simulations. We test the controller in operating environments with and without the addition of noise, i.e., using both traces in Figs. 6 and 7. Our simulations indicate that the above controllers achieve at least 10% in revenue gains and in some cases slightly more than 20% in revenue gains over an uncontrolled system.

Our simulations also sought to quantify the effects of incorporating operating frequency into the control set. The controller may now choose from a set of six different operating frequencies. Assuming a maximum frequency $f_{\mathsf{max}} = 1GHz$ for each server, the various operating frequencies were obtained as $\{0.5 \cdot f_{\mathsf{max}}, 0.6 \cdot f_{\mathsf{max}}, ..., 1.0 \cdot f_{\mathsf{max}}\}$. Adding operating frequencies to the control set has the following advantages when provisioning computing resources. First, frequency scaling can be done almost instantaneously with little transient switching cost. Second, changing the operating frequency allows a cluster to control costs via a cubic relationship between operating frequency and power consumption [24] while still retaining the resource in an active state to handle unpredicted bursts in the workload. In fact, Figs. 13-16 show that a controller the provisions both the cluster size and operating frequency achieves 2-5% more in revenue gains than a controller that provisions only for cluster size.

First, we explore the effect of varying β, the risk preference factor previously introduced in (10), on revenue generation. Given the workload in Figs. 6 and 7, the corresponding performance plots in Figs. 13 and 14 show that a risk-averse

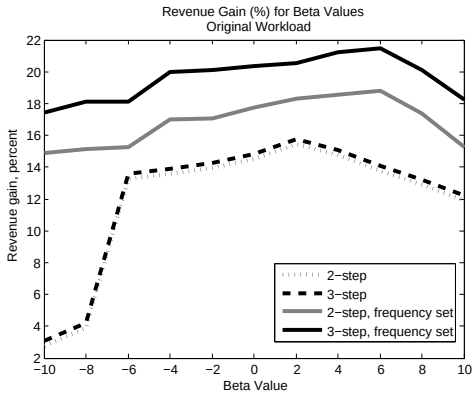

Fig. 13. Comparison of controller performance for various β values using the workload in Fig. 6

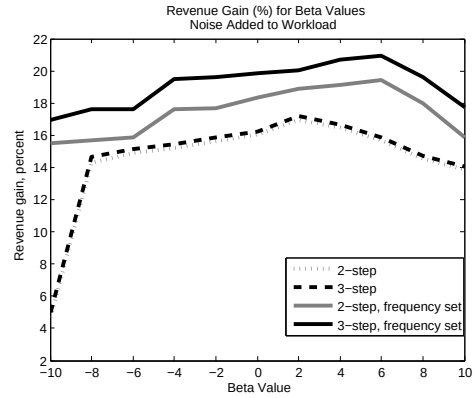

Fig. 14. Comparison of controller performance for β values using the noisy workload in Fig. 7

controller $\beta > 0$ performs better than a risk-seeking one $\beta < 0$, an effect that is somewhat more pronounced when the workload has increased variability. We also see that the performance of the controller drops off for larger β values, both positive and negative. Large negative values for β cause the controller take more risks during resource provisioning, thereby constantly moving servers between the various clusters and reacting quickly to variations in the incoming workload. As noted in Section I, this constant movement of computing resources between clusters can result in lost revenue in the presence of switching costs. On the other hand, when β assumes large positive values, the controller becomes increasingly risk averse to the point of shunning provisioning decisions altogether. This again results in lost revenue.

For the workload in Fig. 6, we see from Fig. 13 that the best performance is obtained around $\beta = 5$. Similarly, for the noisier workload in Fig. 7, the best performance is obtained around a β value slightly greater than 5.

We also quantified the effects of α, the discounting factor previously introduced in (12), on controller performance. Again, considering the workload in Figs. 6 and 7, Figs. 15 and 16 show that varying α has little effect on the various controllers tested in terms of the generated revenue. This is likely due to the limited number of tuning options available to the controller and the short prediction horizon used in our

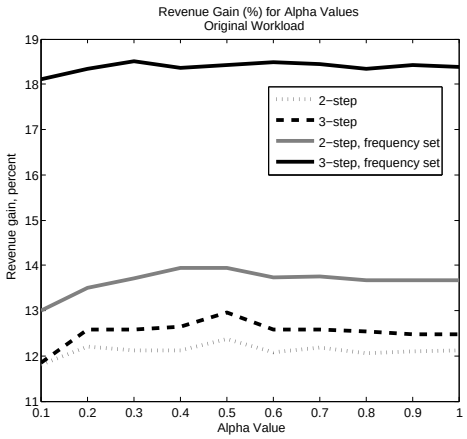

Fig. 15. Comparison of controller performance for a range of α values using the workload in Fig. 6

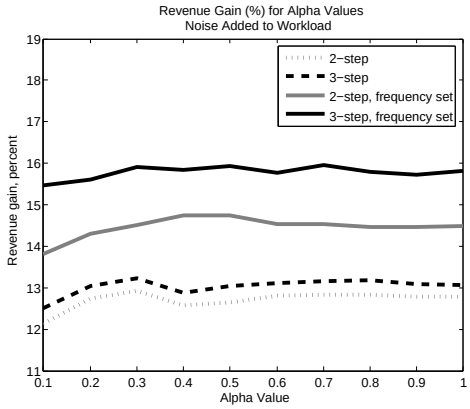

Fig. 16. Comparison of controller performance for various α values using the workload in Fig. 7

simulations.

Finally, it is interesting to note the effect of workload variability and noise on the performance of a controller that uses the discounting factor as a means to tackle forecasting errors. There is a significant performance gap between a controller subjected to the workload in Fig. 6 when compared to the same controller subjected to the noisier workload in Fig. 7. Figs. 15 and 16 show that the control performance of a 3-step controller with frequency tuning suffers substantially as more noise is introduced into the workload. However, the performance of the utility maximizing controller with the risk preference function (Equation 11) is encouraging. The corresponding utility function (see Equation 10) is formulated in such a way to mitigate noise in the predicted workload parameters. Comparing Figs. 13 and 14, we see that controller performance not affected as much by the increased noise in the workload.

To summarize, all our test cases indicate that a 3-step predictive controller outperforms one with a prediction horizon of 2 steps. Expanding the decision space to include frequency tuning also has a positive effect on controller performance. A controller that includes operating frequency as part of its control set generates 2%-7% more revenue than one that provisions only servers between clusters. Adapting the operating frequency also has the advantage of being nearly instantaneous, and with little long-term impact on the wear-and-tear imposed on the servers.

The execution time of a 2-step controller is 0.1 seconds seconds while that of a 3-step controller is about 1.4 seconds. For a sampling interval of 30 seconds, the corresponding control overhead for a 3-step LLC is about 4.6%. We also tested a 4-step controller and found the execution overhead to be about 25 times that of a 3-step controller while achieving only a fraction of a percent in additional revenue gains. Therefore, we conclude that a prediction horizon of 3 steps is sufficient for good control performance.

D. Optimality

In an uncertain operating environment, controller decisions cannot be shown to be optimal since the controller does not have perfect knowledge of future environment parameters, and control decisions are made via a localized search within a limited prediction horizon. Therefore, we must be satisfied with good sub-optimal decisions.

Our final series of tests were aimed at comparing a practical controller implementation against an "oracle" that has perfect knowledge of future environment disturbances. We noted that even if workload predictions are completely error free, the revenue generated is only slightly higher than the practical case where only imperfect predictions can be obtained. We selected the best performing controller after extensive tuning — a 3-step frequency-selecting controller having a risk-averse behavior defined by $\beta = 5$ — and found that having perfect workload predictions increased revenue gains by only 1%, from a 20.5% gain with prediction errors to 21.3% with no errors.

VII. Conclusions

We have presented an optimization framework to enable self-managing behavior in an enterprise computing system serving multiple client classes. The proposed lookahead control algorithm aims to maximize revenue generation by dynamically provisioning computing resources between multiple client clusters. The problem formulation includes switching costs and explicitly encodes the risk associated with making provisioning decisions in an uncertain operating environment. Experiments using the WC98 workload indicate that, over the operating period of one day, our controller generates 10% to 20% more revenue when compared to a computing system without dynamic control, and with a very low cost of control. A risk-averse controller achieved higher revenue gains than a risk-seeking one, and a 3-step lookahead controller capable of tuning both the cluster capacity and operating frequency performed the best over all the controllers that were tested.

References

[1] A. Sahai, S. Singhal, V. Machiraju, and R. Joshi, "Automated policy-based resource construction in utility computing environments," in *Proc. IEEE Symposium on Network Operations and Management*, April 2004, pp. 381–393.

[2] R. Murch, *Autonomic Computing*. Upper Saddle River, NJ: IBM Press/Prentice-Hall, 2004.

[3] IBM, "An architectural blueprint for autonomic computing," IBM Thomas J. Watson Research Center, Tech. Rep., Oct. 2004.

[4] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley-Interscience, 2004.

[5] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online control for self-management in computing systems," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004, pp. 368–375.

[6] A. Bryson, *Applied Optimal Control: Optimization, Estimation, and Control*. Hemisphere Publishing, 1984.

[7] N. Kandasamy, S. Abdelwahed, and J. Hayes, "Self-optimization in computer systems via on-line control: Application to power management," in *Proc. IEEE Int'l Conf. Autonomic Computing*, May 2004, pp. 54–61.

[8] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[9] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," Hewlett-Packard Labs, Tech. Rep., Sept. 1999.

[10] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proc. ACM SIGMETRICS*, June 2004, pp. 303–314.

[11] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Proc. IEEE Intl. Conf. on Autonomic Computing (ICAC)*, June 2005, pp. 217–228.

[12] M. Bennani and D. Menascé, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. IEEE Intl. Conf. on Autonomic Computing (ICAC)*, June 2005, pp. 229–240.

[13] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster based web services," IBM Research Labs, Tech. Rep., May 2003.

[14] J. Zhang, T. Hamalainen, and J. Joutsensalo, "Optimal resource allocation scheme for maximizing revenue in the future ip networks," in *Proc. 10th Asia-Pacific Conf. on Comm. and 5th Intl. Sym. on Multi-Dimensional Mobile Comm.*, Sept 2004, pp. 128–132.

[15] J. Hellerstein, K. Katircioglu, and M. Surendra, "A framework for applying inventory control to capacity management for utility computing," Thomas J. Watson Research Center, Tech. Rep., Oct. 2004.

[16] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf, "Analysis of cycle stealing with switching cost," in *Proc. ACM SIGMETRICS*, June 2003, pp. 184–195.

[17] D. Eager, E. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. on Software Engineering*, vol. 12, no. 5, pp. 662–675, May 1986.

[18] A. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *J. of the ACM*, vol. 32, no. 2, pp. 445–65, May 1986.

[19] T. Mudge, "Power: A first-class architectural design constraint," *IEEE Computer*, pp. 52–58, Apr. 2001.

[20] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, "Power-aware qos management in web servers," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002, pp. 63–72.

[21] R. Mishra, N. Rastogi, Z. Dakai, D. Mossé, and R. Melhem, "Energy-aware scheduling for distributed real-time systems," in *Proc. IEEE Parallel and Distributed Processing Sym. (IPDPS)*, Apr. 2003, pp. 63–72.

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

[23] P. Pillai and K. G. Shin, "Real-time voltage scaling for low-power embedded operating systems," in *Operating Syst. Principles (SOSP)*, 2001, pp. 89–102.

[24] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Proc. 2nd Workshop on Power-Aware Computing Systems (held with HPCA)*, Feb. 2002, pp. 126–37.

[25] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*. Prentice Hall: Upper Saddle River, NJ, 1999.

[26] K. Brammer and G. Siffling, *Kalman-Bucy Filters*. Artec House: Norwood, MA, 1989.

[27] R. Morris and D. Lin, "Variance of aggregated web traffic," in *Proc. IEEE INFOCOM*, May 2000, pp. 360–366.

[28] J. Judge, "A model for the marginal distribution of aggregate per second http request rate," in *Proc. IEEE Wrkshp. on Local and Metropolitan Area Networks*, Nov. 1999, pp. 29–36.

[29] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," in *Proc. ACM SIGMETRICS*, 1996, pp. 126–137.

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

[31] Y. Bar-Shalom, "Stochastic dynamic programming: Caution and probing," *IEEE Trans. on Automatic Control*, vol. 26, no. 5, pp. 1184–1195, Oct. 1981.

[32] C. Rusu, A. Ferreira, C. Scordino, A. Watson, R. Melhem, and D. Mossé, "Energy-efficient real-time heterogeneous server clusters," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006.

[33] M. Arlitt and C. Williamson, "Web server workload characterization: The search for invariants," in *Proc. ACM SIGMETRICS*, 1996, pp. 126–37.

[34] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proc. ACM SIGMETRICS*, 1998, pp. 151–160.