Adaptive Performance Control of Computing Systems via Distributed Cooperative Control: Application to Power Management in Computing Clusters

Mianyu Wang*, Nagarajan Kandasamy†, Allon Guez‡, and Moshe Kam*
Department of Electrical and Computer Engineering
Drexel University
Philadelphia, PA 19104
Email: *{mianyu, kam}@minerva.ece.drexel.edu †kandasamy@ece.drexel.edu ‡guezal@drexel.edu

***Abstract*— Advanced control and optimization techniques offer a theoretically sound basis to enable self-managing behavior in distributed computing models such as utility computing. To tractably solve the performance management problems of interest, including resource allocation and provisioning in such distributed computing environments, we develop a fully decentralized control framework wherein the optimization problem for the system is first decomposed into sub-problems, and each sub-problem is solved separately by individual controllers to achieve the overall performance objectives. Concepts from optimal control theory are used to implement individual controllers. The proposed framework is highly scalable, naturally tolerates controller failures, and allows for the dynamic addition/removal of controllers during system operation. As a case study, we apply the control framework to minimize the power consumed by a computing cluster subject to a dynamic workload while satisfying the specified quality-of-service goals. Simulations using real-world workload traces show that the proposed technique has very low control overhead, and adapts quickly to both workload variations and controller failures.**

I. Introduction

Distributed systems hosting business, e-commerce, and scientific applications must typically satisfy stringent quality-of-service (QoS) requirements while operating in highly dynamic environments. For example, the workload to be processed may be time-varying, and hardware and software components may fail during system operation. To achieve the desired QoS in such systems and applications, numerous performance-related parameters must be continuously tuned. As these systems become more complex, it is highly desirable for them to be largely *autonomic* or *self-managing*, requiring only high-level guidance from administrators [9].

Heuristic or rule-based policies to enforce self-managing behavior, though simple to implement, require a great deal of expert knowledge, and are tightly coupled to the specific application. It is also difficult to analyze their performance in terms of convergence and stability properties. On the other hand, advanced control and mathematical programming techniques offer a theoretical basis to enable adaptation in distributed applications, and have the following advantages over ad hoc methods: (1) one can systematically pose various adaptation problems of interest within the same basic framework and (2) the feasibility of the adaptation algorithms with respect to application QoS goals may be verified prior to deployment.

Using concepts from *optimal control theory*, this paper develops a decentralized optimization framework to enforce self-managing behavior in distributed computing systems. The proposed framework has very low control overhead, is highly scalable, and fault adaptive, in terms of seamlessly tolerating individual controller failures. As a case study, we tackle the problem of power management in a server cluster operating under a time-varying workload.

Control theory is a well-established engineering methodology that has recently been used to achieve adaptive behavior in various computing applications [11]. For example, feedback control has been applied to problems such as task scheduling [18], bandwidth allocation and QoS adaptation in web servers [2], load balancing in e-mail and file servers [17] [11], network flow control [22], and power management [19] [25]. Assuming a linear time-invariant system, an unconstrained state space, and a continuous input and output domain, a closed-loop feedback controller is designed as an open-loop system transfer function under stability and sensitivity requirements. However, in more complex control problems, a pre-specified plan, i.e, the feedback map, is inflexible and does not adapt very well to constantly changing operating conditions. Moreover, the cost of the control actions themselves is not taken into account.

Classical feedback control is not suitable for applications exhibiting *hybrid* behavior (comprising both discrete-event and time-based dynamics), and where control or tuning options must be chosen from a finite set at any given time. Therefore, receding horizon (RH) control concepts, borrowed from model predictive control [20], have been used manage such applications [1] [14] [15]. These methods take into account multi-objective cost functions and dynamic operating constraints while optimizing application performance.

The above-discussed feedback and RH-based control techniques have been used to manage applications executing on

1-4244-0175-5/06/$20.00 ©2006 IEEE.

single or stand-alone processors. A major limitation of these methods is the centralized nature of the controller itself. In a distributed system where the performance of many hardware and software components must be simultaneously managed to achieve system-wide QoS goals, a centralized controller needs to explore a large number of possible tuning options, thereby incurring significant control overhead. Therefore, such designs do not scale well in a distributed setting. Moreover, centralized designs have a single point of failure.

This paper develops a *fully decentralized and cooperative control framework* to enforce adaptive behavior in a distributed computing environment comprising multiple components. The overall performance management problem is decomposed into a set of corresponding sub-problems and each one is mapped to an underlying component. Individual controllers, implemented within each component, solve their respective sub-problems in a cooperative fashion such that system-wide QoS goals are satisfied. The proposed framework has the following desirable characteristics.

- The framework is *highly scalable* since: (1) Each local controller within a component has very low control overhead; (2) Controllers can be *dynamically composed*, i.e., added or removed, during system operation; and (3) Cooperation between controllers is achieved with no inter-controller communication. For example, in the power management case study, the addition or failure of a controller is inferred by other controllers via shared system-state variables without exchanging messages.
- The control framework is *fault adaptive* in that it tolerates individual component and controller failures in seamless fashion. When a failure is detected (or inferred) by other controllers, they adapt their behavior accordingly, while still aiming to satisfy system QoS goals.

Each individual controller within a component is designed using concepts from optimal control theory [16]. Multiple QoS objectives and operating constraints are explicitly specified in the cost function and solved for a limited lookahead control horizon. Future environment inputs as well as the future implications of current control actions on application performance are taken into account during optimization. The optimal tuning decisions are obtained by solving a discrete two-point boundary-value problem and the corresponding control algorithm is derived via Pontryagin's maximum principle [10].

As a case study, the optimization framework is used to manage the power consumed by a *heterogeneous server cluster* processing a time-varying workload. Power has become an important design constraint for densely packed clusters due to electricity costs and heat dissipation issues [23]. To tackle this problem, many modern processors allow their operating frequency and supply voltage to be dynamically scaled [3] [12]. We develop a distributed control solution wherein each self-managing server decides the fraction of the incoming workload to process locally as well as the corresponding frequency setting that satisfies the system-wide QoS goal while minimizing power consumption.

Centralized solutions for energy-efficient cluster operation have been developed previously in [7] [24] [8] assuming *homogeneous* servers with identical processing capabilities. In [24], when the workload is light, some servers are turned off and the workload distributed to the rest of the system while [7] combines voltage scaling with powering on (off) servers. However, the above methods take a heuristic approach wherein the number of servers and their speeds are increased (decreased) if processor utilization exceeds (falls below) specified threshold values. Also, guaranteeing explicit QoS constraints is not addressed. The authors of [8] develop a linear quadratic regulator to dynamically decide the average operating frequency for the cluster while satisfying a specified response time.

We evaluate the performance of the control framework via simulations using a representative e-commerce workload [5], and show that it is scalable, has very low run-time overhead, and adapts quickly to dynamic workload variations and individual controller failures. We also discuss the impact of key design parameters such as the prediction horizon on controller performance.

The paper is organized as follows. Section II discusses key optimal control concepts while Section III formulates and solves the control problem for a single server. Section IV develops the distributed control framework and Section V presents detailed performance evaluation results. Section VI discusses control stability and Section VII concludes the paper with a discussion on future work.

II. Preliminaries

This section describes the dynamical model for a single server, introduces the optimal control concepts and the performance index to be optimized.

System Model. We will use a general nonlinear discrete-time first order dynamical equation

$$x(k+1) = f^k(x(k), u(k)) \tag{1}$$

with initial condition x_0 to model the queueing behavior of a single processor. $x(k+1) \in \mathbb{R}^n$, the system state at time $k+1$, is a function of the state $x(k)$ and input $u(k) \in \mathbb{R}^m$ at time k. The system model f captures the relationship between the observed system parameters, particularly those relevant to the QoS specifications, and the control inputs that adjust these parameters. The superscript on f indicates that it can in general be time-varying.

In our problem, the dynamics of an individual server P is captured by the discrete-time equation (1). Client requests are serviced by P on a first-come first-serve basis. We do not assume an *a priori* stochastic distribution for the request-arrival rate but estimate it using online observations. The request arrival rate during time step k is denoted by $\Lambda(k)$ and the average processing time for each request is $c(k)$. Defining the queue size as our system state $x(k)$, we have the following

state equation.

$$x(k+1) = \max\left\{x(k) + \left(\hat{\Lambda}(k) - \frac{u(k)}{\hat{c}(k) \cdot u_{max}}\right) T_s, 0\right\} \quad (2)$$

and the output equation

$$\omega(k) = (1 + x(k)) \cdot \frac{\hat{c}(k)}{u(k)/u_{max}} \quad (3)$$

The operating frequency $u(k)$ is the constrained control input to the system and the corresponding output is the average response time $\omega(k)$, which includes the waiting time for requests in the queue and the processing time on P.

We assume that P's operating frequency $u(k)$ is tunable input variable over a continuous interval $[u_{min}, u_{max}]$. Thus, if the time needed to process a request while operating at the maximum frequency u_{max} is $c(k)$, then the processing time for the request while operating P at some frequency $u(k) \in [u_{min}, u_{max}]$ is $c/\alpha(k)$, where $\alpha(k) = u(k)/u_{max}$ is the scaling factor. The queue size at the end of the next sampling period T_s is determined by a non-linear and non-differentiable maximum function of current queue size $x(k)$ and input $u(k)$. This function includes the time-varying arrival rate $\Lambda(k)$ and processing time $c(k)$ as disturbance. These signals are usually stochastic and thus replaced by their estimates $\hat{\Lambda}(k)$ and $\hat{c}(k)$. These estimates are obtained using an appropriate forecasting model. This is discussed in greater detail later this section.

Equations (2) and (3) adequately model the server dynamics when the incoming workload is mostly CPU intensive, i.e., the processor is the bottleneck resource. This is especially true for web and e-commerce servers where the both the application and data can be fully cached in memory, thereby minimizing (or eliminating altogether) hard disk accesses.

Control Concepts. Optimal control uses a *predictive* or *proactive* approach to generate a sequence of control inputs over a specified lookahead horizon while estimating changes in operating conditions [16]. The performance index is a convex cost function comprising both the state and control (or decision) vectors that must be optimized within the constraints imposed by the underlying system dynamics. At each time step, the discrete-time optimal control problem is to find the sequence $u(i), ..., u(N)$ within the prediction horizon of length N to minimize

$$J(i) = \Phi(N, x(N)) + \sum_{k=i}^{N-1} L^k(x(k), u(k)) \quad (4)$$

subject to the system model constraint

$$x(k+1) = f^k(x(k), u(k))$$

where $x(k)$ is the system state, Φ is a cost function of the goal state at the end of the horizon, and $L^k(x(k), u(k))$ is a time-varying cost function at each intermediate time step within $[i, N]$. The first control input $u(i)$ in the sequence is applied and the rest are discarded. This process then repeats for the next time step.

Fig. 1 shows the structure of an optimal controller for the queueing system described in (2) and (3). The environment

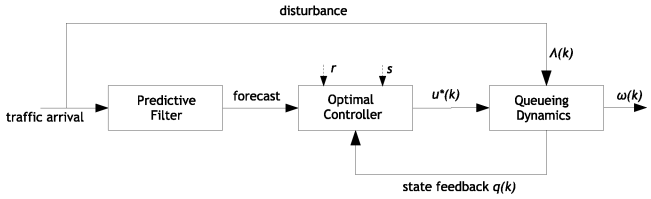

Fig. 1. The overall structure of an optimal controller

inputs (or disturbances) $\hat{\Lambda}(k)$ and $\hat{c}(k)$ are estimated, and the controller computes the optimal inputs over a finite prediction horizon. The control problem is solved as one of *state regulation*, updating both the initial and final states at the end of the prediction horizon. The parameters r and s are weights in the cost function, to be defined in the next section.

Workload Forecasting. Returning to Fig. 1, to estimate processor behavior over the prediction horizon, both $\hat{\Lambda}(k)$ and $\hat{c}(k)$, must be estimated. We use an ARIMA model, implemented using a Kalman filter [6], to provide arrival-rate estimates to the controller. We have previously shown that a number of published e-commerce and web workload traces [4] [21] [5] can be estimated with good accuracy using the Kalman filter. Other appropriate prediction models, for example, the ones proposed in [26], may also be used in our controller design.

The average processing-time per request is estimated using an exponentially-weighted moving-average (EWMA) filter as $\hat{c}(k+1) = \pi \cdot c(k) + (1-\pi) \cdot \hat{c}(k-1)$ where π is the smoothing constant.

III. Power Management on a Server

We now formulate the control problem to be solved by a single server. The multiple objectives for each server are to: (1) regulate the achieved response time $\omega(k)$ around a specified set point ω_o, and (2) minimize the corresponding processor power consumption cost.

Assuming a continuous domain for possible processor operating frequencies, we require an optimal frequency input $u(k)$ at each time step that maximizes the objective function. For the dynamical system described in (2) and (3) and an initial condition $x(0) = x_o$, this objective function is given as follows:

$$\begin{aligned} J &= \Phi(x(N)) + \sum_{k=1}^{N-1} [S(x(k)) + R(u(k))] \cdot e^{-\alpha k} \\ &= \frac{1}{2} v (x(N) - r_N)^2 \\ &\quad + \sum_{k=1}^{N-1} [\frac{1}{2} s (x(k) - x_{\omega_o})^2 + \frac{1}{2} r u^2(k)] \cdot e^{-\alpha k} \end{aligned} \quad (5)$$

where v, s, and r denote the weights of the three terms in the cost function, and x_{ω_o} is the set point for the controller, given in terms of a queue size, where x_{ω_o} is back-calculated using the desired response time ω_o and the output equation (3). The summation of the weighted terms S and R — quadratic

functions of the state $x(k)$ and control input $u(k)$ — over the intermediate time steps specifies the trade-off between the QoS requirement, i.e., minimizing $(x(k) - x_{\omega_o})^2$, and the corresponding power consumption. Finally, $\Phi(x(N))$ is a quadratic cost function that penalizes the number of requests left over in the queue at the end of prediction horizon N when compared to the desired value r_N. In our simulations, we choose $r_N = 0$, thereby forcing the controller to generate a sequence of frequency settings that deplete the queue by the end of the prediction horizon.

The optimal controller explores future (estimated) states within the prediction horizon to obtain a feasible sequence of control decisions. In an uncertain operating environment, however, 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. To counter this problem, we associate a discounting factor $e^{-\alpha k}$ with the cost function in (5). If $\alpha > 0$, 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.

The performance index in (5) is subject to the dynamic system equation constraint, now rewritten as a differentiable state equation

$$x(k+1) = x(k) + \left(\hat{\Lambda}(k) - \frac{u(k)}{\hat{c}(k) \cdot u_{max}}\right) T_s \qquad (6)$$

with state and control inequality constraints.

$$\begin{cases} x(k) \geq 0 \\ u_{min} \leq u(k) \leq u_{max} \end{cases} \qquad (7)$$

The optimal control problem is to find a sequence of frequency settings u^* along the finite horizon $[0, N]$ that drives the system along a state trajectory x^* subject to the above constraints such that the performance index J in (5) is minimized.

We developed the control algorithm to operate a single server using a discrete version of Pontryagin's maximum principle where a *two-point boundary-value problem* is solved, given initial and goal states [16]. During each sliding lookahead horizon N, the controller aims to regulate the queue size to zero, using the current queue size as the initial state. Due to space constraints, we do not show the derivation of the control algorithm here. The interested reader is referred to our technical report [28] for a detailed and mathematically rigorous derivation.

We now show a very simple example of one controller that manages a server whose operating frequency varies from 600 MHz to 1.8 GHz while serving the time-varying traffic shown in Fig. 2(a). We assume that requests have equal processing times. The controller sampling period T_s is set to 5 seconds and the lookahead horizon N to 10 steps. We fix the weights in (5) as $v = 50$ and $s = 5$, and compare the state trajectory and control inputs obtained while varying r, the weight for the energy consumption term. As Figs. 2 (b) and (c) show, larger r values force the controller to reduce server power consumption

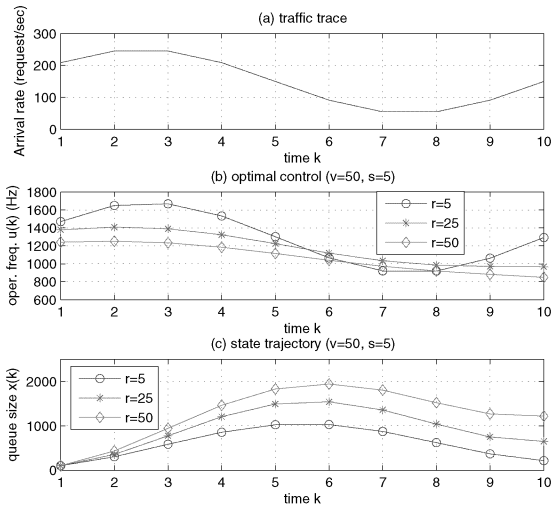

Fig. 2. Power management on a single server where the weights v and s are fixed; (a) the synthetic workload, (b) the operating frequencies decided by the controller, and (c) the corresponding queue sizes

by using lower operating frequencies, thereby slowing down the processing rate and increasing the queue size.

Finally, though the problem posed in this section assumes a continuous domain $[u_{min}, u_{max}]$ for possible control options, the controller can be applied in straightforward fashion to systems having a rich set of *discrete* control options. In this case, the problem is first formulated and solved assuming a continuous approximation of the discrete domain, and the obtained solution is mapped to an appropriate value within the discrete set. We provide examples of this approach in the section on performance evaluation (see Section V).

IV. Distributed Control Framework

We now use the controllers developed in Section III as building blocks in a distributed framework to manage the power consumption of a server cluster.

Our system model is a cluster comprising m servers where a global buffer stores incoming client requests. Each server retrieves some fraction of these requests and processes them in first-come first-serve fashion. We assume heterogeneous servers, and that the processor within each server supports dynamic voltage scaling by varying both its supply voltage and operating frequency. Therefore, the overall power consumption of the cluster at any given time instant includes a constant base cost for each operating server (due to the energy requirements of its power supply, hard disk, etc.) and the dynamic power consumed to process the workload. The optimization problem addressed here is to operate this server cluster in energy-efficient fashion by minimizing its dynamic power consumption while processing a time-varying workload. The QoS goal to be achieved by the cluster is an average request response time ω_o.

We can develop a centralized optimal controller to minimize

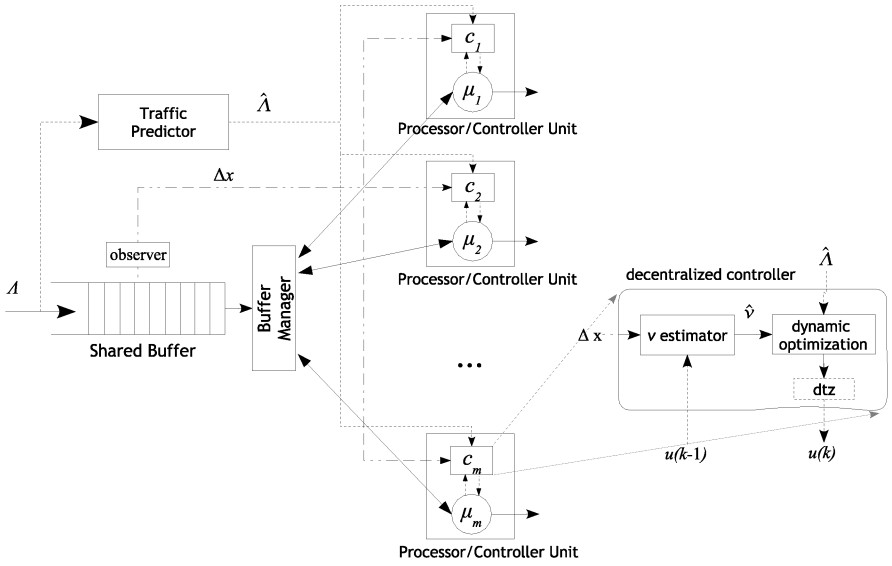

Fig. 3. The distributed system and control architecture

the following global cost function over all m servers

$$J_{global} = \sum_{i=1}^{m} \frac{1}{2} v_i (x_i(N) - r_N)^2 + \sum_{i=1}^{m} \sum_{k=1}^{N-1} [\frac{1}{2} s_i (x_i(k) - x_{\omega_o})^2 + \frac{1}{2} r_i u_i^2(k)] \cdot e^{-\alpha k} \quad (8)$$

where each server i is subject to state and control-input constraints previously introduced in (6) and (7). However, a centralized control solution is not a practical option for large clusters.

Fortunately, for the power management problem of interest, the integrated cost J_{global} is decomposable into sub-problems in such a way that the summation recovers the centralized cost. In other words, given multiple sub-systems whose dynamics and operating constraints are uncoupled, and whose local cost functions are convex, the global optimality may be obtained by simply having each sub-system independently optimize its local cost function. Therefore, for the centralized cost structure in (8), we generate distributed optimal control problems for each server to solve, and the problem to be solved by a server i was previously introduced in Section III. We also note that other performance management problems in utility computing such as resource provisioning and allocation have the above-described characteristics, and we expect the proposed control technique to be applicable to these as well.

Fig. 3 shows the distributed system and control architecture. Each server independently manages its operation using a local cost function (Equation 5) by deciding the optimal frequency settings, and thereby, the fraction of requests it plans to process from the shared buffer. The set of self-optimizing servers can be treated as *non-communicating agents*, where each agent need not have information about the exact behavior of other agents. Moreover, since servers can be heterogeneous in terms of their processing and power consumption characteristics, the weights r and s in their local cost functions can be different.

Each server in Fig. 3 executes the control algorithm developed in Section III. However, in a distributed setting, the state-space model for the i^{th} server must now consider the processing capabilities of other servers in the system and the corresponding impact on the shared buffer (or queue). Therefore, the queue dynamics from the viewpoint of the i^{th} server is as follows.

$$x(k+1) = \max\left\{x(k) + \left(\hat{\Lambda}(k) - \frac{u(k) + \hat{\nu}(k)}{\hat{c}(k) \cdot u_{max}}\right) T_s, 0\right\} \quad (9)$$

The estimate $\hat{\nu}(k)$, computed locally and independently by server i, predicts the cumulative operating frequencies of other servers (normalized to the frequency range of the local server if the cluster is heterogenous) in the system. It takes the following form:

$$\nu(k) = \left(\Lambda(k) - \frac{\Delta x(k)}{T_s}\right) c(k) \cdot u_{max} - u(k) \quad (10)$$

where $\Delta x(k) = x(k) - x(k-1)$ is the change in queue size due to request arrivals and consumption by the servers. Thus, from the i^{th} server viewpoint, $\nu(k)$ is simply the residue of $\Delta x(k)$, obtained after deducting from the total number of request arrivals, the number of requests processed locally by server i.

In our simulations, for the sake of convenience, a single Kalman filter broadcasts arrival-rate estimates for each step within the N-step horizon to all controllers. (Note that this filter can also be implemented within each controller itself.) The average request processing time is also predicted for the next N periods by an EWMA filter. Each controller also has to

locally maintain a Kalman filter to estimate ν, the processing capacity of other servers in the system. At each time step, every controller generates an optimal sequence of frequency settings within the prediction horizon, and applies the first input in this sequence. During the next sampling period, the various filters are updated with new information (the queue size, request arrival-rate and processing time), and the whole control process is repeated.

It is clear from the foregoing discussion that very little overhead is incurred when adding new controllers to the distributed framework. Recall that controllers themselves are non-communicating agents, and the only overhead incurred is in broadcasting the shared state and environment variable, the queue size and arrival rate estimate, respectively, to the newly added controllers. Therefore, the proposed scheme is highly scalable.

V. Performance Evaluation

The performance of the distributed control scheme as well as the impact of tuning key parameters such as r, s, v, T_s, and N on controller performance is now evaluated using a representative e-commerce workload.

Our experiments simulated multiple servers processing a synthetic workload, derived, in part, using HTTP requests made to an Internet service provider in the Washington DC area over a week [5]. Portions of this workload are shown in Figs 4(a) and 6(a).

The processing times for individual requests within the arrival sequence in Figs 4(a) and 6(a) were obtained as follows. We generated a virtual store comprising 10,000 objects (or requests), and the time needed to process an object was randomly chosen from a uniform distribution between (4, 11) 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 some files are more popular than others, and that the popularity distribution commonly follows Zipf's law. 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.

Performance Analysis. We simulated a cluster of four servers whose processors have adjustable frequencies in the continuous domain. First, we ensured that this cluster could satisfy the desired response time, set as $\omega_o = 4$ seconds, under a sustained worst-case workload scenario with each processor operating at its maximum frequency. The worst-case scenario is simply the maximum arrival rate observed within the trace in Fig. 4 where each request has the maximum processing time requirement of 11 ms. The control framework then optimizes

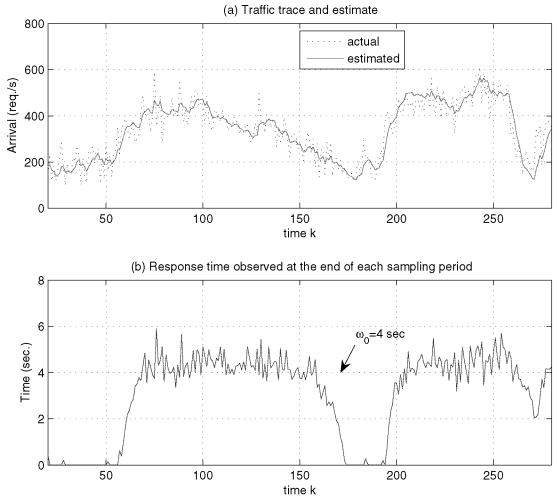

Fig. 4. (a) The synthetic workload and the corresponding predictions and (b) the average response time achieved by the cluster

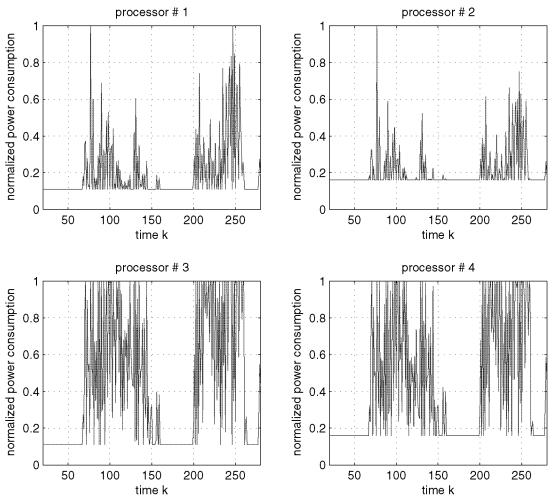

Fig. 5. The normalized power consumption $(u/u_{max})^2$ incurred by each processor

the performance of the cluster when the workload is time-varying.

Note that we use a cluster of four servers to clearly explain the obtained results. However, it follows from the discussion in Section IV that our approach can be extended in straightforward fashion to much larger clusters with very little overhead. In our simulations, the operating frequencies for servers 1 and 3 range from 600 MHz to 1.8 GHz while those for servers 2 and 4 range from 800 MHz to 2.0 GHz. The desired response time for the cluster was $\omega_o = 4$ sec. For each controller, we set the prediction horizon N to 5 steps and the sampling period T_s to 1 second.

Fig. 4(a) shows a portion of the workload and the corre-

sponding predictions obtained by a Kalman filter [6]. Each controller acquires predictions for 5 lookahead steps and computes the sequence of frequency settings over this sliding control horizon. To show how different weights in the cost function (Equation 5) affect controller behavior, we set $r = 1$ and $s = 50$ as weights on the energy consumption and response-time terms, respectively, for processors 1 and 2, and $r = 0.5$ and $s = 100$ for processors 3 and 4. Thus, processors 1 and 2 aim to minimize their power consumption while 3 and 4 prioritize the response time. The weight v was set to 5 for all processors, dictating how queue sizes are regulated to the desired value of zero at the end of the prediction horizon.

Fig. 4(b) shows that the controllers cooperate well to maintain the cluster-wide response time close to $\omega_o = 4$ sec. The dynamic power consumption cost incurred by each processor is shown in Fig. 5. We use a simple model proposed in [27] for this cost as $(u(k)/u_{max})^2$. The frequencies selected by the controllers clearly achieve the desired behavior, as dictated by the weights s and r. Observe that for the same r and s values on processors 1 and 2, each generates a different power consumption profile due to corresponding differences in their maximum operating frequencies (1.8 GHz versus 2.0 GHz).

Finally, MATLAB simulations on a 2.4 GHz Pentium 4 processor indicate that for a 5-step prediction horizon, the estimation and control computations take about 7 ms on each server. Therefore, for a sampling interval of $T_s = 1$ second, the control overhead is only 0.7%. Intuitively, one would expect that increasing the prediction horizon N should result in better control performance while incurring greater computational overhead. We will study how the choice of N affects controller behavior is a later section.

Self-Adaptive Behavior. The following series of experiments demonstrate the fault-adaptive properties of the control framework. We use the workload in Fig. 6(a) to test the system reaction to server and/or controller failures. Note that this workload is lighter than the one in Fig. 4(a) to guarantee that the desired response time can be achieved using just two servers (since we will be simulating the failure of two servers in the original cluster of four).

The controller parameters including the weights, control horizon, and sampling period remain unaltered from our previous experiments. Given the workload in Fig. 6(a), failures of servers 2 and 3 are simulated at time steps 120 and 150, respectively. (Note that in Fig. 7, the operating frequencies of servers 2 and 3 suddenly drop to zero.) However, from Fig. 6(b), we note that the response time achieved by the cluster continues to be maintained around $\omega_o = 4$ sec.

Fig. 7 also shows the reaction of servers 1 and 4 once servers 2 and 3 fail at $k = 120$ and $k = 150$, respectively. The surviving cluster members rapidly increase their frequencies to process the backlog created by these failures. More importantly, the reaction by servers 1 and 4 is achieved without any explicit communication. As shown in Fig. 8, the ν estimates, computed *independently and locally* by servers 1 and 4, make them aware that the overall cluster throughput has suddenly decreased after time steps $k = 120$ and $k = 150$. The servers,

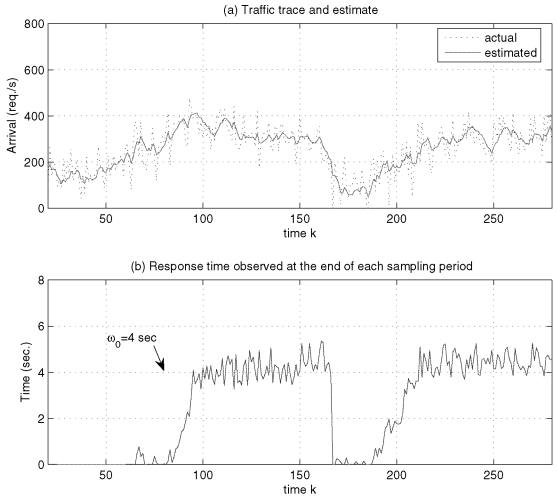

Fig. 6. (a) The synthetic workload and predicted values and (b) the average response time achieved by the cluster

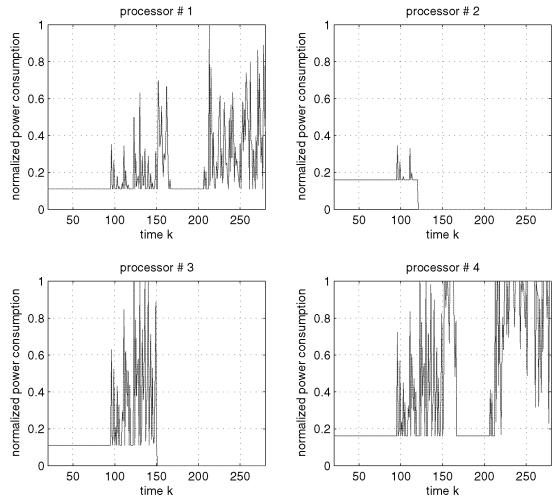

Fig. 7. The power consumption cost $(u/u_{max})^2$ on processors 1 and 4 in response to the failure of processors 2 and 3

therefore, increase their respective processing rates. Also note that of the two survivors, Server 4 processes more requests than 1. This difference in behavior is due to the setting of weights within each server's local cost function. Recall that the weights are $r = 1$ and $s = 50$ for Server 4, whereas $r = 0.5$ and $s = 100$ for Server 1. Thus, Server 4 is "altruistic" and prioritizes the global response time achieved by the cluster while Server 1 is more "selfish" and prioritizes its own power consumption.

Effect of Parameter Tuning. The following results show the effects of tuning the prediction horizon N and the sampling time T_s on control performance.

We now consider the (somewhat) idealized case of a cluster

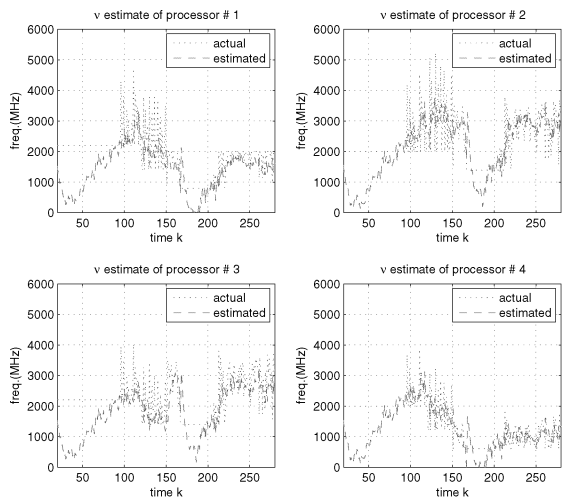

Fig. 8. The aggregate processing capacity estimates ($\hat{\nu}$) computed by each controller; servers 1 and 4 use these estimates to infer the failure of servers 2 and 3

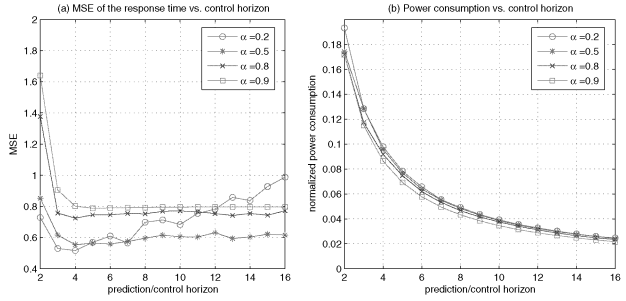

Fig. 9. (a) The MSE of the response time between the achieved and desired values and (b) the average normalized power consumption, as a function of the prediction horizon

comprising four identical servers assuming perfect arrival-rate estimates and a constant request processing time. The control performance of a single server, as a member of the overall cluster, is measured. It is important to note that each server must still estimate the aggregate processing capacity ν of the cluster to decide its operating frequencies — a source of possible estimation errors.

Under the above-described set up, Fig. 9 shows the controller performance, in terms of both the mean square error (MSE) between the achieved response time $\omega(k)$ and the set point ω_o, and power consumption cost as a function of N. Intuitively, as the controller looks further ahead, it can anticipate future workload demands and start preparing accordingly at the current time step itself. However, as noted in Section II, control performance does not necessarily improve by increasing N, since as N increases, so does the error in the estimated parameter ν. Therefore, N must be chosen carefully, considering the trade-off between look-ahead performance and estimation errors.

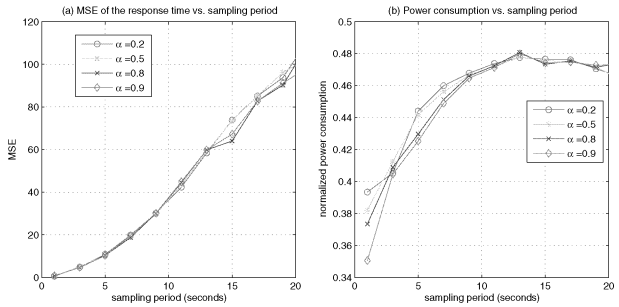

Fig. 10. (a) The MSE of the response time between the achieved and desired values and (b) the average normalized power consumption as a function of the sampling period T_s

Fig. 9 compares controller performance for different values of α, the tunable parameter of the discounting factor $e^{-\alpha k}$ in the cost function (5). We first observe that α has no appreciable effect on the power consumption costs shown in Fig. 9(b). We can see from Fig. 9(a) that small α values, for example, $\alpha = 0.2$, decrease the achieved MSE, which is preferable, but only when the prediction horizon is small. The control performance for $\alpha = 0.2$ actually deteriorates for larger prediction horizons, since $e^{-0.2k}$ cannot sufficiently discount the large estimation errors introduced in ν as one goes deeper into the prediction horizon. Also, a larger prediction horizon will increase the execution overhead of the controller.

To summarize, a prediction horizon between 4 to 7 time steps seems appropriate in this specific case to balance the trade-off between good lookahead performance and estimation errors.

We now examine the effect of varying the controller sampling time T_s on its performance. The sampling period dictates how often the controller provides a new control input to the underlying processor. This affects the average response times achieved by the incoming requests as well as the processor power consumption. Fig. 10(a) shows an almost linear relationship between the MSE versus the sampling period. We also show the power consumption incurred by a controller versus its sampling period in Fig. 10(b). We see that the power consumption saturates at around $T_s = 12$ sec. Here the choice of discounting coefficient α has no effect on both the MSE and the power consumption. Our results indicate that the sampling period must be chosen to be small (around 1 second) to obtain the best performance. A sampling time of $T_s = 1$ sec. is a practical option in our case since the execution time of the controller is very small — approximately 7 ms.

Effects of a Discrete Control-Input Set. The final set of experiments assume that server frequencies are not continuously tunable, but must be selected from a discrete domain. As discussed in Section IV, our approach is still applicable to such systems by simply discretizing the obtained solutions. Figs. 11 and 12 show cluster performance where servers 1, 2, 3, and 4 allow their frequencies to be tuned in discrete steps of 200 Hz. As the results show, overall system performance is still good since control errors introduced by previous discretization

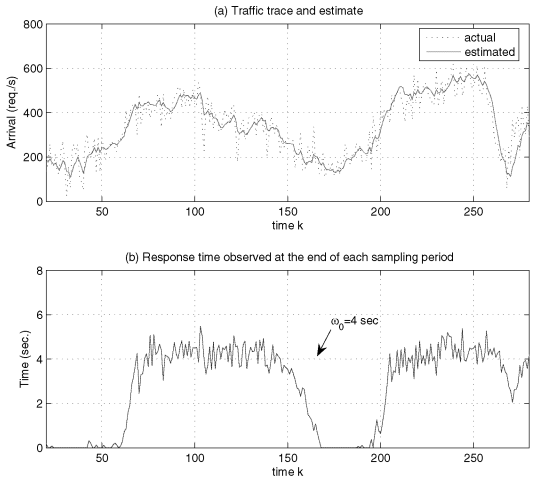

Fig. 11. (a) The synthetic workload and the corresponding predictions and (b) the average response time achieved by the cluster

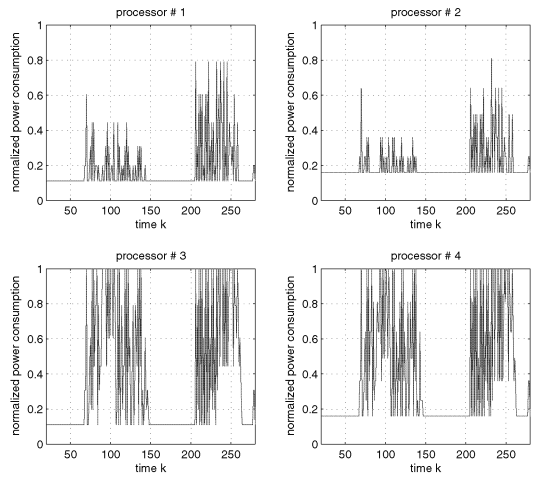

Fig. 12. The normalized power consumption where servers allow their frequencies to be tuned in steps of 200 Hz

steps are compensated by future control actions.

VI. Control Stability

We now informally show bounded-input bounded-state stability for the control architecture developed in IV. To simplify the discussion, we rewrite the system dynamics in a more general form as

$$x(k+1) = \max\{x(k) + T_s \cdot \Lambda(k) - T_s \cdot u(k), 0\} \qquad (11)$$

Without loss of generality, we replace the frequency scaling and processing-time terms by a generalized input $u(k)$, and use x to denote the state variable. We discuss the stability of an autonomous system (one without any external inputs), and the input-state stability of a non-autonomous system, i.e., stabilization.

The system is *(marginally) stable* for a finite horizon N, if, for each $\varepsilon > 0$, there is $\delta = \delta(\varepsilon) > 0$ such that $x(0) < \delta \Rightarrow x(k) < \varepsilon, \forall k, 0 \leq k \leq N$. It is *asymptotically stable* if it is stable and δ can be chosen such that $x(0) < \delta \Rightarrow \lim_{k \to N} x(k) = 0$. Our definition of stability is similar to [13] but with non-negative state variables. We specifically consider a finite N, since, if the state decays too slowly, we claim the controller is unstable within this horizon, although it might be asymptotically stable in the long run, or globally asymptotically stable. The arrival rate $\Lambda(k)$ is a perturbation of the nominal system $x(k+1) = x(k)$, which is marginally stable. However, the perturbed system is unstable because $\Lambda(k)$, though bounded, is nonnegative. Therefore, for a non-autonomous system, a necessary condition for stabilization is $\sum_{i=0}^{N} \Lambda(i) \leq \sum_{i=0}^{N} u(i)$, i.e., the processing capability over N must at least equal the worst-case arrival rate. It is straightforward to extend this argument to multiple servers.

VII. Conclusions

We developed a distributed optimization framework using concepts from optimal control theory, and as a case study, minimized the power consumed by a cluster operating processing a time-varying workload while satisfying QoS requirements. Using realistic workload traces, we showed that the proposed framework is highly scalable and has desirable self-healing properties. Future research will develop controllers for differentiated service and resource provisioning in clusters.

Future research will address differentiated service and resource provisioning in clusters where client arrivals are grouped into multiple classes based on their SLAs. The problem formulation must be extended to tackle this more complex problem, where at each time instant, the controller must now decide two variables, the operating frequency and the fraction of processing capacity that must be given to each client queue.

References

[1] S. Abdelwahed, N. Kandasamy, and S. Neema, "A Control-based Framework for Self-managing Distributed Computing Systems", *Proc. ACM Workshop Self-managing Systems*, 2004.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-system: A Control Theoretic Approach", *IEEE Trans. Parallel & Distributed Syst.*, 13(1):80-96, Jan. 2002.

[3] AMD Corp, *Mobile AMD-K6-2+ Processor Data Sheet*, Publication 23446, June 2000.

[4] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site", *Tech. Report HPL-99-35R1*, Hewlett-Packard Labs., Sep. 1999.

[5] M. F. Arlitt and C. L. Williamson, "Web Server Workload Characterization: The Search for Invariants," *Proc. ACM SIGMETRICS Conf.*, pp. 126-137, 1996.

[6] G. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd Edition, Prentice-Hall, Upper Saddle River, NJ, 1994.

[7] M. Elnozahy, M. Kistler and R. Rajamony, "Energy-efficient server clusters", Proc. Workshop Power Aware Computing Syst. (PACS), 2002.

[8] Y. Chen et al., "Managing Server Energy and Operational Costs in Hosting Centers", *Proc. ACM SIGMETRICS'05*, Jun. 2005.

[9] A. G. Ganek and T. A. Corbi, "The Dawn of the Autonomic Computing Era", *IBM Systems Journals*, 42(1):5-18, 2003.

[10] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A Survey of the Maximum Principles for Optimal Control Problems with State Constraints", *SIAM Review*, Vol.37, No.2, pp.181-218, June, 1995.

[11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*, Wiley-IEEE Press, Hoboken, NJ, 2004.

[12] Intel Corporation, *Enhanced Intel SpeedStep Tecnology for the Intel Pentium M Processor*, White paper: 301170-001, 2004.

[13] H. K. Khalil, *Nonlinear Systems*, 3rd Edition, Prentice Hall, 2002.

[14] N. Kandasamy, S. Abdelwahed, and J. P. Hayes, "Self-Optimization in Computer Systems via Online Control: Application to Power Management", *IEEE Intl. Conf. Autonomic Computing*, 2004.

[15] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu, "Hybrid Supervisory Control of Real-Time Systems," *Proc. Real-Time and Embedded Technology and Applications Symposium*, 2005.

[16] F. L. Lewis and V. L. Syrmos, *Optimal Control*, Wiley-Insterscience, 2nd Edition, 1995.

[17] C. Lu, G. A. Alvarez, and J. Wilkes. "Aqueduct: Online Data Migration with Performance Guarantees", *USENIX Conf. File Storage Tech.*, pp.219-230, 2002.

[18] C. Lu, J. Stankovic, G.Tao, and S. Son, "Feedback Control Real-time Scheduling: Framework, Modeling and Algorithms", *J. Real-time Syst.*, 2002.

[19] Z. Lu, et al., "Control-theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads", *Intl. Conf. Compilers, Architectures, & Synthesis Embedded Systems*, pp.156-163, 2002.

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

[21] D. Menasce, et al., "In Search of Invariants for e-Business Workloads", *ACM Conf. Electronic Commerce*, pp.56-65, 2000.

[22] S. Mascolo, "Classical Control Theory for Congestion Avoidance in High-speed Internet", *Conf. Decision & Control*, pp.2709-2714, 1999.

[23] T. Mudge, "Power: A First-Class Architectural Design Constraint," *IEEE Computer*, 34(4), pp. 52-58, April 2001.

[24] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems", *Workshop Compilers & Operating Syst. for Low Power*, Sept. 2001.

[25] V. Sharma et al., "Power-aware QoS Management in Web Servers", *Real-Time Syst. Symp.*, pp.63-72, 2003.

[26] D. Shen and J. L. Hellerstein, "Predictive Models for Proactive Network Management: Application to a Production Web Server," *Proc. Network Operations & Management Symp.*, pp. 833-846, 2000.

[27] A. Sinha and A. P. Chandrakasan, "Energy-Efficient Real-Time Scheduling," *Proc. Intl Conf. Computer Aided Design* (*ICCAD*), pp. 458-463, 2001.

[28] M. Wang et al., "Adaptive Performance Control of Computing Systems via Distributed Cooperative Control," *Technical Report ACL-2005-02*, Drexel University, December 2005. $(www.ece.drexel.edu/faculty/kandasamy/wang05.pdf)$.