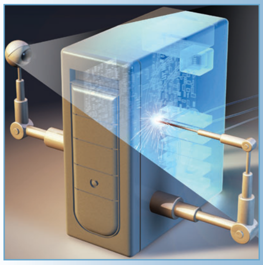

Distributed Cooperative Control for Adaptive Performance Management

The authors' distributed cooperative-control framework uses concepts from optimal control theory to adaptively manage the performance of computer clusters operating in dynamic and uncertain environments. Decomposing the overall performance-management problem into smaller subproblems that individual controllers solve cooperatively allows for the scalable control of large computing systems. The control framework also adapts to controller failures and allows for the dynamic addition and removal of controllers during system operation. This article presents a case study showing how to manage the dynamic power consumed by a computer cluster processing a time-varying Web workload.

Mianyu Wang, Nagarajan Kandasamy, Allon Guez, and Moshe Kam
Drexel University

Web-based services such as online banking and shopping are hosted on distributed computing systems comprising heterogeneous and networked servers. To operate such systems efficiently while satisfying stringent quality-of-service (QoS) requirements, multiple performance-related parameters must adapt to changing operating conditions. The workload a system must process, for example, might be time-varying, and hardware and software resources could fail or need replacing during system operation. For service providers to cope with growing scale and complexity, such systems must become largely *autonomic* — capable of managing themselves with minimal human intervention.[1] Key management tasks that we can automate in computing systems include power management, load balancing, and dynamic resource provisioning between competing workload classes.

A promising method for automating system management tasks is to formulate them as control problems in terms of cost or performance metrics. This approach offers some important advantages over heuristic or rule-based policies in that we can design a generic control framework — that is, it can address a class of problems (such as power management or resource provisioning) using the same basic control concepts. Moreover, we can verify such a control scheme's feasibility with respect to the performance goals prior to actual deployment.

Here, we describe a *cooperative-control framework* that we developed to manage distributed computing systems.[2] We decomposed the overall management task into a set of corresponding subproblems, mapping each to an underlying system component. Controllers — implemented locally within each component — solve their respective subproblems cooperatively, satisfying the overall system's specified performance goals. To solve optimization problems under uncertainty, each local controller uses a *receding horizon control* scheme in which the idea is to solve the optimal control problem over a given time horizon and then continuously extend this horizon forward.[3] This scheme is robust with respect to environmental disturbances and can simultaneously solve for multiple QoS objectives. We also present a case study that demonstrates how to use this distributed control framework to manage the power consumed by a heterogeneous server cluster when processing a dynamic Web workload.

Control-Based Performance-Management Approaches

Researchers from academia and industry have recently applied control theory to several system-management tasks. They've used classical *proportional, integral, and derivative* (PID) control in CPU provisioning, load balancing, and power-management problems in Web servers.[4] Others have used more advanced concepts from model-predictive and optimal control to pose system performance goals as optimization problems and help solve them online under dynamic operating constraints.[5,6] These techniques also accommodate nonlinear system behaviors.

Controlling distributed computing systems with multiple interacting components, however, is especially challenging because the number of available tuning options is typically quite large, and the corresponding search space grows exponentially with increasing system size. Also, the controller must carefully manage inter-component interactions to achieve system-wide performance goals and incorporate notions of uncertainty and risk to cope with dynamic workload arrivals, component failures, and unexpected changes to these goals. The resulting control problem is very complex, and a centralized controller implementation becomes computationally intractable even for relatively small systems.

The Optimal Control Problem

Our system architecture assumes a cluster of m servers. A dispatcher routes incoming client requests to individual servers, where they're queued and processed in a first-come, first-served fashion. We assume heterogeneous servers in which the processor operating frequency within each server is a tunable control variable over a continuous interval between some minimum and maximum frequency values. The cluster's overall power consumption at any given time includes a constant base cost for each operating server (due to the energy requirements of its power supply, hard disk, and so on) and the dynamic power consumed to process the workload.

The cluster's performance goal is to achieve an average response time r^* for the incoming requests while minimizing dynamic power consumption. We aim to achieve this goal using the predictive or receding horizon control scheme that Figure 1 illustrates. Here, we obtain the control actions governing system operation by optimizing the forecast system behavior for the performance metric over a limited prediction horizon. The controller obtains the sequence of control actions that results in the best system behavior over this horizon and applies the first action within this sequence as input during the current time instant. It then discards the rest and repeats this process each time step. Thus, in a predictive-control design, the controller optimizes the performance metric at each sampling-time instance, taking into account future variations in the environment inputs and their effects on system behavior.

We can provide an intuitive understanding of receding horizon control using driving as an analogy.[3] While driving, humans usually consider the road several hundred yards ahead to anticipate driving conditions and adjust speed and gear settings accordingly. As the car moves along the road, we can always see the next few hundred yards, and we're continuously picking up new information from the far horizon and using it to update our control decisions. Predictive control works similarly: it always considers the predicted system behavior over some time horizon into the future and thus, at each successive sampling instant, predicts one further sample into the future. As new information becomes available, the controller uses it to modify the current trajectory.

Estimating Future Environment Inputs

The controller treats the time-varying environment inputs — the number of HTTP requests and the per-request processing time — as disturbances. Because these inputs are usually stochastic, the system model uses their online estimates to track future behavior. A typical Web workload shows pronounced time-of-day variations in which the number of HTTP request arrivals can change considerably in just a few minutes,[7,8] so it's necessary to develop good predictive filters to obtain accurate estimates for these environment inputs. We've previously shown how to design and tune Kalman filters to predict variations in Web workloads with a good degree of accuracy.[9]

Tracking Future System Behavior

The dynamical system model uses the current system state (the queue length), the estimated environment inputs, and the control set (the set of available operating frequencies) to track the system's evolution along the prediction horizon. Here, the system model is a difference equation that captures the relationship between the system variables relevant to the performance metric — that is, the achieved response time and the power consumption, as well as the control inputs that influence these variables. The response time a server achieves is a function of the queue length and the per-request processing time. As the server's operating frequency is tuned, the time needed to process a request scales linearly, affecting both the queue length and the response time. This type of difference equation adequately models the server dynamics when the incoming workload is CPU-intensive — that is, the processor is the bottleneck resource.

Specifying the Performance Metric

We define the QoS goal for the cluster as a *set-point specification* in which the controller aims to operate the system within a close neighborhood of the desired response time r^*. We can also consider *control* or *transient costs* as part of the system operating requirements, indicating that certain trajectories toward the desired state are preferable over others in terms of their cost to the system. In our case, we'd like each server to achieve the QoS goal of r^* while minimizing the corresponding power consumption. We can specify this performance metric for each server using the multiobjective function

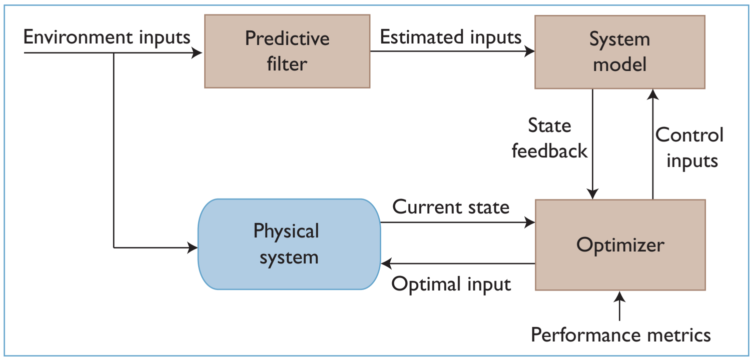

Figure 1. The receding horizon control scheme. In this scheme, we obtain the control decisions that affect system behavior by optimizing the forecast system behavior during a limited prediction horizon.

$$J(k) = \frac{1}{2}s\left(r(k) - r*\right)^2 + \frac{1}{2}w\left(\frac{f(k)}{f_{\max}}\right)^2, \qquad (1)$$

where k denotes the sampling time instant, and $r(k)$ and $f(k)$ denote the achieved response time and operating frequency, respectively. The cost function J captures the inherent trade-off between achieving the desired set point of r^* and the corresponding power consumption cost, given as $(f(k)/f_{\max})^2$, where $f_{\max}$ is the server's maximum operating frequency. The weights s and w denote these terms' relative importance in the overall cost function. To obtain a numerical solution to the control problem, we require that the function J have the quadratic form Equation 1 shows.

From a cluster operator's viewpoint, a practical problem with this multiobjective optimization function is how to determine the weights s and w to achieve acceptable controller performance — an iterative and somewhat time-consuming process. Although we've used abstract weights in this article to clearly illustrate the key control concepts, we can assign an actual dollar amount to each term in the function J — for example, dollars earned by achieving a response time (specified by the service-level agreement) and the cost of operating the server (dollars per kilowatt-hour consumed).

The overall control problem is to minimize the performance metric J for all m servers over a prediction horizon of length N. At each time instant k, the optimal controller finds a sequence of frequency settings for the m servers within the prediction horizon $[0, N]$ that drives the system along a trajectory, minimizing the performance metric J.

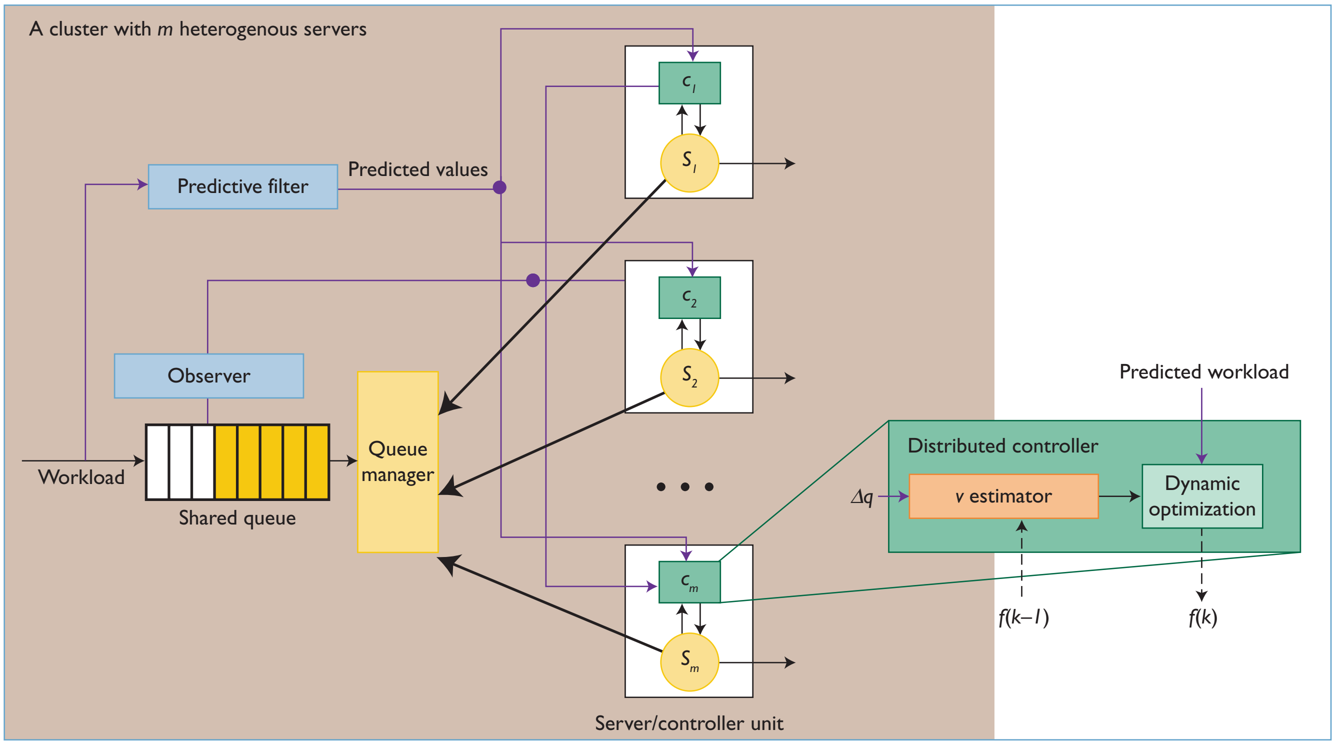

Figure 2. Distributed cooperative-control structure. Here, S_i denotes a server and C_i is its local controller.

The controller applies the first control input in this sequence to the system and discards the rest. This process repeats for the next time step.

The Distributed Control Framework

We formulate the power-management task in a server cluster as a cooperative distributed-control problem. Before describing our approach, we must note that not all control problems can be decomposed in a distributed fashion. However, it's well known that, given multiple subsystems whose dynamics and operating constraints are uncoupled and whose local cost functions are quadratic, simply having each subsystem independently optimize its local cost function can potentially achieve the global optimal.[10,11] The power-management problem falls in this category, and we can decompose it into subproblems for each server to solve such that the summation of the local costs recovers the centralized cost.

Figure 2 shows the distributed control architecture for a cluster. Each server independently manages its operation using a local cost function that decides the optimal frequency settings and, thereby, the fraction of requests to process from the shared global queue. We can treat the set of self-optimizing servers as *noncommunicating agents*, in which each agent needn't have information about other agents' exact behaviors.

Tracking System Behavior

In a distributed setting, each server must independently track the global queue's dynamic growth. From an individual server's viewpoint, the global queue's length at a time instant k is given by the number of requests arriving into the system $a(k)$ and the number of requests consumed from the queue by other servers. The following difference equation F can capture this dynamic on each server:

$$\begin{aligned} q(k+1) &= F\big(q(k), f(k), \hat{a}(k), \hat{c}(k)\hat{v}(k)\big) \\ r(k) &= G\big(q(k), c(k)\big) \end{aligned} \qquad (2)$$

Here, $q(k)$ denotes the global queue's length and $f(k)$ and $r(k)$ denote the operating frequency and response time the server achieves, respectively; $\hat{a}(k)$ and $\hat{c}(k)$ are the estimated number of requests arriving into the system and their average processing time, respectively.

The variable $\hat{v}(k)$, which each server estimates independently, predicts the cumulative processing capacity of all other servers (excluding itself) in the cluster. Thus, $\hat{v}(k)$ represents the implicit coupling between the various distributed controllers via the shared system variable — the global queue length. We compute it using the change in the global queue length $\Delta q(k) = q(k) - q(k-1)$, the number of requests arriving into the system, and the number of requests that the server processes locally.

The Distributed Control Problem

Each server i minimizes the performance metric

$$J = \frac{1}{2}v\left[q(N)\right]^2 + \sum_{j=0}^{N-1}\left\{\begin{matrix}\frac{1}{2}s\left[r(j) - r*\right]^2 \\ +\frac{1}{2}w\left[\frac{f(j)}{f_{\max}}\right]^2\end{matrix}\right\}\cdot e^{-\alpha j} \quad (3)$$

to obtain a sequence of frequency settings along the prediction horizon [0, N]. Given that servers can be heterogeneous in terms of their processing and power-consumption characteristics, the weights in their local cost functions can differ. Also, each controller must choose its server's operating frequency under the following state and control-input constraints:

$$\begin{cases} q(j) \geq 0 \\ f_{\min} \leq f(j) \leq f_{\max}. \end{cases} \quad (4)$$

These constraints simply state that at each point within the prediction horizon, the global queue length must be greater than zero, and the frequency setting must fall within the range [$f_{\min}$, $f_{\max}$], where $f_{\min}$ and $f_{\max}$ denote the minimum and maximum available frequencies, respectively.

The first term in Equation 3 penalizes the number of requests left over in the global queue at the end of the prediction horizon. This is a terminal cost added to improve controller stability. The second term specifies the trade-off between achieving the set point $(r - r^*)^2$ and the corresponding power consumption on the server. Finally, to deal with an uncertain operating environment in which the estimated system states become increasingly inaccurate as we go deeper into the prediction horizon, we use a discounting factor $e^{-\alpha k}$, $\alpha > 0$, with the cost function in Equation 3. Thus, states further out in the prediction horizon have less impact on current control action.

In Figure 2, a Kalman filter broadcasts arrival-rate estimates to every controller for each step within the N-step horizon. (We can also implement this filter within each individual controller.) Using an exponentially weighted moving-average filter, we predict the average request-processing time for the next N periods. Each controller maintains another Kalman filter to locally estimate υ, the aggregate 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 system updates the various filters with new information (the queue size, request-arrival rate, and processing time) and then repeats the whole control process.

The foregoing discussion demonstrates that the control scheme incurs very little overhead when adding new controllers to the distributed framework. Because the controllers themselves are noncommunicating agents, the only overhead incurred is in broadcasting the shared state and environment variables — the queue size and arrival-rate estimate, respectively — to the newly added controllers. Thus, the control scheme is highly scalable.

Controller Stability

Finally, a key driver for using control-based approaches is that we can verify the proposed scheme's feasibility with respect to the performance goals prior to deploying the system. This is analogous to the concept of stability when controlling mechanical systems. The bounded-input bounded-state stability of the control scheme in Figure 2 is easily shown; assuming a bound on the request-arrival rate and processing time, we can derive the conditions (the number of servers required and their processing capacity) under which the response time the system achieves remains bounded. The mathematical details are available elsewhere.[2] Showing other forms of stability — such as asymptotic stability, in which the achieved response time tracks the set point for all time — is a topic of ongoing research.

Performance Evaluation

In our simulations, we assume that the system designer has performed capacity planning to ensure that enough servers are available to handle the desired QoS requirements under the peak or worst-case workload scenario. The online control scheme is responsible for operating the cluster in a power-efficient fashion while satisfying QoS goals under a time-varying workload. Power has become an important design constraint for densely packed clusters due to electricity costs and heat

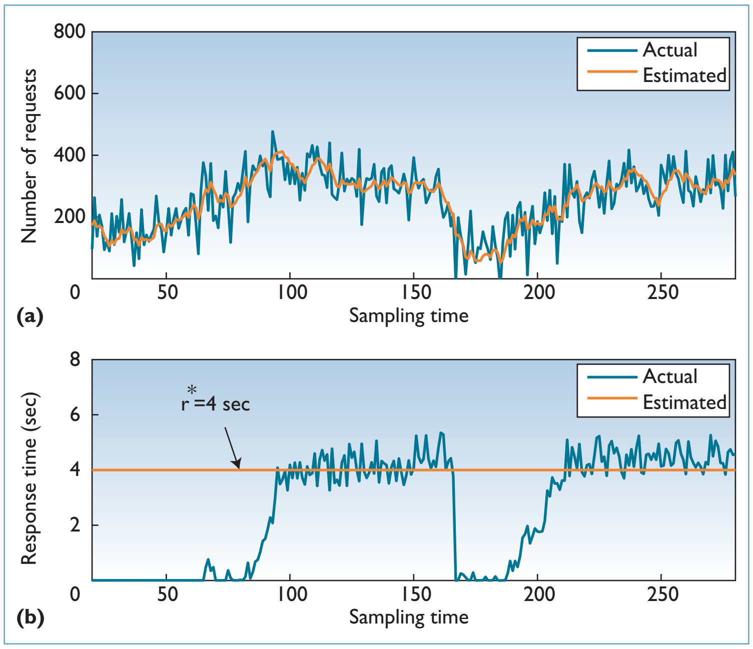

Figure 3. Average cluster-wide response time achieved by the distributed control scheme. In our simulation, we used the Web workload in (a) and obtained corresponding predictions using a Kalman filter. (b) The cluster achieved an average response time close to the desired set point of $r^* = 4$ seconds.

dissipation issues. To tackle this problem, many modern processors allow their operating frequency and supply voltage to be dynamically tuned.[12] To clearly explain the distributed control scheme's behavior, including fault-adaptive behavior, we first analyze its performance on a small cluster of four servers. Later, we'll show that the control structure can easily scale to a larger system processing a heavier workload.

Fault-Adaptive Behavior

In our simulations, the operating frequencies for servers 1 and 3 range from 600 MHz to 1.8 GHz, whereas those for servers 2 and 4 range from 800 MHz to 2.0 GHz. The controller can tune the frequencies in discrete steps of 200 Hz within their respective ranges. We can adapt the control solution developed in the previous section in a straightforward fashion and apply it to a case in which the controller must choose frequency settings from a discrete set. We first formulate and solve the problem assuming a continuous approximation of the discrete domain and then map the solution we obtain to the closest value within the discrete set.

The average response time the cluster must achieve for the incoming requests is set to $r^* = 4$ seconds. Figure 3a shows the workload to be processed in which we chose the processing times for individual requests within the arrival sequence from a uniform distribution between (4, 11) milliseconds (ms). The distribution of requests within the arrival sequence follows Zipf's law (a few files are extremely popular whereas many others are rarely requested). We set the prediction horizon within each local controller to five look-ahead steps, and the controller sampling interval is one second.

Figure 3a also shows the workload predictions we obtained via a Kalman filter. Each controller acquires predictions for five look-ahead steps and computes the sequence of frequency settings over this receding horizon. The weights in the cost function (Equation 3) are set such that servers 1 and 2 aim to minimize their power consumption while severs 3 and 4 prioritize meeting the set response time.

Given this simulation set up, the system operates normally up to time step 120, with all four servers operational. As Figure 3b shows, the controllers cooperate well to maintain the response time close to 4 seconds, the desired value. At time steps 120 and 150, we simulate failures of servers 2 and 3, respectively. (In Figure 4, servers 2 and 3 suddenly drop their operating frequencies to zero.) The response time the cluster achieves, however, remains around the set point.

Figure 4 shows the reactions of servers 1 and 4 after servers 2 and 3 fail at $k = 120$ and $k = 150$, respectively. The surviving servers increase their frequencies to process the queuing backlog these failures create. More important, servers 1 and 4 adapt without any explicit communication. The υ estimates that servers 1 and 4 compute independently and locally make them aware that the overall cluster throughput has suddenly decreased after time steps $k = 120$ and $k = 150$. The servers thus increase their respective processing rates. Of the two survivors, server 4 processes more requests than 1 because of the weightings within each server's local cost function. Thus, server 4 is "altruistic" and prioritizes the global response time the cluster achieves, whereas server 1 is more "selfish" and prioritizes its own power consumption.

Our 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. Thus, for a sampling interval of one second, the control overhead is only 0.7 percent.

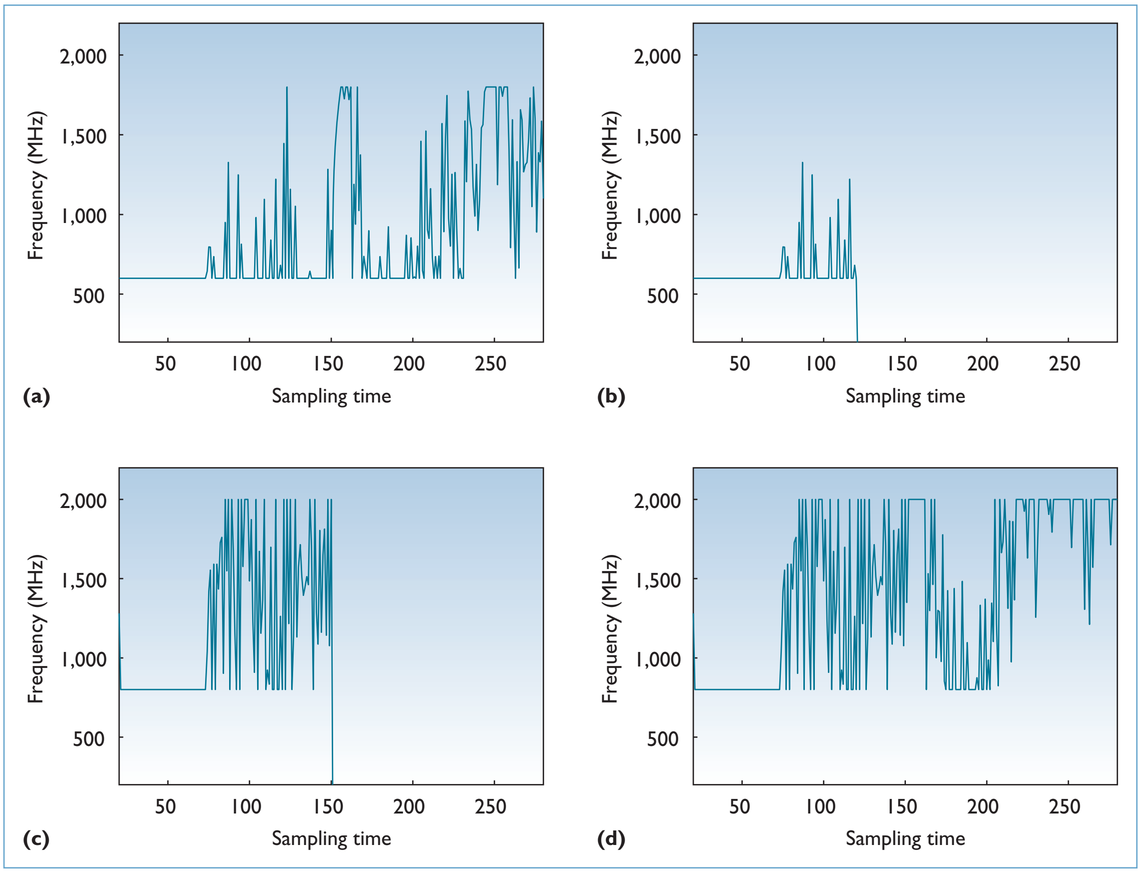

Figure 4. Operating frequencies decided by the local controller on each server. We simulated failures on servers 2 and 3 at time steps 120 and 150, respectively. Servers 1 and 4 react to these failures by increasing their operating frequencies correspondingly to continue to achieve the set point.

Effects of Parameter Tuning

We expected that increasing the controllers' prediction horizon N would result in better overall control performance. If a controller looks ahead further, it can anticipate future workload demands and start preparing accordingly at the current time step itself — say, by slightly increasing the operating frequency. This typically lets it track the set point more smoothly when compared to a situation in which the controller reacts to workload changes, which results in more abrupt oscillations around the set point. Also, smoother set-point tracking reduces power consumption. The dynamic power a server consumes is quadratically related to its operating frequency, and so an abrupt change to a higher frequency is penalized more heavily than a smooth ramp up. Increasing the prediction horizon, however, has its own problems — as N increases, so too do errors in the estimated parameters, which can lead to poor control quality. Thus, we must choose N carefully by considering the trade-off between look-ahead performance and estimation errors.

We consider an idealized case in which a cluster has four identical servers with perfect arrival-rate estimates and a constant request processing time. We measure the control performance of a single server as a member of the overall cluster. Here, each server must still estimate cluster υ's aggregate processing capacity to decide its operating frequency — a source of possible estimation errors. Figure 5 shows the controller performance as the prediction horizon N increases, in terms of the mean square error (MSE) between the desired and achieved response times. We see that MSE decreases with the increasing prediction horizon up to a point before the estimation errors start to slowly degrade control performance.

Figure 5 also shows the controller behavior for different values of α, the tunable parameter of the discounting factor $e^{-\alpha k}$ in Equation 3. We see that small α values (for example, $\alpha = 0.2$) decrease the achieved MSE — which is desirable — but only when the prediction horizon is also small. The control performance for $\alpha = 0.2$ actually deteriorates

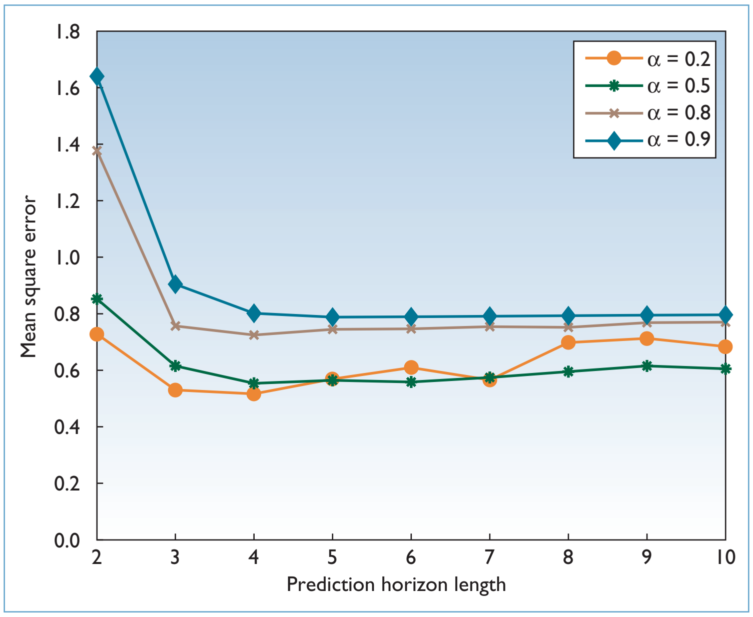

Figure 5. The mean square error (MSE) between the achieved and desired response times. The MSE initially decreases quickly as we increase the length of the prediction horizon and then stabilizes, indicating that the length of the prediction horizon does indeed affect controller performance.

for larger prediction horizons because $e^{-0.2k}$ can't sufficiently discount the large estimation errors introduced in υ as we go deeper into the prediction horizon. Also, a larger prediction horizon will increase the controller's execution overhead.

To summarize the results of our tuning experiments for this case study, a prediction horizon between four to seven time steps and a low value for the discounting factor seems appropriate for balancing the trade-off between good look-ahead performance, estimation errors, and control overhead.

We found that the controllers achieved a 55.5 percent reduction in power consumption when compared to an uncontrolled system in which each server operated at its maximum frequency at all times.

Distributed Control Structure Scalability

The control framework easily scales to larger systems, as we demonstrated by applying it to a cluster of 12 heterogeneous servers processing a much heavier workload (see Figure 6a), derived using HTTP requests made to the France World Cup Web site on 26 June 1998. We set the desired response time to two seconds; Figure 6b shows that the controllers cooperate to keep the actual response time close to this set point. As before, local controllers incurred approximately 7 ms in overhead for a prediction horizon of five steps.

We are currently extending our framework to allow servers to independently switch themselves on or off according to dynamic workload intensity. Our proposed control solution is quite general and can address other important system-management tasks, such as differentiated service and resource provisioning in server clusters in which client arrivals are grouped into multiple classes based on their service-level agreements. We can formulate the optimal control problem to tackle this more complex problem, in which the controller must decide two variables at each time instant: the operating frequency and the fraction of processing capacity that each client queue must receive.

Finally, many enterprise computing systems are realized as multitier architectures comprising clusters of Web, application, and database servers. Such systems demand performance management across multiple tiers in which client requests must traverse multiple stages while satisfying a desired end-to-end response time r^*. We can decompose this requirement across multiple clusters as time-varying response times to be achieved by individual clusters, implying that these clusters' performance must now be coordinated.

One possible approach to achieving end-to-end performance management of multitier systems is to use a supervisory controller to coordinate the activities of different clusters, each of which is managed via the distributed architecture Figure 2 shows. If, for example, recent history suggests that the application cluster has been achieving faster-than-required response times at the expense of increased power consumption, the supervisor can slow the cluster down. The supervisory controller can influence a subordinate cluster's behavior by tuning its cost function appropriately — specifically, the weights s and w in Equation 3, which specify the desired trade-off between the response time achieved by a local controller and the corresponding power consumption.

References

1. A.G. Ganek and T.A. Corbi, "The Dawn of the Autonomic Computing Era," *IBM Systems J.*, vol. 42, no. 1, 2003, pp. 5–18.
2. M. Wang et al., "Adaptive Performance Control of Comput-

ing Systems via Distributed Cooperative Control: Application to Power Management in Computing Clusters," *Proc. 3rd IEEE Int'l Conf. Autonomic Computing* (ICAC), IEEE CS Press, 2006, pp. 165–174.
3. J.A. Rossiter, *Model-Based Predictive Control*, CRC Press, 2003.
4. J.L. Hellerstein et al., *Feedback Control of Computing Systems*, Wiley-IEEE Press, 2004.
5. S. Abdelwahed, N. Kandasamy, and S. Neema, "Online Control for Self-Management in Computing Systems, *Proc. 10th IEEE Real-Time and Embedded Technology and Application Symp.* (RTAS), IEEE CS Press, 2004, pp. 368–376.
6. X. Koutsoukos et al., "Hybrid Supervisory Control of Real-Time Systems," *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symposium* (RTAS), IEEE CS Press, 2005, pp. 12–21.
7. M. Arlitt and T. Jin, *Workload Characterization of the 1998 World Cup Web Site*, tech. report HPL-99-35R1, Hewlett-Packard Labs, Sept. 1999.
8. D. Menascé et al., "In Search of Invariants for e-Business Workloads," *Proc. ACM Conf. Electronic Commerce*, ACM Press, 2000, pp. 56–65.
9. N. Kandasamy, M. Khandekar, and S. Abdelwahed, "A Hierarchical Optimization Framework for Autonomic Performance Management of Distributed Computing Systems," *Proc. 26th IEEE Int'l Conf. Distributed Computing Systems* (ICDCS), IEEE CS Press, 2006.
10. W.B. Dunbar and R.M. Murray, "Distributed Receding Horizon Control for Multivehicle Formation Stabilization, *Automatica*, vol. 42, no. 4, 2006, pp. 549–558.
11. A. Guez, I. Rusnak, and I. Bar Kana, "Multiple Objectives Optimization Approach to Adaptive and Learning Control," *Int'l J. Control*, vol. 56, no. 2, 1992, pp. 469–482.
12. Intel, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*, 2004; www.intel.com/design/intarch/papers/30117401.pdf.

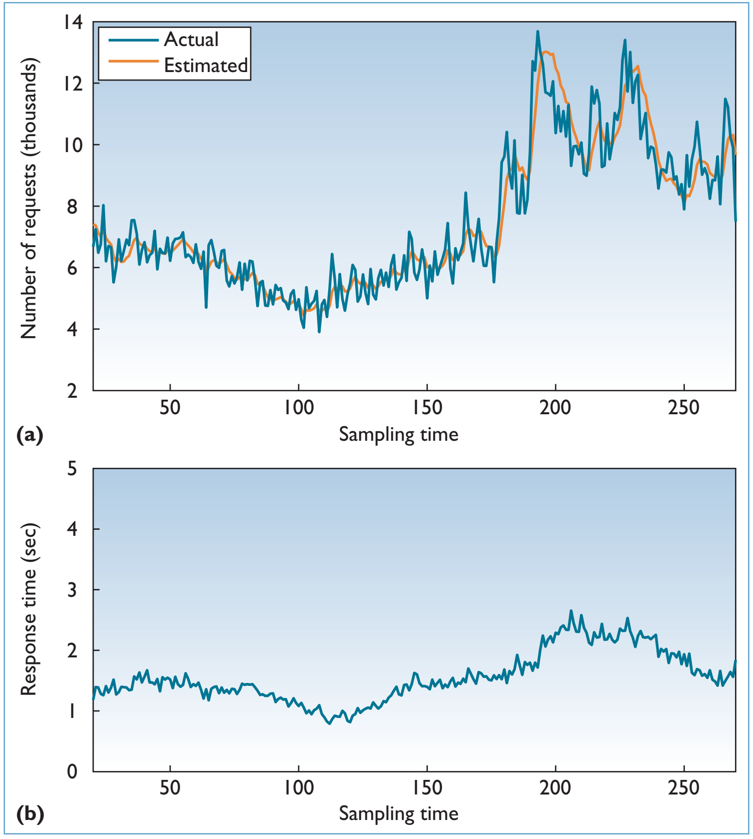

Figure 6. Performance evaluation of a 12-server cluster using a heavier workload. We used the workload shown in (a) in our simulations. (b) The cluster achieved an average response time around the set point of 2 seconds.

Mianyu Wang is a PhD student in the Department of Electrical and Computer Engineering and an MS student in the Department of Mathematics at Drexel University. His research interests include autonomic computing and performance management and optimization in distributed systems. Wang has a BS in control engineering from Zhejiang University, China. He is a student member of the IEEE. Contact him at jeremy@minerva.ece.drexel.edu.

Nagarajan Kandasamy is an assistant professor in the Department of Electrical and Computer Engineering at Drexel University. His current research interests include autonomic systems, embedded systems, reliable and fault-tolerant computing, and computer architecture. Kandasamy has a PhD in computer engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE. Contact him at kandasamy@ece.drexel.edu.

Allon Guez is a professor in the Department of Electrical and Computer Engineering at Drexel University. His interests lie in applying the principles of intelligent decision making, adaptation, optimization, and control to automation, robotics, business, and other areas. Guez has a PhD in electrical engineering from the University of Florida. Contact him at guezal@drexel.edu.

Moshe Kam is the Robert Quinn professor of electrical and computer engineering at Drexel University, the director of Drexel University's NSA Center of Excellence in Information Assurance Education, and technical coordinator of the US Department of Defense-sponsored project Applied Communications and Information Networking (ACIN). His professional interests are in system theory, detection and estimation, information assurance, robotics, navigation, and control. Kam has a BS from Tel Aviv University and an MSc and PhD from Drexel University, all in electrical engineering. Contact him at kam@minerva.ece.drexel.edu.