

A Hierarchical Optimization Framework for Autonomic Performance Management of Distributed Computing Systems

Nagarajan Kandasamy
Drexel University
Philadelphia, PA 19104, USA

Sherif Abdelwahed
Vanderbilt University
Nashville, TN 37203, USA

Mohit Khandekar
Drexel University
Philadelphia, PA 19104, USA

Abstract

This paper develops a scalable online optimization framework for the autonomic performance management of distributed computing systems operating in a dynamic environment to satisfy desired quality-of-service objectives. To efficiently solve the performance management problems of interest in a distributed setting, we develop a hierarchical structure where a high-level limited-lookahead controller manages interactions between lower-level controllers using forecast operating and environment parameters. We develop the overall control structure, and as a case study, show how to efficiently manage the power consumed by a computer cluster. Using workload traces from the Soccer World Cup 98 web site, we show via simulations that the proposed method is scalable, has low run-time overhead, and adapts quickly to time-varying workload patterns.

1 Introduction

A distributed computing system (DCS) hosting e-commerce, business, and scientific applications must typically satisfy stringent quality-of-service (QoS) requirements while operating in a dynamic environment. For example, the workload to be processed may be time varying, and hardware and software components may fail during operation. To achieve QoS goals in such systems, numerous performance-related parameters must be continuously optimized to respond rapidly to time-varying computing demands. As these systems increase in size and complexity, manually tuning key operating parameters will become very difficult. Therefore, it is highly desirable that future systems manage themselves, given only high-level guidance by administrators. Such *autonomic computing systems* aim to maintain the specified QoS by adaptively tuning key operating parameters with minimum manual intervention [15].

Advanced control and mathematical programming techniques offer a formal framework to design performance-control schemes for a DCS. If the system of interest is correctly modeled and the effects of its operating environment accurately estimated, control algorithms can be developed to achieve the desired QoS goals. The key advantages of using such a framework instead of ad hoc heuristics are: (1) one can systematically pose various performance control problems of interest within the same basic framework, and (2) the feasibility of the proposed algorithms with respect to the QoS goals may be verified prior to deployment. This paper develops such a framework and applies it to the specific problem of operating a computing cluster in energy-efficient fashion while satisfying QoS goals.

Classical control theory has been successfully applied to selected performance management problems in stand-alone computing systems including task scheduling [13, 20], bandwidth allocation and QoS adaptation in web servers [4], load balancing in e-mail and file servers [19, 24], and processor power management [21, 26]. These methods use *feedback* or *reactive control* to first observe the current system state and then take corrective action to achieve the specified QoS. Traditional feedback control, however, has some inherent limitations. It assumes a linearized and discrete-time model for system dynamics with an unconstrained state space, and a continuous input and output domain. Many practical systems of interest, on the other hand, have the following characteristics.

- *Hybrid behavior.* Many systems exhibit behavior comprising both discrete-event and time-based dynamics where the control or tuning options are limited to a finite set at any given time. We refer to such systems as *switching hybrid systems* [2].
- *Optimization under constraints.* In addition to the regulation problem where the system state must be maintained within a desired operating region, the corresponding operating cost must also

be minimized. The (nonlinear) cost function can include multiple variables and must be optimized under dynamic operating constraints.

- *Control actions with dead times.* Actions such as (de)activating computing resources in a DCS often incur a substantial dead time (the delay between a control input and the corresponding system response), requiring *proactive control* where control inputs must be provided in anticipation of future changes in operating conditions.

A performance manager for a DCS must, therefore, tackle complex discrete combinatorial problems such as the allocation and provisioning of hardware and software resources, which may need continuous re-solving with observed environmental events such as time-varying workload patterns and failures of individual components. Since the underlying control set is finite, traditional optimal control techniques [12] cannot be applied directly to switching hybrid systems, and in most cases, a closed-form expression for a feedback-control map cannot be established.

We have recently developed control strategies for optimizing the QoS of computing systems exhibiting hybrid behavior. This paper builds on our prior work in [1,18], and develops a hierarchical framework to solve performance management problems in a DCS modeled as a switching hybrid system. The problems of interest include power management, load balancing, and dynamic resource provisioning in server and storage clusters. We use the following key concepts, aimed at “lifting” the dual curses of dimensionality and modeling, to develop the proposed framework.

- *Temporal decomposition.* The problem of interest is posed as sequential optimization under uncertainty and solved via a *limited-lookahead control* (LLC) approach. At each time step, the control actions governing DCS 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* (MPC) [22], sometimes used to solve optimal control problems for which classical feedback solutions are extremely hard or impossible to obtain.
- *Control decomposition.* A hierarchical structure, where multiple controllers interact with each other to satisfy system-wide QoS goals, is used to reduce the dimensionality of the overall control problem. In this control structure, a controller is responsible for optimizing the behavior of the component(s) under its control while satisfying the constraints imposed on it by a higher-level controller.

- *Function approximation.* Complex component behavior, not easily modeled from first principles, is captured by an approximation architecture obtained via *simulation-based learning*. The aggregate behavior of multiple components can be similarly approximated to reduce the dimensionality of the search space.

As a case study, we show how to operate a heterogeneous computer cluster in energy-efficient using the proposed optimization framework. Incoming service requests are distributed among multiple computers to satisfy a desired response time — the QoS goal. At each time instant, the controller decides both the number of computers to operate and the frequency setting on each processor to satisfy the QoS goal while minimizing overall energy consumption. Using workload traces from the France’98 World Cup web site (WC’98) [6], we show via simulations that our method is scalable, has low run-time overhead, and adapts quickly to dynamic workload variations.

Energy-efficient cluster management has been a topic of active research, and [14,25] are examples of some recent work. In [25], when the workload is light, some computers are turned off and the workload distributed to the rest of the system while [14] combines voltage scaling with powering on (off) computers. However, the above methods assume a homogeneous system and take a heuristic approach wherein the number of computers and their speeds are increased (decreased) if processor utilization exceeds (falls below) specified threshold values. By contrast, the optimization framework developed here provides a systematic way to manage a computer cluster, taking into account explicit QoS requirements, operating constraints, and control delays inherent in switching on computers.

The rest of this paper is organized as follows. Section 2 describes the system model and discusses key LLC concepts. Sections 3, 4, and 5 apply these concepts to develop the hierarchical framework and evaluate it under different workload scenarios. We conclude the paper in Section 6.

2 Preliminaries

This section describes the system model and discusses key LLC concepts.

2.1 System Model

Fig. 1(a) shows a computer cluster where a global buffer stores incoming requests. Some fraction of these requests is dispatched to each computer C_i where they are locally queued and processed in first-come first-

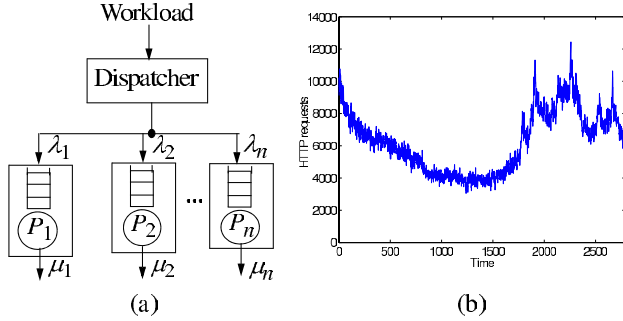


Figure 1: (a) A computer cluster processing a time-varying workload where λ_i and μ_i denote the arrival and processing rates, respectively, on computer C_i , and (b) a sample workload from the WC'98 web site comprising HTTP requests plotted at 2-minute intervals

serve fashion. We assume heterogeneous computers, and that the processor P_i within each computer supports dynamic voltage scaling by varying both its supply voltage and operating frequency [17]. Therefore, 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.

The optimization problem addressed in this paper is how to operate the above cluster in energy-efficient fashion under a time-varying workload while achieving a desired response time. The workload in Fig. 1(b) is a typical example, and comprises HTTP requests made to the WC'98 web site on June 26, 1998 [6]. The workload clearly shows time-of-day variations and is representative of other published web and e-commerce workloads [7, 23]. The cluster in Fig. 1(a) is a good example of a switching hybrid system since we can only choose from a limited number of computers and processor frequencies to affect cluster operation.

The continuous dynamics of the cluster is described by the following discrete-time state-space equation.

$$x(k+1) = f(x(k), u(k), \omega(k)) \quad (1)$$

where $x(k) \in \mathbb{R}^n$ is the system state at time step k , and $u(k) \in U \subset \mathbb{R}^m$ and $\omega(k) \in \mathbb{R}^r$ denote the control inputs and environment parameters, respectively. The system model f captures the relationship between the observed system parameters, particularly those relevant to the QoS goals, and the control inputs that adjust these parameters.

The look-ahead nature of LLC requires that the relevant environment parameters be estimated for the corresponding prediction horizon. Though environmental

inputs to a DCS, such as the workload in Fig. 1(b), are typically uncontrollable, they can, however, be estimated using well-known forecasting techniques. We use a Kalman filter [16] to estimate future environment inputs $\hat{\omega}(k)$ for the controller. 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.

$$\hat{x}(k+1) = f(x(k), u(k), \hat{\omega}(k)) \quad (2)$$

2.2 Performance Specifications

The performance goal for the cluster in Fig. 1(a) is to achieve an average response time for incoming requests. This is expressed as a *set-point specification* where the controller aims to operate the system within a close neighborhood of a desired state $x^* \in X$ where X is the set of valid system states.

The computer cluster must operate within strict constraints on both the system variables and control inputs; for example, limitations on the number of available computers, operating frequencies on individual processors, and the overall energy budget for the cluster. We use a general form to describe the operating constraints of interest as $H(x) \leq 0$ and $u(x) \subseteq U$ where $u(x)$ denotes the control-input set permitted in state x and $H(x) \leq 0$ simply defines the system state space X .

It is also possible to consider transient or control costs as part of the system operating requirements, indicating that certain trajectories towards the desired state are more preferable over others in terms of their cost to the system. For example, transient costs may be incurred when a computer is switched on (off). The overall specification will then require that the system reach its set-point while minimizing transient costs. To this end, we use the following norm-based function to define the overall operating cost.

$$J(x(k), u(k)) = \|x(k) - x^*\|_Q + \|u(k)\|_R + \|\Delta u(k)\|_S \quad (3)$$

where $\Delta u(k) = u(k) - u(k-1)$ is the change in the control inputs, and Q , R , and S are user-defined weights denoting the relative importance of the variables in the cost function. These weights must be chosen to prioritize between the primary and secondary control goals, i.e., driving the system closer to x^* against minimizing the corresponding operating cost.

2.3 Control Concepts

The primary goal of the controller is to drive the system to the desired operating state x^* using an admissible

trajectory, defined by the constraints $H(x(k)) \leq 0$ and $U(x(k))$, and maintain the system close to this state. The cost function $J(k)$ must also be minimized as the system moves toward the desired state.

The LLC approach constructs a set of future states from the current state $x(k)$ up to a prediction horizon N . A trajectory $\{u^*(q)|q \in [k+1, k+N]\}$ that minimizes the cumulative cost while satisfying both state and input constraints is selected within this horizon. The first input leading to this trajectory is chosen as the next control action. The process is repeated at time $k+1$ when the new system state $x(k+1)$ is available. The LLC formulation in (4) accounts for future changes in the desired system trajectory and measurable disturbances (e.g., environment inputs) and includes them as part of the overall control design.

$$\min_U \sum_{q=k}^{k+N} J(x(q), u(q)) \quad (4)$$

$$\text{Subject to: } H(f(x(q), u(q), \hat{\omega}(q))) \leq 0 \\ u(q) \in U(x(q))$$

In a switching hybrid system where control inputs must be chosen from discrete values, the above optimization problem will show an exponential increase in worst-case complexity with an increasing number of control options and longer prediction horizons. In this paper, we substantially reduce the dimensionality of the optimization problem shown in (4) via hierarchical control decomposition. Exhaustive and bounded search strategies are then used at different levels of the hierarchy to solve the corresponding optimization problems with low run-time overhead.

Finally, a detailed optimality analysis of the proposed algorithm is beyond the scope of this paper which focuses on techniques for scalable control. We refer the reader to [3] for the mathematical under-pinnings of how to characterize LLC performance in an uncertain environment. Also, analyzing the optimality of hierarchical control schemes such as the one developed here is a focus of our ongoing research efforts.

3 Hierarchical Control

We now apply the basic concepts introduced in Section 2 to develop a multi-level optimization structure aimed at operating the cluster in Fig. 1(a) in energy-efficient fashion while satisfying our QoS goal — an average response time r^* for incoming requests. First, this cluster is logically partitioned, for the purposes of scalable control, into *modules* where each module M_i itself comprises multiple processors. Fig. 2(a) shows

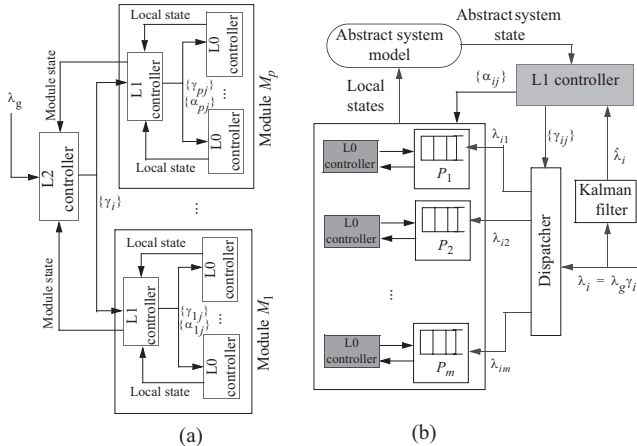


Figure 2: (a) The hierarchical control structure imposed on the cluster in Fig. 1(a), and (b) hierarchical control within module M_i

the various controllers within a three-level hierarchy, all working together to achieve the QoS goals.

The L2 controller must distribute the workload appropriately to the various modules. Therefore, given a request-arrival rate of λ_g and the current state of each module, the controller must decide the vector $\{\gamma_i\}$, the fraction of requests dispatched to each M_i . Fig. 2(b) shows the hierarchical control structure within a single module. The L1 controller within M_i must decide two variables: (1) the operating state of M_i given by the vector $\{\alpha_{ij}\}$, where $\alpha_{ij} = 1$ ($\alpha_{ij} = 0$) denotes the on (off) state of a computer j within M_i , and (2) the fraction $\{\gamma_{ij}\}$ of the requests seen by M_i to distribute to each computer. Given γ_{ij} , the L0 controller on computer j must optimize the processor operating frequency u_j to achieve the desired response time r^* .

The hierarchical structure in Fig. 2(a) reduces the dimensionality of the original optimization problem substantially. Where a centralized controller must decide the variables $\{\gamma, \alpha, u\}$ for each of the n computers in the cluster, in our method, the L2 controller only decides a single-dimensional variable $\{\gamma\}$ for k modules, where $k \ll n$. Similarly, the L1 controller decides control variables only for those computers within its module — far fewer compared to the total number of computers in the cluster. To realize this hierarchical structure, however, each high-level controller must know the approximate behavior of the components comprising the immediate lower level. For example, to solve the combinatorial optimization problem of determining $\{\gamma_i\}$, the fraction of requests to be dispatched to the modules, the L2 controller must be able to quickly approximate the behavior of each M_i (including its L1 and L0 controllers) for various choices of

γ_i . Function approximation techniques are discussed in greater detail in later sections.

Controllers at various levels of the hierarchy can operate at different time scales. Since the L1 controller in Fig. 2(b) uses the aggregate behavior of lower-level controllers to make decisions, it typically operates on a longer time scale when compared to a L0 controller. We assume that $T_{L1} = l \cdot T_{L0}$, $l > 1$, where T_{L1} and T_{L0} denote sampling times for the L1 and L0 controllers, respectively. Similarly, the L2 controller may operate on a longer time scale compared to a L1 controller.

We develop the control hierarchy in a bottom-up fashion. Section 4 develops the L0 and L1 controllers within a single module, corresponding to the structure in Fig. 2(b). Section 5 shows how to compose multiple such modules and develops the L2 controller to manage the performance of the large system.

4 Module-level Controllers

This section develops the control structure for a single module shown in Fig. 2(b). Its performance is then evaluated using a synthetic workload.

4.1 L0 Controller Design

We first develop the system model and the cost function for a L0 controller within module M_i . Though computers are heterogeneous and have different power-consumption and processing profiles, the model and the cost function remains the same for each computer j (though the parameters themselves may vary). Therefore, we drop the subscripts i and j for readability purposes whenever the context is clear. Also, we use k_{L0} to denote the time step for a L0 controller.

System Model. We use a queuing model to capture the dynamics of a processor P where $\lambda(k_{L0})$ and $\mu(k_{L0})$ denote the arrival and processing rates, respectively, of incoming requests, and $q(k_{L0})$ denotes the queue size at time k_{L0} . Each P operates within a limited set of frequencies (control inputs) U . Therefore, if the time required to process a request while operating at the maximum frequency u_{\max} is c , then the corresponding processing time while operating at some frequency $u(k_{L0}) \in U$ is $c/\phi(k_{L0})$ where $\phi(k_{L0}) = u(k_{L0})/u_{\max}$ is the scaling factor. The response time achieved by P during time step k_{L0} is denoted by $r(k_{L0})$, and includes both the waiting time in the queue and the processing time on P . We use the model proposed in [27] to estimate the average power consumed by P while operating at $u(k_{L0})$ as $\psi(k_{L0}) = \phi^2(k_{L0})$. An operating computer also incurs a fixed base cost given by a .

The following equations describe the dynamics of a computer.

$$\hat{q}(k_{L0} + 1) = q(k_{L0}) + \left(\hat{\lambda}(k_{L0}) - \frac{\phi(k_{L0})}{\hat{c}(k_{L0})} \right) \cdot T_{L0} \quad (5)$$

$$\hat{r}(k_{L0} + 1) = (1 + \hat{q}(k_{L0} + 1)) \cdot \frac{\hat{c}(k_{L0})}{\phi(k_{L0})} \quad (6)$$

$$\hat{\psi}(k_{L0} + 1) = a + \phi^2(k_{L0}) \quad (7)$$

Given the observed queue length $q(k_{L0})$ on P , the estimated length $\hat{q}(k_{L0} + 1)$ for $k_{L0} + 1$ is obtained using the predicted workload arrival and processing rates. The average response times of requests arriving during the time interval $[k_{L0}, k_{L0} + 1]$ is estimated as $\hat{r}(k_{L0} + 1)$ and the corresponding power consumption estimate is $\hat{\psi}(k_{L0} + 1)$. Note that $\hat{\lambda} = \gamma \hat{\lambda}_i$, where $\hat{\lambda}_i$ is the load estimate for module M_i and γ is the fraction decided by the L1 controller for the computer of interest.

Equations (5), (6), and (7) adequately model the system dynamics when the processor is the bottleneck resource — web and e-commerce servers where both the application and data can be fully cached in memory, thereby minimizing (or eliminating) hard disk accesses.

We use an ARIMA model [10], implemented by a Kalman filter, to predict load arrivals at both levels of the control hierarchy within a module. The processing-time estimate for time $k_{L0} + 1$ on P is given by an exponentially-weighted moving-average (EWMA) filter as $\hat{c}(k_{L0} + 1) = \pi \cdot c(k_{L0}) + (1 - \pi) \cdot \hat{c}(k_{L0} - 1)$ where π is the smoothing constant.

Control Problem. The optimization problem for the L0 controller is given as follows.

$$\min_U \sum_{q=k_{L0}+1}^{k_{L0}+N_{L0}} J(x(q), u(q)) \quad (8)$$

$$\text{Subject to: } \hat{x}(q + 1) = f(x(q), u(q), \hat{\omega}(q))$$

where J is the cost function for the processor state x (determined by the achieved response time and the corresponding power consumption for a control input u), and N_{L0} is the prediction horizon for the L0 controller. The estimated environment parameters $\hat{\omega}$ include the arrival rate $\hat{\lambda}$ and processing time \hat{c} . Soft constraints are added to function J using *slack variables*, defined such that they are non-zero only if the corresponding constraints are violated. Their non-zero values may be heavily penalized in the cost function. Therefore, the controller has a strong incentive to keep them at zero if possible. If r^* is the desired response time, we define $\epsilon(k_{L0})$ as

$$\epsilon(k_{L0}) = \begin{cases} 0 & : r(k_{L0}) \leq r^* \\ r(k_{L0}) - r^* & : \text{otherwise} \end{cases}$$

The cost function is $J(x(k_{L0}), u(k_{L0})) = \|\epsilon(k_{L0})\|_Q + \|\psi(k_{L0})\|_R$ where Q and R denote user-defined weights. (We ignore the control cost, since switching between different operating frequencies incurs negligible power-consumption overhead.)

The L0 controller uses an *exhaustive search strategy* where a tree of all possible future states is generated from the current state up to the specified depth N_{L0} . If $|U|$ denotes the size of the control-input set, then the number of explored states is $\sum_{q=1}^{N_{L0}} |U|^q$. When both the prediction horizon and the number of control inputs are small, the control overhead is negligible, as confirmed by our simulations in Section 4.3. In fact, processors such as the AMD-K-2 [5] and Pentium M processors [17] offer only a limited number of discrete frequency settings, eight and ten, respectively

4.2 L1 Controller Design

The L1 controller for a module M_i decides the operational settings for each computer j , defined by $\{\alpha_{ij}\}$ and $\{\gamma_{ij}\}$, using future load estimates and the current states of individual computers. We use k_{L1} to denote the sampling time of the L1 controller.

System Model. Since a detailed behavioral model of M_i can be quite complex, we use an approximate model within the L1 controller to describe the composite behavior of the individual computers. If the module comprises m computers, the aggregated state vector seen by the L1 controller is given as

$$x_{L1}(k_{L1}) = \Psi(x_1(k_{L0}, l), \dots, x_m(k_{L0}, l)) \quad (9)$$

where Ψ is the abstraction map, $T_{L1} = l \cdot T_{L0}$, and $x_j(k_{L0}, l) = \{x_j(k_{L0} - l + 1), \dots, x_j(k_{L0})\}$ is the set of local states for the j^{th} computer. The function Ψ simply generates the aggregated state variables of interest — the average queue length and request processing time within M_i . The average queue length over the sampling time T_{L1} is

$$q_{L1}(k_{L1}) = \frac{\sum_{b=1}^l \sum_{j=1}^m q_j(l \cdot k_{L1} - b)}{l \cdot m} \quad (10)$$

Similarly, we can define the environment inputs $\omega_{L1}(k_{L1}) = (\lambda_{L1}(k_{L1}), c_{L1}(k_{L1}))$ seen by the L1 controller as an aggregate of the values seen by each L0 controller over the time interval T_{L1} . Therefore, the arrival rate seen by the module M_i is

$$\lambda_{L1}(k_{L1}) = \sum_{b=1}^l \sum_{j=1}^m \lambda_j(l \cdot k_{L1} - b) \quad (11)$$

and the average processing time is

$$c_{L1}(k_{L1}) = \frac{\sum_{b=1}^l \sum_{j=1}^m c_i(l \cdot k_{L1} - b)}{m \cdot l} \quad (12)$$

Each module M_i is then represented by the following switching hybrid system model.

$$\hat{x}_{L1}(k_{L1} + 1) = g(x_{L1}(k_{L1}), u_{L1}(k_{L1}), \hat{\omega}_{L1}(k_{L1})) \quad (13)$$

where g is the high-level dynamic map — an abstraction of the composite dynamics of the L0 controllers. In this map, $u_{L1}(k_{L1}) \in U_{L1}$ is the input set for the L1 controller defining local settings for individual computers. As discussed before, these settings affect the load distributed to the j^{th} computer (γ_j) and its operating status (α_j). Also, the set of local control settings manipulated by C_{L1} is finite; α_j is a binary variable while γ_j is appropriately quantized.

Informally, g approximates the behavior of the computer (and its L0 controller) under various environment and control inputs supplied to it. For example, given the current state of a computer, g estimates both the average cost and the next state achieved by the L0 controller over the time interval k_{L1} for different values of the arrival rate $\lambda_{L1}(k_{L1})$ and processing time $c_{L1}(k_{L1})$, as well as the operating settings γ_i from the L1 controller's control set. The map g is initially obtained in off-line fashion by simulating the L0 controller using various values from the input set U_{L1} and a quantized approximation of the domain of ω_{L1} , and then (infrequently) adjusted using continuous observations of actual system behavior.

Control Problem. The L1 control problem for M_i comprising m computers is as follows.

$$\min_{U_{L1}} \sum_{q=k_{L1}}^{k_{L1}+N_{L1}} \sum_{j=1}^m (\alpha_{ij}(q) \cdot \tilde{J}(x_{L1}(q), \gamma_{ij}(q)) + \|\Delta\alpha_{ij}(q)\|_W) \quad (14)$$

Subject to: $\hat{x}_{L1}(q+1) = g(x_{L1}(q), u_{L1}(q), \hat{\omega}_{L1}(q))$

$$\sum_{j=1}^m \gamma_{ij}(q) = 1$$

$$\alpha_{ij}(q) - \gamma_{ij}(q) \geq 0, \quad \forall j$$

$$\alpha_{ij}(q) \in [0, 1]$$

$$\gamma_{ij}(q) \geq 0, \gamma_{ij}(q) \in \mathbb{R}$$

Here, $\tilde{J}(x_{L1}(q), \gamma_{ij}(q))$ is the approximate cost provided by the abstraction map g for the j^{th} computer for an operational setting of γ_{ij} , and $\|\Delta\alpha_{ij}(q)\|_W$ specifies the transient cost, in terms of power consumption, associated with switching on that computer where W is an appropriate weight.

The online controller typically operates in a highly dynamic environment. For example, workload traces

from World Cup 98 show high variability and noise, where the request arrival rate λ changes quite significantly and quickly — usually in the order of a few minutes. Under such variability, the estimated arrival rates within the prediction horizon have an “uncertainty band” $\hat{\lambda}(q) \pm \delta(q)$ around them, where $\delta(q)$ denotes the average error between the actual and forecasted values. Such estimation errors may cause the L1 controller to chatter, i.e., switch computers on and off excessively within short time spans in search of an optimal energy-efficient cluster configuration. Clearly, excessive switching is undesirable since it reduces the reliability of a computer.

We reduce chattering effects in our design as follows. Given a state $x(q)$ within the prediction horizon and a possible control input $u(q)$, the expected cost of the next state is obtained by first computing three possible next states from $x(q)$, each corresponding to sampled arrival-rate values $\hat{\lambda}(q) - \delta(q)$, $\hat{\lambda}(q)$, and $\hat{\lambda}(q) + \delta(q)$ within the uncertainty bound. Then, the expected cost of state $x(q+1)$ is the average cost of these three states.

The L1 controller uses a *bounded search strategy* to decide γ_{ij} , where given the current state $x(k_{L1})$, the controller searches a limited neighborhood of this state for a solution. Such a strategy is applicable in our case since the environment parameters are unlikely to change drastically over the sampling period T_{L1} assumed for our experiments, and the possible choices for γ_{ij} at any given time are limited by the maximum processing capacity of each computer j , i.e., we know the peak request arrival rate that can be processed by a computer without queuing instability.

The decisions γ and α of the L1 controller are communicated to each L0 controller which then optimizes its performance under these conditions. Since L0 controllers operate on a shorter time frame than their L1 counterparts, they will, during each time step T_{L0} , estimate and adapt to short-term changes in the environment inputs within the operating conditions set by the L1 controllers.

4.3 Performance Analysis

We now evaluate the performance of the proposed control scheme. Since multiple modules can be composed at the next level in the hierarchy to manage a much larger system, the number of computers within each M_i can be kept low to reduce the L2 controller overhead. We present simulation results for a module comprising four computers C_1, C_2, C_3 , and C_4 , each having the operating frequencies shown in Fig. 3. Without loss of generality, we assume the same base operating and transient power-up costs for each computer.

Computer	Operating frequencies (MHz)
C_1	532, 665, 798, 900, 1081, 1197, 1529
C_2	532, 610, 720, 990, 1197, 1250, 1352, 1529
C_3	532, 665, 798, 1099, 1197, 1280, 1352, 1529
C_4	532, 680, 720, 840, 980, 1100, 1197, 1297, 1529
C_5	532, 840, 1120, 1352, 1529
C_6	532, 1197, 1529

Figure 3: The operating frequencies available within each computer in the module

Workload Generation. As input to the module, we generated a synthetic time-varying workload representative of real-world ones. Using the workload analyzed in [7] — HTTP requests made to one computer at an Internet service provider in the Washington DC area — as our starting point, we scaled this workload appropriately. First, we removed noise from the data to extract its underlying structure, which was then scaled by a factor of four before adding noise. Fig. 4 shows this synthetic workload where request arrivals are plotted at 2-minute granularity. Also, Gaussian noise was added to mimic different operating conditions; for example, the (relatively smooth) period from [0, 300] has a maximum variance of 200 arrivals per 30-second interval while periods [301, 1025] and [1026, 1600] have increased variances of 300 and 500 arrivals, respectively, during each 30-second interval.

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 (10, 25) 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 [7]. 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 [8].

Parameters of the Kalman filter were first tuned using an initial portion of the workload, and then used to forecast the remainder of the load during controller execution. Both the actual and predicted values are

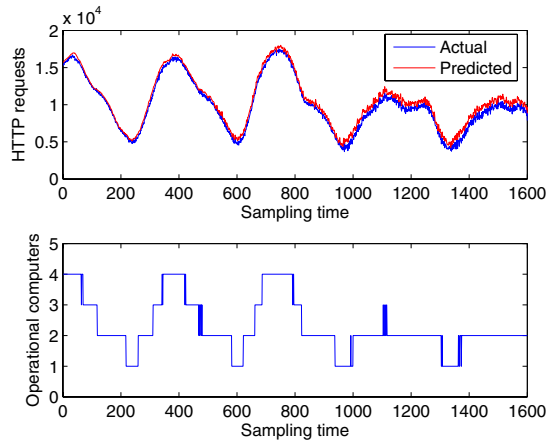


Figure 4: The synthetic workload and the corresponding predictions from the Kalman filter, and the number of computers operated by the L1 controller under load fluctuations

shown in Fig. 4. Request processing times were estimated using an EWMA filter with a smoothing factor of $\pi = 0.1$. Figs. 4 and 5 summarize the performance of the hierarchical control scheme where the average response time to be achieved by the cluster was set to $r^* = 4$ seconds. Also, note that X-axis in Figs. 4 and 5 show the different sampling times of the L1 and L0 controllers, respectively.

Fig. 4 shows how the L1 controller, operating under the one-step lookahead policy ($N_{L1} = 1$), sets the operating state α_{ij} of each computer in anticipation of workload fluctuations. The load distribution factor for the j^{th} computer, $0 \leq \gamma_{ij} \leq 1$, is quantized in intervals of 0.05. The controller sampling time T_{L1} was set to two minutes — the typical time delay incurred in switching on a computer. The penalty for switching on a computer was set to $W = 8$ (much higher than the base operating cost of $a = 0.75$), preventing the L1 controller from switching computers on and off excessively within short time spans. The abstraction map g is obtained off-line as a hash table. (It is also possible to convert this table to an approximation architecture as discussed in Section 5.)

As a representative sample, Fig. 5 shows the operating frequencies of computer C_4 and the achieved response times. The prediction horizon and sampling time for each L0 controller was set to $N_{L0} = 3$ and $T_{L0} = 30$ seconds, respectively. The weights were set to $Q = 100$ and $R = 1$ in the cost function to penalize the controller heavily if a chosen operating frequency failed to satisfy r^* .

Control Overhead. The L1 controller examines an average of 858 system states during each sampling pe-

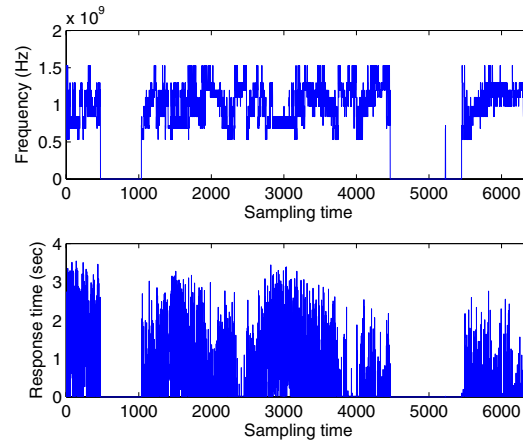


Figure 5: The operating frequencies selected by the L0 controller for computer C_4 and the achieved response times

riod while the number of states explored by the L0 controller depends on the number of operating frequencies available on the underlying computer (see Fig. 3). The combined execution time for the L0 and L1 controllers within the module was approximately 2.0 seconds¹.

The above simulations were also repeated for modules with six and ten computers (after appropriately scaling the original workload in Fig. 4), where γ_{ij} was quantized in coarser intervals of 0.1. For $m = 6$ and $m = 10$ the execution times incurred by the control hierarchy were 1.1 and 2.0 seconds, respectively, indicating that the run-time overhead of the hierarchical control structure in Fig. 2(b) can be kept low for modules of moderate size.

Finally, though we have used abstract weights in the cost functions for the L0 and L1 controllers to illustrate the key concepts, these cost functions can be “scalarized” by assigning an actual dollar amount to each term; for example, dollars earned by achieving the desired response time and the cost of operating the cluster (dollars per Watts consumed).

5 L2 Controller Design

Multiple modules are now composed to achieve scalable control of a larger system. As shown in Fig. 2(a), the L2 controller decides the fraction $\{\gamma_i\}$ of the incoming arrivals to distribute to each of the p modules.

5.1 Control Problem

The design of the L2 controller is conceptually similar to that of the L1 controller. During each sampling time

¹The simulations were run using MATLAB on a 3.0 GHz Pentium processor with 1 GB of RAM

step T_{L2} , the L2 controller obtains the current state of module M_i in the form of the average queue length and request processing time, estimates the environment parameters $\hat{\omega}_{L2} = (\hat{\lambda}_g, \hat{c}_{L2})$, and decides γ_i . The control problem at this level is given as follows.

$$\min_{U_{L2}} \sum_{q=k_{L2}}^{k_{L2}+N_{L2}} \sum_{i=1}^p \tilde{J}_i(x_{L2}(q), \gamma_i(q)) \quad (15)$$

Subject to: $\hat{x}_{L2}(q+1) = h(x_{L2}(q), u_{L2}(q), \hat{\omega}_{L2}(q))$

$$\sum_{i=1}^p \gamma_i(q) = 1$$

$$\gamma_i(q) \geq 0, \gamma_i(q) \in \mathbb{R}$$

Here, x_{L2} is an aggregation of module states in terms of the average queue length and the request processing time, and U_{L2} denotes the set of control options (in terms of γ) available to the L2 controller.

Given the observed state x_{L2} and the estimated environment parameters $\hat{\omega}_{L2}$, the L2 controller must obtain the cost incurred by a module M_i for various choices of γ_i . However, M_i 's behavior includes complex and non-linear interaction between its L0 and L1 controllers, and the resulting dynamics cannot be captured via explicit mathematical equations. A detailed model for each M_i will also increase the L2 controller's overhead substantially, defeating our goal of scalable hierarchical control. Therefore, we apply *simulation-based learning* techniques [9] to generate an architecture that quickly approximates M_i 's behavior. In the above formulation, \tilde{J}_i is the output of this architecture for M_i in response to inputs from the control set U_{L2} .

The behavior of module M_i is learned by simulating the control structure in Fig. 2(b) with a large number of training inputs from the domains of x_{L2} , ω_{L2} , and U_{L2} . Once an approximation is obtained off-line, it can be used by the L2 controller to generate decisions fast enough for use in real time. We use a compact regression tree [11] to store \tilde{J} values for the experiments reported in Section 5.3. A module is first simulated and the corresponding cost values stored in a large lookup table. This table is then used to train a regression tree.

5.2 Performance Analysis

The performance of the hierarchy comprising the L2, L1, and L0 controllers is now evaluated using a workload trace from WC'98. Without loss of generality, we consider a cluster of sixteen heterogeneous computers (after capacity planning for the workload of interest), partitioned into four modules, each comprising four computers. Also, the modules themselves are heterogeneous, i.e., different sets of computers are present

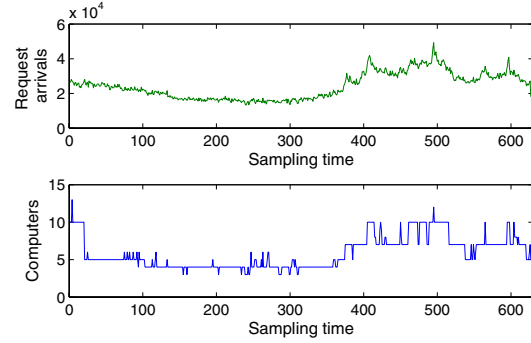


Figure 6: Workload trace from WC'98 and the number of computers operated

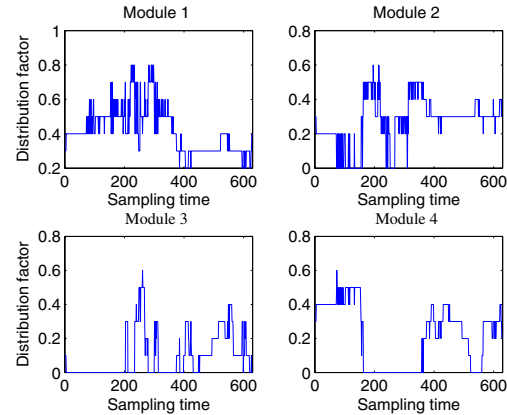


Figure 7: The load distribution factor γ_i decided by the L2 controller for each of the four modules

within each module. Fig. 6 shows the workload of interest, plotted in 2-minute intervals, and the number of computers operated by the control hierarchy in response to workload fluctuations. The desired response time was $r^* = 4$ seconds. The prediction horizon of the L2 controller was set to $N_{L2} = 1$ and the sampling time T_{L2} was two minutes. The load distribution factor for the i^{th} module, $0 \leq \gamma_i \leq 1$, was quantized in 0.1 intervals.

Fig. 7 shows the workload distribution fractions (γ) decided by the L2 controller for each module. The prediction horizons and sampling times for the L0 and L1 controllers were maintained at the values described in Section 4.3. The observed module parameters, in terms of average operating frequencies, response times, and incurred costs, were qualitatively similar to those shown in Fig. 5 and the desired response time $r^* = 4$ was achieved throughout.

The average execution time of the hierarchical optimization scheme is simply the sum of the controller execution times along any one path of the hierarchy shown in Fig. 2(a), and is 2.5 seconds for the cluster

of sixteen computers when γ_{ij} for the j^{th} computer in M_i is quantized in intervals of 0.05, and γ_i is quantized in intervals of 0.1. (We have also performed similar experiments on a cluster of twenty computers, partitioned into five modules, with an average execution time of about 3.4 seconds.)

6 Conclusions

We have presented a hierarchical control framework to design a self-managing DCS that aims to satisfy QoS requirements under dynamic operating conditions. The concepts of temporal and control decomposition, and function approximation were used to achieve scalable control. As a case study, we developed a three-level hierarchical structure to operate a computer cluster in energy-efficient fashion under a time-varying workload. The performance of the controller was evaluated using representative workload from WC'98, and our results indicate that the proposed framework is scalable, has low run-time overhead, and adapts well to dynamic operating conditions.

References

- [1] S. Abdelwahed, N. Kandasamy, and S. Neema. Online control for self-management in computing systems. In *Proc. IEEE Real-Time & Embedded Technology & Applications Symp.*, pages 365–375, 2004.
- [2] S. Abdelwahed, G. Karsai, and G. Biswas. Online safety control of a class of hybrid systems. In *Proc. IEEE Conference on Decision and Control*, pages 1988–1990, 2002.
- [3] S. Abdelwahed, R. Su, and S. Neema. On the feasibility of look-ahead control for systems with a finite control set. In *Proc. IEEE Conference on Control Application*, 2005.
- [4] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control theoretic approach. *IEEE Trans. Parallel & Distributed Syst.*, 13(1):80–96, January 2002.
- [5] Advanced Micro Devices Corp. *Mobile AMD-K6-2+ Processor Data Sheet*, June 2000.
- [6] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-99-35R1, Hewlett-Packard Labs., September 1999.
- [7] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Proc. ACM SIGMETRICS Conf.*, pages 126–137, 1996.
- [8] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. ACM SIGMETRICS Conf.*, pages 151–160, 1998.
- [9] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
- [10] G. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice-Hall, Upper Saddle River, New Jersey, 3 edition, 1994.
- [11] L. Breiman. *Classification and Regression Trees*. CRC Press, Boca Raton, FL, 1998.
- [12] A. E. Bryson. *Applied Optimal Control: Optimization, Estimation, and Control*. Hemisphere Pub. Corp, 1984.
- [13] A. Cervin, J. Eker, B. Bernhardsson, and K. Arzen. Feedback-feedforward scheduling of control tasks. *J. Real-Time Syst.*, 23(1-2), 2002.
- [14] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. Workshop Power Aware Computing Systems (PACS)*, 2002.
- [15] A. G. Ganek and T. A. Corbi. The dawn of the autonomous computing era. *IBM Systems Journal*, 42(1):5–18, 2003.
- [16] A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge, United Kingdom, 2001.
- [17] Intel Corp. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*, 2004.
- [18] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via online control: Application to power management. In *Proc. IEEE Int'l Conf. Autonomic Computing*, pages 54–62, 2004.
- [19] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proc. USENIX Conf. File Storage Tech.*, pages 219–230, 2002.
- [20] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-Time Systems*, 23(1/2):85–126, 2002.
- [21] Z. Lu et al. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proc. Int'l Conf. Compilers, Architectures, & Synthesis Embedded Syst. (CASES)*, pages 156–163, 2002.
- [22] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, London, 2002.
- [23] D. Menasce et al. In search of invariants for e-business workloads. In *Proc. ACM Conf. Electronic Commerce*, pages 56–65, 2000.
- [24] S. Parekh et al. Using control theory to achieve service level objectives in performance management. In *Proc. IFIP/IEEE Int. Symp. on Integrated Network Management*, 2001.
- [25] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proc. Workshop Compilers & Operating Systems for Low Power*, 2001.
- [26] V. Sharma et al. Power-aware qos management in web servers. In *Proc. IEEE Real-Time Systems Symposium*, pages 63–72, 2003.
- [27] A. Sinha and A. P. Chandrakasan. Energy efficient real-time scheduling. In *Proc. Intl Conf. Computer Aided Design (ICCAD)*, pages 458–463, 2001.