An Adaptive Optimal Controller for Non-Intrusive Performance Differentiation in Computing Services

Magnus Karlsson, Xiaoyun Zhu, Christos Karamanolis
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-20
February 7, 2005*

E-mail: {magnus.karlsson, xiaoyun.zhu, christos.karamanolis}@hp.com

QoS, performance differentiation, control theory, adaptive enterprise, utility computing

Shared computing services must control resource usage to meet contractual performance goals for hosted customers. They must ensure performance isolation among the workloads of different customers and enforce prioritization when the service is overloaded. Existing solutions are domain-specific and require modifications to the service. We propose a generic, non-intrusive approach that uses a fair scheduler to intercept incoming requests and enforce proportional sharing of the service resources among workloads. The relation between workload shares and obtained performance varies over time depending on system dynamics. We design an adaptive optimal MIMO controller that dynamically sets the workload shares based on the observed performance. Experimental results from an NFS server and a 3-tier e-commerce site demonstrate that the controller achieves effective performance differentiation, even when the system state or the performance goals change significantly.

* Internal Accession Date Only
To be published in the International Conference on Control and Automation, 26-29 June 2005, Budapest, Hungary
Approved for External Publication

An Adaptive Optimal Controller for Non-Intrusive Performance Differentiation in Computing Services

Magnus Karlsson, Xiaoyun Zhu and Christos Karamanolis
HP Labs, Palo Alto, CA 94304, U.S.A.
{magnus.karlsson,xiaoyun.zhu,christos.karamanolis}@hp.com

***Abstract*— Shared computing services must control resource usage to meet contractual performance goals for hosted customers. They must ensure performance isolation among the workloads of different customers and enforce prioritization when the service is overloaded. Existing solutions are domain-specific and require modifications to the service. We propose a generic, non-intrusive approach that uses a fair scheduler to intercept incoming requests and enforce proportional sharing of the service resources among workloads. The relation between workload shares and obtained performance varies over time depending on system dynamics. We design an adaptive optimal MIMO controller that dynamically sets the workload shares based on the observed performance. Experimental results from an NFS server and a 3-tier e-commerce site demonstrate that the controller achieves effective performance differentiation, even when the system state or the performance goals change significantly.**

I. Introduction

Service providers and enterprises are increasingly hosting services and applications on shared pools of computing and storage resources. Multiplexing workloads onto a shared infrastructure allows for on-demand allocation of resources to workloads and, thus, can improve overall resource efficiency by overbooking. Resource sharing and consolidation create a need to control how competing clients' workloads consume the resources of a shared computing service, such as a database server, a 3-tier e-commerce system, or a file-server.

Ensuring *performance differentiation* among the workloads that share the service resources is a key requirement in such environments. First, the service must enforce *application-level performance goals*, response times and throughputs in particular. Second, it must provide *performance isolation* to ensure that the actions of one workload cannot negatively affect the performance of other workloads. Third, it must enforce a *prioritization* among workloads when the service capacity is not sufficient to meet all the goals, e.g., due to overbooking.

A solution to this problem must satisfy three properties. First, it has to be *adaptive*, as the relationship between allocated resources and the performance a workload receives may vary over time. For example, the performance experienced by the clients of a file server depends on the ratio of requests that are served by the in-memory cache of the server; this ratio may change over time depending also on what other clients are doing. Second, it must be *non-intrusive*, as most computing services have no native support for performance differentiation and, in the general case, cannot be easily modified to do so. Third, it must automatically detect workload correlations due to dependencies on internal service components—a computing service typically comprises an ensemble of hardware and software resources (servers, disks, queues, network links, etc).

This paper focuses on providing performance differentiation according to the above requirements. We propose a solution that is applicable to many computing services. Our approach is based on interposing a fair-queuing scheduler [6] on the request path between a service and its clients, as illustrated in Figure 1. The scheduler enforces a configurable share of the service's capacity that each workload receives. Based on the observed throughputs and response times of the workloads, a MIMO controller sets the share of each workload to achieve performance differentiation.

Intercepting and controlling the workloads that access a computing service has been previously proposed in the context of 3-tiered Web sites [7] and storage systems [2], [8], [13]. All these approaches attempt to enforce some performance goals under certain assumptions about the workload behavior and the system state. With the exception of Triage [8], none of these approaches can ensure performance differentiation when the workloads or system deviate from those specifications. Triage provides differentiated throughput allocation, but only for fixed latency goals. Moreover, no existing approaches can automatically detect and deal with workload dependencies on internal resources. For example, Triage penalizes all workloads when a performance degradation occurs due to an internal conflict between just two workloads. Last but not least, most existing approaches are designed assuming some knowledge of the target computing service. There have been a number of intrusive control theoretic approaches that provide performance differentiation by modifying the target service and/or using application-specific hooks. The systems targeted include Web servers [4], [12], [16], e-mail servers [14], databases [15], file systems [10] and middleware platforms [11].

This paper proposes the use of an *adaptive optimal controller* for providing non-intrusive performance differentiation in computing systems. We formulate an optimal control problem such that conforming to the performance differentiation specification equals a constrained performance optimization. The controller needs to be adaptive to track workload and system dynamics. Finally, the use of a *MIMO*

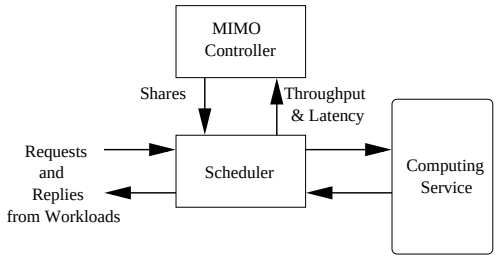

Fig. 1. Architecture of a generic, non-intrusive approach to performance differentiation in computing services.

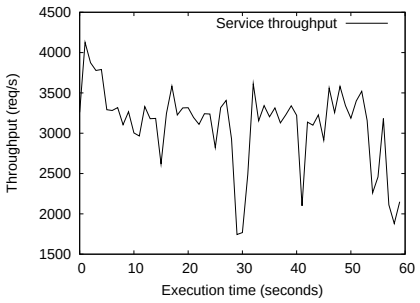

Fig. 2. The throughput of the NFS server described in Section IV as it varies over time.

model is imperative for correctly estimating performance correlations among the workloads.

II. Problem Formulation

We assume a system architecture as depicted in Figure 1. A computing service, comprising an ensemble of resources, is accessed by a number of *clients*. The service can be any computing system that given a request returns a reply with some data. Examples of such computing services include file servers, 3-tier systems, Web servers, and databases. Client requests are grouped into service classes called *workloads*. A performance goal is associated with a workload. The objective is to meet the performance goals of the workloads by controlling the rates at which they consume the service resources. To do this, we use a fair-queuing scheduler, which is interposed on the path between the service and its clients [6]. The scheduler intercepts the requests sent to the service; it limits the resource consumption of each workload in proportion to a *share* assigned to that workload. A controller sets the shares dynamically; it enforces performance differentiation by setting each individual workload's share appropriately given latency and throughput measurements.

To derive a model of throughput and response time as a function of the share assignment, assume for convenience that the shares of a service's workloads sum up to 1. That is, if a workload has a share of 0.3, it is entitled to 30% of the service's capacity to serve requests. If the service is fully utilized, the workload never receives anything above the share it is allocated. Thus, the scheduler enforces performance isolation among workloads. In the case of *throughput goals*, the share is directly related to the portion of the service's total throughput a workload receives. Given that the total service throughput is sufficiently large to avoid quantization effects (i.e., the sample period is adequately long), this relation can be assumed to be linear. However, the total throughput of a computing service varies over time, as shown in Figure 2. This may be due to a number of reasons including other traffic on the network, in-memory caching effects, or background tasks (e.g., backup scripts) executing in the service. In the case of *latency goals*, queuing theory implies that the relation between the share and latency is nonlinear. For a high share, there is little difference in latency when the share is adjusted. But an adjustment when the share is small makes a large difference in the response time. Locally, though, a linear approximation of this relation is adequate. For the same reasons that throughput varies given a fixed share setting, latency also varies. To approximate these system characteristics with a linear model, we use adaptive control.

Given the assumptions above, performance differentiation in a computing service can be formulated as a linear quadratic optimization problem. Let $u(k) = [u_1(k) \dots u_N(k)]^T$ be the vector of the shares for the N workloads of the service during sampling interval k. Let $y(k) = [y_1(k) \dots y_N(k)]^T$ be the vector of the performance metrics (either latency or throughput) of the N workloads, measured at the beginning of interval k. Performance differentiation can then be accomplished by minimizing the quadratic cost function $\|W(y_{ref}(k) - y(k))\|^2$, subject to the constraint that the shares sum up to one, i.e., $\sum_{i=1}^{N} u_i(k) = 1$. Here, the vector $y_{ref}(k) = [y_{ref,1}(k) \dots y_{ref,N}(k)]^T$ captures the desired performance goals of the N workloads and $\|\cdot\|$ is the 2-norm in $\Re^N$. The cost is zero when all workloads achieve the desired performance goals. In this way, we can achieve *independent performance goals* for each workload.

The *weighting matrix* W in the cost function, commonly chosen as a diagonal matrix, provides a means of prioritization among workloads. When the capacity of the system is not adequate to satisfy all workloads and $W = I$, then the cost function is minimized when the performance of all workloads is degraded equally. If, instead, $W \neq I$, then workloads are prioritized. For example, $W_{11} > W_{22}$ implies that workload 1 has higher priority than 2. Under overload conditions, the cost function is minimized by degrading more of the performance of workload 2 than that of workload 1.

III. An Adaptive Controller for Performance Differentiation

A. Recursive Parameter Estimation

We consider a service, which is shared by N concurrent workloads. We assume that the mapping from the N workload shares ($u(k)$) to the N measured performance metrics ($y(k)$), such as response times or throughputs, can be described by the following multiple-input-multiple-output (MIMO) model:

$$A(q^{-1})y(k) = B(q^{-1})u(k) + e(k) \quad (1)$$

where $A(q^{-1})$ and $B(q^{-1})$ are matrix polynomials in the backward-shift operator:

$$\begin{aligned} A(q^{-1}) &= I - A_1 q^{-1} - \ldots - A_n q^{-n} \\ B(q^{-1}) &= B_0 q^{-1} + \ldots + B_{n-1} q^{-n} \end{aligned} \quad (2)$$

Note that $A_i \in \Re^{N\times N}$, $B_j \in \Re^{N\times N}$, $0 < i \leq n$, $0 \leq j < n$, where n is the order of the system. $\{e(k)\}$ is a sequence of independent, identically distributed N-dimensional random vectors with zero means. It is further assumed that $e(k)$ is independent of $y(k-j)$ and $u(k-j)$ for $j > 0$. We use $e(k)$ to represent disturbances in the system that are not accounted for by the model. The linear model was chosen for tractability, as the real system dynamics will indeed, in all but the most trivial cases, be nonlinear.

The reason for choosing a MIMO model over a number of independent SISO models is that it allows us to capture correlations between various shares and performance measurements. For example, increasing the share of one workload may decrease the performance of another if they use the same resource within the service. Such dependencies cannot be captured by individual SISO models. Moreover, a MIMO controller facilitates providing flexible policies when the system does not have enough capacity to meet all individual goals. For example, ensuring that all workloads receive the same performance degradation would be difficult to enforce using independent SISO controllers, as it requires coordination among the controllers.

Some form of system identification is required to estimate the order of the system (n) as well as the parameter matrices A_i and B_j, with $0 < i \leq n$ and $0 \leq j < n$. The estimated parameters are then used in the controller design. n is usually fixed and low in computer systems [5] and can be estimated offline. However, the values of the parameter matrices are likely to vary in a typical computing service due to changes in system operating conditions and workload dynamics. For example, a workload may shift from accessing data from the disk to accessing data from the in-memory cache of a server. Such a change would result in more than an order of magnitude better performance, and thus would require different model parameters. Therefore, we use an adaptive controller where model parameters are estimated on-line and controller parameters are calculated accordingly.

For notational convenience, we rewrite the system model in the following form, which we use in the rest of the paper:

$$y(k+1) = X(k)\phi(k) + e(k+1) \quad (3)$$

where

$$\begin{aligned} X(k) &= [B_0 \quad \ldots \quad B_{n-1} \quad A_1 \quad \ldots \quad A_n] \\ \phi(k) &= [u^T(k) \ \ldots \ u^T(k-n+1) \ y^T(k) \ \ldots \ y^T(k-n+1)]^T \end{aligned}$$

We use a recursive least squares (RLS) estimator with exponential forgetting to estimate the time varying parameter matrix $X(k)$. This type of estimator has been used extensively in adaptive control systems as it converges fast and rejects noise well. The estimator is defined by the following equations:

$$\begin{aligned} \hat{X}(k+1) &= \hat{X}(k) + \frac{\varepsilon(k+1)\phi^T(k)P(k-1)}{\lambda + \phi^T(k)P(k-1)\phi(k)} \\ \varepsilon(k+1) &= y(k+1) - \hat{X}(k)\phi(k) \\ P^{-1}(k) &= P^{-1}(k-1) \\ &\quad + (1 + (\lambda - 1)\frac{\phi^T(k)P(k-1)\phi(k)}{(\phi^T(k)\phi(k))^2})\phi(k)\phi^T(k) \end{aligned} \quad (4)$$

where $\hat{X}(k)$ is the estimate of the true value of $X(k)$, $\varepsilon(k)$ is the estimation error vector, $P(k)$ is the covariance matrix and λ is the *forgetting factor* ($0 < \lambda \leq 1$). We use *directional forgetting* [9] to avoid wind-up of the covariance matrix.

B. Linear Quadratic Controller Design

For the controller design, we aim at minimizing the following quadratic cost function:

$$J = E\{\|W(y(k+1) - y_{ref}(k+1))\|^2 + \|Qu(k)\|^2\} \quad (5)$$

$$s.t. \quad \sum_{i=1}^{N} u_i(k) = 1 \quad (6)$$

where $W \in \Re^{N\times N}$ is a positive-semidefinite weighting matrix on the tracking errors and $Q \in \Re^{N\times N}$ is a positive-definite weighting matrix on the control settings.

The goal of the controller is to steer the system into a state of optimum reference tracking with minimum variance, while penalizing large control settings. The W and Q weighting matrices are commonly chosen as diagonal matrices. Their relative magnitude provides a way to trade-off tracking accuracy for stability. Also, in the case of W, the values of the diagonal terms capture the relative priorities of the corresponding workloads, when the service is overloaded.

The minimization should be over the set of all admissible controllers, where a controller is admissible if each control action $u(k)$ is a function of vector $\phi(k-1)$ and of the new measurements $y(k)$. In the following, we derive the optimal controller by explicitly capturing the dependency of the cost function J on $u(k)$. We first define

$$\tilde{\phi}(k) = [0 \ u^T(k-1) \ \ldots \ u^T(k-n+1) \ y^T(k) \ \ldots \ y^T(k-n+1)]^T \quad (7)$$

Then we have,

$$\begin{aligned} J &= E\{\|W(y(k+1) - y_{ref}(k+1))\|^2 + \|Qu(k)\|^2\} \\ &= E\{\|W(\hat{X}(k)\tilde{\phi}(k) + \hat{B}_0 u(k) + \varepsilon(k+1) \\ &\quad - y_{ref}(k+1))\|^2\} + \|Qu(k)\|^2 \\ &= \|W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1))\|^2 + \|W\hat{B}_0 u(k)\|^2 \\ &\quad + 2u^T(k)\hat{B}_0{}^T W^T W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1)) + \\ &\quad + \|Qu(k)\|^2 + E\{\|W\varepsilon(k+1)\|^2\} \end{aligned}$$

Without the normalization constraint (6), the cost function J is at its minimum where the following derivative is zero.

$$\begin{aligned} \frac{\partial J}{\partial u(k)} = \ & 2(W\hat{B}_0)^T W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1)) \\ & + 2(W\hat{B}_0)^T W\hat{B}_0 u(k) + 2Q^T Qu(k) = 0 \end{aligned} \quad (8)$$

```
1   k = 0, choose X̂(0) and P(0)
2   k = k+1 at the beginning of every sampling interval
3   Obtain new measurements y(k)
4   if total throughput for all the workloads < 5
5       u(k) = u(k−1); go to step 2
6   estimate new model parameters X̂(k) using RLS (4)
7   calculate u(k) according to the optimal control law (14)
8   if min_i{u_i(k)} < 0 ∀i
9       u_i(k) = u_i(k) − min_i{u_i(k)} ∀i
10      normalize the u_i(k)'s s.t. they add up to 1
11  go back to step 2
```

Fig. 3. Pseudo-code description of the adaptive controller.

Solving for $u(k)$ would give us the following unconstrained optimal control law.

$$u^*_{uc}(k) = ((W\hat{B}_0)^T W\hat{B}_0 + Q^T Q)^{-1} (W\hat{B}_0)^T W(y_{ref}(k+1) - \hat{X}(k)\tilde{\phi}(k)) \quad (9)$$

Note that $\hat{X}(k)$ and $\hat{B}_0$ are estimates of the model parameters obtained using the RLS estimator ((4)). The resulting controller is referred to as the linear quadratic self-tuning regulator [1]. The derivation of the control law here is adapted from the controller synthesis in [17].

Now, let us take the normalization constraint (6) into account using a Lagrangian multiplier, λ_L. Let

$$L(u(k),\lambda_L) = J + \lambda_L(\sum_{i=1}^{N} u_i(k) - 1) \quad (10)$$

The derivative equation can be updated as follows:

$$\frac{\partial L(u(k),\lambda_L)}{\partial u(k)} = \frac{\partial J}{\partial u(k)} + \lambda_L I_N = 0 \quad (11)$$

where $I_N = [1\ldots1]^T \in \Re^N$ is a constant vector. Again, solving for $u(k)$ gives us the following adjusted control law:

$$u^*(k) = u^*_{uc}(k) - \frac{\lambda_L}{2} S I_N \quad (12)$$

where $S = ((W\hat{B}_0)^T W\hat{B}_0 + Q^T Q)^{-1}$ can be viewed as a scaling matrix. λ_L can be calculated by substituting the above $u(k)$ into equation (6). Then we have,

$$\lambda_L = \frac{2(I_N^T u^*_{uc}(k) - 1)}{I_N^T S I_N} \quad (13)$$

Therefore, the constrained optimal control law is

$$u^*(k) = u^*_{uc}(k) - \frac{I_N^T u^*_{uc}(k) - 1}{I_N^T S I_N} S I_N \quad (14)$$

As we can see, $\lambda_L = 0$ when $I_N^T u^*_{uc}(k) - 1 = 0$, i.e., the unconstrained optimal control law already obeys the normalization constraint. Otherwise, individual shares are adjusted according to the scaling matrix S so that the sum equals 1.

The pseudo-code in Figure 3 outlines the algorithm that implements the above control law. Since the adaptive controller requires that the structure of the prediction model be fixed, we determine the order and time delay of the system in offline system identification experiments. These experiments also help us choose initial values for $\hat{X}(0)$ and $P(0)$. The above algorithm also handles special cases such as when there are not enough samples or when at least one control setting is negative. It has shown to work well in our experimental evaluations.

IV. Experimental Evaluation

In this section, we present experimental results to demonstrate the applicability of the proposed controller to real systems. In particular, we make the following points:

- The controller can adapt to workload and system dynamics as well as to changes in performance goals.
- The linear quadratic optimal controller can be used to prioritize among workloads when the service is overloaded.

A. Experimental Methodology

We perform our experimental evaluation on two different systems. First, we use a 3-tier e-commerce system that consists of three components: a Web server, an application server and a database. Client requests arrive at the Web server. Unless they are for static content, they are forwarded to the application server, which creates a dynamic page by accessing the database. The generated page is then sent to the client. The workload applied mimics real-world user behavior [3], e.g., browsing, searching and purchasing behaviors including their respective time scales and occurring probabilities. All three components of the system run on separate machines. For the experiments here, we emulate 75 users partitioned into two performance classes. Each class is considered to be one "workload". The scheduler and controller are implemented as part of the workload generator program, `httperf`[1]. The control interval for gathering measurements and setting the shares is 1 s.

Second, we use an NFS file server that is shared by multiple workloads. Each workload performs repeated read operations on its own data set stored on the server. The full data set is large enough that it does not fit in the in-memory cache of the file server. The scheduler and controller are implemented in a proxy server which runs on a separate machine. For NFS, we use a control interval of 500 ms.

B. Evaluation Results

Figure 4 depicts a 90-second execution window from the 3-tier system with two client workloads. The two workloads have throughput goals of 80 and 50 requests/s respectively. Workload 2 has higher priority. Initially both workloads access more static content, e.g., as a result of browsing pre-generated product information pages. Over time, the workloads access increasingly more dynamic content, as a result of performing operations such as purchasing items. At first, both goals could be met. But as the behavior of the workloads changes, the capacity of the service becomes insufficient to meet the goals of both workloads. Thus, the controller tries to meet the 50 requests/s goal of the high-priority workload while penalizing the low-priority one, as

[1] ftp://ftp.hpl.hp.com/pub/httperf

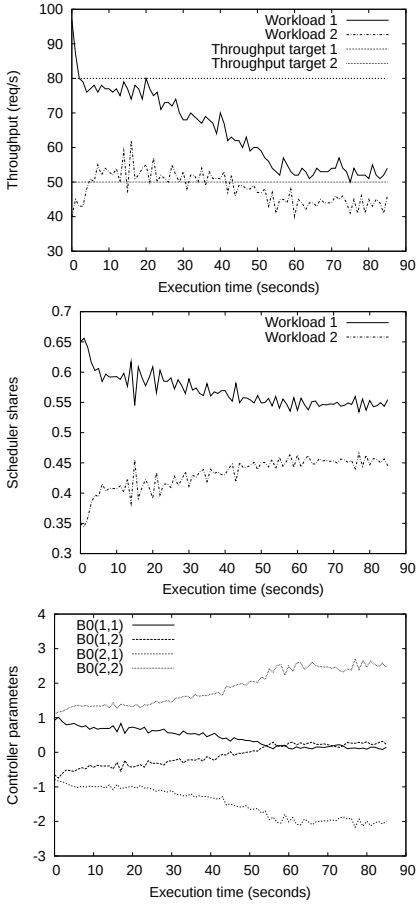

Fig. 4. Results for the 3-tier system with throughput goals.

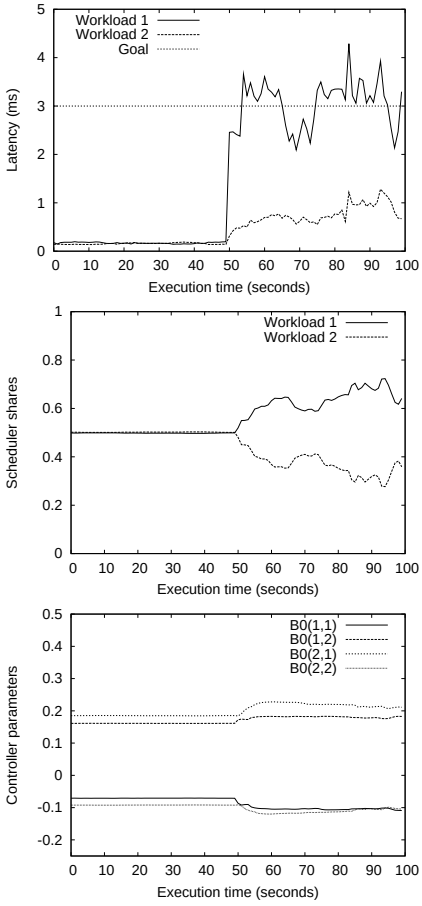

Fig. 5. Results for the NFS system with latency goals.

shown by the top graph in the figure. The throughput of workload 2 drops a little from its goal, but only as much as specified in W. The second graph shows the workload shares in the scheduler, as set by the controller during the execution window. The third graph shows the elements in the estimated B_0 matrix in the controller and demonstrates how they adapt as the workload characteristics change.

Figure 5 depicts a 100-second execution window from the NFS server with two client workloads, each client with two processes. In this case, both workloads are of equal importance and have a response latency goal of 3 ms. For the first 50 seconds of the execution, both workloads access a small percentage of their data sets and thus all requests are served from the server's in-memory cache (the cache has been pre-warmed). Their response latency is well below their goals and thus both have their shares set to 0.5. At 50 seconds, workload 1 becomes disk-bound by iterating over a much larger portion of its data set. As a result, the latency of workload 1 increases and thus the controller has to adjust the weights to track the performance goals. The latency of workload 2 now varies much more as the latency of disk accesses is more unpredictable than that of cache accesses. In this experiment, the controller adapts to a much more rapid change than that of the previous experiment. The controller adapts in a similar fashion when there are changes

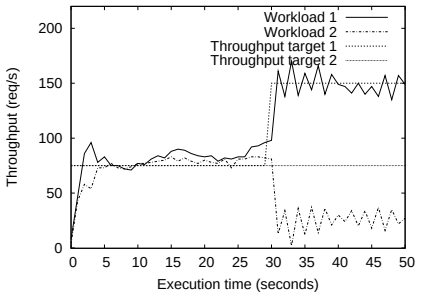

Fig. 6. Adapting to performance goal changes.

in the service (as opposed to the workloads). For example, when a background task starts on the server affecting its overall performance.

Last, we show how the controller adapts to varying performance goals. Figure 6 depicts a 50-second execution window from the 3-tier system. There are two workloads that both initially have throughput goals of 75 requests/s. Workload 1 has a higher priority. At time 30 seconds, the goal of workload 1 is revised to 150 requests/s. We see how the controller adjusts quickly to track the goal change. It meets the new goal of workload 1 while it penalizes the low-priority workload.

V. Conclusions and Future Work

In this paper, we propose a generic, non-intrusive approach for enforcing performance differentiation among workloads that share a computing service. We can enforce both latency and throughput goals, provide performance isolation and prioritize the performance of workloads under overload. An adaptive optimal controller is used to dynamically translate performance goals into the share of the service resources that a workload receives. Experimental evaluation of a prototype has demonstrated that our approach achieves performance differentiation even when the system and the workload conditions change.

Many challenges remain. Currently, manual tuning of the weighting matrices W and Q is required for different target services. We are investigating techniques to simplify or eliminate this step. We also plan to examine controllers that can provide more flexibility in the way workloads are prioritized under overload, by using utility functions.

Acknowledgments

The authors would like to thank Julie Symons for her all help with the experimental system.

References

[1] K. J. Åström and B. Wittenmark. *Adaptive Control*. Electrical Engineering: Control Engineering. Addison-Wesley Publishing Company, 2 edition, 1995. ISBN 0-201-55866-1.

[2] D. Chambliss, G. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. Lee. Performance virtulization for large-scale storage systems. In *Symposium on Reliable Distributed Systems (SRDS)*, pages 109–118, Florence, Italy, October 2003.

[3] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to systems states: A building block for automated diagnosis and control. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, page xxx, San Francisco, CA, December 2004.

[4] Y. Diao, J. Hellerstein, and S. Parekh. MIMO control of an Apache web server: Modeling and controller design. In *American Control Conference (ACC)*, pages 4922–4927, Anchorage, AK, May 2002.

[5] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, August 2004. ISBN: 0-471266-37-X.

[6] W. Jin, J. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. In *International Conference on Measurement and Modelling of Computer Systems (SIGMETRICS)*, pages 37–48, New York, NY, USA, June 2004.

[7] A. Kamra, V. Misra, and E. Nahum. Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web sites. In *International Workshop on Quality of Service (IWQoS)*, pages 47–56, Montreal, Canada, June 2004.

[8] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance Isolation and Differentiation for Storage Systems. In *International Workshop on Quality of Service (IWQoS)*, pages 67–74, Montreal, Canada, June 2004.

[9] R. Kulhavý. Restricted exponential forgetting in real-time identification. *Automatica*, 23(5):589–600, September 1987.

[10] H. D. Lee, Y. J. Nam, and C. Park. Regulating I/O performance of shared storage with a control theoretical approach. In *NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST)*, College Park, ML, April 2004.

[11] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *International Workshop on Quality of Service (IWQoS)*, pages 145–153, Napa, CA, May 1998.

[12] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *IEEE Real Time Technology and Applications Symposium (RTAS)*, pages 51–62, Taipei, Taiwan, June 2001.

[13] C. Lumb, A. Merchant, and G. Alvarez. Façade: Virtual storage devices with performance guarantees. In *International Conference on File and Storage Technologies (FAST)*, pages 131–144, San Francisco, CA, March 2003.

[14] S. Parekh, J. Hellerstein, T. Jayram, N. Gandhi, D. Tilbury, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Journal of Real-Time Systems*, 23(1-2):127–141, July-September 2002.

[15] S. Parekh, K. Rose, Y. Diao, V. Chang, J. Hellerstein, S. Lightstone, and M. Huras. Throttling Utilities in the IBM DB2 Universal Database Server. In *American Control Conference (ACC)*, pages 1986–1991, Boston, MA, June 2004.

[16] A. Robertsson, B. Wittenmark, and M. Kihl. Analysis and design of admission control in web-server systems. In *American Control Conference (ACC)*, Denver, CO, June 2003.

[17] L. Sun, J. Krodkiewski, and Y. Cen. Control Law Synthesis for Self-Tuning Adaptive Control of Forced Vibration in Rotor Systems. In *International Symposium MV2 on Active Control in Mechanical Engineering*, pages S9–25–37, Lyon, France, October 1997.