MIMO Control of an Apache Web Server: Modeling and Controller Design[1]

N. Gandhi and D. M. Tilbury
The University of Michigan
Mechanical Engineering Department
Ann Arbor, MI 48109-2125
{gandhin, tilbury}@umich.edu

Y. Diao, J. Hellerstein, and S. Parekh
IBM T. J. Watson Research Center
30 Sawmill Parkway
Hawthorne, NY
{diao,hellers,sujay}@us.ibm.com

Abstract

This paper considers the efficacy of feedback control in improving the performance of computing systems. Computing systems typically have many competing performance goals which are affected by several external variables. A feedback control strategy is desirable because well established techniques exist to handle these performance trade-offs and external disturbances. In order to employ such a strategy, decisions need to be made about inputs, outputs, sample time, model type, and performance measures. This paper describes this process, which is often nebulous for computing systems, in the context of an Apache web server. A linear multi input multi output model of the system is identified experimentally and used to design several feedback controllers. Experimental results are presented showing the problems associated with a pure pole placement design and the effectiveness of LQR based techniques. The paper concludes with a discussion of future work.

1 Introduction

As computing systems become more widely deployed and used, there is increasing demand for performance improvement. Instead of attempting to optimize software systems for one particular situation, developers often expose many tuning parameters which can be set by the system administrator. Using these parameters, the administrator has the ability to optimize the system's performance in accordance with application specific high-level goals designed to satisfy various business needs.

Choosing the correct settings for tuning parameters is not a straightforward job. The best settings will depend on hardware, workloads, and any concurrent jobs running on the system. Since the workloads and concurrent jobs can change over time, a dynamic feedback control strategy is desirable. In order to employ this strategy, control input(s) and system output(s) must be selected. The control input can be chosen to be one or more of the tuning parameters discussed above. An output must be chosen that has meaning for the system and can be reasonably measured. First-principles models are difficult to construct for most computing systems; a "black-box" approach is typically more appropriate.

[1]This research was supported in part by IBM.

The application of traditional control strategies to computing systems encounters a number of obstacles. Control performance must be defined in a meaningful way, a system model must be constructed, a sample time must be chosen, and the control options must be evaluated. Although many of these steps are well-understood in electromechanical systems, the appropriate analogs for computing systems are still being defined. The definition of "good performance" is especially unclear in computing systems. Certainly, the system should not crash, but there usually does not exist a prespecified trajectory or reference that the system should follow. In addition, workloads in computing systems are highly stochastic, and even with well-defined workloads, the systems themselves exhibit significant stochastic behavior.

In this paper, we will outline the above challenges, and show how these obstacles have been overcome using an Apache web server as an example. Section 2 provides background on Apache and describes how the system outputs and control inputs were chosen. Section 3 details our approach to modeling. Section 4 presents and evaluates two controller designs: pole placement and linear quadratic regulator. Our conclusions are contained in Section 5.

2 System Output/Control Input Selection

The first step when implementing a feedback control strategy is the selection of control input(s) and system output(s). As mentioned in the introduction, the tuning parameters available on the system can be viewed as the control inputs. System outputs, on the other hand, must be chosen to reflect the high-level goal of the control strategy and thus should be representative of system performance.

2.1 Selection of System Outputs

A number of metrics are used to quantify performance of the Apache web server and thus are ideal candidates for system outputs. These metrics include: end-user response times, response times on the server, throughput, utilizations of various resources on the server, etc. Selection of the appropriate performance metrics will not only depend on the high-level goal of the control strategy, but also how easily the metrics are measured. The latter is especially important for feedback control.

One high-level goal of control may be to ensure some bound on end-user response times. This is a client-side metric; it cannot be measured by the server. Instrumentation may be added to measure end-user response times; however, this approach results in additional load on the server and is not always accurate. Using client-side metrics also introduces delays since information needs to be transferred between two systems. Because of these issues, a control strategy that seeks to limit end-user response times is not considered in this paper. However, it is an area of future work.

Another high-level goal may be to limit the CPU and memory utilizations (hereafter denoted by `CPU` and `MEM`) associated with the Apache application. Several business needs make these limits necessary, including: (a) providing sufficient capacity to co-located applications (e.g., file server, database server); (b) avoiding thrashing and failures as a result of overutilization; and (c) ensuring that there is sufficient capacity remaining to handle workload surges and/or server failures. `CPU` and `MEM` are server-side metrics and thus can be easily measured. For these reasons, `CPU` and `MEM` are used as the system outputs for this paper.

2.2 Selection of Control Inputs

There are two considerations when selecting the appropriate tuning parameters to use as control inputs in a feedback strategy: (1) the parameters must be dynamically changeable; (2) the tuning parameters must affect the selected performance metrics in a meaningful way. If the selected tuning parameters have little impact on the selected performance metrics, then creating a system model will become cumbersome and limit the efficacy of the control strategy. In the case of Apache, none of the available tuning parameters are dynamically changeable in the release version (it is necessary to reboot after any change is made). Hence, parameters that require minimal changes to the Apache source code are desired.

Apache v1.3 on Unix [1] is structured as a pool of worker processes monitored by a master process. Each worker process is responsible for handling communication with the web clients and can handle at most one connection at a time. In HTTP 1.1 [3], a new feature known as *persistent connections* was added where the TCP connection can be left open. This avoids the connection setup overhead for each request and thus reduces the response time perceived by end users. With this feature, either side may close the connection.

Two available tuning parameters that significantly affect utilization of the Apache server are "MaxClients" and "KeepAliveTimeout". The "MaxClients" parameter (abbreviated by `MC`) limits the size of the worker pool, thereby imposing a limitation on the processing capacity of the server. A higher `MC` value allows Apache to process more client requests increasing both `CPU` and `MEM`. But if `MC` is too large, there are excessive resource utilizations that degrade performance for all clients.

The "KeepAliveTimeout" parameter (abbreviated by `KA`) limits the *user think time*, the time between an HTTP reply and the receipt of the next client request. If this value is exceeded by the client, the connection is closed by the server. If `KA` is too large, `CPU` and `MEM` are underutilized since clients with requests to process cannot connect to the server. Reducing `KA` means that workers spend more time processing HTTP requests and so `CPU` increases. Although `KA` indirectly affects memory by allowing more clients to eventually connect to the server, increasing `KA` does not have the same effect on `MEM` that it does on `CPU`. A too small `KA` terminates the TCP connection prematurely and reduces the benefits of persistent connections.

As mentioned above, in the default version of Apache, `KA` and `MC` cannot be changed dynamically. As a result, the Apache source had to be modified to enable real-time control. A detailed description of these modifications can be found in [2]. For the remainder of the paper, the tuning parameters `KA` and `MC` are referred to as the control inputs.

3 Modeling Apache

This section describes our "black box" approach to modeling Apache. In mechanical or electrical systems, modeling is relatively straightforward because there are physical laws that govern the interaction between control inputs and system outputs (e.g., Newton's law). In computing systems, the relationship between control inputs and system outputs is not as clearly defined.

One approach is to start from first-principles and create a queueing model of the Apache server. However, this approach is not only complicated but would require detailed knowledge about the inner workings of the server. So instead of proceeding from first-principles, an empirical approach is used to quantify the relationship between the control inputs and the system outputs. This approach involves four main steps: (1) designing a sufficiently rich input signal; (2) collecting the data from the server; (3) using system identification techniques to construct statistical models from the data; (4) validating the models.

3.1 Experimental Environment

Our testbed consists of one server running Apache connected through a LAN to a client machine running a synthetic workload generator that simulates the activity of many clients. The workload model used to generate synthetic transactions is based on the WAGON model of Liu et al. [5] that has been validated in extensive studies of production web servers. The file access distributions we use are from the Webstone 2.5 reference benchmark [7].

Both control inputs have saturation regions implemented by the control code. `MC` is always an integer value in the range [1; 1024]. `KA` is in integral seconds, with a minimum of 1. No maximum value is enforced, however `KA` values larger than 50 have only a small effect on the system outputs given the nature of the workload. While it is feasible to have fractional values for `KA`, we have not changed default implementation of Apache, which uses integral values.

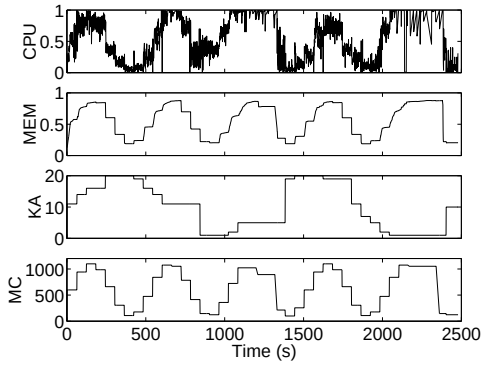


Figure 1: Experimental data with discrete sine wave inputs.

3.2 Design of Input Signals

The control inputs must be varied in a manner so that two properties are satisfied. First, the input signal should be persistently exciting; it should contain enough frequency content to excite all of the dynamics of the system [6]. In addition, there should be dense and uniform coverage of the operating region in which the model will be used. However, care is required to avoid highly nonlinear regions since a poor model fit will result, although separate models can be constructed for these regions.

In the case of the Apache server, the operating region is constructed by considering the saturation limits of the control inputs. Discrete sine waves are used for both `KA` and `MC`. This is done so that there are both high frequency components and low frequency components. The frequencies of the two sine waves were designed to be relatively prime within the length of time of the server run so that the individual effect of each control input can be properly determined. Figure 1 plots the data from the server run using the discrete sine waves as the control inputs.

3.3 System Identification

There are a number of methods available to aid in the construction of a model that captures the relationship between the control inputs and system outputs. We chose to fit a linear time invariant (ARX) model to the data. If a linear model adequately captures the relationship between control inputs and system outputs, then linear control theory can be used to design a relatively simple feedback controller with guaranteed properties within a certain operating region. Even when a full nonlinear model of a system is available, the first step is often to design a controller based on its linearization. In addition, the model is used specifically for controller design and hence extremely accurate predictions are not required.

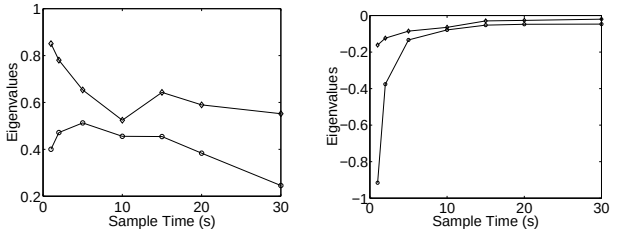
The form of the linear model is shown in (1), with parameters A and B estimated using least squares regression. Note that this is a MIMO model; A and B are both 2×2 matrices. A first-order model was chosen for simplicity and because increasing the order of the model did not significantly increase the quality of the model (the R^2 increased by less than 2%). We use time-averaged values of the system outputs to reduce measurement overhead and also because the inherent variability of the metrics (`CPU` in particular) makes instantaneous control impractical. Hence, $\texttt{CPU}_k, \texttt{MEM}_k$ in (1) denotes the average value of `CPU`, `MEM` over the time interval k.

$$\begin{bmatrix} \texttt{CPU}_{k+1} \\ \texttt{MEM}_{k+1} \end{bmatrix} = A \cdot \begin{bmatrix} \texttt{CPU}_k \\ \texttt{MEM}_k \end{bmatrix} + B \cdot \begin{bmatrix} \texttt{KA}_k \\ \texttt{MC}_k \end{bmatrix} \quad (1)$$

3.3.1 Selection of Sample Time: The choice of sample time is a key factor that affects the performance of the controller. In this system, the sample time not only determines the length of time between successive updates of the control inputs, but also the length of time system outputs are averaged over. In this sense, the sample time is also an averaging interval. A short sample time enables the controller to react to changes in the system quickly but increases measurement overhead. A long sample time keeps the controller from overreacting to random fluctuations by averaging out the stochastics of the metrics, but will also yield a slow response.

In order to negotiate these competing goals, a sample time study is performed. First-order linear models of the form in (1) were created at many different sample times using the data set plotted in Figure 1. Figure 2(a) and (b) plot the eigenvalues of the identified A matrices as well as their continuous-time equivalents (converted using zero-order hold). The eigenvalues of the continuous-time equivalent should be fairly constant, if a continuous-time model of the system exists. Inspecting Figure 2(b), it is clear that at low sample times different dynamics are being captured by the model than at high sample times. This is probably because at low sample times, the variability in the `CPU` metric is skewing the model parameters. However as sample times increase above 5 seconds, the eigenvalues seem to converge.

For the rest of this paper, a sample time of 5 seconds is used. From Figure 2, we know that the identified model is similar at sample times larger than 5 seconds. In addition, a sample time of 5 seconds is large enough to filter out the stochastics of the metrics while at the same time allowing for a decent speed of response. The parameters of the model are given in (2).



(a) Discrete-time (b) Continuous-time

Figure 2: Eigenvalues of discrete-time model (a) and its continuous-time equivalent (b) plotted verses sample time.

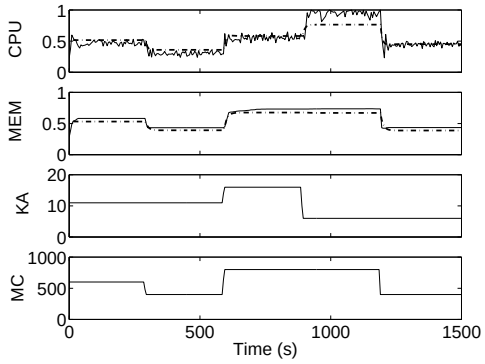


Figure 3: Results of multi-step prediction. In each plot, the solid line is the experimental data and the dashed line is the model prediction.

$$y_{k+1}=\begin{bmatrix} 0.537 & -0.109 \\ -0.0256 & 0.630 \end{bmatrix}\cdot y_k+\begin{bmatrix} -84.5 & 4.39 \\ -2.48 & 2.81 \end{bmatrix}\times 10^{-4}\cdot u_k \quad (2)$$

3.4 Model Evaluation

Two model evaluations are performed: one-step prediction using the same data set used to identify the model given in (2) and multi-step prediction on a independent data set. R^2 is an indicator of the amount of variability in the data that is captured by the model. This measure was used to quantify the quality of fit of the one-step predicted values. For a perfect model, the predicted values equal the measured values resulting in an R^2 value of 100%. The R^2 value for `CPU` is greater than 90%. The R^2 value for `MEM` is greater than 98%.

Figure 3 plots the response of both the real system and the multi-step prediction of the model given in (2) to a series of step changes in `KA` and `MC`. Overall, the model does a good job of predicting system response (especially for `MEM` where there is little variability). However, there are regions in which there is a degradation of accuracy due to limitations of the linear model. It is most accurate near the center of the operating region (`KA` = 11 and `MC` = 600) and less accurate near the edges or outside of this region.

4 Controller Design

The block diagram of the closed loop system is shown in Figure 4. The reference is comprised of the desired utilizations for `CPU` and `MEM`, denoted by `CPU`* and `MEM`*. In this approach, the job of the administrator is shifted from directly setting the tuning parameters to supplying the desired utilization values. However, determining feasible regions for the desired utilizations (i.e. the reference) is not a straightforward process, especially since the two utilizations are interrelated. Section 4.1 will explore how the DC gain of the MIMO model in (2) can be used to determine feasible regions for the reference.

The goal of the feedback control strategy is to track the desired utilizations, which implies responding to changes in these values in a reasonable amount of time. However unlike traditional control systems, a reasonable amount of time may be on the order of minutes.

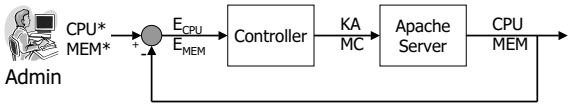


Figure 4: Block diagram of feedback system for control of CPU and memory utilizations.

An aggressive controller is not necessary to realize this type of response and is, in many ways, undesirable in computing systems. Aggressive controllers often utilize high gains which cause excessive reactions to stochastics, which act like noise in system outputs and cannot be controlled. Aggressive controllers also mandate drastic changes in control inputs. These drastic changes might lead to saturation, which can cause instability in the form of limit cycles as seen in [4]. For these reasons, our methodology focuses on the design of a low gain controller. Design of this controller will involve negotiating the trade-offs between speed of response and overreaction to noise in system outputs.

Although it is not necessary that the controller yield "fast" response, it is desired that the controller be robust. That is, it should able to handle changes in workload. This is necessary because workloads are often unknown; even when they are known, they change over time. Because the controller is designed using a model of the system identified at a specific workload, it is desired that the controller be robust to changes in that model (at a different workload, different model parameters may be identified).

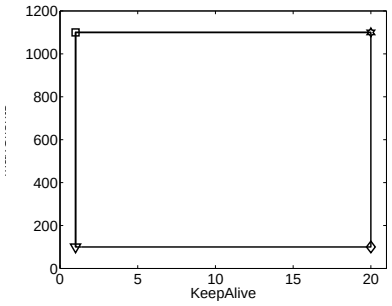
Because of its robustness and in an attempt to narrow down the space, a proportional integral (PI) controller is selected. The PI control control law is shown in (3). In this law, $u_k = [\texttt{KA}_k \;\; \texttt{MC}_k]^T$ and $e_k = r_k - y_k$ is the 2×1 vector of errors between the reference values, $r_k = [\texttt{CPU}^*_k \;\; \texttt{MEM}^*_k]^T$, and the system outputs $y_k = [\texttt{CPU}_k \;\; \texttt{MEM}_k]^T$. Note that this is a MIMO controller; K_P and K_I are both 2×2 matrices.

$$u_k \;\; = \;\; K_P \cdot e_k + K_I \cdot \sum_{j=1}^{k-1} e_j, \qquad (3)$$

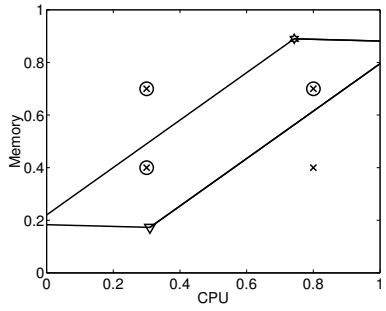
Initially, pole-placement was used to determine appropriate values for the matrix gains, K_P and K_I. However, the standard algorithm to solve for the gains using the desired closed loop pole locations and the open loop system model decouples the closed loop system. This implementation results in unnecessarily large control gains which results in a degradation of control performance. In order to overcome this shortcoming, LQR was used to design the gains. The LQR approach gives us greater authority to negotiate the trade-off between speed of response and overreaction to noise in system outputs.

4.1 Feasible Reference Values

A common problem in practice is determining the feasibility of references for interrelated metrics. In the case of the Apache server, there may exist combinations of `CPU` and `MEM` that cannot be achieved at a given workload, at least not using the control inputs `KA` and `MC`.



(a) Range of inputs



(b) Predicted range of feasible outputs

Figure 5: The parallelogram in (b) displays the regions into which the input range of KA and MC pictured in (a) are mapped by the model.

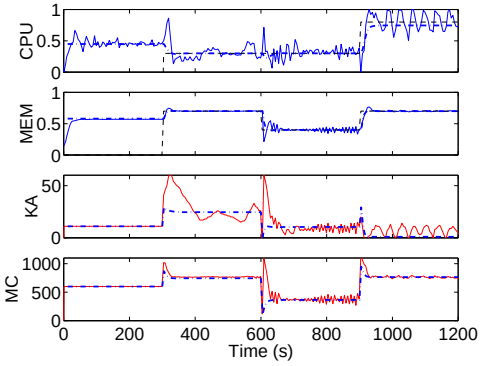
Using (2), we can predict feasible reference values of the system outputs based on the ranges of the control inputs. This is illustrated in Figure 5. The range of KA and MC considered (from the data set plotted in Figure 1) is pictured in (a). In (b), the solid parallelogram is the feasible region predicted by the model in (2) using the range in (a).

In (b), the bold x's represent candidate (CPU, MEM) references for the Apache system. Inverting the DC gain of the model, we can determine the inputs needed to realize these reference values. For (CPU = 0.3, MEM = 0.7), we determine that the inputs should be (KA = 30, MC = 800). That is, the model predicts that this point cannot be achieved within the range of inputs considered in (a). Our experimental results confirm this, although they also show that (CPU = 0.3, MEM = 0.7) can be realized if larger values of KA are used. For (CPU = 0.8, MEM = 0.4), the model determines that (KA = −10, MC = 450). This cannot be attained since KA cannot be negative. Our experimental results confirm that (CPU= 0.8, MEM= 0.4) is not a feasible combination of reference values. The three circled x's are all feasible and are used as references in our experiments.

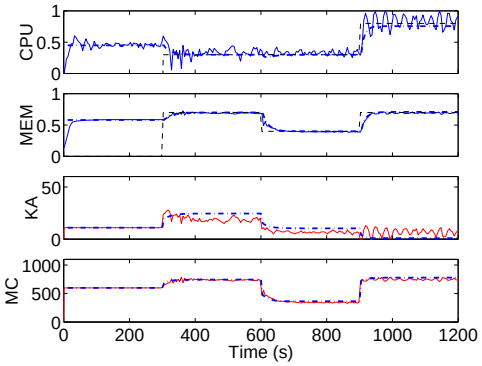
4.2 Controller Design using Pole Placement

The starting point when designing a controller via pole placement is usually desired transient specifications such as maximum overshoot and settling time. As mentioned earlier, precise specifications for the transient response of the closed loop system do not exist. Instead, a controller with low gain is desired.

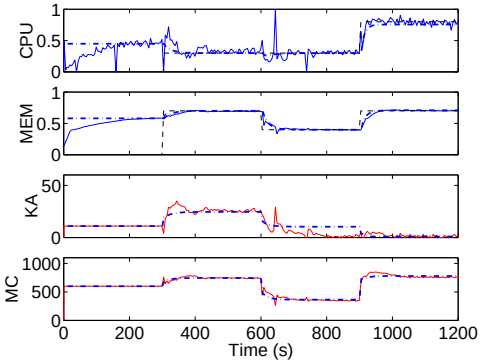
In an attempt to design a low gain controller, we specify the transient specifications of our closed loop system based upon the transient response (settling time) of our open loop system. By doing this, we hope to use minimal control effort because the controller is not attempting to move the poles of the open loop system. However because we are using a PI control law, two additional poles are introduced by the controller. Hence an overshoot specification was added to fully constrain the four desired closed-loop poles given in Table 1. The standard algorithm as implemented in MATLAB was used to calculate a set of gains that would yield the desired closed loop response. These gains result in a decoupled closed loop system.



(a) Pole Placement



(b) LQR, heavy workload



(c) LQR, light workload

Figure 6: Performance of the two controllers. The solid lines are experimental data, the dash-dotted lines show the prediction of the model, and the dashed lines indicate the reference values for CPU and MEM utilizations.

The control performance is shown in Figure 6(a). The large control gains result in excessive control reaction to the stochastics of the system and thus large changes in the control inputs KA and MC. Moreover, since the

closed loop poles have imaginary parts, the closed loop response is oscillatory.

4.3 Controller Design using LQR

Because the transient response of the closed loop system is not the key design consideration, the pole-placement approach is not really the best method to use for this system. LQR allows us to better negotiate the trade-offs between speed of response and over-reaction to noise in system outputs. LQR finds gains that minimize the quadratic cost function shown in (4), where e_k is defined as in (3) and v_k is the state added by the integrator. By selecting appropriate weighting for Q and R, we can ensure that the control inputs (tuning parameters) do not get too large, and thus, in effect, design a low gain controller.

$$J = \sum_{k=1}^{\infty} [e_k \; v_k]^T \cdot Q \cdot \begin{bmatrix} e_k \\ v_k \end{bmatrix} + {u_k}^T \cdot R \cdot u_k \qquad (4)$$

The control design problem has now shifted to choosing the weighting matrices Q and R. The ranges for `CPU` and `MEM` are both [0,1], for `KA` [1,50], and for `MC` [1,1024]. Thus, we choose $R = diag(1/50^2, 1/1000^2)$ to scale the inputs to be on the same order of magnitude as the control errors. Then, we choose $Q = diag(1, 1, 0.1, 0.2)$ to weight the control errors more heavily than the accumulated control errors. The resulting closed loop poles are given in Table 1.

Using this method, we have designed a less aggressive controller with smaller gains. Moreover, since this closed loop pole has no imaginary part, the closed loop system should be less oscillatory. The control performance is shown in Figure 6(b). It is clear that both control inputs have been reduced, and `CPU` and `MEM` are less oscillatory than before. At the same time, the controller is still fast enough to track the desired utilizations.

Table 1: Closed Loop Pole Locations

Controller	Poles
Pole Placement	$\mathbf{0.66 \pm 0.25}\,i,\; 0.51 \pm 0.16\,i$
LQR	$\mathbf{0.81},\; 0.72,\; 0.68,\; 0.36$

4.4 Controller Robustness

The experimental results presented thus far used only a single workload. In practice, however, the workload is unknown *a priori* and changes over time. To determine how well our feedback control design performs in the presence of these unknowns, we ran an experiment with a different workload. In the original workload, the session arrival rate was 0.05; we change this to 0.5, creating a lighter workload. We did not rebuild the system model or redesign the controller; the LQR controller of Section 4.3 is used. The experimental results, shown in Figure 6(c), indicate that even though the model prediction is not very accurate, the controller still performs well.

5 Conclusions

The widespread use of information technology has motivated the need for performance management of computing systems. To this end, system administrators attempt to translate desired performance into appropriate settings of available tuning parameters. We propose the use of a feedback control strategy to achieve this goal. This strategy requires selection of control inputs and system outputs, creation of a system model, and evaluation of various control options. The issues mentioned above are addressed in the context of the Apache web server in the hopes of providing a general framework for the control of computing systems. Creation of a MIMO model captures the interactions between `CPU` and `MEM` and thereby provides an accurate model of the real system. Having this model is of particular benefit in determining feasible reference values. It is shown that controllers designed using the model work well when designed properly and are robust to changes in workload.

Most computing systems have many tuning parameters (dozens or more) that all interact to affect the metrics of the system. This work is the first step towards showing how MIMO techniques can be used. It would be particularly interesting to see how well MIMO techniques work on systems that have more tuning parameters than metrics, which is typical in computing systems. In addition, we would like to understand the limits of the models we employ when dealing with notoriously nonlinear metrics such as response times. We believe that linear models will be effective if the models are applied within appropriate operating regions (e.g., workloads), and the models are adapted appropriately as the operating region changes. Finally, the selection of sample time remains an ongoing research issue.

References

[1] Apache Software Fndn. http://www.apache.org.

[2] Y. Diao, N. Gandhi, S. Parekh, J. Hellerstein, and D. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server. In *Proc. of the Network Operations and Management Symposium*, 2002. To Appear.

[3] R. Fielding and et. al. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.* IETF, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[4] N. Gandhi, S. Parekh, D. Tilbury, and J. Hellerstein. Feedback control of a Lotus Notes server: Modeling and control design. In *Proc. of the American Control Conference*, 2001.

[5] Z. Liu, N. Niclausse, C. Jalpa-Villanueva, and S. Barbier. Traffic model and performance evaluation of web servers. Technical Report 3840, INRIA, December 1999.

[6] L. Ljung. *System Identification: Theory for the User.* Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1999.

[7] Mindcraft, Inc. Webstone 2.5 web server benchmark, 1998. http://www.mindcraft.com/webstone/.