RC22216 (W0110-057) October 22, 2001
Computer Science

IBM Research Report

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

Y. Diao, J. Hellerstein, and S. Parekh
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 708
Yorktown Heights, NY 10598

N. Gandhi and D. M. Tilbury
The University of Michigan
Mechanical Engineering Department
Ann Arbor, MI 48109-2125

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Paper #: ACC02-INV2703

MIMO Control of an Apache Web Server: Modeling and Controller Design*

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

Submitted to the 2002 American Control Conference
September 21, 2001

1 Introduction

As computing systems become more widely deployed and more heavily used, there is increasing demand for performance improvement. Software developers may not know in advance the types of workloads that their systems will face, or the variety of hardware platforms that they will run on. Instead of attempting to optimize the software systems for one particular situation, they often expose many tuning parameters which can be set by the system administrator. The administrator thus has the ability to set these parameters to optimize the system's performance in accordance with some high-level goals. These high-level goals are often application specific and are designed to satisfy various business needs.

Choosing the correct settings for tuning parameters is not a straightforward job. The best settings will depend on the hardware on which the system is running, the workloads that the system experiences, as well as what other jobs are running concurrently on the system. Since the workloads and concurrent jobs can change over time, a dynamic feedback control strategy is desirable.

In order to employ a feedback control 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; some effort may be required to allow the parameter to be dynamically changeable (as opposed to only set when the system is booted). An output must be chosen that has meaning for the system and can be reasonably measured. For example, the response time seen by the user may be a desirable output but not one that is readily measured. First-principles models (such as those based on Newton's laws) may be difficult to contruct for most computing systems; a "black-box" approach is typically called for. Finally, if multiple inputs and multiple outputs are chosen for the model, the appropriate MIMO control strategy must be chosen.

*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 traditional 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 in the case of an Apache web server. The remainder of the paper is organized as follows. 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 Selection of System Outputs and Control Inputs

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. In Section 2.1, some background will be given on the structure of Apache so that readers will have some intuition as to how the chosen control inputs work. In Sections 2.2 and 2.3, the process used to select the system outputs and control inputs used in this paper is outlined.

2.1 Apache Architecture

Apache v1.3 on Unix [1] is structured as a pool of worker processes monitored by a master process. The master process manages the creation, health, and destruction of the worker processes. Each worker process is responsible for handling communication with the web clients and can handle at most one connection at a time. The worker process can be in three states: "Idle", "User Think", and "Busy". In the "Idle" state, the worker is waiting for a client connection. Once a connection is accepted, the worker enters the "User Think" state, where it waits for an HTTP request. Upon receipt of the request, the worker enters the "Busy" state, where it remains until the reply is sent. The time between sending an HTTP reply and the receipt of the next client request is known as the *user think time*. In HTTP 1.1 [2], 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 persistent connections, either side may potentially close the connection.

2.2 Selection of System Outputs

A number of metrics are used to quantify performance of the Apache web server and make 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 a feedback control strategy.

One high-level goal of control may be to ensure some bound on end-user response times. For example, service level agreements (SLAs) between internet service providers (ISPs) and their customers often involve an end-user response time specification. However, because this is a client-side metric which incorporates network delays, it can not be measured by the server. As a result, it cannot be used in a control strategy that is implemented on the server-side. One way to overcome this problem is to add instrumentation in the form of probes that supply an approximation of end-user response times. However this approach results in additional load on the server and is not always accurate. In addition, using client-side metrics introduces delays since information needs to be transferred between two systems (the client and the server). 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.

2.3 Selection of Control Inputs

There are a number of tuning parameters that are exposed to optimize the performance of the Apache web server that can be used as control inputs. Some of these tuning parameters include: "MaxClients", "KeepAliveTimeout", "MaxKeepAliveRequests", "MaxRequestsPerChild", and "HostNameLookups". There are two considerations when selecting the appropriate tuning parameters to use in a feedback control strategy. First, the parameters must be dynamically changeable. That is, in order to change the parameters the system should not have to be rebooted. In the case of Apache, none of the available tuning parameters are dynamically changeable. Hence, parameters that require minimal changes to the Apache source code are desired. Second, the tuning parameters must affect the selected performance metrics in a meaningful way. The goal is to create a system model that relates how the tuning parameters affect the performance metrics. If the selected tuning parameters have little impact on the selected performance metrics, then the task of creating a system model will become quite cumbersome and the efficacy of the control strategy will be limited.

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

The "KeepAliveTimeout" parameter (`KeepAlive`, abbreviated by `KA`) controls the maximum time a worker can remain in the "User Think" state before its client connection is closed. If `KeepAlive` is too large, `CPU` and `MEM` are underutilized since clients with requests to process cannot connect to the server. Reducing `KeepAlive` means that workers spend less time in the "User Think" state, and more time in the "Busy" state (if connection overheads are modest compared with the time for processing requests). Hence, `CPU` increases. Although `KeepAlive` indirectly affects memory by allowing more clients to eventually connect to the server,

increasing KeepAlive does not have a direct effect on MEM like it does on CPU. If KeepAlive is too small, the TCP connection terminates prematurely and reduces the benefits of having the persistent connections.

As mentioned above, in the default version of Apache, MaxClients and KeepAlive 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 [3]. For the remainder of the paper, the tuning parameters KeepAlive and MaxClients 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 easy 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. Looking at experimental data from the Apache server, it is clear that a relationship exists between the control inputs, the tuning parameters KeepAlive and MaxClients, and the system outputs, the utilization values CPU and MEM. However, it is not clear how to derive a functional relationship between KeepAlive, MaxClients and CPU, MEM.

One approach is to start from first-principles and create a queueing model of the Apache server. However, this approach will quickly become very complicated and would require detailed knowledge about the inner workings of the server. In addition, the first-principles approach does not translate well once the application changes. So instead of proceeding from first-principles, an empirical approach is used to quantify the relationship between the control inputs (KeepAlive and MaxClients) and the system outputs (CPU and MEM). This approach involves four main steps: (1) determining the appropriate experiments to run (i.e., designing appropriate input signals); (2) collecting the data from the server; (3) using system identification techniques to construct statistical models from the data; and (4) validating the models.

The following sections focus on the steps listed above. Section 3.1 describes the experimental environment used to obtain the data from which the models are constructed. Section 3.2 outlines how input signals were chosen. Section 3.3 details the methodology used to construct the models, and Section 3.4 evaluates these models.

3.1 Experimental Environment

Our testbed consists of one server running Apache connected through a 100 Mbps LAN to one or more clients running a synthetic workload generator. All the machines are Intel-based with a Linux operating system. The server is a Pentium III 800 MHz with 256 MB RAM. Each client machine runs 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. [4] that has been validated in extensive studies of production web servers. WAGON structures the workload into multiple sessions (which represent a series of user interactions) during which there are multiple clicks (end-user requests), each of which generates a burst of HTTP requests (which represents web pages with multiple objects). Table 1 summarizes the parameters used based on the data reported in [4]. The file access distributions we use are from the Webstone 2.5 reference benchmark [5].

For our experiments, we have increased the default process limit on the server to 1024. For the heavier workload, the server CPU can be 100% utilized with a few hundred processes. Hence, the control code ensures that MaxClients is always an integer value in the range [1; 1024]. For KeepAlive, no maximum value is

Table 1: Workload Parameters

Parameter Name	*Distribution*	*Parameters*
Session Length (# clicks)	LogNormal	$\mu = 8$, $\sigma = 3$
Burst Length (# URLs)	Gaussian	$\mu = 5$, $\sigma = 3$
User Think Time (s)	LogNormal	$\mu = 30$, $\sigma = 30$
Session Inter-arrival time (s)	Exponential	$\mu \in \{0.05(\text{heavy}) \ , 0.5(\text{light}) \ \}$

enforced, but the timeout values are in integral seconds, with a minimum of 1 second. With the nature of the workload, `KeepAlive` values above 50 do not significantly affect server utilizations, and `KeepAlive` values larger than 20 have only a small effect (depending on the current value of `MaxClients`). While it is feasible to have fractional values for `KeepAlive`, the default implementation of Apache uses integral values, and we have not changed this.

3.2 Design of Input Signals

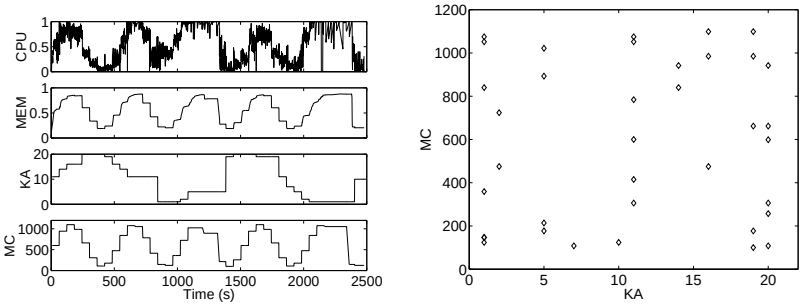
The main task when deciding which experiments to run whose data will be used to construct a model of the system is designing a sufficiently rich input signal. The control inputs must be varied in a manner so that two properties are satisfied. First, the input signal should be persistently exciting; it 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 (`KeepAlive` and `MaxClients`). Discrete sine waves are used for both `MaxClients` and `KeepAlive`. This is done so that there are both high frequency components (in the form of the steps) and low frequency components (the frequency of the sine wave) that are sufficient to identify A and B. The `MaxClients` sine wave has a period of 500 seconds, a mean of 600, and an amplitude of 500; values of `MaxClients` greater than 1024 are saturated to this value by the control implementation as noted in Section 3.1. The `KeepAlive` sine wave has a period of 1200 seconds, a mean of 11, and an amplitude of 10. 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(a) plots the data from the server run using the discrete sine wave input signals as the control inputs and the heavy workload as parameterized by Table 1. Figure 1(b) shows the coverage that the two discrete sine wave signals provide of the operating region when used concurrently.

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. A linear model does not even attempt to capture the stochastic or nonlinear nature of this system. So why did we choose it? 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



(a) (b)

Figure 1: (a) Experimental data with discrete sine wave inputs. (b) Coverage of operating region when discrete sine waves are used concurrently as in (a).

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.

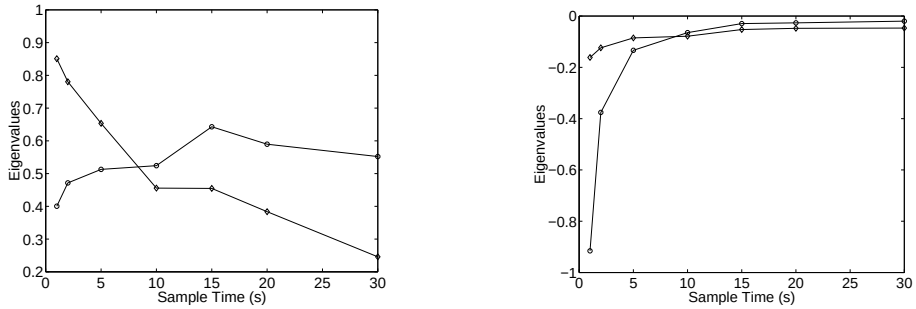
The form of the linear model is shown in Equation (1), with parameters A and B estimated using least squares regression. Note that this is a MIMO model; A and B are both 2×2 matrices. A first-order model was chosen for simplicity and because increasing the order of the model did not significantly increase 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, $y_k[u_k]$ in (1) denotes the average value of $y[u]$ over time interval k.

$$\begin{aligned} y_k &= \begin{bmatrix} \texttt{CPU}_k \\ \texttt{MEM}_k \end{bmatrix} \\ u_k &= \begin{bmatrix} \texttt{KA}_k \\ \texttt{MC}_k \end{bmatrix} \\ y_{k+1} &= A \cdot y_k + B \cdot u_k \end{aligned} \tag{1}$$

3.3.1 Selection of Sample Time

The choice of averaging interval (sample time) is a key factor that affects the performance of the controller. However, because the system in question is inherently discrete and stochastic in nature and not a sampled continuous process, it is not clear what measure should be used to determine how often sampling should occur. 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. Sample times were chosen so that they divide the amount of time each step in the discrete sine waves is held for (60 seconds). This prevents an interval from sharing two different values of a control input and hence ensures that the effect of changing control inputs is clearly captured.



(a) Eigenvalues of discrete-time model. (b) Eigenvalues of continuous-time equivalent.

Figure 2: Models of the form in (1) were identified using seven different sample times (averaging intervals). At each sample time, the eigenvalues of the A matrix were calculated both for the identified discrete-time model and its continuous-time equivalent (converted using zero-order hold). The resulting collection of eigenvalues are plotted verses sample time.

Figure 2(a) and (b) plot the eigenvalues of the A matrix of the identified discrete-time models as well as their continuous-time equivalents (converted using zero-order hold). It is expected that the eigenvalues of the discrete-time model will change as sample time changes ($z = e^{T \cdot s}$). On the other hand, the eigenvalues of the continuous-time equivalent should be fairly constant, regardless of sample time, 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.

Figure 3 plots the frequency response of the identified discrete-time models. Because a full first-order model was identified (no assumption was made about the form of the A or B matrices), four transfer functions exist: (1) `KA` $\rightarrow$ `CPU`, (2) `KA` $\rightarrow$ `MEM`, (3) `MC` $\rightarrow$ `CPU`, and (4) `MC` $\rightarrow$ `MEM`. The number of each transfer function corresponds to the number of the subplot in Figure 3 in which the frequency response of the transfer function is plotted. Why is the gain of all these transfer functions so small? The control inputs were chosen because they impacted the system outputs in a meaningful way. The answer lies in the ranges of the system outputs and control inputs.

The range of both `CPU` and `MEM` is [0,1]. The range of `KeepAlive` is [1,20] and the range of `MaxClients` is [1,1024]. Thus, the transfer functions from `KA` to `CPU` and from `KA` to `MEM` include a scaling factor on the order of 10^{-1}. Similarly, the transfer functions from `MC` to `CPU` and from `MC` to `MEM` include a scaling factor on the order of 10^{-3}. From this analysis, it is expected that at low frequencies the response in subplots (1) and (2) should lie around -20 dB ($20 \cdot log_{10}(10^{-1})$). Similarly, the response at low frequencies in subplots (3) and (4) should lie between -60 dB ($20 \cdot log_{10}(10^{-3})$). Subplots (1),(3), and (4) confirm this prediction. Subplot (2) does not; at low frequencies, the response lies around -60 dB. As mentioned in 2.3, `KeepAlive` does not directly affect `MEM`. Hence, the model should identify a zero transfer function from `KA` to `MEM`. This explains why the response in subplot (2) is almost 30 dB lower than the response in subplot (1) at low frequencies.

It is clear that all the models have similar frequency responses over the low frequency range. Thus, even though the dynamics captured by the model are different at lower sample times, the steady state gain of the system is the same. At higher frequencies, the response of the identified models starts to diverge as expected. This is because as sample time decreases, the bandwidth of the system increases.

For the rest of this paper, a sample time of 5 seconds is used. From Figure 2, we know that the identified

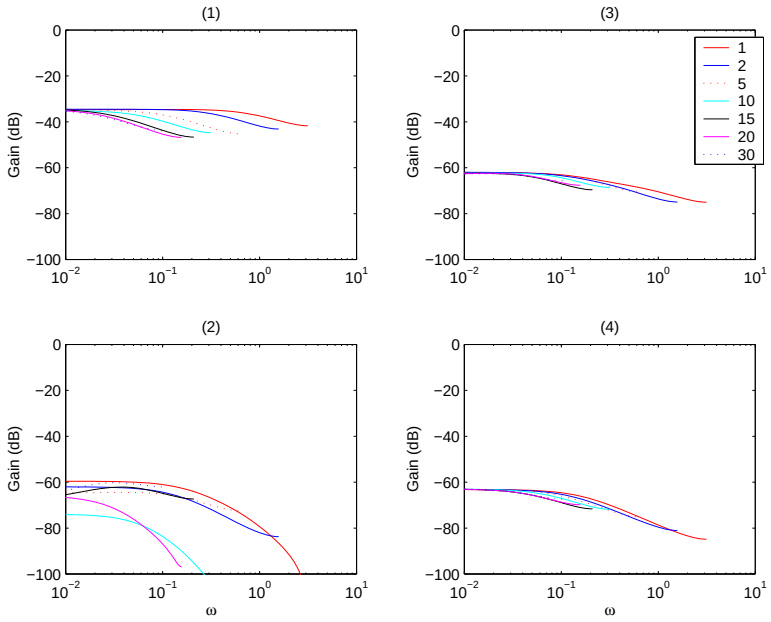
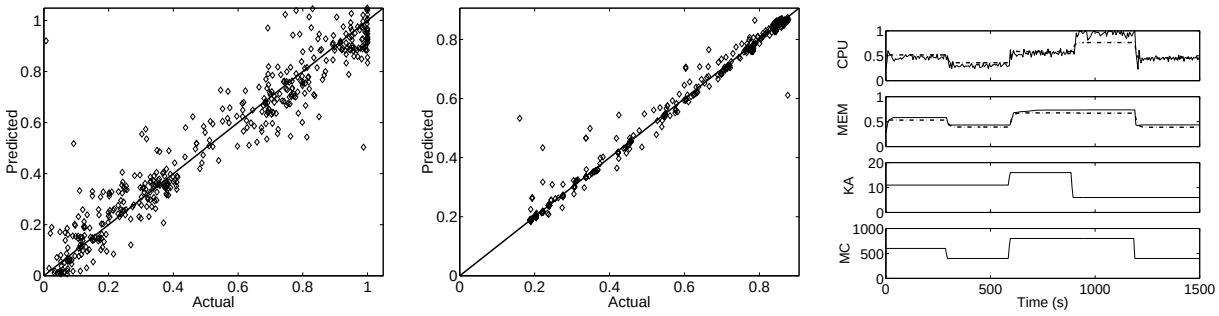


Figure 3: The frequency response of the discrete-time models at various sample times. The four subplots contain the Bode magnitude plots (in dB) of the four transfer functions contained in our system. In the first row, `CPU` is the output; in the second row, `MEM` is the output. In the first column, `KA` is the input; in the second column, `MC` is the input. In each subplot, the seven lines represent the response of the seven models (each identified at a separate sample time).



(a) One-step: CPU, $R^2 = 91.5$ (b) One-step: MEM, $R^2 = 98.9$ (c) Multi-step

Figure 4: (a),(b) Results of one-step prediction. In each plot, the x-axis is the actual value and the y-axis is the predicted value. The line indicates when the actual equals the predicted value, which occurs when the model is perfect. (c) Results of multi-step prediction. In each plot, the solid line is the experimental data and the dashed line is the model prediction.

model is similar at sample times larger than 5 seconds. Hence, we chose 5 seconds as the smallest sample time with this similarity property. 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).

$$\begin{bmatrix} \texttt{CPU}_{k+1} \\ \texttt{MEM}_{k+1} \end{bmatrix} = \begin{bmatrix} 0.537 & -0.109 \\ -0.0256 & 0.630 \end{bmatrix} \cdot \begin{bmatrix} \texttt{CPU}_k \\ \texttt{MEM}_k \end{bmatrix} + \begin{bmatrix} -84.5 & 4.39 \\ -2.48 & 2.81 \end{bmatrix} \times 10^{-4} \cdot \begin{bmatrix} \texttt{KA}_k \\ \texttt{MC}_k \end{bmatrix} \tag{2}$$

3.4 Model Evaluation

Two model evaluations are done. The first is one-step prediction in which the value of a metric at time $k+1$ is predicted based on the *measured* values at time k. The second is multi-step prediction in which the value at $k+1$ is predicted based on the *predictions* at time k.

Figure 4(a) and (b) plot the quality of fit for the identified model given in (2) using one-step prediction on the same data set used to identify the model. For a perfect model, the predicted values equal the measured values and thus lie on the line of unit slope. The R^2 value (a measure of the amount of variability in the data captured by the model) for CPU is greater than 90%. The R^2 value for MEM is greater than 98%.

Figure 4(c) 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 KeepAlive and MaxClients. 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 the accuracy of the model degrades, especially when KA = 6 and MC = 800. This occurs 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 5. The reference is comprised of the desired utilizations for `CPU` and `MEM`, denoted by `CPU`* and `MEM`*. The controller acts on the error between the desired and measured utilizations, denoted by $E_{\texttt{CPU}}$ and $E_{\texttt{MEM}}$, to determine the next values of the control inputs, `KeepAlive` (`KA`) and `MaxClients`(`MC`). 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 (they both depend on `MaxClients`). The model of the system created in Section 3.3 can be of great assistance in this process. Section 4.1 will explore how the DC gain of this model can be used to determine feasible regions for the reference.

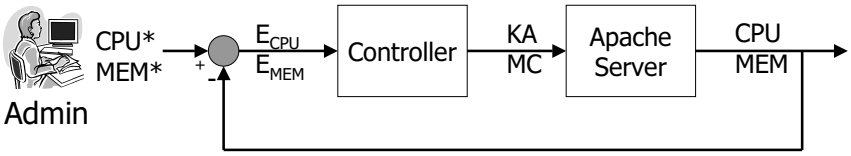


Figure 5: Block Diagram of Feedback System for Control of CPU and Memory Utilizations

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. An aggressive controller is not necessary to realize this type of response.

In fact, in many ways, aggressive controllers are undesirable in computing systems. Aggressive controllers often utilize high gains which cause excessive reactions to stochastics present in computing systems. Stochastics act like noise in system outputs and cannot be controlled. In addition, aggressive controllers mandate drastic changes in control inputs, which is also undesirable in computing systems. Often, these drastic changes will lead to saturation which can cause instability in the form of limit cycles as seen in [7]. 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; and 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 would 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 is the 2×1 vector of control inputs given in (1) 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 as given in (1). Note that this is a MIMO controller; K_P and K_I are both 2×2 matrices

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

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. Sections 4.2 and 4.3 will go into greater detail about these two design approaches. Section 4.4 will explore the robustness of our best LQR controller.

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 least not using the control inputs `KeepAlive` and `MaxClients`. Thus, it is desirable for administrators to know whether or not a reference is feasible using the available controls.

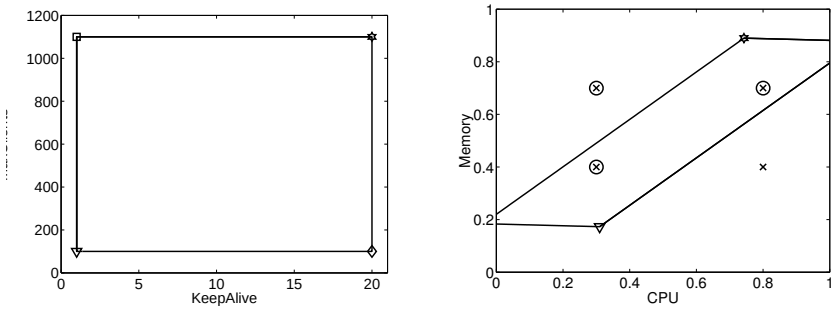
Using (2), we can predict feasible reference values of the system outputs, `CPU` and `MEM` based on the ranges of the control inputs, `KeepAlive` and `MaxClients`. This is illustrated in Figure 6. Part (a) displays the range of `KeepAlive` and `MaxClients` used. This range was created using the range of the control inputs in the data set plotted in Figure 1. In part (b), the bold x's represent candidate (`CPU`, `MEM`) references for the Apache system. The solid parallelogram is the feasible region predicted by the model in (2). Only two of the x's are predicted as being feasible by the model for the inputs in Part (a).

We now explore the unfeasible references (as predicted by the model) in more detail. 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 Part (a). Our experimental results confirm this, although they also show that (`CPU`= 0.3, `MEM`= 0.7) can be realized if larger values of `KeepAlive` are used. For (`CPU`= 0.8, `MEM`= 0.4), the model determines that (`KA`= -10, `MC`= 450). This cannot be attained since `KeepAlive` 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 in Sections 4.2 and 4.3.

4.2 Controller Design using Pole Placement

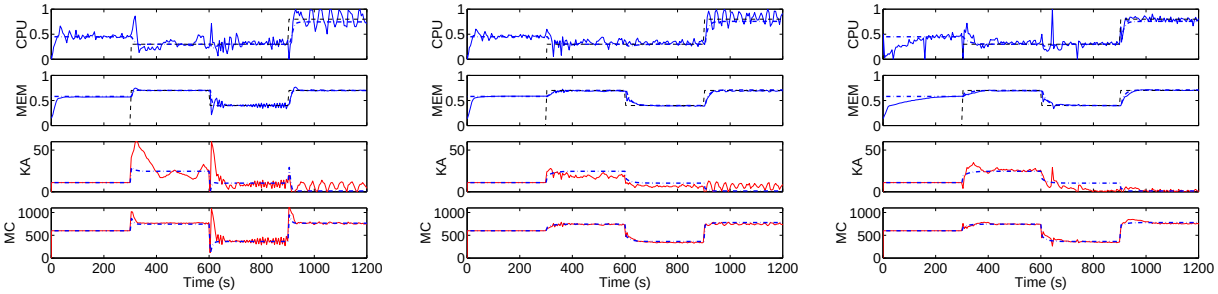
The starting point when designing a controller via pole placement is usually the desired closed loop pole locations. These pole locations are typically derived from desired transient specifications such as maximum overshoot and settling time. As mentioned earlier, precise specifications for the transient 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 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. Our experimental measurements suggest that the settling time is 60 seconds for `CPU` and 40 seconds for `MEM`. However because we are using a PI control law, two additional poles are introduced by the controller. The settling time specifications only consider the poles of the open loop system, and hence another set of specifications is needed in order to fully constrain the four desired closed-loop poles. So the maximum overshoot is chosen to be 5% for `CPU` and 0.2% for `MEM`. Using these transient specifications, the standard



(a) Range of inputs (b) Predicted range of feasible outputs

Figure 6: Determining feasible regions for reference values. Part (a) shows the range of `KeepAlive` and `MaxClients` considered, as indicated by the markers on the corners of the rectangle. Part (b) displays the regions into which the inputs are mapped by the model. The model prediction is enclosed by the parallelogram. The markers on the parallelogram indicate their correspondance to input values. Candidate reference are denoted by an "x". Circled x's are feasible.



(a) Pole Placement (b) LQR, heavy workload (c) LQR, light workload

Figure 7: 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.

algorithm as implemented in MATLAB was used to calculate a set of gains that would yield the desired closed loop response. These gains are shown in Table 2 and result in a decoupled closed loop system.

The control performance is shown in Figure 7(a). The large control gains result in excessive control reaction to the stochastics of the system and thus large changes in the control inputs `KeepAlive` and `MaxClients`. 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. As mentioned before, LQR allows us to negotiate the trade-offs between speed of response and overreaction 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.

Table 2: Control Gains and Resulting Closed Loop Poles

Controller	K_P	K_I	Closed Loop Pole Locations	Settling Time (s)
Pole Placement	$\begin{bmatrix} -31 & 114 \\ -106 & 2121 \end{bmatrix}$	$\begin{bmatrix} -22 & 44 \\ -14 & 921 \end{bmatrix}$	$\mathbf{0.66} \pm \mathbf{0.25}\,i,\ 0.51 \pm 0.16\,i$	57
LQR	$\begin{bmatrix} -32 & 31 \\ 193 & 890 \end{bmatrix}$	$\begin{bmatrix} -12 & 9 \\ 75 & 335 \end{bmatrix}$	$\mathbf{0.81},\ 0.72,\ 0.68,\ 0.36$	91

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

The control design problem has now shifted from determining the desired closed-loop poles to choosing the weighting matrices Q and R. Since `CPU` and `MEM` are in the range from 0 to 1, the valid region for `KeepAlive` is from 1 to 50, and `MaxClients` can vary from 1 to 1024, 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.

Using this method, we have designed a less aggressive controller with smaller gains; K_P and K_I are given in Table 2. The transient performance of the closed loop system can be inferred from the closed loop pole locations [8]. The dominant closed loop pole is located at 0.81, which predicts a settling time of 91 seconds. 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 7(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.

4.4 Controller Robustness to Changes in Workload

The experimental results presented thus far used only a single workload – the heavy workload described in Section 3.1. 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 heavy workload, the session arrival rate was 0.05; we change this to 0.5. 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 7(c), indicate that even though the model prediction is not very accurate, the controller still performs well.

5 Conclusions

The wide-spread use of information technology has motivated the need for performance management of computing systems. Frequently, software developers expose a number of tuning parameters to aid in the optimization of system performance. The challenge then becomes to devise a method for translating the desired performance into appropriate settings of the available tuning parameters. The best settings will depend on the hardware on which the system is running, the workloads that the system experiences, as well

as what other jobs are running concurrently on the system. Since the workloads and concurrent jobs can change over time, a dynamic feedback control strategy is desirable.

The application of traditional control strategies to computing systems encounters a number of obstacles. Control inputs and system outputs must be selected, 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 traditional electromechanical systems, the appropriate analogs for computing systems are still being defined. In addition, workloads in computing systems are highly stochastic, and even with well-defined workloads, the systems themselves exhibit significant stochastic behavior.

This paper describes the use of MIMO techniques to track desired CPU and memory utilizations of an Apache web server. The issues mentioned above are addressed in the context of this application 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. Future work will involve applying the same approach to more general examples with more tuning parameters and metrics. 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 non-linear 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 Foundation. http://www.apache.org.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force (IETF), June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[3] 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 *Proceedings of the Network Operations and Management Symposium*, 2002. Submitted, August 2001.

[4] Z. Liu, N. Niclausse, C. Jalpa-Villanueva, and S. Barbier, "Traffic model and performance evaluation of web servers," Tech. Rep. 3840, INRIA, Dec. 1999.

[5] I. Mindcraft, "Webstone 2.5 web server benchmark," 1998. http://www.mindcraft.com/webstone/.

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

[7] N. Gandhi, S. Parekh, D. Tilbury, and J. Hellerstein, "Feedback control of a Lotus Notes server: Modeling and control design," in *Proceedings of the American Control Connference*, 2001.

[8] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Reading, Massachusetts: Addison-Wesley, third ed., 1998.