Dynamic Black-Box Performance Model Estimation for Self-Tuning Regulators

Magnus Karlsson and Michele Covell
HP Labs
Palo Alto, CA 94304, U.S.A.
{magnus.karlsson,michele.covell}@hp.com

Abstract

Methods for automatically managing the performance of computing services must estimate a performance model of that service. This paper explores properties that are necessary for performance model estimation of black-box computer systems when used together with adaptive feedback loops. It shows that the standard method of least-squares estimation often gives rise to models that make the control loop perform the opposite action of what is desired. This produces large oscillations and bad tracking performance. The paper evaluates what combination of input and output data provides models with the best properties for the control loop. Plus, it proposes three extensions to the controller that makes it perform well, even when the model estimated would have degraded performance.

Our proposed techniques are evaluated with an adaptive controller that provides latency targets for workloads on black-box computer services under a variety of conditions. The techniques are evaluated on two systems: a three-tier e-commerce site and a web server. Experimental results show that our best estimation approach improves the ability of the controller to meet the latency goals significantly. Previously oscillating workload latencies are with our techniques smooth around the latency targets.

1 Introduction

The increasing costs associated with managing computer systems have spurred a lot of interest in automatically managing system with little or no human intervention. Examples of this includes, managing the energy consumption of servers [4, 14], automatically maximizing the utility of data centers [22], and meeting performance goals in file systems [15, 16], 3-tier e-commerce sites [13, 16], disk arrays [3, 18, 23], databases [19] and web servers [1, 20].

For a solution to a specific management problem to be applicable to as many systems as possible it should be *non-intrusive*. The reason for this is that most computing systems have no native support for automatic management, and in the general case, cannot be modified easily to do so due to proprietary sources and/or the complexity of the modifications. We refer to such a system as a *black-box* system. It has a number of adjustable *actuators* (e.g., per workload CPU share or throughput per workload) that will change a number of *measurements* (e.g., latency, availability and throughput). Any management solution for a black-box system has to be able to discover a relationship between the actuators and the measurements, and then be able to set the actuators so the the management goals are achieved.

One technique that has successfully solved management problems of black-box systems is control-theoretic feedback loops [8, 9]. These methods can deal with poor knowledge of the system, changes in the system or the workloads, and other disturbances. Classical non-adaptive control-theoretic methods are usually not adequate for two reasons. First, for many systems it is not even possible to use non-adaptive control as the system changes too much [13, 15, 16]. For example, the performance experience by a client of a three-tier e-commerce site depends on many things: what tier a request is served from, if it was served from the disk or the memory cache of that tier, what other clients are in the system, and so on. Second, to be applicable to more than one specific system configuration, it is unreasonable to require a lot of tuning for each system change. Thus, we will focus on *adaptive controllers* that automatically tune themselves while the system is running.

Self-tuning regulators (STR) [2] are one of the most commonly used and well studied adaptive controllers. They consist of two parts: an *estimator* and a *control law*, which are usually invoked at every sample period. The most commonly used estimator in STR is *recursive least-squares (RLS)* [10]. The purpose of this estimator is to dynamically estimate a model of the system relating the measured metrics with the actuation. The control law will then, based on this model, set the actuators such that the desired perfor-

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**

IEEE COMPUTER SOCIETY

mance is achieved. The ability of the controller to achieve the performance goals is explicitly tied to how well the model represents the system at that instant.

This paper shows that standard RLS does not provide good control performance for STRs used in black-box systems, and proposes and evaluates a number of extensions of RLS that provide good control performance for STRs. More specifically, we evaluate what measurement and actuation combinations provides the best models, and we propose three methods of dealing with poor models such that they do not decrease controller performance. These three methods are: (1) to remember a good model and use that one when a poor model is produced, (2) to modify the poor model such that it becomes good, and (3) to run multiple estimators and pick the best model out of all of them.

We have evaluated the proposed extensions using an adaptive controller that provides latency targets for workloads on black-box computer services under a variety of conditions [16]. The evaluation is performed both analytically and experimentally on two real systems: a three-tiered e-commerce site and a web server. The results show that our best proposed method offers superior control performance, compared to the baseline of standard RLS without any of our techniques. This best method uses an affine function of throughput, and it modifies the models it detects are poor.

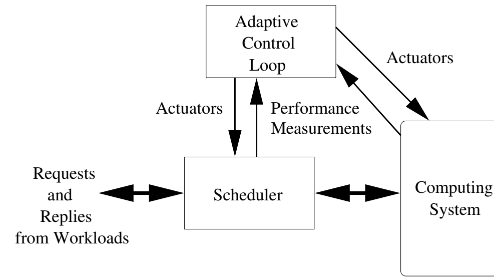

Figure 1. General architecture of the black-box systems studied in this paper.

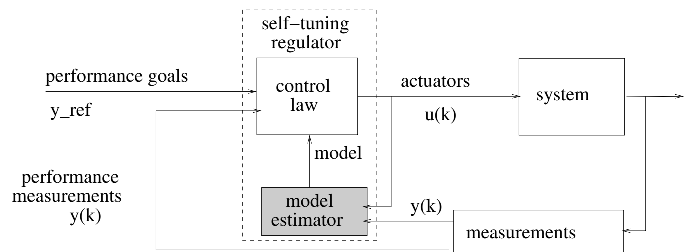

Figure 2. The self-tuning regulator studied.

2 Background and Problem

Throughout this paper we use a black-box computing system executing workloads, as depicted in Figure 1. The computing system has a number of control parameters that affect the system. These controls are called *actuators*. We measure the effect these actuators have on the system using the measurements that the system exports. If the computing system does not export suitable actuators or measurements, we can interpose the flow of requests with a scheduler [12], allowing us to control (actuate) the flow into the system and measure the performance at the scheduler. A system might have actuators and measurements from either the scheduler or the black-box system, or from both of the two parts.

The methods we propose are applicable to any combination of actuators and measurements, as long as the expected relationship between them is monotonic. Examples of this include: controlling the percentage of CPU resources each workload gets and measuring performance in terms of transfered bits/s; changing the amount redundancy and measuring the dependability; and adjusting CPU frequency to meet power consumption targets.

The adaptive controller used in this paper is the widely used self-tuning regulator (STR) depicted in Figure 2. It consists of two parts: an estimator and a control law. In order to explain why the standard estimator does not perform well and what kind of problems it can give the control loop, we first describe the estimator in Section 2.1 and the control law in some detail in Section 2.2.

2.1 Recursive Least-Squares Estimation

Recursive least-squares (RLS) [10] is one of the most widely used estimation algorithm in adaptive controllers, due to its robustness against noise, its good convergence speed and proved convergence properties that can be used to prove the stability of the whole control loop. Our proposed methods are also valid for other versions of least-squares, such as extended least-squares, total least-squares, the projection algorithm, the stochastic approximation algorithm and least-mean squares. It is applicable to any technique estimating a model on the same form as RLS.

In order to explain least-squares estimation we have to introduce some notation. Let $u(k) = [u_1(k) \dots u_M(k)]^T$ be the vector of the M actuator setting during sampling interval k, and let $y(k) = [y_1(k) \dots y_N(k)]^T$ be the vector of the performance measurements of the N workloads, measured at the beginning of interval k. The symbols have been collected in Table 1 for easy reference. The relationship between $u(k)$ and $y(k)$ can be described by the following multiple-input-multiple-output (MIMO) model:

$$y(k) = \sum_{i=0}^{n} A_i y(k-i-1) + B_i u(k-i-1) \qquad (1)$$

where A_i and B_i are the model parameters. Note that $A_i \in \mathbb{R}^{N \times N}$, $B_j \in \mathbb{R}^{N \times M}$, $0 < i \leq n$, $0 \leq j < n$, where n is the order

IEEE COMPUTER SOCIETY

Symbol	Meaning
$y(k)$	Performance measurements at time k.
$u(k)$	Actuator settings at time k.
A_i	Model parameters in front of y.
B_i	Model parameters in front of u.
$X(k)$	Model parameter matrix at time k.
$\phi(k)$	Regression vector at time k.
$y_{ref}(k)$	Desired performance at time k.
$T(k)$	Throughput at time k.
$T_{\Sigma}(k)$	Total throughput at time k.
$L(k)$	Latency at time k.
$S(k)$	Share setting at time k.

Table 1. Frequently used symbols.

of the model. This linear model was chosen for tractability as we know that the relationship will indeed, in all but the most trivial cases, be nonlinear. However, it will be a good local approximation of the nonlinear function, and this will often be good enough for the controller as it usually only makes small changes to the actuator settings.

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

$$y(k+1) = X(k)\phi(k) \tag{2}$$

where

$$X(k) = [B_0 \ \dots \ B_{n-1} \ A_0 \ \dots \ A_{n-1}]$$
$$\phi(k) = [u^T(k) \ \dots \ u^T(k-n+1) \ y^T(k) \ \dots \ y^T(k-n+1)]^T$$

where $X(k)$ is called the *parameter matrix* and $\phi(k)$ is referred to as the *regression vector*.

RLS is then defined by the following equations:

$$\hat{X}(k+1) = \hat{X}(k) + \frac{\varepsilon(k+1)\phi^T(k)P(k-1)}{\lambda + \phi^T(k)P(k-1)\phi(k)} \tag{3}$$

$$\varepsilon(k+1) = y(k+1) - \hat{X}(k)\phi(k) \tag{4}$$

$$P(k) = \frac{P(k-1)}{\lambda} - \frac{P(k-1)\phi(k)\phi^T(k)P(k-1)}{\lambda(1+\phi^T(k)P(k-1)\phi(k))} \tag{5}$$

where $\hat{X}(k)$ is the estimate of the true value of $X(k)$, $\varepsilon(k) \in \mathbb{R}^{N\times 1}$ is the estimation error vector, $P(k) \in \mathbb{R}^{NMn\times NMn}$ is the covariance matrix and λ is the *forgetting factor* ($0 < \lambda \leq 1$). A high λ means that RLS remembers a lot of old data when it computes the new model. Conversely, a low λ means that it largely ignores previous models and only focuses on producing a model from the last few samples.

The intuition behind these equations is quite simple. (4) computes the error between the latest performance measurements and the performance prediction of the model $\hat{X}(k)\phi(k)$. We refer to this as the *rls error*. The model parameters are then adjusted in (3) according to the rls error and another factor dependent on the covariance matrix P computed in (5). P contains the covariances between all the measurements and the actuators. The model $\hat{X}$ is then used by the control law described next to set the actuators correctly.

2.2 Control Law

The only way RLS defines an error in the model is through the rls error. In this section, we will show that even if a model has no rls error, it still might give rise to unacceptable controller performance. To be able to explain why this is the case and what other property other than rls error the model need to have, we need to explain a bare minimum self-tuning regulator (STR). While this basic STR has a number of drawbacks, it serves the purpose of explaining the problem with as little as possible of control-theoretic details. All other direct STRs have this problem, the equations are just more elaborate.

Assume for notational convenience that the order of our system model is one. The model (1) is then:

$$y(k) = A_0 y(k-1) + B_0 u(k-1) \tag{6}$$

To turn this model into a controller, we observe that a controller is a function that returns $u(k)$. If we shift equation (6) one step ahead in time and solve for $u(k)$, we get:

$$u(k) = B_0^{-1}(y(k+1) - A_0 y(k)) \tag{7}$$

If this equation is to be used to calculate the actuation setting $u(k)$, then $y(k+1)$ represents the desired performance to be measured at the next sample point at time $k+1$, i.e., it is $y_{ref}(k)$. Thus, the final control law is:

$$u(k) = B_0^{-1}(y_{ref}(k) - A_0 y(k)) \tag{8}$$

This is a simple STR for the model given by (6).

To illustrate the point that a model with zero rls error can still cause the controller to underperform, consider a black-box computing system where the actuator is the share of CPU we give to each workload, and the performance metric we care about is latency. For this system, consider two estimators that produce these two models with only one input and one output:

$$X_1(k) = [-0.2 \ \ 0.6]$$
$$X_2(k) = [0.2 \ \ 0.4]$$

If $y(k-1) = y(k) = 4$ and $u(k-1) = 2$ i.e., $\phi(k-1) = [2 \ \ 4]^T$, then $X_1(k)\phi(k-1) = X_2(k)\phi(k-1)$ and both have the same rls error. Consider now what happens when we use these two models that are equivalent in the least-squares sense, in the control law (8) when $y_{ref}(k) = 1$. For model 1, the actuator setting would be $u_1(k) = \frac{1}{-0.2}(1 - 0.6\cdot 4) = 7$. Model 1 does what we would expect. When the controller

IEEE COMPUTER SOCIETY

observes a latency that is higher than desired it should increase the CPU share of the workload that its latency will go down. But with model 2, it does the complete opposite of what is desired, as $u_2(k) = \frac{1}{0.2}(1-0.4\cdot 4) = -3$. It decreases the CPU share so that the latency is increased and the latency target is missed even more. If the model was constant, the controller using model 2 would eventually reach the latency goal. But with an adaptive controller, the model changes, so this bad behavior might go on for a much longer time. We will show in the experimental section that this happens frequently and gives rise to performance degradations.

In the example above, B_0 is negative when the controller works and positive when it does not work. The physical meaning of B_0 in (6) should reflect the fact that if more share is given to a workload, the latency should go down, and conversely, if less is given the latency should go up. For that to be true, B_0 has to negative. To illustrate why it is more important for B_0 to have the correct sign than any other model parameter, consider this control law derived using a second order model.

$$u(k) = B_0^{-1}(y_{ref}(k) - A_0 y(k) - A_1 y(k-1) - B_1 u(k-1)) \tag{9}$$

If one of A_0, A_1 or B_1 has the incorrect sign, the other parameters might correct for this and the whole expression within the parenthesis comes out with the right sign. However, if B_0 has the wrong sign there is no single parameter that can compensate for this, unless the whole expression within the parenthesis comes out with the wrong sign too. The higher the order, the more critical it is that B_0 has the right sign, compared to the all the other model parameters.

When the model has more than one input and one output, B_0 is a matrix. The diagonals of this matrix should then be negative as an increase in CPU share of a workload should result in lower latency. Each entry in the anti-diagonal captures the effect increasing the share of one workload has on the latency of one other workload. These should then be positive or zero. Positive if the two workloads compete for some resource in the system as increasing the share of one would decrease it for the other, and zero if they do not compete for a resource[1]. A correct B_0 would then for the combination of a CPU share actuator and latency as the performance measurement, look like this:

$$\begin{pmatrix} <0 & \geq 0 & \cdots & \geq 0 \\ \geq 0 & <0 & \cdots & \geq 0 \\ \vdots & \vdots & \ddots & \vdots \\ \geq 0 & \geq 0 & \cdots & <0 \end{pmatrix} \tag{10}$$

[1]It is actually possible for the anti-diagonal entries to be negative as increasing the amount of resources for one workload might positively help another. This might occur if one workload loads data into a cache that the other workload then reads. However, we have not been able to observe this on our systems.

Other actuator and performance measurement combination would have other rules on what a correct B_0 is. For example if the performance measurement was throughput instead of latency, the diagonals of (10) would be > 0, as higher CPU share means higher throughput. The anti-diagonals would then be ≤ 0.

3 Solutions

In this section, we propose two types of solutions, both aimed at improving models and thus the control performance. First in Section 3.1, we look at what combination and arithmetic manipulation of actuators and measurements produces the best models when considering the rls error and the error to B_0. Second, we propose a number of techniques that can alleviate the impact of a B_0 with incorrect signs in Section 3.2.

3.1 Possible Input Vectors

As an example in this section we will use the measurements and actuators we use in the evaluation in Section 4. The techniques presented in this section can be used with other actuators and measurements as long as the relationship is monotonic. Actuators can be grouped into two main types: *ratios* and *absolute* actuators. Our example system uses a ratio actuator namely the share ($S(k)$) of the number of requests that is submitted to the system on a per workload basis. However, it is also possible to map the absolute throughput ($T(k)$) to the real physical actuator, by just dividing each individual workload's throughput with the total throughput from all the workloads. We then have two actuators we can use, $S(k)$ and $T(k)$. In the system, we can measure latency ($L(k)$) and we are interested in meeting latency goals. $y(k)$ should then be set to $L(k)$ as we are interested in predicting and controlling latency. The question is then, what should we then put in $\phi(k)$ to get an estimation that is as good as possible?

There are some rules of thumb to think about when making this decision. First, the fewer the variables, the faster RLS will generally converge. To leave out critical variables is not good either, for obvious reasons. Second, we do not include variables that are heavily dependent on each other as the covariance matrix P will become rank deficient. At that point, strange results might occur due to very low or high numbers being produced coupled with the limited floating point accuracy of CPUs. Models can sometimes become random when this occurs.

For notational convenience, let $V_{-1}(k)$ signify a row vector with the last entry removed from the original vector V, and $V_\Sigma(k)$ signify the sum off all the elements in row vector V. The most obvious choice of $\phi(k)$ would be to just enter the share percentages $S(k)$. However, as $S_\Sigma(k) = 100$,

Short-hand	$\phi(k)$	Comments
Ratio (R)	$S_{-1}^T(k)$	Fewer variables, but a nonlinear ratio.
Throughput (T)	$T^T(k)$	Provides absolute numbers.
RatioAffine (RA)	$[S_{-1}^T(k)\ 1]$	As Share but affine.
ThroughputAffine (TA)	$[T^T(k)\ 1]$	As Throughput but affine.
ThroughputSum (TS)	$[T_{-1}^T(k)\ T_\Sigma(k)]$	Adds sums to Throughput for scaling.
ThroughputSumAffine (TSA)	$[T_{-1}^T(k)\ T_\Sigma(k)\ 1]$	Affine function plus sum for Throughput.

Table 2. Various ways of forming $\phi(k)$ and some comments.

entering all the entries in $S(k)$ does not provide any extra information. Instead, we drop the last entry of $S(k)$ and use $S_{-1}(k)$. The good thing about this choice is that $\phi(k)$ has few variables. One drawback is that the share ratios hide the absolute throughput of the system, which also affects latency. When the capacity of the system changes, this shortcoming results in poor tracking. This choice of $\phi(k)$ is called Ratio. It, and all others described below, have been tabulated in Table 2.

The second choice is to use the throughput $T(k)$. This corresponds to the share ratio prior to normalization by the summed inputs and therefore has a more constant relationship relative to latency. We refer to this choice as Throughput. Even if the total capacity of the system changes, this model will be good. However, it would still not be a good approximation if the service time of the individual requests changes.

A number of tricks used in other fields to improve estimation, that we could use to extend the list of possible $\phi(k)$. The models we have proposed so far are not affine, i.e., they all start from the origin. This is a drawback when using share ratios or throughputs to model latency. No share, or close to no share, for one workload will not mean zero latency to that workload. One way of dealing with this is to extend $\phi(k)$ by an additional input dimension that is always constant [21], say 1. The model equations will then become:

$$y(k) = \sum_{i=0}^{n} (A_i y(k-i-1) + B_i u(k-i-1)) + C \cdot 1 \quad (11)$$

where $C \in \mathbb{R}^{N \times 1}$ can take on any value estimated by RLS. Allowing affine functions produces two new possible ways to form $\phi(k)$ referred to as RatioAffine and ThroughputAffine as shown in Table 2.

A second trick used in image processing [6] is to add the sum of the throughputs to $\phi(k)$. When the throughput is changing, having this extra dimension gives a way for least squares to approximately scale the effects expected from $T(k)$. It is analogous to the scaling that you do to get a ratio, but instead of being a division that RLS does not have access to (and which introduces a non-uniform rescaling on its input vectors), it can be used to estimate a linear approximation to the normalized ratio. Adding this to the shares only creates rank deficiencies, so they have not been included. Instead, we end up with two more possible $\phi(k)$ called ThroughputSum and ThroughputSumAffine that are shown in Table 2.

Short-hand	Description
None	Do nothing and hope for the best.
Remember	Remember a good model from the past and use that one if B_0 is bad.
Modify	Modify the model so that B_0 becomes good.
RunAll	Estimate models from all estimators all the time and use the model that is the best.

Table 3. Our three techniques of dealing with a B_0 with incorrect signs plus the baseline.

3.2 Dealing with an Incorrect B_0

As we will see in the experimental section, none of the input data permutations above gets rid of erroneous B_0 matrices. Therefore, in this section we will present three methods to alleviate or completely remove the effects of B_0 errors. The methods are presented in Table 3.

The Remember method always saves the latest known good model that had a correctly signed B_0. When a new model is estimated with a B_0 sign error, it uses the last known good model instead of the new model. It keeps on using the last known good model until the new model recovers and gets a correctly signed B_0. While simple, the drawback of this method is that the model might never recover, and in that case we will use an outdated model forever that might not at all represent the system any more.

The Modify method tries to tackle this problem by forcefully modifying the model when a B_0 error is detected. Let B_0 be a matrix with a B_0 error and let $B_0 + \tilde{B}$ be a matrix that does not have a B_0 error. $\tilde{B}$ is chosen so that an entry with the wrong sign gets the value 0.001 if it should be positive and -0.001 if it should be negative. We did not chose the value 0 as it might give rise to divisions by zero and singular matrices. If we were to modify B_0 alone in this way, the model would suddenly predict completely different $u(k)$

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**
IEEE COMPUTER SOCIETY

values than before and have an rls error. This might actually make the controller perform much worse than before. Therefore, we need to modify the whole model such that it predicts the same model locally with the modified $B_0 + \tilde{B}$ matrix as it did before with only B_0.

Let K be the predicted output of the unmodified model without the terms B_i and A_i for $i > 0$, that can safely be ignored for our purpose. Then the unmodified model is:

$$K = A_0 y(k) + B_0 u(k) \quad (12)$$

Then the problem can be specified as:

$$\begin{aligned} K &= (A_0 + \tilde{A}) y(k) + (B_0 + \tilde{B}) u(k) \\ &= A y_0(k) + \tilde{A} y(k) + B_0 u(k) + \tilde{B} u(k) \end{aligned} \quad (13)$$

Using (12), this is equivalent to

$$\tilde{A} y(k) + \tilde{B} u(k) = 0 \Leftrightarrow \tilde{A} y(k) = -\tilde{B} u(k) \quad (14)$$

This equation has multiple solutions as $-\tilde{B}u(k)$ is a row vector and $\tilde{A}$ is a matrix. One possible solution is to set $\tilde{A}$ to the following:

$$\tilde{A} = \begin{pmatrix} -V_1/y_1(k) & 0 & \cdots & 0 \\ 0 & -V_2/y_2(k) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -V_N/y_N(k) \end{pmatrix} \quad (15)$$

where V_i is row i of $\tilde{B}u(k)$ and $y_i(k)$ is row i of $y(k)$. Note that this modified model will only be the same as the unmodified one locally around $\phi(k)$.

The last method `RunAll` runs six estimators in parallel. One for each single possible $\phi(k)$ in Table 2. Out of those six models, we will the pick the model that has no B_0 sign errors and that has the lowest rls error. If there is no model with a correctly signed B_0, we pick the one with the least amount of B_0 error. The intuition behind this is that hopefully at least one of the estimators will always produce a good model that we can use, and we believe that a good B_0 is more important than a perfect rls error. Many other ways to pick the model to use exists, e.g., forming a weighted sum of the two errors and the pick the one with the lowest. We chose the former one for its simplicity and its lack of tunable parameters.

4 Experimental Results

In this section we present the experimental methodology and the results. In summary, the results show that:

- Standard RLS estimation does not work for the black-box systems examined.
- There is no single regression vector that consistently produces good results. This is highly dependent on workload and system.
- The `Modify` method of mitigating B_0 errors works the best and manages to quickly correct any errors in B_0, and improve both rls error and controller performance.
- The combination of inputting individual workload throughputs together with a constant input parameter (`ThroughputAffine`) combined with the `Modify` technique consistently provides the best controller performance.

4.1 Experimental Methodology

The experimental evaluation is performed on two different systems. First, we use a three-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 web, application, and database servers are hosted on separate server blades, each with two 1 GHz Pentium III processors, 2 GB of RAM, one 46 GB 15 krpm SCSI Ultra160 disk, and two 100 Mbps Ethernet cards. The web server is Apache version 2.0.48 with a BEA WebLogic plug-in. The application server is BEA WebLogic 7.0 SP4 over Java SDK version 1.3.1 from Sun. The database client and server are Oracle 9iR2. All three tiers run on Windows 2000 Server SP4. The site hosted on the 3-tier system is a version of the Java PetStore [11] that has been tuned in order to support a large number of concurrent users.

The workload applied mimics real-world user behavior [5], e.g., browsing, searching and purchasing behaviors including their respective time scales and probabilities. For the experiments here, we emulate 75 users partitioned into two classes. Each class is considered to be one “workload”. The control interval for both the controller and all the measurements is 1 s.

As a second experimental setup, we use the web server in the above setup with only static content of size 64K. The client accesses are generated by `httperf` and has a Poisson arrival process. The sampling interval is also 1 s for the web server setup.

In order to be able to control the performance of the black-box systems above, we have used an approach based on interposing a fair-queuing scheduler [12] on the path between the system and its clients, as depicted in Figure 1. The scheduler enforces a configurable share of the system’s capacity that each workload receives. The only available

measurements from the outside is the latency and throughput each workload receives. We will focus on latency as it is harder to estimate due to its nonlinear dependency on the share. The order of the model and the controller is set to two, as it has proven to be a good approximation of the two systems examined.

4.2 Estimation Results

In this section, we evaluate what regression vector makes the best predictions of future latencies. To make the comparison as fair as possible, the estimation is performed on two sets of measurement data that has been previously gathered from the two systems. Each set consists of 14,400 samples from a 4 hour run. The data was gathered by picking a uniformly random number $i \in [30, 70]$ at beginning of each sample interval and then setting the shares to $u(k) = [i \;\; 100-i]^T$. This is white noise input and provides the estimator with values (and frequencies) from all over the interval. The reason that the interval is not larger than $[30, 70]$ is that there is no good linear approximation between share and latency for intervals larger than that. In Section 4.3, we will see that when an STR is put on top of the estimator, the estimator performance will often be degraded as the controller will make sure that only certain values of the shares and latencies are seen by the estimator as it aims for a specific latency goal that usually corresponds to a small subset of share values.

The regression vectors are evaluated using two metrics: *rls error* and B_0 *error*. RLS error is the error according to recursive least-squares and tells us how well the current model predicts the next $y(k)$ vector. It is defined as

$$E_{RLS}(k) = \frac{\sum_{i=1}^{N} |y_i(k) - \hat{X}_i(k-1)\phi(k-1)|}{N} \quad (16)$$

where $\hat{X}_i(k)$ is row i of $\hat{X}(k)$. The B_0 error, on the other hand, will tell us how wrong the signs of B_0 are. If there are any entries with wrong signs, the metric will have a value greater than zero. The greater the impact of the wrong signs to the controller output $u(k)$, the higher the B_0 error. A B_0 with no sign errors has a B_0 error of zero. It is formally defined as

$$E_{B0}(k) = \frac{\sum_{j=1}^{N}\sum_{i=1}^{M} |\beta_{ji}(k)u_i(k)|}{N} \quad (17)$$

where each entry of $\beta(k) \in \mathbb{R}^{N\times M}$ is zero if the corresponding entry in B_0 has the correct sign or the value of the corresponding B_0 entry if that entry has the wrong sign.

The top two graphs in Figure 3 show the mean and the standard deviation of the rls error as a function of the forgetting factor (λ) for the six ways of forming $\phi(k)$ for the three-tier system. From the graph, we can see that `Ratio` and

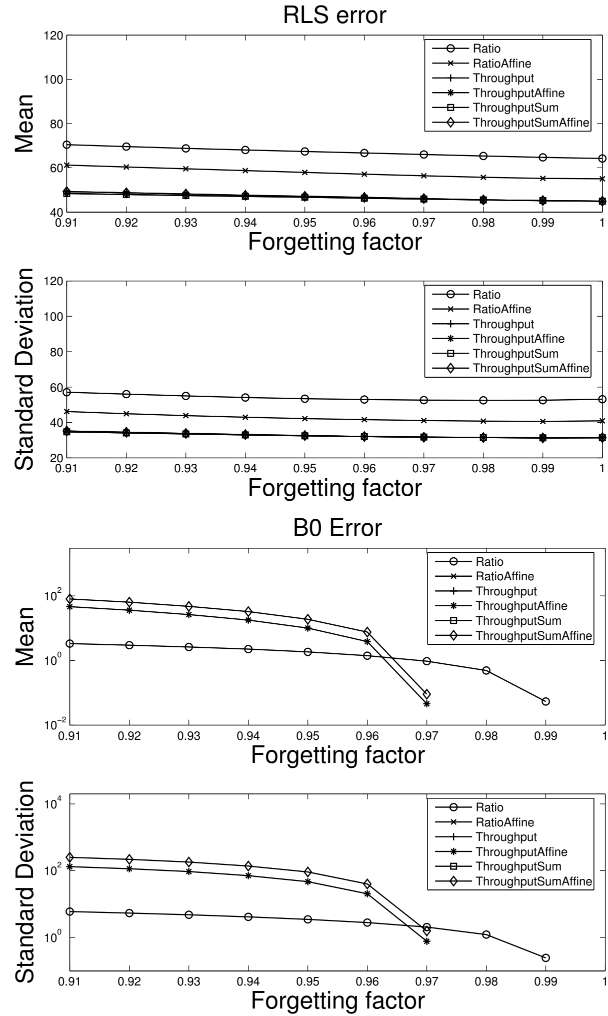

Figure 3. The mean and standard deviation of the RLS error and the B_0 error for the three-tier system.

`RatioAffine` has a worse rls error than the ones based on throughput measurements that are more or less overlapping in the graph. However, the errors are within one standard deviation of the mean, so we cannot say with certainty that there is a statistically significant difference in rls error. Figure 4 plots the rls error as a function of time for two of the input vectors, showing that the rls error indeed varies a lot over time with no clear winner.

The bottom two graphs of Figure 3 show the B_0 error of the regression vectors as a function of the forgetting factor (λ). Note, that the graph is logarithmic, thus any point not found on the graph has a B_0 error of zero. The results in terms of B_0 error are much clearer than those given by rls error. From the graph we can see that for $\lambda \leq 0.97$, `RatioAffine`, `Throughput` and

IEEE COMPUTER SOCIETY

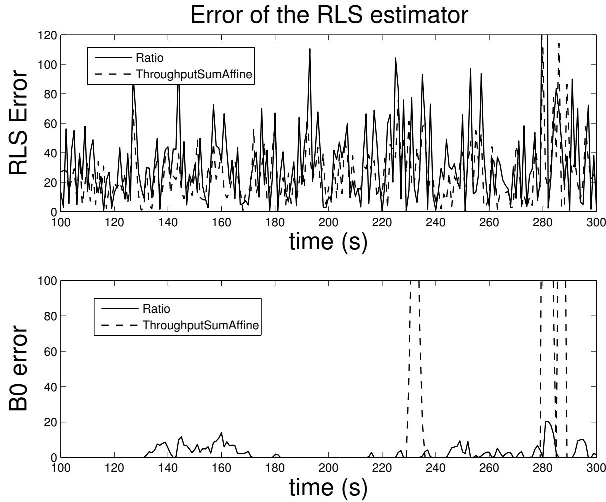

Figure 4. RLS and B_0 error as a function of time for the three-tier system when $\lambda = 0.95$.

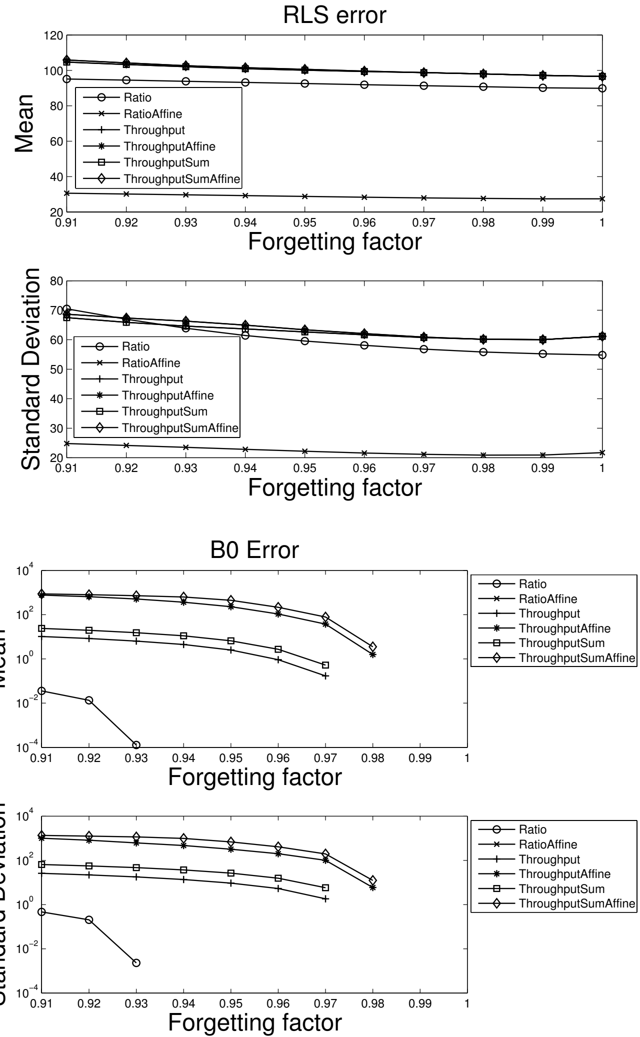

Figure 5. The mean and standard deviation of the RLS error and B_0 error for the web server.

ThroughputAffine are clearly better than the rest, with no B_0 error at all. A $\lambda = 1$ also gives no B_0 error, but that model is probably useless, as it will never change after a while and be unresponsive to changes in the system. Looking at the B_0 error as a function of time in Figure 4, we can see that the B_0 errors occur in bursts. In between, B_0 has correct signs.

To see how these results change for other systems, consider Figure 5. For the web-server, RatioAffine is clearly the best choice as it provides the lowest rls error and has no B_0 error. We have run experiments with a number of other workloads on top of the three-tier system, where we varied the ratio of CPU intensive content to more light-weight content and the rate at which this changed during the execution. We also changed the distribution of document sizes for the web server and changed the interval that is used to draw random share value. (The results from all these experiments are not shown in the paper due to space considerations.) By looking at these experiments and the ones shown here, there is no clear method that is better than the other. If one is the best for one workload, it is the worst for another. But as will be shown in the next section, there is a combination of regression vector and B_0 method that consistently outperforms all the other approaches.

4.3 Controller Results

To be able to study how the results of the previous section hold and how the B_0 methods work, we evaluate them within a self-tuning regulator. The STR used [16] is designed using the basic principles of Section 2.2. However, it contains more additions in order to improve the performance of the control loop. These additions are among others, to deal with actuator constraints, maximize system usage while meeting the latency goals, handle drastic changes in the system, and improve stability. The details of the controller is outside the scope of this paper and can be found in Karlsson *et al.* [16]. In order to get a concise metric on how well the controller meets its latency target we define *control error* as:

$$E_{ctrl} = \frac{\sum_{j=1}^{N} \sum_{i=1}^{K} |y_{ref,j}(i) - y_j(i)|}{KN} \qquad (18)$$

where K is the number of samples during the execution. This is the mean absolute deviation from the performance reference throughout the execution and among all workloads. We will focus on the results from the three-tier system, as the results from the web-server produced the same conclusions.

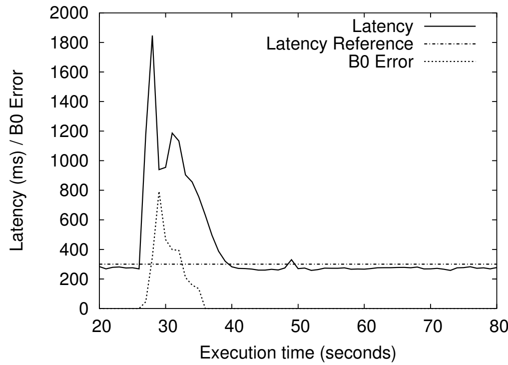

Figure 6. How the controller manages to track the latency reference with time and how that ability degrades when there is a B_0 error. The B_0 error method is None and Throughput is used for $u(k)$.

To see that B_0 errors reduce the performance of the control loop, consider Figure 6, plotting the request latency and the B_0 error as a function of time. The system uses Throughput for the RLS estimation and None for correcting the B_0 errors. The rls error is small during the whole time interval. From the graph, we can see that B_0 errors have a large negative impact on the ability of the controller to meet the latency target. When there is a B_0 error, the latency is up to 6 times as high as targeted. When there is no B_0 error, the controller manages to track the latency target well. Note, that the estimator eventually recovers from the B_0 error.

Table 4 shows statistics from the various runs. The metrics are the means across the whole 4 h execution. All of the regression vectors suffers from B_0 errors. There are two main observations to take away. First, that the control error is a function of both the rls error and the B_0 error. If at least one of them is high, the control error also tends to be high. Second, that ThroughputSumseems to be better than the others when there is no B_0 method.

Let us try to decrease the errors of the estimated model and thus improve the control performance, by studying the Remember method. Figure 7 shows the latency and B_0 error as a function of time for an illustrative time period of an execution using ThroughputAffine and Remember. At around time 50 s, there is a sudden change in the system performance and the model estimated suddenly starts to have B_0 errors. As soon as this is observed, the controller will start to use the latest good model without a B_0 error. However, the estimated model never recovers from the B_0 error and the old model is used for the rest of the execution. The key point to note here is that as the system changes, the model used will not as we are using an old model without a B_0 error. The negative impact of this can be seen in

Method	None					
Input vector	R	RA	T	TA	TS	TSA
rls error	118	88.1	50.8	90.6	27.5	39.5
B_0 **error**	37.8	88.4	47.6	25.4	69.6	67.4
control error	61.2	41.4	45.3	55.4	29.8	86.4

Method	Remember					
Input vector	R	RA	T	TA	TS	TSA
rls error est.	134	60.5	6.86	7.40	12.0	40.8
rls error used	319	127	73.0	75.3	75.2	143
B_0 **error est.**	31.4	92.2	63.0	118.3	219	484
B_0 **error used**	2.34	2.70	0.52	0.38	0.35	0.34
control error	133	31.2	14.4	18.3	14.8	38.5

Method	Modify					
Input vector	R	RA	T	TA	TS	TSA
rls error	72.0	36.5	13.4	8.79	11.4	13.4
B_0 **error est.**	4.77	4.37	8.98	6.29	6.43	5.05
B_0 **error used**	0	0	0	0	0	0
control error	44.1	23.3	14.7	12.0	14.1	16.4

Method	RunAll
Input vector	ALL
rls error used	22.3
B_0 **error used**	15.4
control error	19.3

Table 4. Overall error statistics for all the input vector and B_0 error methods. The "est." rows are for the model produced by the estimator and "used" rows are for the model used by the control law.

the latency slowly diverging from the goal as time goes by and the performance characteristics of the system changes. Remember only works well when there are short bursts of B_0 errors, not when they last for a long time.

Looking at the compounded results for Remember in Table 4, the method does improve the control performance of all input vectors except Ratio. While this method decreases the B_0 error, it does so at the cost of an increase in the rls error of the used model, as can be seen in Table 4.

The main drawback of Remember was that the model used in the controller might get a high rls error if the estimated model most of the time has a B_0 error. Modify on the other hand, will try to use the estimator to get a low rls error and modify that model so that B_0 has the correct signs. Figure 8 shows the latency and B_0 error of ThroughputAffine for an illustrative time period. With this method, B_0 errors are quickly correct and have little impact on the control performance. As can be seen in Table 4, Modify offers the least rls error as well as B_0 error, and thus offers the best control performance of all the

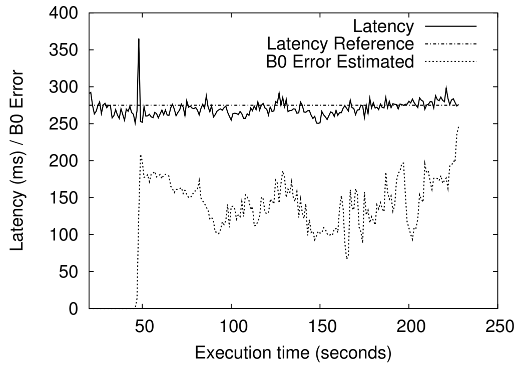

Figure 7. How the controller manages to track the latency reference with time and how that ability degrades when there is a B_0 error. The B_0 error method is Remember.

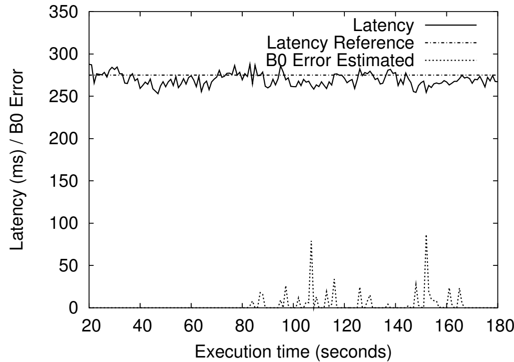

Figure 8. How the controller manages to track the latency reference with time and how B_0 errors are quickly corrected. The B_0 error method is Modify.

methods. The only exception to is Throughput that suffers a small degradation compared to Remember. Modify reduces the control error by up to 11 times compared to the previous methods.

The results from the RunAll method are tabulated in Table 4. Compared to the other techniques, RunAll performs as well as Remember with the best regression vectors, but not as well as Modify with a throughput based regression vector. We also tried RunAll together with Modify, but the improvement compared to Modify and ThroughputAffine was small. As the computational burden of RunAll is six times that of just having a single regression vector, the best choice is to go with Modify.

To conclude the results of this section, the overall best method is to use ThroughputAffine with Modify.

5 Related Work

There are a few examples of self-tuning regulators used to control computer systems. Lu *et al.* [17] estimates a model between cache size and the hit ratio a workload receives. Triage [15] estimates a model between the total number of requests sent to the system and the latency. Karlsson *et al.* [16] estimates a model between shares and both latency and throughput. None of these approaches consider B_0 errors or evaluates different ways of estimating the performance models. All of them could probably benefit from the techniques presented in this paper.

There are also a number of adaptive controllers that use ad-hoc estimation techniques that are not part of the estimation literature. For example, the control law and estimator of Yaksha [13] could probably benefit from our techniques.

While RLS is the most widely used estimation technique for STR, there are others including extended least-squares (ELS), total least-squares (TLS), the gradient estimator, the projection algorithm, the stochastic approximation algorithm and least-mean squares (see Åstrom *et al.* [2] for details on all of these methods). All, with the exception of ELS and TLS, are computationally more efficient but generally do not provide as good estimates as RLS. However, our techniques are applicable to all the above estimators too, without any modifications. In fact, our techniques are applicable to any estimator using the same model used in this paper.

Some of the techniques we have employed have been successfully used in other fields. The trick of including scaling separately, as in the ThroughputSum method has been used for handling perspective projection in images and video [6]. To include a constant as input to least-squares has been used in many fields, for example in the off-line estimation of streaming media server performance [7], in order to get affine functions. The use of multiple concurrent estimators is not uncommon in adaptive control theory [2]. But it has been used in order to lessen the impact of covariance windup, not errors in the signs of B_0.

6 Conclusions

This paper is concerned with dynamically estimating a performance model of a black-box computer system using a least-squares estimator that provides a self-tuning regulator with good control performance. We show that regular least-squares does not produce models that lead to good control performance. This paper identifies that the signs of the B_0 model parameter matrix is critical to good controller performance, as well as the least-squares minimized model error. To alleviate this situation, we propose and evaluate two sets of techniques. First, we examine what combination and arithmetic manipulation of measurements and actuators that

IEEE COMPUTER SOCIETY

provide the best models. Second, we propose three methods that can mitigate the effects of an incorrect B_0 parameter produced by the estimator, and provide the controller with better performance under those circumstances.

The techniques are evaluated with a self-tuning regulator that provides latency targets for workloads on black-box systems. The experimental evaluation was conducted on two systems: a three-tier e-commerce site and a web server. The results show that the overall best controller performance is achieved by estimating the latency as an affine function of throughput coupled with the technique of modifying models with B_0 errors. This combination results in up to 11 times average lower error between the desired latency and the latency the workloads receive. Previously oscillating workload latencies are with our technique smooth around the latency targets.

Acknowledgment

The authors are indebted to Christos Karamanolis and Zhikui Wang for their invaluable help with the paper.

References

[1] T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), 2002.

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

[3] 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.

[4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centres. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, Banff, Canada, October 2001.

[5] 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)*, pages 231–244, San Francisco, CA, December 2004.

[6] M. Covell, A. Rahimi, M. Harville, and T. Darrell. Articulated-Pose Estimation using Brightness- and Depth-Constancy Constraints. In *IEEE International Conference on Computer Vision and Pattern Regocnition (CVPR)*, pages 2438–2445, Head Island, SC, June 2000.

[7] M. Covell, B. Seo, S. Roy, M. Spasojevic, L. Kontothanassis, N. Bhatti, and R. Zimmermann. Calibration and prediction of streaming-server performance. Technical Report HPL-2004-206R1, HP Laboratories, 2004.

[8] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley Publishing Company, 3 edition, 1998. ISBN 0-201-82054-4.

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

[10] M. Honig and D. Messerschmitt. *Adaptive Filters: Structures, Algorithms, and Applications*. Kluwer Academic Publishers, Hingham MA, 1984. ISBN 0-898-38163-0.

[11] *Java PetStore*. www.middleware-company.com.

[12] 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.

[13] 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.

[14] N. Kandasamy, S. Abdelwahed, and J. Hayes. Self-Optimization in Computer Systems via On-Line Control: Application to Power Management. In *International Conference on Autonomic Computing (ICAC)*, pages 54–61, New York, NY, May 2004.

[15] 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.

[16] M. Karlsson, X. Zhu, and C. Karamanolis. An Adaptive Optimal Controller for Non-Intrusive Performance Differentiation in Computing Services. In *IEEE Conference on Control and Automation (ICCA)*, Budapest, Hungary, June 2005. To appear.

[17] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework for QoS guarantees and its application to differentiated caching services. In *International Workshop on Quality of Service (IWQoS)*, pages 23–32, Miami Beach, FL, May 2002.

[18] 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.

[19] 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.

[20] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Design and Evaluation of Load Control in Web Server Systems. In *American Control Conference (ACC)*, pages 1980–1985, Boston, MA, June 2004.

[21] G. Strang. *Linear Algebra and its Applications*. Brooks Cole, 3 edition, 1988. ISBN 0-155-51005-3.

[22] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 239–254, Boston, MA, December 2002.

[23] Z. Zhang, S. Lin, Q. Lian, and C. Jin. RepStore: A Self-Managing and Self-Tuning Storage Backend with Smart Bricks. In *International Conference on Autonomic Computing (ICAC)*, pages 122–129, New York, NY, May 2004.