Controlling the Performance of 3-Tiered Web sites: Modeling, Design and Implementation

[Extended Abstract]

Abhinav Kamra
Dept. of Computer Science
Columbia University
New York, NY
kamra@cs.columbia.edu

Vishal Misra
Dept. of Computer Science
Columbia University
New York, NY
misra@cs.columbia.edu

Erich Nahum
T.J. Watson Research Center
IBM Corporation
Yorktown Heights NY
nahum@watson.ibm.com

Category and Subject Descriptors: C.4 Computer Systems Organization:Performance of Systems:Design Studies

General Terms: Design, Experimentation, Measurement

Keywords: Control Theory, TPC-W, E-Commerce, Admission Control

1. INTRODUCTION

E-Commerce is rapidly becoming an everyday activity as consumers gain familiarity with shopping on the Internet. The infrastructure behind E-Commerce Web sites is typically composed of a three-tiered architecture, consisting of a front-end Web server, an application server and a back-end database.

Two problems are frequently encountered with deploying such Web sites. First is *overload*, where the volume of requests for transactions at a site exceeds the site's capacity for serving them and renders the site unusable. Second is *responsiveness*, where the lack of adequate response time leads to lowered usage of a site, and subsequently, reduced revenues.

This paper presents a method for controlling multiple-tiered Web site performance, both by bounding response times and preventing overload. Our approach uses a *self-tuning proportional integral (PI) controller* for admission control, enabling overload protection and bounding response time based on an administrator-based policy (e.g., 90 percent of the requests should see a response time of less than 100 milliseconds). By using a self-tuning controller, our system automatically adapts to variation in load and requires only two parameter settings.

Our method requires no changes to the operating system, Web server, application server or database. This allows rapid deployment and use of pre-existing components.

We present an implementation of our controller in a proxy, called *Yaksha*[1]. We evaluate our system with standard software components used in multiple-tiered e-Commerce Web sites, namely Linux, Apache, Tomcat, and MySQL. We drive the system using the industry-standard TPC-W [2] benchmark, and demonstrate that Yaksha achieves both stable behavior during overload and bounded response times. Our results show that a properly designed and implemented controller be used in a complex environment, such as multi-tiered Web sites.

[1] *Yakshas* are the guardians of wealth in the Hindu pantheon.

SIGMETRICS/Performance'04 June 12-16, 2004, New York, NY, USA
ACM 1-58113-664-1/04/0006.

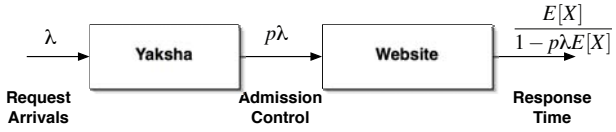

Figure 1: System model for Yaksha and the Web site

.

2. MODELING AND DESIGN

In this section we present the system model and design procedure for Yaksha. Our abstraction for the E-commerce server is an $M/GI/1$ Processor Sharing queue. We denote by $T(x)$ the mean response time of a job whose job size (or service time) is x. The job size in an $M/GI/1$ is an i.i.d. random variable, denoted by X, whose probability distribution function is $F(x)$, with a mean $E[X]$.
It is well known that

$$T_{PS}(x) = \frac{x}{1-\rho}. \tag{1}$$

where ρ is the load of the queue. The mean response time for all jobs, T_{RT}, is simply

$$T_{RT} = \int_0^\infty T_{PS}(t)dF(t), = \frac{E[X]}{1-\rho} \tag{2}$$

The task of Yaksha is to control T_{RT}, by controlling an acceptance probability p_a of requests to the system. We model the feedback control system using a fluid model.

The system model is depicted in Figure 1. Requests arrive to Yaksha with a mean rate $\lambda(t)$, and get modulated with an admission probability $p_a(t)$, with the Web server observing a mean arrival rate of $p_a(t) \cdot \lambda(t)$. The mean response time is then given by

$$T_{RT}(t) = \frac{E[X]}{1 - p_a(t)\lambda(t)E[X]}$$

We assume that there is an operating point p_0 of the system, that achieves $T_{RT} = T_{ref}$ for a given λ, where T_{ref} is our desired response time. We then linearize the response function about the operating point, and work with a small signal model. The linearization is a simple gain function, which is the derivative of the expression for the mean response time with respect to p_a, evaluated at p_0.

The linearized closed loop system model is then depicted in Figure 2. The PI controller has a transfer function in the frequency

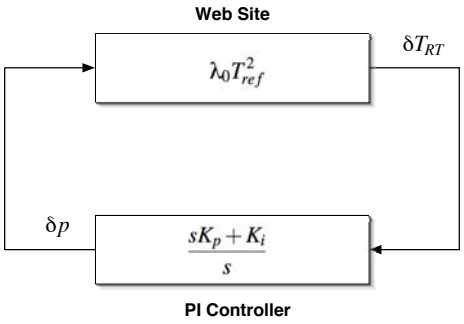

Figure 2: Small signal model for the feedback loop

.

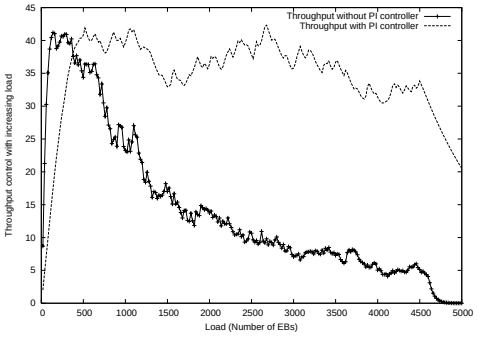

Figure 3: Throughput with increasing load

.

domain given by

$$PI(s) = K_p + \frac{K_i}{s}$$

We use the bilinear transform [1] to convert the transfer functions into digital form.

We assume a nominal λ_0 in the design process. However, the *effective* arrival rate of jobs that the system observes is $p_a\lambda$, where p_a is the admission probability that the controller computes. Hence, by observing p_a, assuming p_a converges, we can estimate the true arrival rate λ and then recalculate the controller parameters, thereby self-tuning the controller.

[4] gives a more detailed analysis of the model.

3. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We use tinyproxy v1.6.1 [3] and modify it to act as a controller. All HTTP requests from the client are directed towards this proxy, which then relays them to the Web and application server after the control decision. All responses from the Web and application server to the client also go through the proxy.

Our experimental testbed consists of five machines. Three of them are used as clients to generate HTTP requests. One machine is used as the Web and application server and another hosts the database server. The client machines drive the system with the Java TPC-W workload generator. We use Jakarta Tomcat v4.1.27 [6] as the Web and application server and MySQL v4.1.0-max-alpha [5] as the database server. The controller embedded inside tinyproxy resides on the machine hosting the Web and application server.

As client load increases, the throughput increases until the load reaches a threshold after which the throughput drops and response times grow. We show that the Yaksha controller is able to bound response times and maintain significant throughput levels even at excessive load levels.

Figure 3 shows how throughput varies with increasing load for both controlled and uncontrolled experiments. Note the downward trend for the uncontrolled case as overload takes its toll. The load increases from zero to 4500 EBs (Emulated Browsers).

Figure 4 shows the average response times of the servlets for increasing load for both the "controlled" and "uncontrolled" cases. As can be observed, Yaksha prevents server overload and so is able to effectively bound request response times while maintaining high throughputs. We set the reference value T_{ref} to be 150ms for the experiment, and that is the value Yaksha tries to control the running average of the response time.

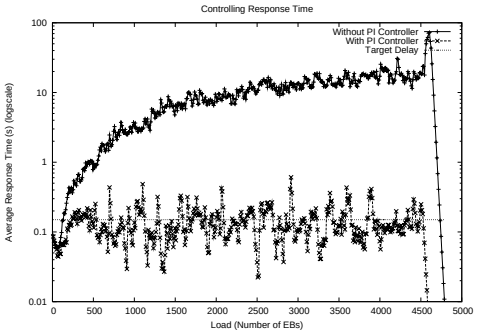

Figure 4: Response Time Control: Without the PI controller the response time can grow unbounded

.

4. SUMMARY AND CONCLUSIONS

We designed a completely self-tuning admission controller for 3-tiered websites based on classical control theoretic ideas. We implement our controller in the form of a proxy, Yaksha. By performing extensive experiments we show that Yaksha is able to bound response times for requests while maintaining a high throughput under overload. [4] contains a more detailed analysis of the model and extensive experimental results.

5. REFERENCES

[1] K. J. Åström and B. Wittenmark. *Computer Controlled Systems: Theory and Design.* Prentice-Hall, 1984.

[2] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti. Characterizing a java implementation of tpc-w. In *Third Workshop on Computer Architecture Evaluation Using Commercial Workloads*, Toulouse, France, January 2000.

[3] R. J. Kaes and S. Young. tinyproxy. http://tinyproxy.sourceforge.net/.

[4] A. Kamra, V. Misra, and E. Nahum. Controlling the performance of 3-tiered websites: Modeling, design and implementation, 2004. Technical Report, Department of Computer Science, Columbia University, http://www.cs.columbia.edu/ kamra/papers/control-3tier.pdf.

[5] MySQL. The MySQL database. http://www.mysql.com.

[6] The Apache Jakarta Project. Jakarta Tomcat servlet container. http://jakarta.apache.org/tomcat.