Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 1174–1188
computer communications
www.elsevier.com/locate/comcom

An observation-based approach towards self-managing web servers ☆

Abhishek Chandra[c,*], Prashant Pradhan[a], Renu Tewari[b], Sambit Sahu[a], Prashant Shenoy[d]

[a] IBM T.J. Watson Research Center, Hawthorne, NY 10532, USA
[b] IBM Almaden Research Center, San Jose, CA 95120, USA
[c] Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
[d] Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA

Received 29 June 2004; revised 28 June 2005; accepted 14 July 2005
Available online 15 August 2005

Abstract

As more business applications have become web enabled, the web server architecture has evolved to provide performance isolation, service differentiation, and QoS guarantees. Various server mechanisms that provide QoS extensions, however, rely on external administrators to set the right parameter values for their desirable performance. Due to the complexity of handling varying workloads and bursty traffic, configuring such parameters optimally becomes a challenge. In this paper, we describe an observation-based approach for self-managing web servers that can adapt to changing workloads while maintaining the QoS requirements of different classes. In this approach, the system state is monitored continuously and parameter values of various system resources—primarily the accept queue and the CPU—are adjusted to maintain the system-wide QoS goals. We implement our techniques using the Apache web server and the Linux operating system. We first demonstrate the need to manage different resources in the system depending on the workload characteristics. We then experimentally demonstrate that our observation-based system monitors such as workload changes and adjusts the resource parameters of the accept queue and CPU schedulers in order to maintain the QoS requirements of the different classes.

Keywords: Web server; Self-managing; Dynamic resource allocation

1. Introduction

1.1. Motivation

Current Web applications have evolved from simple file browsing to complex tools for commercial transactions, online shopping, information gathering and personalized service. To accommodate this diversity, Web servers have evolved into complex software systems. Web servers today perform a variety of tasks such as (a) dynamic HTML generation, (b) personalized page assembly using scripting languages (e.g. JSP), (c) SSL processing for secure transmission, (d) persistent HTTP protocol processing, to reduce connection setup over-heads and improve end-user performance, and (e) communication with the application server components via servlets. In doing so, the server interacts in complex ways with the underlying OS mechanisms that manage resources such as the CPU, memory, disk and the network interface. Another emerging trend is the growing popularity of Web hosting services that collocate multiple Web domains on the same host machine or a cluster and provide different levels of service to these domains based on various pricing options. In such environments, service differentiation and performance isolation become necessary for efficient operation.

Numerous mechanisms for service differentiation and performance isolation have been proposed in the literature. Such mechanisms for Web servers include QoS-aware extensions for admission control [1], SYN policing and request classification [2], accept queue scheduling [3], and CPU scheduling [4]. These mechanisms enable a Web server to differentiate between requests from different classes and provide class-specific guarantees on performance (for instance, by providing preferential treatment to

☆ An earlier version of this article appeared in the Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002), Miami Beach, FL, May 2002.

* Corresponding author.
E-mail addresses: chandra@cs.umn.edu (A. Chandra), ppradhan@us.ibm.com (P. Pradhan), tewarir@us.ibm.com (R. Tewari), ssahu@us.ibm.com (S. Sahu), shenoy@cs.umass.edu (P. Shenoy).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.07.003

users who are purchasing items at an e-commerce site over users who are merely browsing, or by providing better service to institutional investors over individual investors at a financial site). One limitation of these QoS mechanisms is that they rely on an external administrator to correctly configure various parameter values and set policies on a system-wide basis. Doing so not only requires a knowledge of the expected workload but also a good understanding of how various operating system and Web server configuration parameters affect the overall performance. Thus, while these QoS mechanisms undoubtedly improve performance, they also *exacerbate the problems of configuration and tuning*—each mechanism provides one or more tunable 'knobs' that the system administrator needs to deal with. More importantly, these mechanisms are not independent of one another—depending on the configuration, each mechanism can have repercussions on the behavior of others, which further complicates the configuration process. Furthermore, past studies have made contradictory claims about the utility and benefits of these mechanisms. For instance, one recent study has claimed that the (socket) accept queue is the bottleneck resource in Web servers [3], while another has claimed that scheduling of requests on the CPU is the determining factor in Web server performance [4]. Thus, it is not evident a priori as to which subset of QoS mechanisms should be employed by a Web server and under what operating regions.

The increasing complexity of the Web server architecture, the dynamic nature of Web workloads [5,6], and the interactions between various QoS mechanisms makes the task of configuring and tuning modern Web servers exceedingly complex. It has been argued that the more complex the system, the greater are the chances of a misconfiguration and sub-optimal performance [7,8]. To address this problem, in this paper, we develop an adaptive architecture to make Web servers *self-managing*. By self-managing, we mean mechanisms to automate the tasks of configuring and tuning the Web server so as to maintain the QoS requirements of the different service classes. The emphasis on manageability of computing systems has gained momentum in recent years with the ever increasing complexity of these systems—in fact, several researchers have argued that, in today's environments, the problems of manageability, availability and incremental growth have overshadowed that of the traditional emphasis on performance [9,10].

1.2. Research contributions

This paper focuses on the architecture of a self-managing Web server that *supports multiple QoS classes*—a scenario where multiple virtual servers run on a single physical server or where certain classes of customers are given preferential service. Assuming such an architecture, we make three key contributions in this paper. (1) We conduct an experimental study using the Apache Web server to identify bottleneck resources for different Web workloads; our study illustrates how the bottleneck resource can vary depending on the nature of the workload and the operating region. (2) Based on the workloads in our study, we identify a small subset of resource control mechanisms—the incoming request queue scheduler and the CPU share-based scheduler—that are likely to provide the most benefits in countering the performance degradation. (3) We then present an observation-based technique to automate the tasks of configuring and tuning of the parameters of these OS mechanisms. A key feature of this technique is that it can handle multiple OS resources in tandem. Our architecture consists of techniques to monitor the workload and to adapt the server configuration based on the observed workload. The adaptation system can adjust to: (i) a change in the request load, (ii) the QoS requirements of the classes, (iii) the workload behavior, and (iv) the system capacity. Since the system dynamically monitors and adjusts the parameters it makes no underlying assumption of the workload characteristics and the parameter behaviors.

We implement our techniques into the Apache Web server on the Linux operating system and demonstrate its efficacy using an experimental evaluation. Our results show that we can adjust dynamically to a change in workload, a change in response time goal and a change in the type of workload.

The rest of this paper is structured as follows. Section 2 presents our experimental study to determine the bottlenecks in the Apache request path. Section 3 discusses the architecture and kernel mechanisms used to support multiple classes of Web requests. Section 4 presents our framework to configure and tune the Web server. Section 5 presents the results of our experimental evaluation. Section 6 discusses related work, and finally, Section 7 presents our conclusions.

2. Analyzing the bottlenecks in web request processing

In this section, we examine the bottlenecks encountered in the processing of Web requests. We use Apache as a representative example of a Web server and subject it to a variety of different workloads. For each workload, we determine the bottlenecks in the request path at different operating regions. In what follows, we first present a brief overview of the software architecture employed by Apache before presenting our experimental results.

2.1. Architecture of the Apache Web Server

Apache employs a process-based software architecture. Apache spawns a pool of child processes at startup time, all of which listen on a common socket (typically, port 80). A newly arriving request is handed over to one of the children for further processing; the process rejoins the pool after it is done servicing the request and waits for subsequent