Performance Modeling of a Web Server with a Dynamic Pool of Service Processes

Tien Van Do, Ram Chakka, Thang Le Nhat, Udo R. Krieger [1]

Abstract

This paper considers the multi-processing module *Prefork* in the Unix version of Apache, one of the most widely applied software solutions for Web servers in todays Internet. It analyzes its dynamic pool size behavior. For the first time, we propose a queueing model to approximate the performance of the latter. Numerical results computed by advanced matrix-analytic methods are compared with measurement results. They clearly indicate that the proposed analytic model can indeed accurately predict the performance of the dynamic pool size behavior of an Apache Web server.

Key words: Web server performance, queueing network model, QBD-M processes

1 Introduction

Previous analytic studies of a Web server have not sufficiently considered the internal structure and operation of its software architecture, which has a crucial impact on the performance. In particular, the dynamic behavior of the available number of operating processes (or threads) serving HTTP requests inside the software architecture plays a critical role. However, so far all presented analytic performance models are not able to capture explicitly the dynamic pool size behavior that handles the concurrency of HTTP requests.
We propose a new queueing model with a variable number of servers to describe the dynamic pool size implemented in an Apache Web server. Applying a batch Markovian arrival process and advanced matrix-geometric solution techniques, results on the system performance are derived and compared with actual measurements.

2 Analysis of a Web server with dynamic pool size

We consider the UNIX software architecture of an Apache Web server, in particular, the operation of the Apache multi-processing module (MPM) and the dynamic pool of the actually available processes serving HTTP requests (cf. [7]). The introduction of the MPM along with the dynamic behavior of the pool size efficiently supports the *stable and scalable* operation of an Apache software architecture and constitutes currently an important topic regarding the optimal resource allocation (cf. [1], [5]). Here, we focus on the MPM non-threaded *Prefork* processing module and try to capture the dynamic behavior of the pool size under various workload conditions by an analytic model.

[1]Tien Van Do is with the Department of Telecommunications, Budapest University of Technology and Economics. Ram Chakka is with Rajeev Gandhi Memorial College of Engineering & Technology (RGMCET), Nandyal, India. Thang Le Nhat is with Post and Telecommunications Institute of Technology, Hanoi, Vietnam. Udo Krieger (`corresponding author`) is with Otto-Friedrich-Universität, Feldkirchenstr. 21, D-96045 Bamberg, Germany, Email: `udo.krieger@ieee.org` .

2.1 Operation of an Apache Web Server with dynamic pool size

At the beginning of the Web server operation, the Apache 2.0 MPM Prefork running on UNIX starts a single control process that is responsible for launching child processes to handle the incoming or waiting HTTP service requests. Hence, the number of available active or idle HTTP service processes depends on the offered load of HTTP requests that are transported by TCP connections (sessions) to the Web server. Thus, it dynamically varies with the load of the TCP session requests. It can be controlled by the following three variables, called directives (cf. [1]):

- The MaxClients directive limits the maximal number of simultaneously served requests, i.e. the maximal number of operating (active or idle) service processes to N. Any further attempt exceeding the MaxClient limit will normally be queued in the Listen Queue up to a number (LB) based on the ListenBacklog directive.
- The MaxSpareServers directive determines the desired maximum number h_{max} of idle child server processes. If the number of idle child server processes exceeds MaxSpare-Servers (h_{max}), then the control process will initiate a killing of the idle processes at a predetermined rate ϵ.
- The MinSpareServers directive determines the desired minimum number h_{min} of idle child processes. If the number of idle child server processes falls below h_{min} and the number of active and idle processes in the system is less than N, then the parent process creates new child processes at a predetermined rate η.

Thus, in the server the MPM is responsible to keep the number of idle child processes less than h_{max} and greater than h_{min} at any given time to serve the incoming HTTP requests.

2.2 Modeling and analysis of a non-threaded Web server at the page level

Considering the operation of the Web server at the page level, we conclude that multiple HTTP requests can be transmitted over a single TCP connection between a client and the server. A new TCP connection requires one server process (either from the idle processes or a newly created one) to serve a new HTTP request on downloading a page and subsequent HTTP requests to load its embedded objects can arrive in the case of a persistent connection. A TCP connection (session) may be initiated when a client starts to download a new object from a Web server if a previously opened TCP connection has been closed.
We primarily focus on the workload model at this TCP session level and use a generalization of the associated WAGON HTTP 1.0 traffic model (cf. [5]). To predict the performance of the dynamic pool size of the available HTTP service processes, we have extended a convenient queueing model fed by these processing requests (cf. [7], [8]). As illustrated by Figure 1, it takes into consideration the impact and interaction of other components such as the client population, the TCP transport protocol stack, as well as the software and resource contention level of a Web server.
In the queueing system of the Web server at the software level we primarily consider the arrivals of initial HTTP requests by TCP sessions which require the service of an idle or new worker process. The HTTP requests compete for worker processes controlled by the Web server software. A request may wait in the software queue until a service process becomes available to handle the request. At the physical level, these processes may wait to use any of

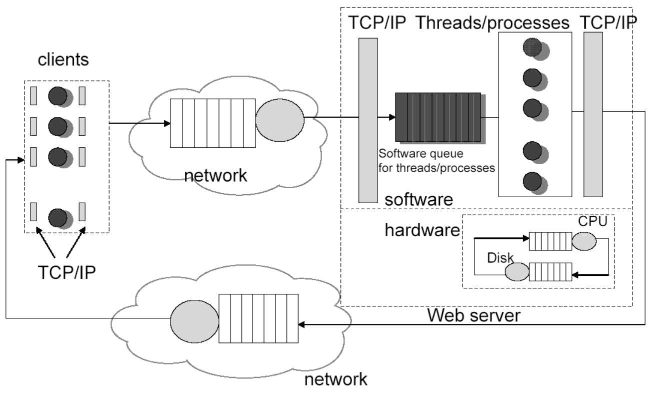

Figure 1: Components of a Web server model with a variable number of service processes

the Web server's physical resources (CPUs and disks). Thus the associated service times of sessions must take into account the interaction of other HTTP requests.
To generate a mathematically tractable model, we propose the following Markovian approach based on a system decomposition of the Web server queue:

- As workload model we use a generalization of the WAGON and assume that the time between successive requests are iid random variables with a Generalized Exponential (GE) distribution and the parameters σ and θ (that means the arrival process of requests is compound Poissonian) (cf. [4], [5]).

- The service time of requests is a result of the contention among the physical resource of the Web server by other concurrently served requests and the interaction with the TCP flow-control algorithm. Following Menasce et al. [6], the service time is approximated by the sojourn time in a closed Queueing Network (CQN) describing the contention among the physical resources of the Web server. The CQN has been analyzed by Mean Value Analysis to determine the relevant parameters including the throughput $X_p(k)$ of a server when there are k requests and p processes in the system.

- A two-dimensional Quasi Simultaneous-Multiple Births-and-Deaths (QBD-M) process (Z) is used to describe the varying number of HTTP service processes at the software level (cf. [3]).

Hereby, we have assumed that the invocation times between the successive creations or terminations of service processes are iid random variables and exponentially distributed with parameters η and ϵ, respectively. It is shown that the combination of the available number $I(t)$ of active or idle service processes and the actual number $0 \leq J(t) \leq L$ of session requests that are served or queued in the process queue of the Web server at time t create a continuous-time Markov chain $Z = \{[I(t), J(t)]; t \geq 0\}$ with a discrete state space $\{1, \ldots, N\} \times \{0, \ldots, L\}$ on a rectangular lattice strip. It belongs to the class of level-dependent Quasi Simultaneous-Multiple Births-and-Deaths (QBD-M) processes (cf. [3]) and its transition behavior can be explicitly specified.
Applying a computationally efficient advanced matrix-analytic method, the solution of the overall queueing model with batch Markovian arrivals and a variable number of servers has

Table 1: Average number of idle processes

σ	$\theta = 0.5$, $\eta = 0.000898445$, $\epsilon = 0.000608624$		$\theta = 0.7, \eta = 0.00607108$ $\epsilon = 0.00559304$		$\theta = 0.9, \eta = 0.0540266$ $\epsilon = 0.0526692$	
3.0	Model	Measurement	Model	Measurement	Model	Measurement
	9.981257	9.96753	9.947323	9.92116	9.688130	9.60742
σ	$\theta = 0.5$, $\eta = 0.0011335$, $\epsilon = 0.000680102$		$\theta = 0.7, \eta = 0.00817092$ $\epsilon = 0.00742811$		$\theta = 0.9, \eta = 0.0840194$ $\epsilon = 0.0819702$	
5.0	Model	Measurement	Model	Measurement	Model	Measurement
	9.969591	9.91731	9.910794	9.84428	9.471635	9.40582
σ	$\theta = 0.5$, $\eta = 0.00273584$, $\epsilon = 0.0019068$		$\theta = 0.7, \eta = 0.0149412$ $\epsilon = 0.0136071$		$\theta = 0.9, \eta = 0.162362$ $\epsilon = 0.158922$	
10.0	Model	Measurement	Model	Measurement	Model	Measurement
	9.940920	9.90265	9.846855	9.83912	9.25642	9.17

been obtained. It generates the required performance metrics of the system including the mean number of idle processes, the utilization and throughput of the Web server as well as the corresponding server-side response time characteristics of requests.

To validate the proposed modeling approach, a measurement configuration has been set up. The corresponding workload of an Apache Web server is generated by the traffic generator `ab` [2] and the major statistics, e.g. the creation and killing time, the idle and busy time of each HTTP process are captured.

In Table 1, we show the results of the average number of idle processes versus σ and θ. They justify our claim that the proposed mathematically tractable Markovian model can be used to predict accurately the performance of the corresponding MPM prefork in an Apache server although the creation and termination of service processes do not guarantee the assumed Markovian property in practice. Moreover, we have studied other features of Web server characteristics such as the relationship of the average number of idle processes and the killing and creation rates ϵ and η that are relevant to allocate processing resources efficiently.

3 Conclusions

We have proposed a queueing model with a variable number of servers to describe the dynamic pool size behavior implemented in the MPM Prefork worker module of a non-threaded Apache Web server. Applying a batch Markovian arrival process and advanced matrix-geometric solution techniques, results on the performance of the worker process have been derived and compared with actual measurements of the dynamic pool size behavior. The latter justify and validate the proposed modeling and analysis approach.

In conclusion, we believe that our matrix-analytic approach can also be used to evaluate other Web server software architectures such as the Apache 2.2 hybrid multi-threaded multi-process MPM worker module which is the subject of current research.

Bibliography

[1] Apache Web Server, “http://www.apache.org”.

[2] ab – Apache HTTP server benchmarking tool, “http://httpd.apache.org/docs/2.2/programs/ab.html”.

[3] R. Chakka and T. V. Do, “Some new Markovian models for traffic and performance analysis in telecommunication networks, Tutorial Paper,” in *Proceedings of the Second International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs 04)*, D. D. Kouvatsos, Ed., Ilkley, UK, July 2004, pp. T6/1–31.

[4] D. Kouvatsos, “Entropy maximisation and queueing network models,” *Annals of Operations Research*, vol. 48, pp. 63–126, 1994.

[5] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, “Online Response Time Optimization of Apache Web Server,” in *Eleventh International Workshop on Quality of Service (IWQoS 2003)*, 2003, pp. 461–478.

[6] D. A. Menasce, R. Dodge, and D. Barbara, “Perserving QoS of E-commerce Sites Through Self-Tunning: A Performance Model Approach,” in *Proceedings of the ACM Conference on E-Commerce, Tampa, Florida, USA*, October 14-17 2001, pp. 224–234.

[7] D. A. Menasce, “Web Server Software Architectures,” *IEEE Internet Computing*, vol. 7, no. 6, pp. 78–81, November/December 2003.

[8] P. Reeser and R. Hariharan, “Analytical Model of Web Servers in Distributed Environments,” in *Proceedings of the 2nd ACM International Workshop on Software and Performance WOSP’2000, Ottawa, Canada*, Sept. 17-20 2000, pp. 158–167.