Performance modeling of an Apache Web server with a dynamic pool of service processes

Tien Van Do · Udo R. Krieger · Ram Chakka

Abstract

In the current Internet the performance of service delivery crucially depends on the proper and efficient operation of Web servers. It is determined by their software architecture and characterized by the applied processing model.

Here we consider the Unix software architecture of an Apache Web server with its non-threaded multi-processing module *Prefork*. We propose a tractable multi-server model to approximate the performance of the load-dependent dynamic behavior of Apache's resource pool of available HTTP service processes, which has not been done before. Furthermore, we show that this Markovian queueing model can be solved by advanced matrix-geometric methods. Then the efficiently computed performance results of this analytic model are compared with measurements of a real Apache Web server. The outcome clearly indicates that our analytic model can very accurately predict the mean-value performance of Apache under the Prefork policy.

U.R. Krieger acknowledges the support by the EU IST-FP6 NoE project "EuroNGI/EuroFGI".

T.V. Do
Department of Telecommunications, Budapest University of Technology and Economics, Magyar tudósok körútja 2, 1111 Budapest, Hungary
e-mail: do@hit.bme.hu

U.R. Krieger (✉)
Faculty WIAI, Otto-Friedrich-Universität, 96045 Bamberg, Germany
e-mail: udo.krieger@ieee.org

R. Chakka
Meerut Institute of Engineering and Technology (MIET), Meerut 250005, India
e-mail: ramchakka@yahoo.com

Keywords Web server modeling · Queueing network model · QBD-M process · GE distribution

1 Introduction

Considering the service infrastructure of the current Internet, Web servers play a dominant role. It is their main task to receive and process requests of clients demanding specific objects from the servers and to return them by related responses. The associated request and response messages are transported by the Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) using TCP connections between the clients and the server. The resulting performance of the service delivery crucially depends on the proper and efficient processing of these tasks.

Regarding the software design of the HTTP server the concurrency of connection requests poses a critical issue. To tackle it, two orthogonal components have been implemented in the Web server software architecture on the host machine: the processing model and the dynamic size of a resource pool of instantiated parallel processes serving the requests (cf. [23]). Considering the processing model of the software architecture of a HTTP server, there are several choices, viz. a process based, a thread based, or a hybrid model which is a combination of the former two (cf. [13]). The behavior of the pool size is characterized by a fixed or varying number of concurrent HTTP processes (or threads) to serve incoming connection requests and, thus, determines the performance of the server. When a dynamic behavior of the pool size is incorporated into the server software, the number of processes (or threads) can vary depending on the load of the offered connection requests.

Nowadays, the Unix version of the Apache Web server constitutes one of the most frequently used software so-

lutions of the Internet. In Apache 2.0 the introduction of the multi-processing module (MPM) combined with the dynamic pool size efficiently supports a stable and scalable operation. Moreover, it is currently considered as an important topic regarding the optimal resource allocation (cf. [11, 23]).

Therefore, we need to understand both the performance of different software solutions of a Web server, and the dynamic behavior of the pool size subject to various workload conditions. Slothouber [19] has proposed an open queueing network to model a Web server and predicted the response time versus the load. Dilley et al. [7, 8] have used layered queueing models regarding the performance evaluation of a Web server in the Internet and Intranet. Squillante et al. [20, 21] developed a Web traffic model using the access logs of the Web site of the Winter Olympic Games in Nagano, Japan, 1998. To analyze the tail behavior of the request latency, the authors fed these traffic processes into a Web server modelled by a $G/G/1$ queue. Reeser and Hariharan [18] have presented an analytic queueing model of Web servers in a distributed environment. They have argued that a Poisson assumption regarding the process of HTTP (object) requests is reasonable. They have also shown that the performance predictions of their analytic model match well with results obtained by simulations. In Cao et al. [3] a queueing model $M/G/1/K^*$PS has been proposed to predict the performance metrics of a Web server in terms of the average response time, throughput and blocking probability. Recent work of the same authors [1] was carried out assuming that the arrival stream of HTTP requests forms a Markov-Modulated Poisson Process (MMPP).

However, all these previous investigations of a Web server have not studied in more detail the internal software architecture and the dynamic behavior of the number of available service processes and their impact on major performance indices applying analytic means.

Menasce [13] has been one of the first who considered the software architecture of a Web server. In his performance evaluation approach a queueing network (QN) models the physical resources of the Web server and the software architecture of Apache is represented by a Markov chain. Liu et al. [11] have recognized the importance of the Apache directive *MaxClients* as a tuning parameter and proposed an online optimization to determine an optimal operating point. However, so far all presented models have not been able to capture explicitly the dynamic behavior of the pool size that handles the concurrency of requests. Moreover, they cannot cope with simultaneous arrivals of multiple requests arising from the client population.

In this paper we consider the non-threaded multi-processing module (MPM) *Prefork* of Apache's UNIX architecture. For the first time, we propose a mathematically tractable analytic queueing model to predict the performance of an Apache server with its load-dependent dynamic pool size. It is derived from certain Markovian assumptions concerning the number of active service processes and applies a decomposition approach to the client population, the Web server and the TCP transport system. In particular, we suppose that the arrival process of TCP connection requests demanding the support of a HTTP service process follows a Markov-Modulated Compound Poisson Process (MMCPP). Here the inter-arrival times are governed by Markov-Modulated Generalized Exponential (MMGE) distributions. We use the MMCPP because it can cope with the fluctuations of the arrival rate, the autocorrelation of inter-arrival times of requests and also with simultaneous arrivals of multiple requests. As a consequence the well-known WAGON traffic model of Liu et al. [12] is a special case of our more general workload model.

In addition to the development of a tractable analytic model we have also measured the performance of a real Apache Web server where the number of service processes does not follow a Markovian property. To generate the workload as a generalization of the WAGON traffic model (cf. [11]), either the benchmarking program `ab` (cf. [22]) or `httperf` (cf. [16]) can be used. Comparing the numerical results of the analytic model with those obtained by the actual measurements, we have realized a rather good coincidence of mean values and an acceptable accuracy of our predictions regarding control purposes. Moreover, the results are used to identify the impact of major control parameters of an Apache configuration on relevant performance measures.

Since we are only dealing with non-threaded multi-processing, modeling the thread-based Apache multi-processing module winnt[1] used by Microsoft Windows operating systems is out of the scope of our study. However, by appropriate modifications the presented methodology can be used to analyze this Apache MPM winnt as well.

The rest of the paper is organized as follows. In Sect. 2 we provide an overview of the operation of the Apache MPM *Prefork* with dynamic pool size. In Sect. 3 the proposed queueing model and its analysis are introduced. Section 4 presents some performance results and the validation of the proposed model by measurements. Finally, some conclusions are drawn in Sect. 5.

2 Non-threaded multi-processing operation of an Apache Web server with dynamic pool size

In this section we sketch the operation of an Apache 2.0 Web server that implements the non-threaded multi-processing

[1] Winnt has a single control process which gives birth to a single child process. The latter creates in turn threads to handle HTTP requests.

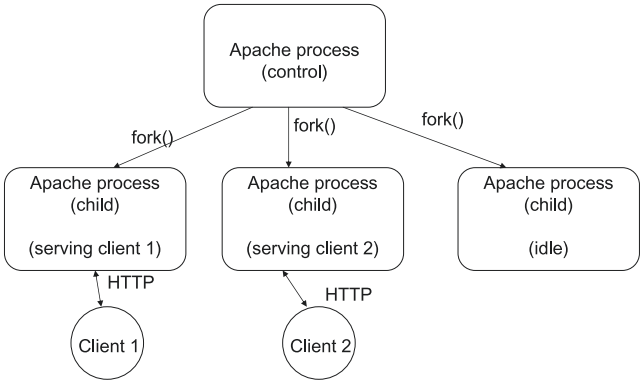


Fig. 1 MPM prefork operation of an Apache Web server with a Unix operating system

module (MPM) *Prefork*. At the beginning of the operation, the Apache 2.0 running on top of UNIX starts a single master control process. The latter is responsible for launching child processes to handle the incoming or waiting HTTP service requests (see Fig. 1). The Web server listens for requests at the well-known TCP port number 80. Web browsers, i.e. the clients, are normally used to communicate with the Web server. When a client opens or clicks on the first URL during a specific session of the Web browser, the set-up of a TCP connection is initiated and one of the following three alternatives may happen (see Fig. 1):

1. One idle child process of the Apache HTTP server is allocated to handle the incoming TCP connection and the associated HTTP requests conveyed by that connection.
2. The TCP request waits in the software queue for an idle process and will get service either after the MPM has successfully launched an idle child process or when a child process has completed the service of another request.
3. The TCP connection and associated HTTP request is blocked if the Listen Queue[2] of the server system is full. The client receives a message that the Web server is not available. In our model the request is considered to be lost.

If a free child process is allocated, the related service process locates the Web page and its referenced resources as requested objects. It normally consists of a single HTML file, an HTML file with several inline images or other items. An object may be a static HTML file stored at the Web server or may be dynamically generated from databases by a special script engine. An obtained object is encapsulated as HTTP response message and handled by the TCP/IP stack. The latter encapsulates and segments the message into TCP/IP packets, and sends the resulting packet stream through the already opened TCP connection from the HTTP server to the browser. The service time of a HTTP request, i.e. the interval between the arrival time of the request at the HTTP server and the completion instant of the initiated processing and its resource allocations, i.e. the tear-down of the TCP connection and the release of the related HTTP service process, constitutes a basic performance index of the system. Obviously, the service time of a specific request and, hence, the overall response time of the HTTP server depend in a very complex manner on a number of different items, such as the physical resources of the operating machine with regard to CPU power and disks' operation, the number of other concurrently served requests competing for these physical resources of the Web server as well as the network conditions influencing the TCP flow-control algorithm.

In the Unix version of Apache 2.0, the number of available busy and idle service processes varies dynamically in terms of the load of TCP session requests. It creates a dynamic resource pool of HTTP service processes whose size is controlled by the following three main control parameters of the server software, called *directives* (cf. [23]):

- The *MaxClients* directive limits the maximal number of operating busy and idle HTTP service processes, therefore it sets a threshold N for the maximal number of simultaneously served requests. Any further connection attempt exceeding the MaxClient limit will normally be queued in the Listen Queue based on the *ListenBacklog* directive. Once a child process is released from serving a previous connection request, a new connection from the queue will be served in FCFS order.
- The *MaxSpareServers* directive determines the desired maximum number $h_{\max}$ of idle child service processes which are not handling a connection request. If the number of idle child processes exceeds MaxSpareServers $h_{\max}$, then the control process will kill 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 processes drops below $h_{\min}$ and the number of busy and idle processes in the system is less than N, then the parent process creates new child processes at a predetermined rate η.

  However, when the number of processes reaches the limit MaxClients (N), e.g. due to heavy traffic, and the number of idle process is smaller than $h_{\min}$, no creation of a new child process is allowed.

In summary, it is the rationale of this design to serve the incoming HTTP requests in an efficient way saving the spare resources of the operating system. Considering the number of idle child processes in the host machine of the Web server, the MPM Prefork module tries as effectively as possible to keep this number at any given time less than or equal to $h_{\max}$ and greater than or equal to $h_{\min}$.

[2]Note that the terms "software queue" and "Listen Queue" are interchangeable throughout the paper.

3 Modeling and analysis of an Apache Web server with the MPM strategy prefork

3.1 Hierarchical workload modeling of a Web server

Following the approach of [6] the workload of a Web server can be described by a hierarchically layered model. It is generated by traffic streams of a client population and depicted in Fig. 2. The associated processes that characterize the traffic patterns arising from the activity of a Web user and their corresponding time scales are related to the following levels:

- *Level of Web requests by users*: The behavior of a Web user can be characterized as an ON-OFF process. During the HTTP ON state, called activity period, the user is downloading a specific Web page and waiting for its presentation. The OFF state, called think time, simply describes the following silence period. Here the user is not sending any requests, only viewing and perhaps reading the downloaded document. During the OFF state no user traffic is generated at the other traffic layers beneath.
- *Level of Web pages with their corresponding objects*: A requested Web page with its objects denotes a HTML file and the referenced inline objects, e.g. images, included in such a HTML document.
- *Level of TCP sessions*: When the user clicks on a URL reference or enters a URL into the command line of the Web browser, the client first opens a TCP connection to the specified Web server. After the TCP connection is established, the client sends a HTTP request in the HTTP protocol format typically by one TCP segment, that is encapsulated in one IP packet, over the already opened TCP connection to the HTTP server. The request normally specifies which HTTP version the client is using. If it is HTTP/0.9 or HTTP/1.0, the server will automatically work in the transitory connection mode, will only send one reply and then close the connection. If it is HTTP/1.1, it is assumed that a persistent connection is desired. This means that multiple HTTP requests can be sent through the opened TCP connection.
- *Packet level*: The HTTP object request and response messages for HTML files or object files and their referenced inline images etc. are encapsulated into TCP packets and segmented into a stream of IP packets. The resulting TCP/IP packet stream is transmitted along a path between the server and the requesting client inside the Internet and the transport is governed by the TCP congestion-control mechanism.

In this paper we primarily focus on the workload modeling at the TCP session level. Liu et al. [11] have proposed a related stochastic model (called WAGON) of the generated HTTP traffic between a specific user and a Web server at this time scale. The WAGON model can be described by a marked point process where the arrival times correspond to the start times of sessions followed by a cluster of further object requests and think times (see Fig. 3). We follow this line of reasoning and subsequently extend the corresponding approach.

3.2 Resource contention during Web server operation

Considering a scenario of a single Web server and multiple clients, contention about its resources may happen at the following two traffic layers (see Fig. 4):

- *TCP/IP flow layer*: Flows of IP packets encapsulating TCP segments that are carrying HTTP protocol requests of clients and responses of the Web server compete with other traffic streams in the switching nodes and on the links of the underlying transport network.
- *TCP session layer*:
  - *Child process level of the Web server*: At the child process level of the Web server (called software level), HTTP requests compete for such service processes controlled by the related software components of the server. An arriving request may wait in a Listen Queue

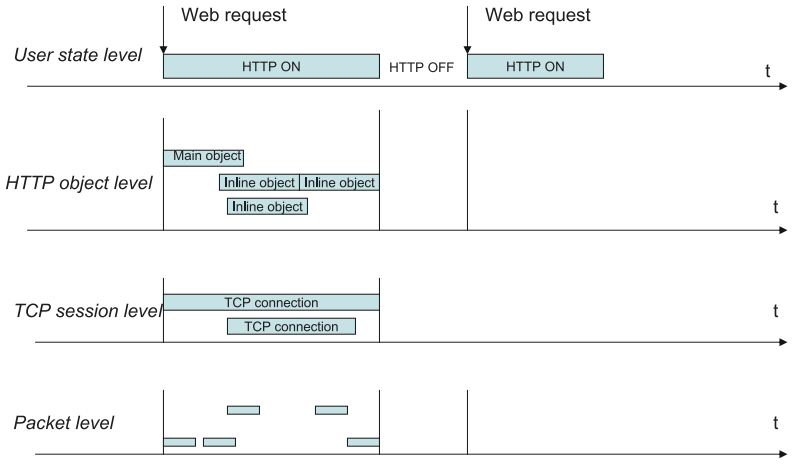
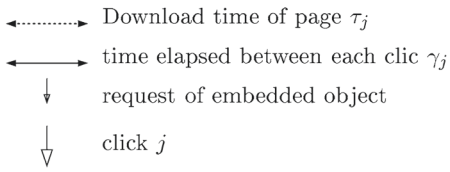


Fig. 2 Hierarchical workload modeling of Web traffic



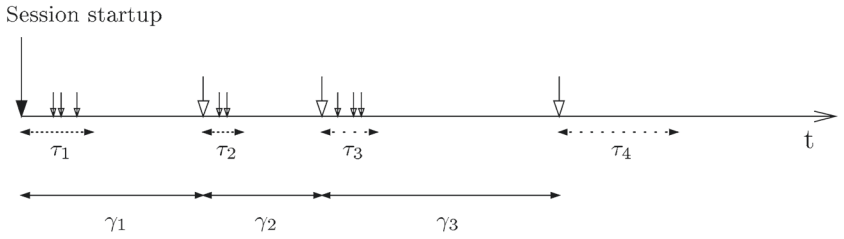


Fig. 3 The WAGON traffic model

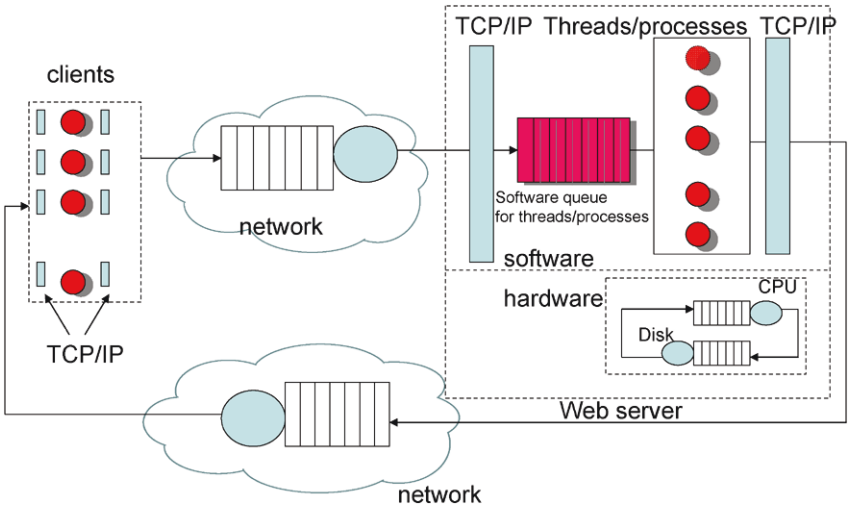


Fig. 4 Resource contention and queueing in the operation of a Web server

for an available service process that can handle the request regarding a demanded page and its objects.

– *Level of physical resources*: At the kernel level of the host machine running the Web server (called hardware level), processes may wait to use any of the server's physical resources, i.e. CPU power and disk processing of the underlying machine etc. (see Fig. 4).

All these contentions have an impact on the performance experienced by the Web clients of the users. As a consequence of these mutual interactions of the corresponding queues at two different time scales, a queueing model which takes into consideration all these factors, such as the arrivals of requests, the transmission of every TCP/IP packet governed by TCP congestion control, contentions both in the transport network and about the physical resources of the server etc., becomes mathematically intractable. Therefore, we will study the Web server in isolation and try to model its offered workload in an appropriate manner (see Fig. 4).

3.3 The Markovian performance model

Considering the operation of a Web server at the page level in more detail, we can make the following observations:

– A TCP connection (i.e. a session) may be initiated when a client starts to download a new page and its objects from the Web server if a previously opened TCP connection has been closed.
– Multiple HTTP object requests can be transmitted over a single TCP connection between a client and the Web server.
– One new TCP connection requires a HTTP service process either from the group of idle processes or it is a newly created child process. It serves the new HTTP request to download a page and subsequent HTTP requests to further transfer its embedded objects in the case of a persistent connection.

To predict the performance of the dynamic pool size of HTTP service processes in the UNIX operating system of an

Apache Web server, we have extended a convenient queueing model fed by these processing requests (cf. [13, 18]). As illustrated by Fig. 4, it takes into consideration the impact and interaction of other components such as the client population, the transport protocol stack of TCP, as well as the software and resource contention level of a Web server. In the queueing system of the Web server at the child process level we primarily consider the arrivals of initial HTTP requests by TCP sessions which require the service of an idle or a new service process. Here all HTTP requests compete for service processes controlled by the software of the Web server. A request may wait in the software queue, i.e. Listen Queue, until a service process becomes available to handle it. At the physical level these processes may wait until they can use any of the server's physical resources, i.e. CPUs and disks. Thus, the associated service time of a specific HTTP request is strongly influenced by the interaction of other HTTP requests.

To generate a mathematically tractable model, we propose a system decomposition of the queueing network describing the Web server (see Fig. 4). Our approach is based on the following Markovian workload model.

3.3.1 Arrival process of HTTP requests

Considering multiple simultaneous arrivals of HTTP requests to the Web server arising from the client population, we use the generalization of the WAGON traffic model at the TCP session level. Therefore, we assume that the arrival process of TCP connection requests demanding the service of child processes follows a Markov-Modulated Compound Poisson Process (MMCPP). Its inter-arrival times are governed by a Markov-Modulated Generalized Exponential (MMGE) distribution.

The arrival process is modulated by an irreducible continuous-time Markov chain X with M states called phases of modulation. Let Q_X denote its generator matrix. Then an off-diagonal element $Q_X(i,k) = qx_{i,k}$, $i \neq k$ describes the instantaneous transition rate from phase i to phase k, and the ith diagonal element is given by $Q_X(i,i) = -qx_i = -\sum_{l=1}^{i-1} qx_{i,l} - \sum_{l=i+1}^{M} qx_{i,l}$. Let Ω_X denote the off-diagonal matrix of the transition rates of process X defined by $\Omega_X(i,k) = Q_X(i,k)$, $i \neq k$, $\Omega_X(i,i) = 0$, $1 \leq i \leq M$. Let $I_1(t)$, $1 \leq I_1(t) \leq M$, represent the phase of the modulating process X at any time t.

The arrivals in each modulating phase follow a Compound Poisson Process (CPP) process (cf. [9]). Strictly speaking, during the modulating phase i, $1 \leq i \leq M$, the probability distribution function of the inter-arrival times τ_i is governed by a Generalized Exponential (GE) distribution $\mathbb{P}(\tau_i = 0) = \theta_i$ and $\mathbb{P}(0 < \tau_i \leq t) = (1-\theta_i)(1-e^{-\sigma_i t})$ with the associated parameters (σ_i, θ_i). Note that the GE distribution is the least biased distribution with two parameters which can approximate other distributions by matching the first two moments (cf. [9]).

Thus, the point process of session arrivals during a given modulating phase can be considered as a batch Poisson process with a geometric batch size distribution, i.e. during phase i size $s \geq 1$ happens with probability $(1-\theta_i)\theta_i^{s-1}$.

3.3.2 Service time of HTTP requests

The service time of HTTP requests is a result of the complex competition for physical resources of the Web server, the interactions with all HTTP requests concurrently served by other service processes and the TCP flow-control algorithm. Thus, it strongly depends on the contention for physical resources of CPUs and disks in the host machine. To approximate the latter a closed queueing network (CQN) is used. It has been validated by Menasce et al. [14] in their experimental test-bed that such a CQN model is accurate enough to estimate the response time of a HTTP server.

Let $T_p(k)$ denote the throughput of the Web server when there are k requests and p processes in the system (note that $T_p(k) = T_p(p)$, if $k \geq p$, i.e. there is no idle process in the system). Following the approach of Menasce [13] and describing the physical resources of the Web server by a CQN (see Fig. 4), we approximate the service time by the mean sojourn time of a request in this CQN. In this manner the associated conditional service rates of an equivalent server are determined by $T_p(k)$.

3.3.3 Modeling the varying number of the HTTP service processes

We introduce an integer-valued random variable $I_2(t)$ to describe the number of busy and idle HTTP service processes at time t (it does not include the master process in the case of Apache 2.0). $I_2(t)$, $1 \leq I_2(t) \leq N$, varies due to the creation of a new service process by forking of the Apache master process or due to a process cancellation by killing an existing idle child process. When the number j of concurrent connection requests is larger than $I_2(t)$, the excess of requests beyond $I_2(t)$ resides in a software queue. We assume that at any time the number j of connection requests in the Web server is bounded by the fixed value L. It means that the software queue is resized accordingly to keep the maximum number of connection requests in the system equal to L. Note that this assumption is quite reasonable for the software of the Web server because this behavior guarantees the fairness for those requests arriving after different periods of low and high traffic load.

We model the changing number $I_2(t) = i_2$ of child processes that are available in the Web server to process the HTTP requests by $N \times N$ rate matrices U_j, $j = 0, 1, \ldots, L$. They represent the applied server strategy to regulate this

number. To capture correctly the dynamics of this mechanism, that handles the idle service processes in the Apache MPM Prefork, the U_j's are constructed as follows:

- if $i_2 - j < h_{\min}, \forall j = 0, \ldots, L, i_2 = 1, \ldots, N-1$ then $U_j(i_2, i_2+1) = \eta$,
- if $i_2 - j > h_{\max}, \ \forall j = 0, \ldots, L, i_2 = 2, \ldots, N$ then $U_j(i_2, i_2-1) = \epsilon$,
- otherwise $U_j(i_2, k) = 0$.

Thus, the number of idle processes is kept smaller than or equal to $h_{\max} \geq 1$ and larger than or equal to $h_{\min}$. In state $i_2 = N$ a creation and in state $i_2 = 1$ a cancellation of a service process is not possible.

Here, we assume that the times between successive creations or terminations of HTTP service processes are generalized exponentially distributed and, hence, $I_2(t)$ is a Markov chain. As a consequence we propose a mathematically tractable performance model of the software architecture of the Web server.

We will show in the subsequent Sect. 4 that despite this approximation our computationally efficient model can successfully and rather accurately predict the performance of the real operation of an Apache server. There the times between successive creations/terminations of service processes are neither exponentially distributed nor independent, but correlated, generally distributed entities.

3.4 Analysis and solution of the multi-server model

The Apache Web server with its dynamic pool size is modelled by a continuous-time Markov chain $Z = \{[I_1(t), I_2(t), J(t)]; t \geq 0\}$ with state space $\{1, \ldots, M\} \times \{1, \ldots, N\} \times \{0, \ldots, L\}$. Here $J(t)$, $0 \leq J(t) \leq L$, represents the number of connection requests that are actually served in the HTTP server or wait to get service at time t.

Then the two state variables $(I_1(t), I_2(t))$ are lexicographically sorted to form a single variable $I(t)$ as illustrated by Table 1. The index transformation is determined by the following equations:

$$I(t) = I_1(t) + (I_2(t) - 1)M,$$

$$I_2(t) = f_2(I(t)) = \left\lfloor \frac{I(t)-1}{M} \right\rfloor + 1,$$

$$I_1(t) = f_1(I(t)) = (I(t) - 1) \bmod M + 1.$$

Then we get the equivalent Markov chain $Y = \{[I(t), J(t)]; t \geq 0\}$ with a generator matrix Q_Y to describe the operation of the non-threaded Web server with the MPM module Prefork. It evolves on a finite rectangular strip and belongs to the class of level-dependent Quasi Simultaneous-Multiple Births and Simultaneous-Multiple Deaths (QBD-M) processes (cf. [4]). Its possible transitions are determined by the following events with corresponding rates:

Table 1 Ordering of the phase variable $I(t)$

I	(I_1, I_2)
1	(1, 1)
2	(2, 1)
⋮	⋮
M	$(M, 1)$
$M+1$	(1, 2)
⋮	⋮
$2M$	$(M, 2)$
⋮	⋮
$MN - M + 1$	$(1, N)$
⋮	⋮
MN	(M, N)

- $A_j(i,k)$ describes the purely lateral transition rate from state (i, j) to (k, j), for all $0 \leq j \leq L$ and $1 \leq i, k \leq MN$, $i \neq k$. It is caused by either a phase transition of the modulating Markov process of the arrival stream or a change of the number $I_2(t)$ of available HTTP service processes. We can write $A_j = U_j \oplus \Omega_X$, where $\oplus$ denotes the Kronecker sum of two matrices.
- $B_{i,j,j+s}$ denotes the s-step upward transition rate from state (i, j) to $(i, j+s)$, $1 \leq s \leq L-j$, $0 \leq j \leq L$ and $1 \leq i \leq MN$. It is caused by a new batch arrival of size $s \geq 1$ of connection requests. For a given j, s can be considered as bounded variable if L is finite and unbounded if L is infinite. When $s \geq L - j \geq 0$ the queue gets full and $j+s-L$ requests are lost. Then it holds for $1 \leq i \leq MN$:

$$B_{i,j-s,j} = (1 - \theta_{f_1(i)})\theta_{f_1(i)}^{s-1}\sigma_{f_1(i)}$$
$$0 \leq j - s \leq L - 2; 1 \leq j \leq L - 1$$

$$B_{i,j,L} = \sum_{s=L-j}^{\infty} (1 - \theta_{f_1(i)})\theta_{f_1(i)}^{s-1}\sigma_{f_1(i)} = \theta_{f_1(i)}^{L-1-j}\sigma_{f_1(i)}$$
$$0 \leq j \leq L - 1.$$

- $C_{i,j,j-1}$ is the departure rate of HTTP connections, i.e., the Web server finishes the service of a request and the downward transition from state (i, j) to $(i, j-1)$, $1 \leq j \leq L$, $1 \leq i \leq MN$, represents the departure of the corresponding connection request from the HTTP server.

We assume that a maximum of one request departs at any time. Therefore, the one-step downward transitions take place by the departures of single connection requests after their service completion. The service of requests depends on the contention for physical resources, i.e. CPU processing and disks' operations, in the host machine.

Applying Menasce's [13] approach with a CQN model of the Web server's physical resources (see Fig. 4) and Mean Value Analysis to solve it, we obtain the corresponding values of the throughput $T_p(j)$ for j requests and p active processes in the system. Hence, $C_{i,j,j-1} = T_{f_2(i)}(j)$ for $1 \le i \le MN$.

Since the Markov chain $Y(t)$ with its finite state space is ergodic, its corresponding steady-state probabilities $\{p_{i,j}\}$, $p_{i,j} = \lim_{t\to\infty} \mathbb{P}(I(t) = i, J(t) = j)$, exist. Let $\mathbf{v}_j = (p_{1,j}, \ldots, p_{NM,j})$ denote the row vector of these probabilities corresponding to level $J(t) = j$, $0 \le j \le L$, in steady state and $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_L)$.

We define the diagonal matrices

$$B_{j-s,j} = \operatorname{diag}\Big[B_{1,j-s,j}, B_{2,j-s,j}, \ldots, B_{NM,j-s,j}\Big],$$
$$0 \le j - s < j \le L$$
$$B_s = B_{j-s,j}$$
$$= E_N \otimes \operatorname{diag}\Big[\sigma_1(1-\theta_1)\theta_1^{s-1}, \ldots, \sigma_M(1-\theta_M)\theta_M^{s-1}\Big],$$
$$1 \le s \le j \le L-1$$
$$\Sigma = E_N \otimes \operatorname{diag}[\sigma_1, \sigma_2, \ldots, \sigma_M]$$
$$\Theta = E_N \otimes \operatorname{diag}[\theta_1, \theta_2, \ldots, \theta_M]$$
$$C_j = \operatorname{diag}[C_{i,j,j-1}]$$
$$= \operatorname{diag}[T_1(\min(j,1)), \ldots, T_N(\min(j,N))] \otimes E_M,$$
$$0 < j \le L$$

where $\operatorname{diag}(\cdot)$ represents a diagonal matrix of the corresponding elements and E_N, E_M denote an $N \times N$ and $M \times M$ identity matrix, respectively.

Then we get (since $\Theta^0 = E_{NM}$ is the identity matrix):

$$B_s = \Theta^{s-1}(E_{MN} - \Theta)\Sigma, \quad 1 \le s \le L-1$$
$$B_{L-s,L} = \Theta^{s-1}\Sigma, \quad 1 \le s \le L$$

Let $\mathbf{e}_{NM}$ and $\mathbf{e}_M$ be a column vector with all ones of size NM and M, respectively. D_j^A, D_j^C are those diagonal matrices whose diagonal elements are the sums of the elements in the corresponding rows of A_j and C_j, respectively.

Consequently, the steady-state balance equations $\mathbf{v} \cdot Q_Y = 0$ and normalization condition read as follows:

(1) For the L^{th} column of Q_Y, i.e. level $j = L$:

$$\sum_{s=1}^{L} \mathbf{v}_{L-s} B_{L-s,L} + \mathbf{v}_L[A_L - D_L^A - D_L^C] = 0$$

(2) For the j^{th} column of Q_Y, i.e. level $0 < j \le L-1$:

$$\sum_{s=1}^{j} \mathbf{v}_{j-s} B_s + \mathbf{v}_j[A_j - D_j^A - \Sigma - D_j^C] + \mathbf{v}_{j+1} C_{j+1} = 0,$$

(3) For the 0^{th} column of Q_Y, i.e. level $j = 0$:

$$\mathbf{v}_0[A_0 - D_0^A - \Sigma] + \mathbf{v}_1 C_1 = 0$$

(4) Normalization condition:

$$\sum_{j=0}^{L} \mathbf{v}_j \cdot \mathbf{e}_{NM} = 1$$

The steady-state probabilities $\{p_{i,j}\}$ can be obtained either by directly solving the steady-state balance equations or by advanced matrix-geometric methods (cf. [2, 10, 15, 17]). Before we can use these advanced matrix-geometric techniques, the computational methodology proposed by Chakka and Do [5] has to be applied. Here we follow this approach.

Then important performance measures can be determined in terms of the steady-state probabilities:

– The mean number of idle processes (and in a similar way the variance) is determined by

$$E(Idle) = \sum_{i=1}^{NM} \sum_{j=0}^{L} p_{i,j} \cdot \max(f_2(i) - j, 0). \tag{1}$$

– The probability p_W that a new TCP request has to wait until the Web server provides a new idle child process is determined as follows. Upon the arrival of a batch of size s, the number of idle processes is given by $\max(f_2(i) - j, 0)$ and $\max(j - f_2(i), 0)$ is the number of waiting processes. Hence, $\max(s - \max(f_2(i) - j, 0), 0)$ is the number of requests which cannot "immediately" get service by idle processes, i.e. they have to wait in the software queue for the birth of new `httpd` processes or get lost due to the shortage of buffering in state (i, j). Taking into account that a maximum of $L - j$ requests can be admitted in state (i, j), then $\min(\max(s - \max(f_2(i) - j, 0), 0), L - j - \max(f_2(i) - j, 0))$ is the number of requests which are forced to wait. The arrival rate of batches is given by $\sigma_{f_1(i)}$ and $\sigma_{f_1(i)} \cdot (1 - \theta_{f_1(i)}) \cdot \theta_{f_1(i)}^{s-1}$ determines the arrival rate of batches of size s in state (i, j). Therefore, p_W can be written as

$$p_W = \sum_{i=1}^{NM} \sum_{j=0}^{L} \sum_{s=\max(f_2(i)-j,0)+1}^{\infty} p_{i,j} \cdot \frac{\sigma_{f_1(i)}(1-\theta_{f_1(i)})\theta_{f_1(i)}^{s-1} s}{\overline{\sigma}} \times \frac{\min(\max(s-\max(f_2(i)-j,0),0), L-j-\max(f_2(i)-j,0))}{s}$$

$$= \sum_{i=1}^{NM} \sum_{j=0}^{L} \sum_{s=\max(f_2(i)-j,0)+1}^{\infty} p_{i,j} \cdot \frac{\sigma_{f_1(i)}(1-\theta_{f_1(i)})\theta_{f_1(i)}^{s-1}}{\overline{\sigma}} \cdot \min(s-\max(f_2(i)-j,0), L-j-\max(f_2(i)-j,0)) \quad (2)$$

where $\overline{\sigma}$ denotes the overall mean arrival rate of the individual requests given by

$$\overline{\sigma} = \sum_{l=1}^{M} \frac{\sigma_l}{(1-\theta_l)} r_l.$$

Here $\mathbf{r} = (r_1, r_2, \ldots, r_M)$ is the vector of the equilibrium probabilities of the modulating Markov chain X that is uniquely determined by the equations:

$$\mathbf{r} \cdot Q_X = 0; \qquad \mathbf{r} \cdot \mathbf{e}_M = 1.$$

4 Performance results

4.1 Validation of the analytic model

To validate the proposed modeling approach, a measurement configuration of an Apache Web server operating on a host with a 3.00 GHz Intel(R) Pentium(R) D CPU, 2 Mbyte cache, and 2 Gbyte memory has been set up under Linux. The corresponding workload described in Sect. 3.3.1 has been generated by an appropriate shell script running on another Linux machine in combination with the traffic generator `ab`[3] (see The Apache Software Foundation [22]). Moreover, the major statistics have been captured, e.g. the creation and killing times, the idle and busy times of each HTTP process. In the following, we present some numerical results related to a HTTP/1.0 workload model. The Web server system has the parameters $h_{\min} = 5$, $h_{\max} = 10$, $N = 30$, $L = 40$. In our corresponding traffic model the modulating process X has only one state. This means that the arrivals of TCP connection requests constitute a recurrent process and its inter-arrival times are approximated by a GE distribution with the parameters (σ, θ). Hence, we capture only the burstiness of the arrivals without considering a sophisticated autocorrelation structure of the request process. Despite of this restriction, the derived results already indicate a sufficient level of accuracy of our new performance model. To generate the numerical results of the latter analytic model, we need the service time parameters as well as the killing and the creation rate of the `httpd` process. These parameters were determined based on the statistics, i.e. the creation and killing times, the idle and busy times, of each HTTP service process arising from the measurements.

In Table 2 we show a comparison between the measurements and the results arising from the solution of our analytic model by a procedure for QBD-M processes (cf. [4, 5]). The performance metric includes the mean $E(Idle)$ and the variance $Var(Idle)$ of the number of idle processes as function of the workload parameters σ and θ, as well as the probability p_W that a new TCP request needs to wait until the Web server offers a new `httpd` process, i.e., the probability that a new request does not find an idle `httpd` process upon its arrival.

In the selected scenario the measurements generate an average of 9.5 idle processes and a minimum and maximum of 8.2 and 9.9, respectively, whereas the analytic model has a range of [7.1, 9.9] and a mean of 9.3 idle processes. Regarding the mean number of idle processes the analytic model generates an average relative error of less than 3% subject to the measured values, the relative error of the variance is about 20%. Considering the waiting probability the absolute error is less than 11.3%, however, the relative error is larger since the correlation of the arrivals is not fully considered in the selected scenario. For a properly tuned system, the waiting probability can be kept small. Regarding control purposes the model is accurate enough and can be successfully applied to predict the average behavior of idle processes.

In conclusion, the numerical results justify the claim that our mathematically tractable model can be used to predict rather accurately the mean-value performance of the MPM Prefork process, despite of its explicit assumption about the Markovian property of $I_2(t)$.

[3] We have used the tool `ab` since it is capable to initiate simultaneously a batch of HTTP requests. To generate complex arrival distributions of sessions, an additional shell script is needed.

Table 2 Comparison of the measurement results and the analytic solution

Parameters				Measurement study			Analytic model		
σ	θ	η	ϵ	$E(Idle)$	$Var(Idle)$	Waiting prob.	$E(Idle)$	$Var(Idle)$	Waiting prob.
5	0.4	0.003960	0.001500	9.92740	0.4323	0.000087	9.933774	0.333595	0.0003301
5	0.5	0.007042	0.003130	9.89174	0.6153	0.003466	9.906068	0.437426	0.0023988
5	0.6	0.011198	0.007337	9.83198	0.8833	0.015859	9.858125	0.602280	0.0122027
5	0.7	0.020465	0.016675	9.70886	1.3312	0.056009	9.762665	0.895090	0.0485737
5	0.8	0.062304	0.058661	9.47287	2.0765	0.175968	9.545315	1.460498	0.1602376
5	0.9	0.230746	0.227524	9.23774	3.8538	0.385522	8.876794	2.732862	0.4270081
10	0.4	0.006120	0.001749	9.87653	0.5695	0.000028	9.865037	0.477537	0.0001713
10	0.5	0.007774	0.003455	9.82228	0.7829	0.002989	9.812958	0.618592	0.0029087
10	0.6	0.016354	0.012161	9.72373	1.1271	0.015115	9.716133	0.852531	0.0146051
10	0.7	0.040675	0.036607	9.54612	1.6814	0.056359	9.525983	1.265490	0.0576934
10	0.8	0.095757	0.091942	9.15435	2.5931	0.178238	9.098354	2.055569	0.1883662
10	0.9	0.330487	0.327367	9.00565	4.5981	0.360757	7.918966	3.700704	0.4729701
20	0.4	0.007869	0.002951	9.82219	0.6871	0.000782	9.733936	0.671095	0.0006167
20	0.5	0.012012	0.007207	9.74126	0.9354	0.003211	9.628082	0.872976	0.0041383
20	0.6	0.020740	0.016131	9.60848	1.3171	0.015597	9.447366	1.195716	0.0200142
20	0.7	0.057788	0.053410	9.37257	1.9583	0.054940	9.083828	1.772014	0.0775575
20	0.8	0.130866	0.126864	8.88637	2.9865	0.166680	8.328340	2.831075	0.2433198
20	0.9	0.408192	0.405091	8.23474	5.3645	0.536450	7.083550	4.933103	0.5040300

4.2 Performance impact of Apache directives

In real operation it is of major importance to analyze the impact of basic Apache directives such as *MaxClients*, *MaxSpareServers* and *MinSpareServers* on the performance indices perceived by the users such as the waiting probability and the efficiency observed by operators, e.g. in terms of the average number of idle processes. Therefore, we have depicted in Fig. 5 the average number of idle processes as function of the creation rate η and the killing rate ϵ for some specific parameter values of the request rate. The results quantitatively show that ϵ has a crucial influence on the number of idle processes. The average number is almost independent of the creation rate η if the killing rate ϵ is large enough. The rationale of this behavior is as follows: when the killing rate ϵ is large enough, the control policy of a Web server can achieve its goal in a very fast manner, i.e. to enforce a number of idle processes less than $h_{\max}$. This means that the interval determined by the reaction to the critical load period until the compliance with the control policy is very short. From the point of view of an autonomous Web server operation, this behavior of the MPM Prefork module is useful. It is proved by the operational success of Apache Web servers every day.

The overall response time of the Web server observed by a user sending HTTP requests depends upon the physical configuration of the corresponding machine running the Web server, the parameter setting of the Web server, the size

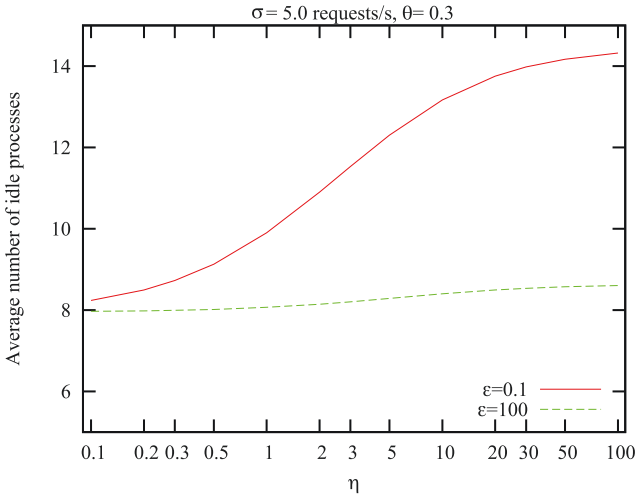


Fig. 5 Average number of idle processes vs. η and ϵ

of the objects associated with the specific request and the network status. Since the response time at the server side plays a very significant role in the overall response time perceived by the users, we focus on those factors which have a major impact on the latter performance index.

Following the approach in [13] we have assumed in our analysis that after the successful TCP connection set-up the residual service time upon the acceptance of a HTTP request is approximated by a CQN. The latter provides the input of our service time model. If the machine running the

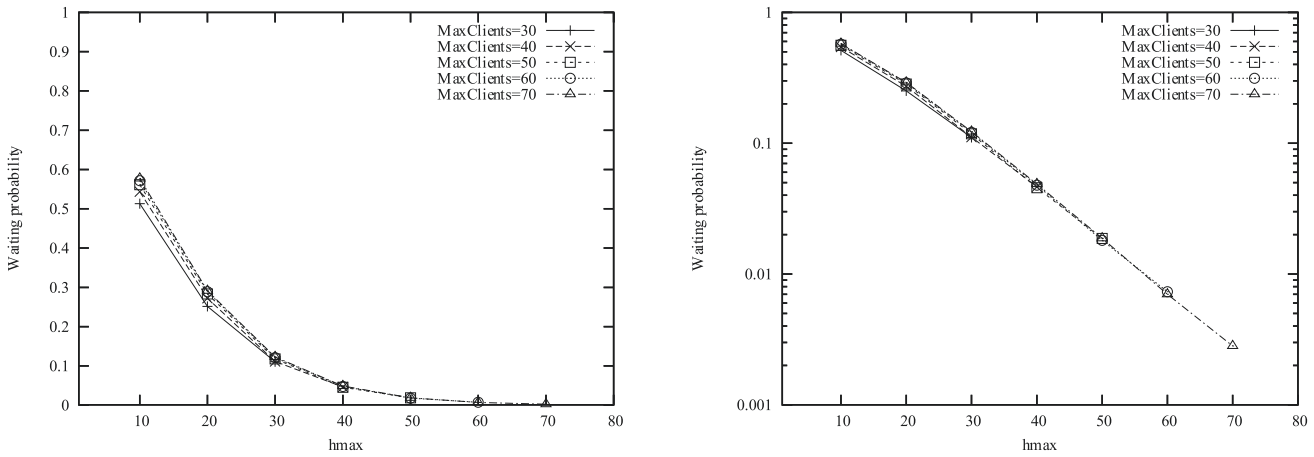


Fig. 6 Waiting probability p_W vs. MaxClients N and MaxSpareServers $h_{\max}$ on a linear (*left*) and a logarithmic scale (*right*) for the parameters $\sigma = 20$ and $\theta = 0.9$

Web server is powerful enough, then after the TCP connection set-up the service time should be small enough to satisfy the QoS requirement of a specific user. From the users' perspective, the most disturbing event happens when there is no idle service process available upon the arrival of a request. Then the user does not receive any response and has to wait until a new service process is spawned by the Web server. Therefore, this waiting time to generate a new service instance can significantly contribute to the overall response time. In Table 2 we can observe, for example, that the resulting waiting probability is around 0.5 when $\sigma = 20$ and $\theta = 0.9$.

To further explore the impact of the Apache directives on the waiting probability p_W, we illustrate in Fig. 6 the latter performance index as function of the parameters *MaxClients* N and *MaxSpareServers* $h_{\max}$ on a linear and a logarithmic scale of the vertical axis, respectively. Note that *MaxSpareServers* is upper bounded by *MaxClients*. However, the *MaxSpareServers* directive has a much stronger influence on the waiting probability governed by a nearly linear shape on the logarithmic scale, i.e. it follows an exponential decay characteristic in $h_{\max}$.

In summary, we have clearly shown by our analytic means that for the given non-threaded multi-processing module *Prefork* both the *MaxClients* and the *MaxSpareServers* directive have a strong impact on the waiting probability of HTTP requests and, therefore, on the response time of the Apache Web server.

Considering the autonomous operation and optimized control of an Apache Web server with MPM Prefork, we see that the appropriate dynamic tuning of these major directives and control parameters provides the highest potential to satisfy both the operator's efficiency and the users' QoS requirements.

5 Conclusions

In this paper we have studied the dynamic behavior of the resource pool of service processes that is implemented in the non-threaded multi-processing module (MPM) *Prefork* of an Apache Web server. To describe the latter and to analyze its performance we have proposed a new Markovian queueing model with a variable number of servers.

Applying a batch Markovian arrival process and advanced matrix-geometric solution techniques, numerical results on the performance of Apache's dynamic pool of service processes have been derived from this tractable analytic multi-server model. We have also compared it with actual measurements of the real dynamic pool-size behavior. The latter justify and validate the proposed decomposition modeling and analysis approach. It is derived from our major assumption that the modulating joint arrival and service process $I(t)$ is governed by a Markovian property. Furthermore, the model can be easily extended to capture multiple arrivals and multiple departures of connection requests.

In conclusion, we believe that our matrix-analytic approach is a useful tool to predict and to tune the performance of an Apache Web sever with the non-threaded multi-processing module *Prefork*. It can also be extended to evaluate other Web server software architectures under UNIX such as the Apache 2.2 hybrid multi-threaded multi-processing module *Worker* which is the subject of future research.

Acknowledgements The authors express their sincere gratitude to Thang Le Nhat, now with Post and Telecommunications Institute of Technology, Hanoi, as well as Cuong T. Van and Nam H. Do, Budapest University of Technology and Economics, who supported the monitoring of the Apache Web server and the validation of the performance model.

Tien Van Do acknowledges the support of Siemens Hungary by the Werner von Siemens Excellence Award, and the grants by the EU IST-FP6 NoE project "EuroNGI/EuroFGI" and the IKMA (Ipar a Korszerű Mérnökképzésért Alapítvány) foundation.

Finally, the authors would like to express their sincere thanks to the reviewers for their constructive criticism and useful comments which helped them to improve the presentation of the paper.

References

1. Andersson, M., Cao, J., Kihl, M., & Nyberg, C. (2003). Performance modeling of an apache web server with bursty arrival traffic. In *Proceedings of the international conference on Internet computing*, Las Vegas, USA.
2. Bini, D., & Meini, B. (1996). On the solution of a nonlinear matrix equation arising in queueing problems. *SIAM Journal Matrix Analysis and Applications*, *17*, 906–926.
3. Cao, J., Andersson, M., Nyberg, C., & Kihl, M. (2003). Web server performance modeling using an M/G/1/K*PS queue. In *Proceedings of 10th IEEE international conference on telecommunications* (Vol. 2, pp. 1501–1506).
4. Chakka, R., & Do, T. V. (2004). Some new Markovian models for traffic and performance analysis in telecommunication networks, Tutorial Paper. In D. D. Kouvatsos (Ed.), *Proceedings of the second international working conference on performance modelling and evaluation of heterogeneous networks* (*HET-NETs 04*), Ilkley, UK (pp. T6/1–31).
5. Chakka, R., & Do, T. V. (2007). The $MM\sum_{k=1}^{K} CPP_k$/GE/c/L G-queue with heterogeneous servers: steady state solution and an application to performance evaluation. *Performance Evaluation*, *64*(3), 191–209.
6. Choi, H.-K., & Limb, J. O. (1999). A behavioral model of Web traffic. In *Proceedings of the seventh IEEE international conference on network protocols* (*ICNP'99*) (pp. 327–334).
7. Dilley, J., Friedrich, R., Jin, T., & Rolia, J. (1997). Measurement tools and modeling techniques for evaluating Web server performance. In *Proceedings of the 9th international conference on modelling techniques and tools for computer performance evaluation*.
8. Dilley, J., Friedrich, R., Jin, T., & Rolia, J. (1998). Web server performance measurement and modeling techniques. *Performance Evaluation*, *33*, 5–26.
9. Kouvatsos, D. (1994). Entropy maximisation and queueing network models. *Annals of Operations Research*, *48*, 63–126.
10. Latouche, G., & Ramaswami, V. (1999). Introduction to matrix analysis methods in stochastic modeling. *ASA–SIAM series on statistics and applied probability*.
11. Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J. L., & Parekh, S. (2003). Online response time optimization of Apache Web server. In *Eleventh international workshop on quality of service (IWQoS 2003)* (pp. 461–478).
12. Liu, Z., Niclausse, N., Jalpa-Villanueva, C., & Barbier, S. (1999). *Traffic model and performance evaluation of Web servers* (Technical Report 3840). INRIA.
13. Menasce, D. A. (2003). Web server software architectures. *IEEE Internet Computing*, *7*(6), 78–81.
14. Menasce, D. A., Dodge, R., & Barbara, D. (2001). Preserving QoS of e-commerce sites through self-tunning: a performance model approach. In *Proceedings of the ACM conference on e-commerce*, Tampa, Florida, USA (pp. 224–234).
15. Mitrani, I., & Chakka, R. (1995). Spectral expansion solution for a class of Markov models: application and comparison with the matrix–geometric method. *Performance Evaluation*, *23*, 241–260.
16. Mosberger, D., & Jin, T. (1998). httperf: A tool for measuring Web server performance. In *First workshop on Internet server performance* (*WISP 98*) (pp. 59–67). New York: Asoc. Comput. Math.
17. Naoumov, V., Krieger, U. R., & Warner, D. (1996). Analysis of a multi-server delay–loss system with a general Markovian arrival process. In S. R. Chakravarthy & A. S. Alfa (Eds.), *Lecture notes in pure and applied mathematics: Vol. 183. Matrix–analytic methods in stochastic models*. New York: Dekker.
18. Reeser, P., & Hariharan, R. (2000). Analytical model of web servers in distributed environments. In *Proceedings of the 2nd ACM international workshop on software and performance WOSP'2000*, Ottawa, Canada (pp. 158–167).
19. Slothouber, L. P. (1995). A model of Web server performance. http://www.geocities.com/webserverperformance.
20. Squillante, M. S., Yao, D. D., & Zhang, L. (1999). Web traffic modeling and Web server performance analysis. In *Proceedings of the 38th IEEE international conference on decision & control*, Phoenix, Arizona, USA (pp. 4432–4439).
21. Squillante, M. S., Yao, D. D., & Zhang, L. (1999). Web traffic modeling and Web server performance analysis. *ACM SIGMETRICS Performance Evaluation Review*, *27*(3), 24–27.
22. The Apache Software Foundation (1995–2007). ab—Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.2/programs/ab.html.
23. The Apache Software Foundation (1995–2007). Apache HTTP server project. http://www.apache.org.

Tien Van Do obtained a M.Sc and a Ph.D degree from the Budapest University of Technology and Economics in 1991 and 1996, respectively. He is an associate professor at the Department of Telecommunications, Budapest University of Technology and Economics. He has participated in some European projects like ACTS ELISA, EU Framework 5 HELINET and EU Framework 6 CAPANINA and led various software development projects for T-COM Hungary, NOKIA Hungary and NOKIA Siemens Networks Hungary.

Udo R. Krieger obtained an M.Sc. degree in Applied Mathematics and a Ph.D. degree in Computer Science from the Technische Hochschule Darmstadt, Germany. From 1985 to 2003 he worked as professional researcher and technical project manager in the areas of telecommunications, queueing and teletraffic theory at the Research and Technology Center of Deutsche Telekom in Darmstadt and additionally as a part-time lecturer at the Computer Science Department of J.W. Goethe University, Frankfurt. Since October 2003 he is head of the computer networks group at Otto-Friedrich-University Bamberg, Germany. Dr. Krieger has led national projects of Deutsche Telekom and participated in several EU projects like COST 257 and 279, Eurescom P1112 and the IST-FP6 Network of Excellence EuroNGI/EuroFGI. At present, he is serving on the editorial board of the journal Computer Networks.

Ram Chakka received his B.Engg. (Electrical and Electronics, 1980), M.S. (Engg) (Computer Science and Automation, 1986), both from the Indian Institute of Science, Bangalore, India and a Ph.D. degree (Computer Science, 1995) from the University of Newcastle upon Tyne, UK. Presently he is a Professor in Computer Science and Engineering and Director Research at MIET, Meerut, India. Earlier, he worked at Indian Institute of Science, University of Newcastle upon Tyne, Imperial College (London), Middlesex University (UK), Norfolk State University (NSU, USA), Sri Sathya Sai Institute of Higher Learning (India) and RGMCET (India). At NSU, Dr. Chakka was awarded the Certificate of Excellence for Outstanding Scholarship from the School of Science and Technology. He published over 40 papers in performability modeling and evaluation of computing systems, communication networks and other discrete event systems. Dr. Chakka is a member of IEEE and also IEEE Vehicular Technology Society.