Simulation Based Performance Analysis of Web Servers

Lisa Wells, Søren Christensen, Lars M. Kristensen, and Kjeld H. Mortensen
CPN Centre, Department of Computer Science
University of Aarhus
DK-8000 Aarhus C, Denmark
{wells,schristensen,lmkristensen,khm}@daimi.au.dk

Abstract

This paper presents a general framework for modeling distributed computing environments for performance analysis by means of Timed Hierarchical Coloured Petri Nets. The proposed framework was used to build and analyze a Coloured Petri Net model of a HTTP web server. Analysis of the performance of the web server model reveals how the web server will respond to changes in the arrival rate of requests, and alternative configurations of the web server model are examined. These are the results of a research project conducted in cooperation between the CPN Centre and Hewlett-Packard Corporation on capacity planning and performance analysis of distributed computing environments.

1. Introduction

The Internet and the World Wide Web (WWW) have experienced exponential growth since their inception. Popular web sites receive millions of hits per day, and it is not uncommon for these sites to exhibit extremely high response times. High response times are a source of frustration for users, and with the growing use of web sites for e.g., electronic commerce this may damage the reputation of the company offering the web site, leading to loss of business. As a consequence, it is important to be able to identify bottlenecks, predict future capacity shortcomings, and determine the most adequate or cost effective way to reconfigure such distributed computing environments to overcome performance problems and cope with increasing workload demands.

This paper presents one of the first results of the CPN Centre which is a research project conducted as a cooperation between the CPN Group at the University of Aarhus and Hewlett-Packard (HP) Corporation. One of the goals of the CPN Centre is to investigate the use of Coloured Petri Nets (CP-nets or CPNs) [14, 17] and simulation as an underlying technology for performance analysis and capacity planning of distributed computing environments. In the initial project phase the goal has been to establish a proof-of-concept. It was therefore decided to consider a concrete representative example of a distributed computing environment in the form of a simple web server environment. It is the main results of this first subproject which is the subject of this paper.

The main result of the first subproject has been the development of a modeling framework for distributed computing environments. This framework is based on a building-block approach dividing the components of the CPN models into three distinct layers: a structural layer describing clients, servers, networks, and their relationship; an application layer describing the applications running on the servers and the clients; and a resource layer describing the resources, i.e., CPUs, disks, and communication channels, of the system. This means that the CPN models include both a functional view of the system represented by the application layer, and a performance view represented by the resource layer.

A second result has been the simulation based performance analysis of the constructed CPN model. The CPN model was validated and calibrated by comparing performance results from simulation with the performance which can be observed in a corresponding physical environment, for details see [16]. Using the validated model, simulation experiments were run in order to examine the effects of varying the arrival rates of requests on the performance of the web server. Additional experiments were undertaken in order to compare the performance of different configurations, e.g. faster CPU and faster disk, with the basic configuration corresponding to the actual web server.

For construction and simulation of the CPN models we rely on the Design/CPN tool [6, 9]. The simulator of Design/CPN has previously been used in other projects on performance analysis e.g., in the areas of high-speed interconnects [7], and ATM networks [8]. For collecting data about the performance of the CPN web model during simulations,
we have used the Design/CPN Performance Tool [24]. This tool makes it possible to collect data about the performance of the considered system during lengthy simulations without having to modify or make extensions to the CPN model itself.

In the literature, there are several papers on performance analysis of web servers. Many of these papers present measurement studies that focus on workload characterization [2, 3, 18] or measurement of, e.g., resource utilization and response time [1, 10, 13]. Analytic models have been used to analyze the performance of HTTP over several transport protocols [12] and for capacity planning of web servers [10]. As one of the few simulation studies of web servers, [23] presents an end-to-end queueing model of a web server environment. Although these studies provide excellent insight into the performance of web servers, and have made significant contributions to the understanding of web workloads, none provide a framework that can be used for both performance analysis and functional analysis of web servers, in particular, and of distributed systems, in general. Furthermore, the proposed framework can be useful for answering "what if" questions concerning possible alternative web server configurations or workloads.

This paper is organized as follows. Section 2 provides the necessary background on web servers and timed hierarchical CP-nets for understanding the rest of this paper. The reader is assumed to have some background knowledge on the basic concepts of High-level Petri Nets [15]. Section 3 describes the physical web server, and explains how workload for the web server and the simulation model was obtained. Section 4 introduces the framework for modeling of distributed computing environments and illustrates how it can be used to model a HTTP web server. Section 5 presents performance results obtained by making simulations of the constructed CPN model. Finally, Sect. 6 sums up the conclusions from the project.

2. Background

This section provides the background on web servers, and the time and hierarchy concepts of CP-nets sufficient for understanding the rest of this paper. Section 2.1 reviews web servers and the HTTP protocol. A detailed treatment of these concepts can be found in e.g., [22]. Section 2.2 reviews the time concept of CP-nets. Section 2.3 reviews the hierarchy concepts of CP-nets. A detailed treatment of the time concept and the hierarchy concept of CP-nets can be found in volumes 2 and 3, respectively, of [14].

2.1. Web servers

The fundamental service offered by a web server is access to the documents stored at the web server to web clients. These documents are typically Hypertext Markup Language (HTML) [21] documents but may also be e.g., graphics or plain ASCII files. Documents stored at the web server are typically accessed by web clients using a web browser. The main components of a web server consist of a hardware platform (e.g., CPU and disk), an operating system, the web server application, and the documents stored at the web server.
messages.

Throughout this paper we will consider HTTP/1.0 [4]. This means that if a requested HTML document contains e.g., inline images then such images are requested separately using the same communication pattern as described above. In particular this means that a TCP connection is opened for each subdocument. A newer version of the HTTP protocol, i.e. HTTP/1.1 [11], includes the notion of persistent connections to avoid the overhead of opening separate TCP connections. It is rather straightforward to modify the CPN model presented in this paper to reflect HTTP/1.1.

2.2. Timed Coloured Petri nets

The time concept of CP-nets is based on the introduction of a global clock. The clock values represent model time, and they may either be integers, i.e. discrete time, or reals, i.e. continuous time. In addition to the token value, we allow each token to carry a time value, also called a time stamp. Intuitively, the time stamp describes the earliest model time at which the token can be used, i.e., removed by the occurrence of a binding element.

In a timed CP-net a binding element is said to be colour enabled when it satisfies the enabling rule for untimed CP-nets, i.e., when the required tokens are present at the input places and when the guard of the transition evaluates to true. However, to be enabled, the binding element must also be ready. This means that the time stamps of the tokens to be removed must be less than or equal to the current model time.

The marking of places with a timed colour set (type) is a timed multi-set of token values. A timed multi-set is similar to a multi-set, except that each member of the multi-set carries a time stamp. To model that an event takes \( r \) time units, we let the corresponding transition create time stamps for its output tokens that are \( r \) time units larger than the clock value at which the transition occurs. This implies that the tokens produced are unavailable for \( r \) time units.

The execution of a timed CP-net is time driven, and it works in a way similar to that of event queues found in many languages for discrete-event simulation. The system remains at a given model time as long as there are colour enabled binding elements that are ready for execution. When no more binding elements can be executed at the current model time, the system advances the clock to the next model time at which binding elements can be executed. Each marking exists in a closed interval of model time (which may be a point, i.e., a single moment).

2.3. Hierarchical Coloured Petri nets

The basic idea underlying hierarchical CP-nets is to allow the modeler to construct a large model from a number of smaller CP-nets called pages. These pages are then related to each other in a well-defined way as explained below.

In a hierarchical CP-net, it is possible to relate a so-called substitution transition (and its surrounding places) to a separate CP-net called a subpage. A subpage provides a more precise and detailed description of the activity represented by the transition. Each subpage has a number of port places and they constitute the interface through which the subpage communicates with its surroundings. To specify the relationship between a substitution transition and its subpage, we must describe how the port places of the subpage are related to so-called socket places of the substitution transition. This is achieved by providing a port assignment. When a port place is assigned to a socket place, the two places become identical. The port place and the socket place are just two different representations of a single conceptual place. More specifically, this means that the port and the socket places always have identical markings.

It should be noted that substitution transitions never become enabled and never occur. Substitution transitions work as a macro mechanism. They allow subpages to be conceptually inserted at the position of the substitution transitions – without doing an explicit insertion in the model.

3. The ITCHY environment

In this section we describe the hardware and software constituting the performance analysis and capacity planning laboratory environment. The environment is used for making performance measurements which can be compared with the simulation results of the CPN model. The capacity planning laboratory environment has been named ITCHY after the name of the web server. A detailed description of the ITCHY environment can be found in [16].

3.1. Server configuration

The web server is an Intel Pentium II 266 MHz processor based PC equipped with a local disk, and 160 MB RAM of internal memory. The server is running Windows NT 4.0 and has a Microsoft web server application installed. For the experiments the web server application was configured as a single threaded web server with cache disabled. The aim has been to configure the web server and choose parameters in a way which makes it possible to put heavy load on the server by means of a single client (to which we will return in a moment) and relatively few requests. As a consequence, the configuration is not realistic and is far from optimal with respect to performance of the web server.
However, this rather disabled configuration is better suited for initial experiments and for judging the accuracy of the CPN model. In order to model a multi-threaded server, the only change that needs to be made in the CPN model is to change the value of a parameter which determines the number of web server threads.

3.2. Workload model

It is obviously impractical to involve thousands of users to generate a realistic web workload. Thus we need to generate the workload by means of software. In this section we explain how a sufficiently realistic workload can be generated such that performance measurements for different parameter configurations can be obtained.

The approach taken is that of using a trace client. A trace client takes as input a web access log file and then makes a replay of the get-requests contained in the web access log file. Each get-request (entry) in a web access log contains a time stamp, specifying when the requests are to be made, and a specification of the document requested. The log can be real or automatically generated. A real log extracted from some existing web server is easy to use with the trace client but is lacking flexibility with respect to the variation of experiments. Automatically generated log files are harder to create because one needs to investigate statistical models, however the flexibility is more beneficial compared with using a real log file.

We made a simple statistical model of workloads with focus on file size distribution, because it is one of the important factors in a realistic workload. In order to estimate the distribution of file sizes, a web access log file covering a period of one month was analyzed. The Windows-based application Curve Expert determined that the best fit was the Weibull distribution [19], with the following distribution function: 

$$F(x) = 1 - 1.2393187 \exp^{-0.02445885x^{0.475}}$$

Figure 2 shows the fitted Weibull distribution and the file sizes from the web access log file in question (DAIMI log).

The average file size is approximately 5KB. To avoid any effect related to the file cache of the web server all get-requests in the generated workload are for distinct files.

Due to time constraints, a detailed analysis of neither the request times nor temporal locality of requests in the web access log file was undertaken. The log file covered a period of one month, and it was observed that the average request rate was 1 request/second. In the following, we will assume that the workload during peak hours is 5-10 requests/second, which is not an unreasonable expectation judging by what has been observed within other academic environments [2]. Furthermore, we will assume that the interarrival times between requests are exponentially distributed. This is not the most accurate model for request arrivals [2], but it proved to be sufficient for our purposes. In future experiments, the SURGE [3] tool could be used to generate accurate workload for web servers.

4. Modeling framework

In this section we present the developed modeling framework. As we introduce the basic ideas in the modeling framework, we will also present the constructed CPN model of the web server. In the following, the CPN model of the web server will be referred to as the CPN web model.

The modeling framework is based on the idea of logically dividing the pages of the CPN model into three layers: the structural layer representing the system architecture; the application layer representing the various applications running on the clients and servers of the modeled system; the resource layer representing the resources present in the considered system.

The structural layer of the modeling framework is motivated by the observation that a distributed computing environment, by definition, consists of several interconnected computers or workstations communicating using, e.g., local area networks (LANs), wide area networks (WANs), routers and gateways. Therefore, the model has to identify these components as well as their relationship which constitutes the architecture of the distributed computing environment under consideration.

The application layer of the modeling framework is motivated by the observation that a main component in a distributed computing environment is the applications e.g., the web servers, database servers, and file servers running on the different workstations. The relationship between these applications are often based on the client/server paradigm. The communication between clients and servers in the form of requests and responses has a high impact on the performance of a distributed computing environment. It is typically the performance of the service offered by these applications which is of interest when analyzing the performance of a distributed system.
When applications are executing and communicating, they make use of the resources in the environment e.g., CPUs, disks, LANs, and WAN connections. It is the use of these resources imposed by the different applications which determines the utilization of resources and the performance of the considered system. Since we aim at conducting performance analysis of distributed computing environment, these resources must also be represented in the models.

Although the idea of using a layered approach is not new in the area of distributed systems, e.g. consider the O/S1 layers for network architectures [22], we have not seen another modeling framework that explicitly divides the functional view and the performance view of a system into separate layers within a model. Separating the parts of the model giving a functional description (the application layer) from the part of the model describing the use of resources implies that it is easy to make the transition from a CPN model focusing on performance to a CPN model focusing on the logical correctness of the system. This can be done by simply disabling the pages in the resource layer since all aspects of the CPN model related to performance are isolated in this layer. The separation also means that during the development of a CPN model, one can first construct the structural and application layers of the CPN model, validate that this functional description behaves as expected, and subsequently add the use of resources and conduct performance analysis of the considered system. Of course, one must ensure that the resource layer does not affect the functionality of the application and structural layers. Unfortunately, we were unable to do any functional analysis of our model, due to time constraints. The focus of this particular project was to investigate the use of Coloured Petri Nets and simulation for performance analysis. However, in the future, we do intend to use our modeling framework to do both functional and performance analysis of distributed computing environments.

Figure 3 provides an overview of the pages constituting the CPN web model. Each node in Fig. 3 represents a page in the CPN model, and is named according to the page in the CPN model it represents. An arc leading from one node to another node means that the latter is a subpage of the former. The pages CpnWeb, Server, Network, and Client are the pages at the structural layer of the CPN model. They describe the architecture of the considered system as consisting of a server part (left), a client part (right), and a network part (middle).

The pages Resource and OSandHardware constitute the resource layer of the CPN model. In the CPN web model there are three different resources: the CPU resource of the server, the disk resource of the server, and the network resource corresponding to the LAN between the server and the client. We do not consider, e.g., the CPU resource of the client part since our main concern will be to analyze the performance of the server and not the performance of the clients. The OSandHardware page is used only to initialize the server resources, and the Resource page is described in more detail below.

The remaining pages are associated with the application layer of the CPN model, and they constitute four basic building blocks: a HTTP web server building block, a web client building block, a TCP protocol building block, and a building block describing parts of the HTTP protocol. The TraceClient page generates requests according to a web access log file, such as those described in Sect. 3. Page InputLog (lower right) enables the CPN model to read in web access log files and to use these as workload specification for driving the simulations. The SimpleClient page generates requests on-the-fly based on a user-defined workload specification.

In this paper we will not go into detail with all pages
of the CPN web model. In the following three subsections we will instead present selected examples of pages from the three different layers of the CPN model.

4.1. Structural layer

We now give an example of a page from the structural layer of the CPN web model. Figure 4 depicts the CpnWeb page which provides the most abstract view of the model. The substitution transitions Server, Network, and Client correspond to the three main parts of the CPN model. Place Network Buffer is used to model packets in transit between the clients and the server.

Figure 4. The CpnWeb Page.

4.2. Application layer

The application layer of the CPN web model is made up of the HTTP web server, which is the only application under consideration in the presented model. Below we give two examples of pages from the application layer of the server part of the CPN web model.

Figure 5 depicts the page HTTP-ServerGet. This page models the program executed by the threads of the HTTP server. Each thread repeatedly executes a loop in which get-requests from the clients are processed. The loop consists of the opening a TCP connection, represented by the substitution transition TCP-OpenConn, handling the request (HTTP-GetURL), and closing the TCP connection (TCP-CloseConn). All three phases require the execution of certain jobs which are put as tokens on place Jobs. We will return to how jobs are represented when we present the resource layer of the CPN web model. All three phases require access to the network represented by place Client. To illustrate the interplay between the application layer and the resource layer in terms of how the application layer requests resources at the resource layer, we now consider the page SendResponse which describes the behavior of the HTTP web server when sending the requested document back to the client. This page is shown in Fig. 6. Note that SendResponse is a subpage of HTTP-ServerGetURL (not shown) which is, in turn, a subpage of HTTP-ServerGet.

Sending the response of the get-request back to the client consists of the two phases modeled by the transitions Read URL and Send Response. Transition Read URL models the web server thread, represented by (i,app), checking the file attributes of the requested document, fetching the document from either cache or disk, writing the get-request into the servers web access log, and possibly updating the cache. Jobs for the CPU and disk resources are created by the function createJobs on the arc from Read URL to Jobs. Disk resources are always needed to read the file attributes of the document and to write the server access log. If the response code is HTTP_OK, then additional resources are needed. CPU resource is always needed to search through the cache, and in case of a cache hit, CPU resource is needed to read the document from main memory. In case the document is not cached, some disk resource is needed to read the document from the disk. When these jobs have been executed, as described by the arc inscription on the arc from the place Jobs to the transition Send Response, the response can be sent. Transition Send Response puts a packet on the network according to the determined response.

4.3. Resource layer

A resource is characterized by the capability of executing jobs. Execution of jobs at resources takes time and determines the progress of time in the CPN model. The concepts of resources and jobs of the resource layer facilitate making performance analysis based on the CPN model. In this modeling framework, the resource layer of the CPN model is the only part of the CPN model where the time constructs of CP-nets are directly exploited. They are only exploited indirectly at the application layer using the inter-
Figure 6. The SendResponse Page.

face between the application and the resource layer as illustrated in Fig. 6.

The page Resource, shown in Fig. 7, is the central page at the resource layer of the CPN web model and will be explained in more detail below. Recall that three kinds of resources are currently considered: the CPU resource of the server, the disk resource corresponding to the local disk of the server, and the network resource between the clients and the server. In Fig. 7 the resources are located as tokens on the place Resource. Resources are described by the colour set Resource which identifies the different attributes of a resource, e.g., the state attribute reflects whether the resource is idle or running, and the parameter attribute describes, among other things, the speed of the resource.

Now let us return to Fig. 7. Jobs arrive for the resource at the input/output port place Jobs. The different kinds of jobs are described by the colour set JobKind. Each kind of job has an attribute which specifies the size of the job. When a thread or an application requests use of a resource, e.g., a CPU resource, it will put a token on place Jobs, as was illustrated in Fig. 6. When the job has been executed at the resource, a token will be put on place Jobs to signal that the job has been processed. For example, when a CPU or disk job has been executed, it becomes READY. When a network job has been executed, i.e., the packet has been transmitted, it becomes TRANSMITTED.

Transition Job Arrival adds a newly arrived job to the job queue of the corresponding resource. Function ValidateJob ensures that the job and the resource match. This makes sure, for example, that threads can only request CPU resources of the computer on which the thread is executing. Transition Schedule Job is responsible both for scheduling jobs at the resource and for removing completed jobs.

The colour set Resource of place Resource is timed. The time stamp of a resource token residing on this place is calculated by the function NextEvent, and it corresponds to the time at which the next scheduling of a job on the resource is to take place. This is a very important construction in order to make simulations tractable since it, in practice, significantly reduces the number of steps which have to be executed during simulations.
4.4. Model validation

We will now consider both the validation of the logical behavior of the model and the calibration of the parameters of the model. A detailed description of the validation and calibration processes can be found in [16, 20].

The size and complexity of the CPN web model precluded using state space analysis to fully validate the functional correctness of the model, but interactive simulations and Message Sequence Charts were used extensively to validate the logical behavior of the model. While these techniques can never ensure that a model is error-free, they are useful for finding obvious modeling errors, and they can confirm that the model behaves as expected in a variety of situations.

The goal of the calibration process is to obtain a match between the performance results obtained by simulation and the performance results measured in the ITCHY environment and in this way to obtain a validated CPN web model. In this phase web access log files were used to generate the same requests for both the ITCHY environment and the CPN web model. The CPN web model has a number of parameters which can be adjusted. The values for many of these parameters were known, e.g. size of TCP packets, the speed of the server disk, and the effective speed of the network. However, the main difficulty was estimating the parameters of the CPU and the parameters related to the use of CPU resource, e.g., establishing a TCP connection or processing a get-request. The process of calibrating the CPU parameters consisted of a number of iterations of adjusting parameters in the model, running simulations, and comparing the results to the performance of the actual web server. For a more thorough description of the calibration process, see [20].

5. Performance analysis

In this section we show how the performance of the web server can be analyzed by means of the CPN web model presented in the previous section. We do so by illustrating the kind of performance results which can be obtained from lengthy simulations of the CPN web model.

5.1. Simulation experiments

This section briefly describes the simulation experiments that were run. During the performance analysis phase of the project, workload for the CPN web model was generated on-the-fly. The requests that were dynamically generated during a simulation fulfilled the same criteria as the workload that was described in Sect. 3.2, i.e., the file sizes of the requested documents were generated from the Weibull distribution, and the interarrival periods between requests were exponentially distributed. The performance of the web server was examined for request arrival rates of 5, 10, 20, 40 and 50 requests/second. Ten independent simulations were executed for each system configuration that was examined, and each simulation corresponded to 30 minutes of activity in the web server environment. 95% confidence intervals were calculated for the performance measures in question.

5.2. Performance measures

When considering the performance of a web server, several performance measures are of interest. Resource utilization of the CPU and local disk in the web server and of the network denotes the percentage of time in which each resource is busy processing jobs. Response time, i.e. the time from a client initiates a get-request by opening the TCP connection until the response is received, is also of interest. This is the delay observed from the point-of-view of a client. Response time is highly variable since it depends largely on the size of the document that has been requested.

5.3. Changing arrival rates

This section focuses on the performance of a web server under different workload intensities. Recall that we have assumed that requests arrive at a rate of 5-10 requests/second during peak hours. Let us consider how the performance of the web server may be affected by an increase in the arrival rate of requests.

Figure 8 shows the 95% confidence intervals for the average utilization of the different resources under different workload intensities. The disk utilization is largest and the network utilization is lowest for all arrival rates. When the arrival rate is 10 request/second, the average utilization of the server's disk is 10.67%, the average utilization of the
CPU of the web server is 7.66%, and the average utilization of the network is 1.98%. Increasing the arrival rate of requests to 50 requests/second, increases the average utilization of the web server’s CPU and disk to 40.25% and 56.03%, respectively. The average utilization of the network is also increased to 10.23%. The increase in utilization of the network is not as large as for the resources in the web server since the capacity of the network is quite large (5 Mb/sec.).

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Configuration</th>
<th>Configuration with lowest utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>fastcpu</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>fastdisk</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>cache</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>cpudisk</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>cpucache</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>diskcache</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. Comparison of configurations.

3. As in 1, but with a disk that is twice as fast.
4. As in 1, but using a cache that can hold a large percentage of the most frequently requested documents.

Ten independent simulations were run for each of these configurations, and several comparisons of the different configurations were made. Paired-t confidence intervals [19] were used to determine whether or not there is a significant difference in the utilization of both the CPU and disk for the configurations that were compared.

Table 1 provides an overview of the comparisons that were made and the conclusions that can be drawn from the comparisons. The first column in Table 1 is simply a name of the comparison. The numbers in columns 2-4 correspond to the numbered configurations above. Consider, for example, the second row in Table 1. The name of the comparison that has been made is fastdisk. In this comparison, configurations 3 and 1 were compared. The paired-t confidence intervals for this comparison reveal that the disk utilization for configuration 3 is lower than the disk utilization for configuration 1, as is expected. Based on the observations that have been made, there is no significant difference between the utilization of the CPU for the two configurations.

5.4. Alternative configurations

The CPN web model can also be used to analyze and compare the performance of alternative configurations of the web server. Suppose that the peak arrival rate of requests doubles to approximately 20 requests/second. The utilization of the resources of the server will increase, the quality of service that the web server can provide will be affected, and response times for clients may increase. Let us examine several alternative configurations of the web server. Suppose that the available options are:

1. The configuration of the CPN web model corresponding to the ITCHY web server.
2. As in 1, but with a CPU that is twice as fast.

6 Conclusions

In this paper we have presented a framework for construction of CPN models for performance analysis of distributed computing environments. It was demonstrated how the framework can be used to build a CPN model of a web server environment consisting of a HTTP web server and web clients connected to a LAN. In this modeling framework there is a clear separation within a model between the components which model the functionality of the system and the components which model the performance related aspects of the system. Thus, it is relatively simple to observe the functionality by simply choosing to disable the performance related components in the system. Carefully
designed experiments and the use of sound statistical methods allow us to make informed choices when we need to improve the overall performance of the system. The model can also be used to investigate the effect on performance of reconfiguring the web server in different ways.

It may be argued that the CPN web model is simplistic in the sense that it only considers a web server with clients connected to the same local area network as the web server. In practice, clients often communicate with a web server using the Internet. However, a main goal of the project was to compare the performance results obtained by simulations with the performance monitored in a corresponding physical environment. Therefore, a web server environment which could be setup and controlled within the scope of the project was important. Along these lines one may instead view the CPN model as modeling a web server for an intranet of a company.

Many simplifying assumptions have been made about the web server and its workload during this project. Future work with the web server model could include: using more realistic workload with proper arrival rates and considering temporal locality in the request stream, using a multi-threaded server, and modeling HTTP/1.1. Recent work investigates the impact of different designs of the TCP protocols [12] and caching strategies [2] on the performance of web servers. This is an aspect of web server performance which we have not considered in this paper. However, it is worth mentioning that with the proposed framework for construction of CPN models, which makes a clear distinction between system architecture, applications, and resources, could also serve as a foundation for investigating different web server design alternatives.

References


68