UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
INSTITUTE OF COMPUTER SCIENCE

Yuliya Brynzak

# Probabilistic Performance Testing of Web Applications

Master's Thesis (30 EAP)

Supervisor:
Dr. Michele Mazzucco, University of Tartu

Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . "‥‥‥" May 2011
Supervisor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . "‥‥‥" May 2011
Professor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . "‥‥‥" May 2011

TARTU, 2011

# Contents

# List of Tables

# List of Figures

# Abbreviations and acronyms

| | |
|---|---|
| QN | Queuing Network |
| QoS | Quality of Service |
| SLO | Service Level Objective |
| CBMG | Customer Behavior Model Graph |
| MVA | Mean Value Analysis |
| FCFS | First Come First Served |
| PS | Processor Sharing |
| MPL | Multiprogramming level |
| IP address | Internet Protocol Address |
| ISP | Internet Service Provider |
| ID | Identification Data |
| JMT | Java Modeling Tools |
| Regexp | Regular Expressions |
| API | Application Programming Interface |

# Abstract

Web systems are used widely for reaching different purposes, as remote access to information is very convenient. However, the remote access brings many aspects which should be handled. Users expect predictable performance levels (e.g., acceptable response time), therefore, service providers should know how their system performs under different loading conditions. In this thesis I design an analytical performance model and develop a tool which can solve that model. The tool allows analyzing the performance of web applications and answer the following questions:

1) What is the average response time of the system?
2) What is the utilization of the system as a whole?

The input parameters, such as the average service time of transactions, average arrival rate of requests, and the average think time, are estimated from a real workload (of a system under test). The performance model is developed by means of Queuing Networks, a framework which enables the analysis of a system in terms of mathematical formula.

# Chapter 1

# Introduction

Performance modeling, analysis and prediction is a significant part of web systems' creation and maintenance. Before taking decisions such as whether to add extra resources and, if yes, how many, it is important to detect performance issues and estimate how many resources are needed to meet agreed SLOs. Also, sometimes it is necessary to test how an existing system will work with a new configuration or understand how the system behaves under certain conditions. In order to do that, it is necessary to build a testing environment and perform several tests, which require a non-negligible effort, as, testing web systems requires human and technical resources, which are expensive.

In this thesis I propose a performance model for testing the performance of web systems. Performance models can be classified as either empirical or analytical. An analytic model suits better to the purpose of this thesis.

Web systems differ in their configurations and architectures. It is very hard to take into account all the details and characteristics of a certain system. However, it is always possible to approximate a system and estimate its performance. The analytical model I propose will capture fundamental aspects of web systems and relate them to each other by means of mathematical formulae.

Performance models require inputs in order to be solved. In order to obtain such data one should profile the system, e.g., gather information from system logs. When profiling is detailed the performance model produces more accurate results. In order to profile a system and obtain the input parameters for an analytical performance model, it is necessary to perform a number of operations such as analysis and deriving essential information from a particular workload.

To simplify the performance testing step, I propose to build a black box tool which, given a log file (web traces) as input, obtains relevant information and estimates performance metrics of interest.

As a basic pillar of the proposal I employ Queuing Networks, a mathematical framework that is used to model potential contention and queuing issues, when a set of resources are shared. QN allow to model web systems, where different jobs compete between each other in order to access various resources.

The validation of the performance model is performed by comparing the performance metrics estimated by the tool I will develop throughout this thesis and the empirical results obtained by stress testing a web application.

The rest of the thesis is organized as follows:

1) Chapter 2 provides information on performance models of web systems, the necessary background about performance modeling and queuing networks, and discusses relevant related work.
2) Chapter 3 discusses the design of the performance model and the tool I propose.
3) Chapter 4 presents the results of the designed performance model by running a set of experiments over a real web system.
4) Chapter 5 is dedicated to future work.
5) Chapter 6 contains some concluding remarks.

# Chapter 2
# Background and related work

## 2.1 Performance Modeling

Performance modeling is a structured approach for determining the behavior of a system. In other words it is the process of capturing performance related information. In the following I will discuss some of the non-functional requirements of systems such as availability, security, maintainability, and performance of systems, which are determined by the Quality of Service (QoS) requirements.

Availability is defined as the fraction of time that a system is available to its users. The two main reasons for it being unavailable are failures and overload. Since, it is hard to distinguish whether a system is slow or unavailable [16], web system should limit the number of requests, which are handled concurrently, in order to guarantee an acceptable level of performance for the requests in the system.

Security means that a system should implement authentication and related mechanisms in order to provide the confidentiality of the information. Maintainability means that it should be easy to extend a system in order to cope with new requirements. For example, when a system can not handle the current load, more resources should be added. Such operation should be easy to perform.

In this thesis I focus on performance. When the system performance degrades (e.g., the average response time exceeds a predefined limit), it becomes necessary to assess its performance. In this case, it is efficient to solve a performance model for the system.

Performance models are an abstraction or approximation of real systems and their level of details and specific aspects depend on the purpose the models are built for. There are two major types of performance models: simulation and analytical [13]. Simulation models are based on computer programs that emulate different dynamic aspects of a system and its static structure. Due to the level of details generally necessary, simulators are usually expensive to develop, validate and run.

On the other hand, analytical models are based on a set of formula or algorithms that provide the values of desired performance metrics as a function of some parameters. Analytical models can be less accurate than simulations, but are certainly more effective and not so expensive. Among the main performance attributes of web systems are response time, utilization and throughput. These values are used for analyzing the performance and verifying that the QoS requirements have been fulfilled.

The response time is a significant characteristic, as it determines the time after which a user will receive the response. In other words the response  time is the time it takes for a system to react to a request. In order to have good performance and keep users satisfied the response time should be small and predictable. The response time is composed of three elements: browser time, network time and service time (Figure 2.1). Browser time consists of processing and I/O time required to send the request and display the result page. The network time includes the time spent for the transmission from browser to user's Internet Service Provider (ISP), and the time spent in the network [1]. Finally, the service time includes all the operations involved with processing the request on the server side.

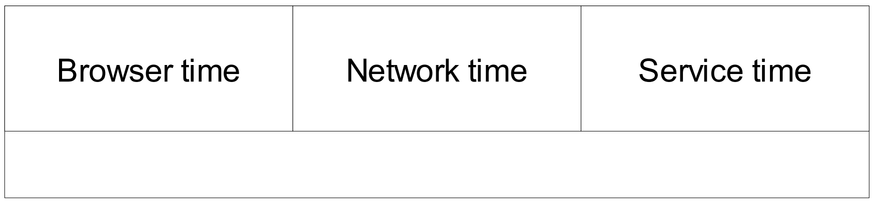


*Figure 2.1 - Response time breakdown*

The throughput is defined as the average amount of work performed by the system. That is, the number of requests processed by the system per unit time. The throughput depends on many factors, including the nature and size of the requests, or the number of users.

The system utilization is the total time that a resource of the system has been busy during a certain observation period. The system utilization is a number whose value ranges in the interval [0,1].

## 2.2 Queuing Network

Complex systems may be represented as a set of queues where every queue represents a resource (device, from now on), which is used to execute requests (jobs, from now on). QN describes system in terms of queues. Queuing network consists of several service centers related between each other by a path. Jobs move through the network from one queue to another in order to obtain the required service, while jobs are served according to a certain scheduling strategy (e.g., First Come First Served or Processor Sharing). An example of a queuing network is shown in Figure 2.2 .

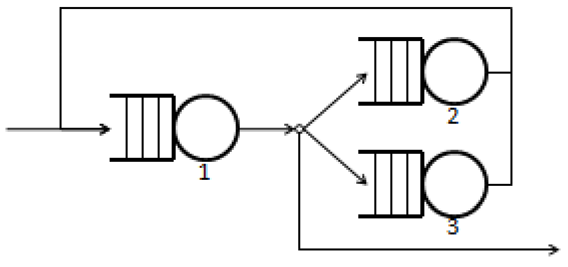


*Figure 2.2 - Simple queuing network*

The most important parameters of a QN are:
1) arrival process;
2) service process;
3) number of servers.

Arrival process determines how the user enters the system. The service process determines the user service times in the system. The number of servers determines the number of servers in the system.

QN can be classified according to the topology of the underlying graph and to the nature of the job population [5]. There are two types of queuing networks: closed and open. Queuing networks without external arrivals and departures, but with a certain number of jobs circulating among the nodes, are said to be closed (Figure 2.3 (a)). In contrast, open queuing networks, shown in Figure 2.2(b), are networks where there is at least one arc along which jobs enter the network and at least one arc along which jobs leave the system, and there is a path which allows the entered jobs to leave the system.

However, to model real systems, open and closed queuing networks are used at the same time, because of the mixed nature of the systems. Combination of the open and the closed models is called partly-open queuing network [2]. Partly-open queuing networks (see Figure 2.3 (c)) are not open because some jobs may never leave the network, while they are not closed because new jobs are allowed to enter the system.

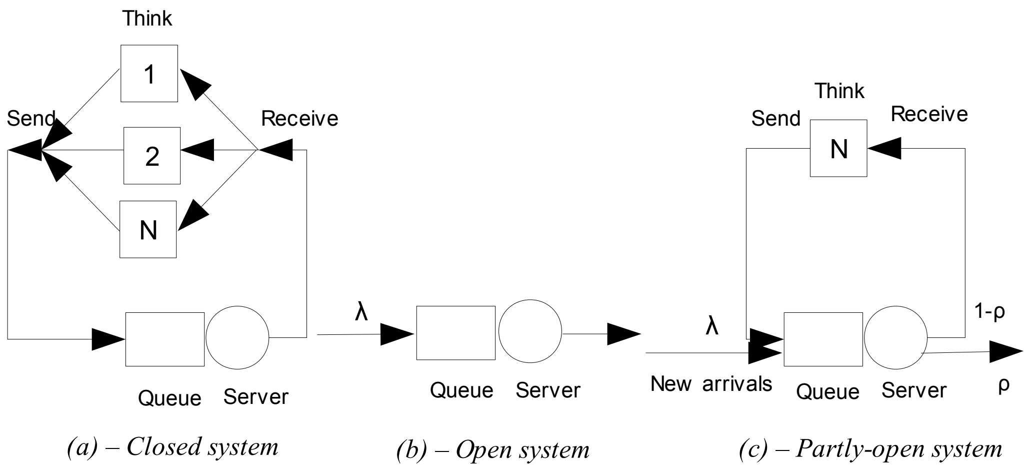


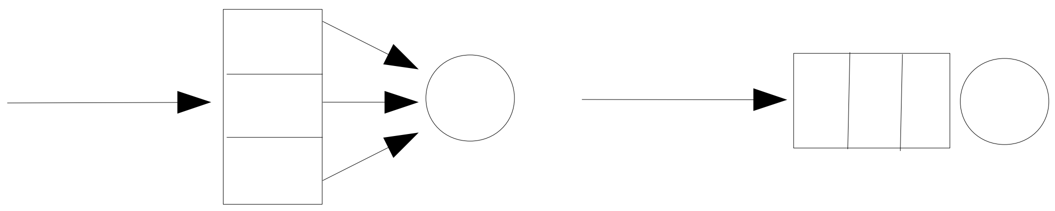
*Figure 2.3 - Illustration of closed, open and partly-open queuing networks*

In closed queuing networks, new jobs are triggered when jobs are completed, and followed by a think time. The think time is defined as the interval between a response and the following request. In contrast to closed systems, in open systems new jobs arrive independently of previous jobs. In an open system, the influence of think time is negligible, and thus it is omitted. The performance of open systems depends on the rate at which new jobs enter the system, while that of closed systems depends on the number of users which produce jobs. This number is known as multiprogramming level (MPL) and denoted by N, see Figure 2.3(c) [22].

Open and closed models when run under the same workload and service times parameters bring significantly different results. Thus, it is important to fix the principles which influence the performance according to queuing network [2].

The first principle is that the distribution of the file sizes, which in its turn affects the service times, significantly influence the average response times in open based models, but has a low impact in closed based models.

The second principle is about the impact of the scheduling policy on the performance of closed and open systems. There are two types of scheduling policies usually used to model systems, First-Come-First-Served (FCFS), see Figure 2.5 (a) and Processor Sharing policy (PS), see Figure 2.5 (b). According to FCFS, jobs are served in the same order as they arrive. PS assumes that jobs are served on a processor in equal quanta of time. In other words, if there are *n* jobs in the system throughout an interval of length *x*, then during that interval each of the *n* jobs receives an amount of service equal to *x*/*n* (e.g., each job receives service at rate 1/*n*) [14]. When the size of the quanta approaches infinity, PS becomes FCFS.



*(a) – Processor Sharing* *(b) - First-Come-First-Served*

*Figure 2.4 - Scheduling policies*

In FCFS small jobs would have to wait for large jobs to complete. In contrast, PS is better for dealing with large job sizes as small jobs are not stuck behind large ones.

## 2.3 Related Work

As web systems become more popular, more demand exists for researching the area of performance engineering. A lot of work has been done in the area of capacity planning and performance analysis.

## 2.3.1 Capacity planning models

[3] proposes a capacity planning framework, which is based on three components:

1) a workload profiler, used to build the workload profile;

2) a regression-based solver, which is used to derive the CPU demand of transactions on a given device;

3) and an analytical model, which is based on a queuing network, where the queues represent different tiers of the system.

This approach aims at estimating the behavior of web systems and help to predict their future performance with the system logs. This work tries to answer the following questions:

1) how many clients can be supported using the current number of requests?

2) does the existing system have enough available capacity for processing additional service for N number of clients?

3) if the current client population doubles, what the average response time will be?

These questions have to be taken into consideration in order to build or maintain a web system, because such aspects in particular circumstances can significantly harm the availability.

The capacity planning framework presented in [3] is aimed at practical capacity evaluation of existing production systems under real workload. The framework, instead of characterizing the overall service time of every server for transactions, uses a statistical regression method to approximate the service cost of individual core transactions. Core transactions are defined as those which are frequently executed. The performance model is modeled as a closed system because of session-based behavior of users. Thus, the performance model is solved by means of MVA.

To apply this framework a service provider should collect logs and CPU utilization at all tiers. It would be useful to avoid direct measuring of devices utilization as it requires additional software available on a server, but retrieve service demands of different transaction types with a simple approach, such as that we present in this thesis.

[4] proposes a model based on regression analysis to translate SLOs into design and operational policies for multi-tier applications. Since web applications vary and can have different complexities, the model is general enough to capture the behavior of different applications. This approach handles both request-based (open model) and session-based (closed model) workloads in contrast to [3] where only a session-based model is available despite the web application's origin. The analytical performance model is used here to relate high-level

performance goals such as the average response time, the application topology, and the resource usage of all components.

Queuing network models are developed here as a multi-station queuing center. The type of a queuing network model is chosen on the base of properties of tested workload. In other words how the system behaves, more as an open based model (which depend on requests rate) or as a closed-based model (which depends on number of terminals/clients in the system).

In [4] the authors profile a web application for generating a resource demand of every transaction type for every resource. In order to obtain the parameters at each resource, collection of utilization data from each resource is performed. To obtain these parameters, system and application monitoring logs should be available, which might be not always available in particular circumstances. After completing the system's profiling, and the performance model is specified for a certain workload, the solver outputs low-level settings for the systems. That is, it should be clear which resource requirements the system should have, for instance, how many servers to allocate to each tier in order to meet agreed SLOs.

[12] presents an approach for planning the capacity of multi-tier applications, which is based on network of queues, by means of which the authors represent how multi-tier applications cooperate for processing requests. The model is designed to handle session-based model with MVA approach. The model parameters are estimated from monitoring the tiers and using workload logs. Additionally the authors use special logs from Java applications involved in the system. Service time at each tier is estimated as the fraction of time the job spends by receiving service from the tier. Service time is estimated from the last tier which does not have sub-tiers and continues with the other tiers backward.

[17] uses a M/GI/1 queue with Processor Sharing (see previous page) for estimating the performance of 3-tiered Web sites. This work employs classical control theoretic techniques to design Proportional Integral (PI) controller for admission control of HTTP clients. The PS model is used for self-tuning the controller, and thus no parameter is required for the target response time. The controller is, simply, a lightweight HTTP proxy which is faster than regular HTTP proxies. All HTTP requests from the client are directed towards this proxy, which then relays them to the Web and application server after the control decision.

Traditional capacity planning methodologies as in papers [19] and [20] involve variations in load under typical behavior of the system, and examine peak loads and system utilization to come up with the number of users the system can handle. In [19] application servers are modeled as a M/G/1 queue with two methods of allocating a fixed number of servers within variety of customers with different service demands. In the first method, the average response time is calculated, and in the second method, the variance of response time is calculated.

## 2.3.2 Modeling tools

The Java Modeling Tools (JMT) is an open source suite consisting of six tools for performance evaluation, capacity planning, workload characterization, and modeling computer and communication systems [21]. The suite implements several state-of-the-art algorithms for the exact simulative analysis of queuing networks, either with or without product-form solution. Models can be described either through wizard dialogs or with a graphical user interface. The workload analysis tool is based on clustering techniques. The suite incorporates an XML data layer that enables full re-usability of the computational engines.

The JMT is composed of the following tools:

1) JSIMgraph: queuing network models simulator with graphical user interface. It allows an easy description of the network structure, as well as simplified definition of the input and execution parameters.

2) JSIMwiz: queuing network models simulator with wizard-based user interface. It supports several probability distributions for characterizing service and inter-arrival times. Also it allows one to choose between load-dependent strategies. The simulation engine supports several extended features, which are finite capacity regions (blocking), fork-join servers (parallelism) and priority classes. The JSIM performs automatically the transient detection, computes and plots on-line the estimated values.

3) MVA: Mean Value Analysis of queuing network models, for exact analysis of single-class or multi-class product-form queuing networks, processing open, closed and mixed workloads.

4) JABA: Asymptotic Analysis of queuing network models which is meant for the identification of bottlenecks in multi-class closed product-form networks using efficient convex hull algorithms.

5) JWAT: Workload Analysis from log and usage data. It supports the workload characterization process. Here are implemented algorithms for data scaling, sample extraction, outliers filtering, k-means and fuzzy k-means clustering for identifying similarities in the provided input data.

6) JMCH: Markov chain simulator (didactic tool). It applies a simulation technique to solve a single station model, with finite (M/M/1/k) or infinite queue (M/M/1), and shows the underlying Markov Chain.

JMT is good in order to analyze or predict performance of web systems. It provides variety of techniques and gives an opportunity to model different aspects of the systems. In general, the tool is very useful for the performance modeling, but it is not very convenient in terms of getting performance solved in simple way. Moreover it does not allow to build user interaction models which is an important aspect in solving a performance model designed in this thesis.

# Chapter 3
# Performance model

## 3.1 Workload analysis

This section covers the workload analysis. In particular, I will discuss how to process log files and assess the performance related data. I describe how to parse logs by means of Regular Expressions, explain the procedure of deriving user sessions from the logs, and show how the user interaction graph is built. The user interaction model is used to explain how to estimate the average user population in each node of the graph. That value is necessary for calculating the service time for the average user session.

### 3.1.1 Log files

There are several aspects of web systems performance which may be analyzed by means of logs (or traces). Web logs (Apache HTTP Server log files [27], for example) contain information about user activities on a web site, that is, how users navigate through the site (see Figure 3.1).

In this thesis I consider the Apache log format. As the first element of each log entry, is the IP/DNS, which may be represented either as "129.188.154.200" or "smyth-pc.moorecap.com", for example. The second element is the timestamp, e.g., the time at which the server has received the request. It includes the time correction, "-0400", according to the timezone. The third element indicates the requested element of a site, e.g., the URL of the resource. The fourth element is the status code. The status codes are numerical codes indicating the status of the requested element. For example:

- 200 indicates that the request is succeeded and the response was delivered;
- 404 means that there is no such an element on the server or the element could not be found;
- 304 indicates that the response has not been modified from the last visit and no content was returned from the server. Instead the requested content might be taken from the cache.

There are many other status codes. For more information, please refer to [18].

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html
HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif
HTTP/1.0" 200 4179
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0"
200 0
205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0"
200 3985
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200
40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200
786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200
1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

*Figure 3.1 - Example of Apache logs*

The fifth element describes the size of the HTTP response in bytes. When the status code of the request is 304, which means that the requested file was not changed, than the fifth element contains “-”, because no bytes have been transferred from the server to the user machine.

## 3.1.2 User sessions

A user session is a set of activities which a user makes during a web site visit. Activities are determined as user transactions, e.g., a request or several requests submitted by a user to a server .

Sessions can not be identified by IP addresses (or DNS) as we would want to, because the same IP address in the logs can “hide” different sessions. However, it is possible to approximate the sessions with the following method.

The duration of each session is different. Every user decides when to make a transaction and when to leave the system. I propose to consider a new session every time the user's think time exceeds 30 minutes. When the logs are parsed, each session will obtain a different session ID. Figure 3.2 shows the derived sessions where 1, 2 and 3 are session IDs.

```
 1 = GET /w ordpress/ HTTP/1.1 | Wed May 04 19:04:14 EEST 2011 | 83130 | 193.40.10.179  ||  GET
/w ordpress/?page_id=8 HTTP/1.1 | Wed May 04 19:04:19 EEST 2011 | 8618 | 193.40.10.179  ||
GET /w ordpress/?page_id=12 HTTP/1.1 | Wed May 04 19:04:22 EEST 2011 | 14034 | 193.40.10.179  ||
GET /w ordpress/?page_id=29 HTTP/1.1 | Wed May 04 19:04:28 EEST 2011 | 10031 | 193.40.10.179  ||

2 = GET /w ordpress/ HTTP/1.1 | Wed May 04 19:04:20 EEST 2011 | 7528 | 95.30.193.60  ||  GET
/w ordpress/?page_id=10 HTTP/1.1 | Wed May 04 19:04:25 EEST 2011 | 9666 | 95.30.193.60  ||
GET /w ordpress/?page_id=27 HTTP/1.1 | Wed May 04 19:04:29 EEST 2011 | 10487 | 95.30.193.60  ||
GET /w ordpress/?page_id=29 HTTP/1.1 | Wed May 04 19:04:32 EEST 2011 | 9950 | 95.30.193.60  ||

3 = GET /w ordpress/ HTTP/1.1 | Wed May 04 19:04:22 EEST 2011 | 82921 | 93.72.195.158  ||  GET
/w ordpress/?page_id=8 HTTP/1.1 | Wed May 04 19:06:19 EEST 2011 | 10760 | 93.72.195.158  ||
GET /w ordpress/?page_id=12 HTTP/1.1 | Wed May 04 19:06:24 EEST 2011 | 10487 | 93.72.195.158  ||
GET /w ordpress/?page_id=29 HTTP/1.1 | Wed May 04 19:06:30 EEST 2011 | 10181 | 93.72.195.158  ||
```

*Figure 3.2 - Example of user sessions retrieved from a log file*

The user sessions are necessary for building the user activities graph, which shows the structure of a web application and helps us to estimate the average total service time. This graph allows us to know how users make their decisions and how they move through the application, and which transactions they execute. Moreover, we can estimate values such as the average arrival rate, think times between requests in a session, and file sizes for each request.

### 3.1.3 Transaction types

One of the issues arising when processing logs is deriving the transaction types. As shown in Figure 3.3, the user with the IP address “193.40.10.9” entered the system with "GET /wordpress/ HTTP/1.1".  This is a single request done by the user, but it triggers the following requests:

- GET/wordpress/wp-content/themes/twentyten/style.css HTTP/1.1;
-GET/wordpress/wp-content/themes/twentyten/images/headers/path.jpg HTTP/1.1;
-GET/wordpress/wp-content/themes/twentyten/images/wordpress.png HTTP/1.1;

```
193.40.10.9 - [04/May/2011:19:04:14 +0300] "GET /w ordpress/ HTTP/1.1" 200 7528
193.40.10.9 - [04/May/2011:19:04:14 +0300] "GET /w ordpress/w p-
content/themes/tw entyten/style.css HTTP/1.1" 200 22817
193.40.10.9 - [04/May/2011:19:04:14 +0300] "GET /w ordpress/themes/tw entyten/headers/path.jpg
HTTP/1.1" 200 51727
193.40.10.9 - [04/May/2011:19:04:14 +0300] "GET /w ordpress/w p-
content/themes/tw entyten/w ordpress.png HTTP/1.1" 200 849
193.40.10.9 - [04/May/2011:19:04:19 +0300] "GET /w ordpress/?page_id=8 HTTP/1.1" 200 7315
193.40.10.9 - [04/May/2011:19:04:19 +0300] "GET /w ordpress/w p-includes/js/l10n.js?ver=20101110
HTTP/1.1" 200 308
193.40.10.9 - [04/May/2011:19:04:19 +0300] "GET /w ordpress/w p-includes/js/comment-reply.js?
ver=20090102 HTTP/1.1" 200 786
193.40.10.9 - [04/May/2011:19:04:19 +0300] "GET /favicon.ico HTTP/1.1" 404 209
95.30.193.5 - [04/May/2011:19:04:20 +0300] "GET /w ordpress/ HTTP/1.1" 200 7528
93.72.195.8 - [04/May/2011:19:04:21 +0300] "GET /w ordpress/ HTTP/1.1" 200 7528
93.72.195.8 - [04/May/2011:19:04:21 +0300] "GET /w ordpress/w p-
content/themes/tw entyten/style.css HTTP/1.1" 200 22817
93.72.195.8 - [04/May/2011:19:04:21 +0300] "GET /w ordpress/w p-
content/themes/tw entyten/images/path.jpg HTTP/1.1" 200 51727
```

*Figure 3.3 - Server logs*

In this example the triggered requests are design elements of the requested page. Then I estimate here that the transaction type is called by the submitted request and includes all requests triggered by it. All of them require some resources from the server and can influence the performance of the system. Thus, they cannot be ignored.

For example, all the requests are of *.css or *.jpg or *.js formats, according to the extensions. Css defines how the content will appear to the human being accessing the document. JPG is one of the images formats. Several image formats exist, for example, gif, jpeg, png etc, which very often are built into web pages. As a consequence, they appear on logs when the page is requested. JS extension defines a file with Java Script program.

The formats can be easily recognized with regular expressions, also known as “regexp”. Regular expressions are a very flexible tool for recognizing and matching strings of a text, such as particular characters, words or patterns of characters.

For example, Figure 3.4 shows a regexp for matching the requests which possibly are elements of the page which have been requested.

```
.+?\\.(gif|png|css|jpeg|js|ico|jpg)\\s?.+?
```

*Figure 3.4 - A regular expression for matching triggered requests*

However, for example, JPG may be not a triggered request, but a separate transaction type. In this case when we parse the logs, we need not only to match the elements with that particular extension, but also check whether the request is a single one or if it was triggered from some other request. From Figure 3.3 we can see that after "GET /wordpress/ HTTP/1.1" request has been submitted at time [04/May/2011:19:04:14 +0300], the subsequent requests were triggered at the same time. Thus, we assume that if the request matches the regular expression and the time does not exceed 3 seconds after the previous request within the same session, than it belongs to the previous request. In all other cases the request is assumed to belong to a different transaction.

After it was defined that the request is an element of the other request we take the size of the request and add it to the value of size of the previous request. For example, as we already know the request "GET /wordpress/ HTTP/1.1" has size 7528 bytes and triggers the following requests:

– "GET /wordpress/wp-content/themes/twentyten/style.css HTTP/1.1" with size 22817 bytes;

– "GET /wordpress/wp-content/themes/twentyten/images/headers/path.jpg HTTP/1.1", size 51727 bytes;

– "GET /wordpress/wp-content/themes/twentyten/images/wordpress.png HTTP/1.1", size 879 bytes.

$$7528+22817+51727+879=82951\,bytes$$

It means that the transaction of type "GET /wordpress/ HTTP/1.1" will have size not 7528 but 82951 bytes.

### 3.1.4 User interaction model

The service time of a transaction is one of the required parameters for solving the performance model and its total value can be obtained by using a user interaction model, e.g., a model which describes the interaction between the users and the system. This model is useful, as it describes how users obtain service form the system.

The interaction between users and the system can be represented by means of a Customer Behavior Model Graph (CBMG), see Appendix 2 for more details. The CBMG model captures the possible states a user can reach, as well as the transitions (movements) between these states. It should be noted, however, that not all users behave in the same way. Users visit the web site according to the service they would like to obtain, thus the CBMG model can include cycles and several entry points (the states at which users arrive to the system). Different sessions can be described by means of the CBMG.

Each node of the graph (see Appendix 2) represents a transaction. Directed edges of the graph represent possible paths through which users receive different kinds of service, and are called "transfers". The values next to each node specify the probability with which a user might go to a specific state within the same session, and obtain one or another service by completing one or another transaction.

Once the sessions have been determined it is possible to calculate how many transfers have been made from one state to another. For example, see Appendix 2, assume that there are 10 user sessions, and each session first state is "A". That means that when the user starts a new session, his/her first transaction is "A". Each new user enters the system with the transaction, thus the probability of entering the system with that transaction is equal to 1. After some time (think time) the user decides whether to visit state "B" or state "C". According to the graph 70 % of sessions contain "B" and 30 % contain "C", which means that 7 of 10 users moved to the first state and 3 of 10 users moved to the second state respectively.

Transfers which are going from one state to outside of the graph mean that some of the sessions have ended on a certain state. For example, the outgoing edge to outside of the graph at the state "GET /wordpress/?page_id=12 HTTP/1.1" means that 20 % of the user sessions have ended once they had reached the state.

If we analyze the CBMG using the method discussed in [5] we can determine the average number of times per session that each state is visited. For example, for session of type "D" from Appendix 2, the average time the state is visited can be estimated as the fourth equation from Figure 3.5.

A = 1 entry
B = 0. 7 A
C = 0.3 A + 0.5 B
D = 0.3 B + 0.6 C
E = 0.3 C + D

*Figure 3.5 - Average time each state have been visited*

Now we can calculate the average total service time, which is equal to the sum of the average service times of each state.

In real world, users behave much different, therefore the resulting CBMG may be complex. As a consequence, it is not easy to find the average number of visits for each state. However, there is one technique which can help to overcome this problem.

### 3.1.5 Estimating the number of concurrent users

Figure 3.5 shows a system of linear equations. Thus, the population of the states (rate of jobs entering the state) can be derived by solving a system of linear equations with Gaussian elimination. Gaussian elimination is the elementary row operations which are used to reduce a matrix to row echelon form [7].

In the previous section, we determined the transaction probabilities between states (see Figure 3.6 (a)), thus, according to it we can create a map which will contain transactions (states) as keys and all possible incoming transactions to as values. An algorithm for deriving the incoming transactions map from the transfers probabilities map is shown in Figure 3.7, where the input parameter is the map containing the transition probabilities.

The Figure 3.6 (b) has to be interpreted as which portion of users will transfer from different states to a certain state. Consider for example third line from Figure 3.6 (b). 30 % of users move from state "A" to state "C", and 50 % of users move from state "B" to state "C". The sum of these portions will result in what portion of users come to "C" in total. That is to find by solving the system of equations, the value we want.

The equations (see Figure 3.5) are represented as follows. The left-hand part of each equation is its body, and the body contains variables and their coefficients. The right-hand part of each equation contains a resulting value of the equation.

```
enter → {A = 1}
A → { B = 0.7, C = 0.3}
C → { D = 0.6 , E = 0.3, out = 0.1 }
B → {out= 0.2, D = 0.3, C = 0.5}
D → { E = 1 }
E → { out = 1}
```

*(a) –Transfers probabilities map*

```
A → { enter = 1 }
B → { A = 0.7}
C → { A = 0.3 , B = 0.5 }
D → { B = 0.3, C = 0.6 }
E → { C = 0.3, B = 1}
out → {E=1,  B=0.2, C = 0.1}
```

*(b) – Incoming transactions map*

*Figure 3.6 - Conversion transition probabilities map to incoming transactions map*

```
Input:        [map0] MapOfProbabilities <String, <String, Double>>

Initialization:     [map1] <String, <String, Double>>;
                    [map2] <String, Double>

    for each  element of  [map0] {

            get [key] <String>
            get [value] <String, Double>

            for each [value] {

                    get [key2] <String> ;
                    get [value2] <Double> ;

            check whether [key2] in [map1]
                    no: {
                            new [map2] ;
                       }
            [map2].put([key], [value2]);
            [map1].put([key2], [map2]);
            }
    }
    return [map1];
```

*Figure 3.7 - Algorithm for converting transfers probabilities map to incoming transactions map*

From the incoming transactions map (Figure 3.6(b)) we have to obtain two matrices in order to solve the system of linear equations with Gaussian eliminations. The first matrix, two-dimensional one, contains the coefficients of the variables, and the second matrix, one-dimensional, contains the resulting values of the equations. The matrices are called coefficient matrix (see Figure 3.8(a)) and right-hand matrix (see Figure 3.8(b)) respectively.

$$\begin{array}{c|rrrrr} & A & B & C & E & D \\ \hline A & -1 & 0 & 0 & 0 & 0 \\ B & 0.7 & -1 & 0 & 0 & 0 \\ C & 0.3 & 0.5 & -1 & 0 & 0 \\ D & 0 & 0.3 & 0.6 & -1 & 0 \\ E & 0 & 1 & 0.3 & 0 & -1 \end{array} \qquad \begin{array}{c|r} A & -1 \\ B & 0 \\ C & 0 \\ E & 0 \\ D & 0 \end{array}$$

*Figure 3.8 - Coefficient and right-hand side matrices*

The rows of the coefficient matrix represent the equations, while the columns of the matrix represent the variables of the equation. We have as many equations as many variables, thus the coefficient matrix is always quadratic. If there is no particular variable (state) in an equation, then coefficient of it is equal to 0 in other case the coefficient is equal to the value of the coefficient (see Figure 3.8 (a)).

There are two special cases: when the state is “out”, that means that users leave the system, and when the state is “enter”, that denotes the entry states, from which users start to discover the system (because it is possible that there are more then one entry point).

We write the right-hand value only for equations which contain the variable “enter”. For the rest of equations the right hand value is equals to 0. For example, consider the first line from a Figure 3.6 (b) “A → {enter = 1}”. For this record we can write the following equation (see Figure 3.5):

$$A = 1 \times enter \qquad (3.2)$$

as “enter” = 1 always, thus from (3.2) we obtain

$$A=1 \qquad (3.3)$$

According to (3.3) the coefficient of variable A equals to 1 and the right-hand value is equal to 1. This is the first equation, so in a coefficient matrix it will belong to the first row and in the right-hand matrix to the first element of the matrix. For more convenience, we write the values with opposite sign. Thus, we assign -1 to the coefficient and -1 to the right-hand matrix according to the provided addresses in the matrices (see Figure 3.8).

Consider, for example, the second equation, i.e. "C → {A = 0.3 , B = 0.5)", which does not contain the "enter" state. Hence, we can write:

$$C=0.3\times A+0.5\times B \qquad (3.4)$$

Since A = 1, we can re-write Eq. (3.4) as

$$C-0.3-0.5\times B=0 \qquad (3.5)$$

Equation (3.5) computes the third row in the coefficient matrix and the third element of the right-hand matrix. Each coefficient is written with the opposite sign (see Figure 3.8).

The state "out" is ignored, as it is not suppose to be measured. In other words, we do not need to now which part of users in average leave the system.

As the linear equations are solved we obtain the fraction of users for each graph node. After that, we can estimate the total average service time for the average user session. This is the subject of the following section.

## 3.2 Service time

The service time is associated with the usage of hardware (e.g., CPU) and software (e.g., locks) resources. In other words, the service time of a transaction is the time the transaction spent on a resource in order to obtain service and does not include queuing time. When a transaction contains several requests then the service time is the sum of service times each request.

According to [6] the service time of HTTP requests depends on the amount of exchanged information. That means that the service times for different requests can be approximated from statistical data, which might be obtained by measuring the services times for files of different size.

In the following, the file sizes of various transactions are obtained from Web logs (see Section 3.1.1). The service times of different file sizes are measured with Tsung [8], an application that can be employed to test Web applications. More in detail, Tsung measures the average response time of each request type (i.e., a request for a specific file). The load on the server is kept to a low level so that it is possible to approximate the average service time as the measured response time (there is no contention on the server side). The measured value of the response time contains the connection time. Hence, in order to improve the quality of the approximation, the connection time is subtracted from the response time.

File sizes of possible transaction types of a system are obtained from web logs (see 3.1.1 Log files). With help of regression-based method we obtain service times for particular file sizes.

Service times are stochastic variables, generally distributed. However, if the estimates are accurate enough, the service times can be approximated as independent and identically distributed variables.

Service times of different file sizes can be measured by using a stress test tool, specified on testing the performance of IP based client/server applications called Tsung [8]. Thus, the idea is to create a set of files with different size. These files are located on a server and with the help of the tool the average response time of each request is recorded. The load on the server is small, therefore, the measured response time of each request can be used to approximate its service time. The measured value of the response time contains the connection time. However, if the connection time is subtracted from the response time then we obtain a value which is close to the service time.

Figure 3.9 shows the average service time as a function of the file size for various file sizes. In order to estimate the service time of a request which has not been previously measured we employ an interpolation method. Thus, by knowing the file size we can get service time from interpolated data by means of [14].

In order to estimate the service time of a request which has not been previously measured we employ polynomial interpolation, a method that allows the construction of new data points within the range of a discrete set of known data points.

The interpolation function is in the form

$$y=c[0]+c[1]*x+c[2]*x^2+c[3]*x^3+c[4]*x^4 \quad .$$

The parameters estimation as well as the degree of polynomial are performed at runtime by means of the Java Flanagan library (14). Given the parameters from (Figure 3.10), we obtain

$$y = 13.2915 - 0.0943\,x + 8.6756\text{E-}4\,x^2 - 8.6335\text{E-}7\,x^3 + 2.7594\text{E-}10\,x^4.$$

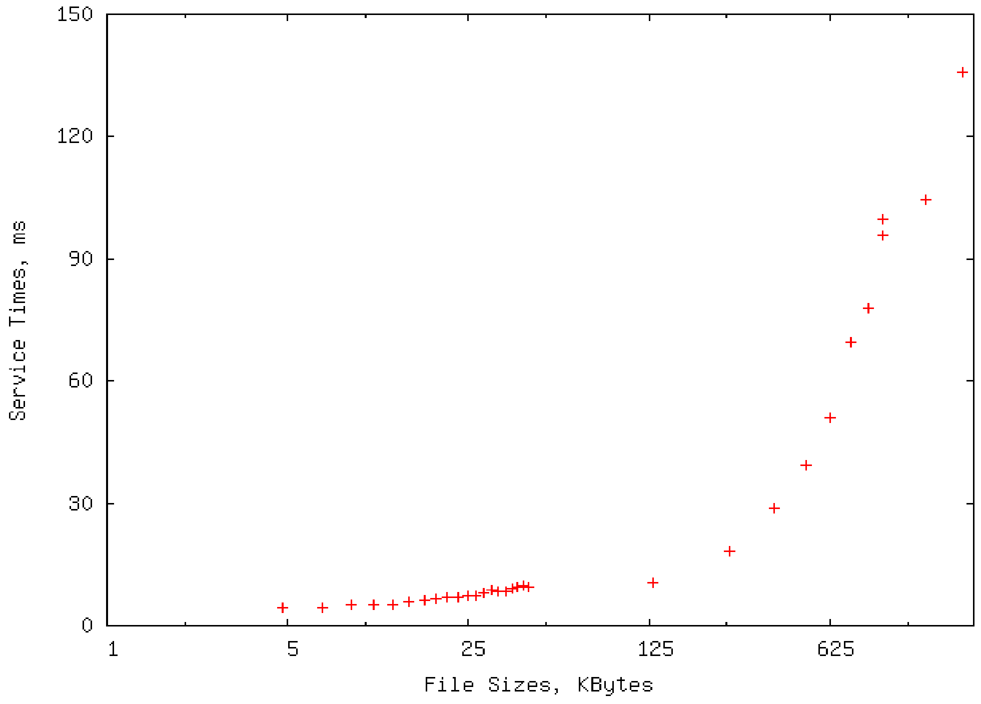


*Figure 3.9 - Service time dependency from file size requested*

Polynomial (with degree = 4)
Fitting: Linear Regression

| | Best Estimate |
|---|---|
| c[0] | 13.2915 |
| c[1] | -0.0943 |
| c[2] | 8.6756E-4 |
| c[3] | -8.6335E-7 |
| c[4] | 2.7594E-10 |

*Figure 3.10 - Polynomial and its coefficients for linear regression*

## *3.3 Model approximation*

The performance of an open web system is estimated by means of a M/G/1 PS queue, where "M" stands for memoryless. The memoryless property means that the future does not depend on the past, e.g., when a job has not arrived to a system yet we can not say how much time it left for the job to arrive. "G" indicates that the service time distribution is general, with finite variance.

The M/G/1 is queuing model characterized by the following assumptions:

- jobs arrival rate is exponentially distributed with parameter λ. For t >= 0, the probability density function is

$$f(t)=\lambda\times e^{-\lambda\cdot t} \qquad (3.6)$$

- the service times are independent and identically distributed random variables with some general distribution function;
- the buffer is of infinite size;
- the number of potential jobs is infinite.

The utilization of the devices should be less then 1, that is, $\rho < 1$. If that is not a case, the device is overloaded, i.e., the queue grows unbound. Figure 3.11 [10] shows a single-server system with Processor Sharing scheduling.

The average arrival rate of the customers is denoted as λ. The arrival process, formed by merging together requests from a large number of independent sources, is approximately exponentially distributed [13]. When users arrive to a server they are served in equal quantity of time, when the server is not idle. The average required job service time is denoted by S ( or E[S], where "E" stands for expected).

The offered load, that is the average amount of work arriving into the system per unit time, is utilization. According to Utilization Law [15] traffic intensity will be equal to

$$\rho=\lambda\times E[S] \qquad (3.7)$$

According to Little's Law [11] the average response time, W, for the system with Processor Sharing scheduling policy can be estimated as

$$W=\frac{E[S]}{1-\rho} \qquad (3.8)$$

In other words λ and E[S] are the parameters which influence the average performance of M/G/1 system.

The model employs Processor Sharing (PS) queuing discipline, which was discussed in sections 2.2. I use PS because it is a reasonable approximation to represent processor scheduling disciplines of modern operating systems.

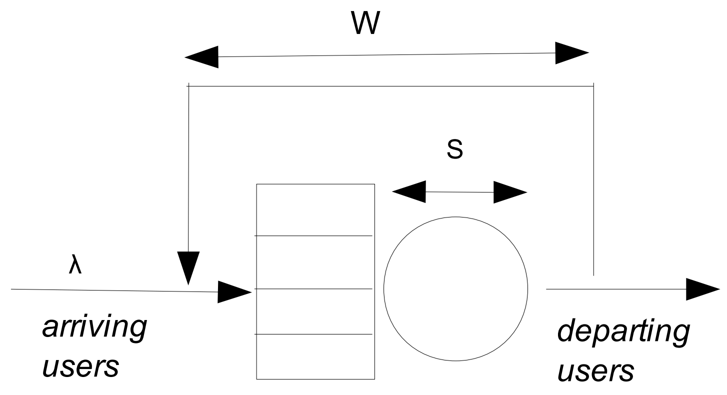


*Figure 3.11 - Processor Sharing in a single server system*

# Chapter 4

# Performance Evaluation

This chapter discusses several experiments which were coined out with the aim of evaluating the tool designed in this thesis. The experiments show that my tool (P0.1 further on) can be used for estimating the performance of web applications and it produces results close to those measured by stress test tools.

Before running the experiments we need to create a web site with some workload on a remote server. The web site creation is based on WordPress technology (more details next page). We assume that workload of the site consists of 11 web pages.

The specifications are:

- CentOS, release 5.5 (Final)
- Apache HTTP Server, version 2.2.18;
- MySQL
- WordPress 3.1.2;
- PHP 5.3.6.

To measure the performance of the web site, we use a load generator tool Tsung, version 1.3.3 [8]. It allows to specify the following parameters in an execution file:

1) inter-arrival interval;
2) transaction types;
3) think time between transactions;
4) sessions;
5) probabilities of choosing particular session.

Tsung produces the average response time of the web site. We will measure the average response time with different values of arrival rate. After the experiment we obtain the following results:

1) average response times for different traffic intensities;

2) log files with user sessions with respective arrival rates;

After we obtained the results with Tsung we run a set of experiments with P0.1 according to the logs obtained from the previous set of experiments. We get a set of results showing the average response time as a function of the arrival rate.

Afterwards, we analyze the results and show the similarities between the results obtained from the experiments and the estimates an approximate error of measures.

## 4.1 Workload

We build a web site which contains 11 pages. Each page has a unique name to recognize it and unique content in order to have different sizes of the pages (requests). The web site is deployed on a web server running the Apache HTTP server [25]. I restart the server each time we perform some measures over it. Restarting the server prevents from different aspects, which may influence accuracy of results[26].

The web site is built using WordPress, a popular open source application implementing a blog and running on top of the LAMP stack [24]. LAMP is an acronym for a solution stack of free, open source software, originally coined from the first letters of Linux (operating system), Apache HTTP Server, MySQL (database software) and PHP, principal components to build a viable general purpose web server.

| Page name | Page URL | Page size |
|---|---|---|
| / * | http://ats.cs.ut.ee:11018/wordpress/index.php | 7528 |
| A | http://ats.cs.ut.ee:11018/wordpress/?page_id=12 | 10011 |
| B | http://ats.cs.ut.ee:11018/wordpress/?page_id=14 | 10194 |
| C | http://ats.cs.ut.ee:11018/wordpress/?page_id=27 | 10487 |
| D | http://ats.cs.ut.ee:11018/wordpress/?page_id=35 | 10804 |
| E | http://ats.cs.ut.ee:11018/wordpress/?page_id=29 | 10165 |
| F | http://ats.cs.ut.ee:11018/wordpress/?page_id=37 | 10084 |
| G | http://ats.cs.ut.ee:11018/wordpress/?page_id=31 | 10534 |
| H | http://ats.cs.ut.ee:11018/wordpress/?page_id=8 | 7483 |
| K | http://ats.cs.ut.ee:11018/wordpress/?page_id=10 | 9666 |
| Z | http://ats.cs.ut.ee:11018/wordpress/?page_id=33 | 9950 |

*Table 4.1-Web Pages of a Web site*

Table 4.1 shows the web pages of the web site. Each page has a name (see the first column in the table) for simplifying its recognition. The symbol “/” denotes the index-page.

After users visit the web site, they choose whether to go through the site further and which web pages to visit. In our case the traffic is synthetic, e.g., generated by Tsung (see next page for more details). Thus, we defined 8 sessions, which are displayed in Figure 4.1.

1 / → H → A → E → G
2 / → H → A → B → C → D → F
3 / → H → C → G
4 / → H → C → D → F
5 / → K → C → G
6 / → K → C → D → F
7 / → K → C → Z → F
8 / → K → C → Z

*Figure 4.1 - User sessions*

From the sessions defined above we can draw a user interaction graph which shows which options a user can have in order to move through the web site (see Figure 4.2).

This graph may be characterized as follows:

– it has one entry node “/”, that is index page, where users enter the graph;

– it has three nodes “Z”, “F” and “G”, after visiting which, users leave the graph (the web site).

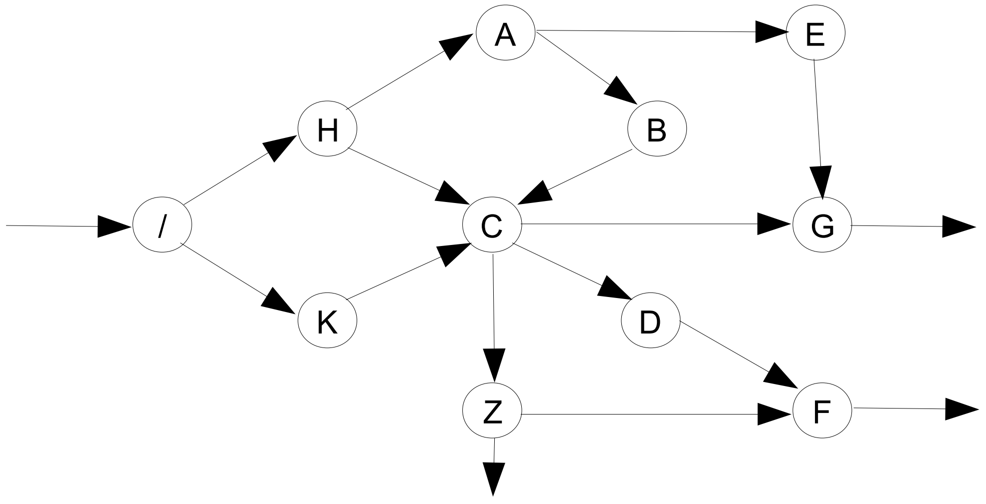


*Figure 4.2 - User interaction graph*

## 4.2 Measuring performance with Tsung

### 4.2.1 Setting up Tsung

Tsung is a distributed load testing tool, which allows to stress web applications. It is able to emulate a large number of simultaneous users from a single CPU. It includes a set of tools which can measure the average response time. We set up Tsung with its configuration file.

Tsung configuration file is in XML format. It contains a set of parameters. The first parameter is the inter-arrival interval, e.g. the reciprocal of the arrival rate. For example, if the arrival rate is 2 sessions/seconds, then the inter-arrival interval between two consecutive sessions is 0.5 sec on average.

Inter-arrival intervals, in Tsung, are distributed exponentially. For example, if we specify inter-arrival interval as 0.5, see Figure 4.3, then the inter-arrival interval will be distribute exponentially with mean equals to 0.5, while new arrivals will be generated for 60 minutes.

```
<arrivalphase phase="1" duration="60" unit="minute">
        <users interarrival="0. 5" unit="second"/>
</arrivalphase>
```

*Figure 4.3 - Inter-arrival parameter of Tsung configuration file*

After we have specified the inter-arrival interval, we have to define the sessions. Each session takes place with a certain probability. For example, the probability that a user takes the path " / → H → A → E → G " is equal to 0.16 (see Appendix 1). In total, all probabilities of the sessions have to be equal 1. We assume that all sessions have the following probabilities:

1) / → H → A → E → G = 0.16 ;
2) / → H → A → B → C → D → F = 0.15;
3) / → H → C → G = 0.07;
4) / → H → C → D → F = 0.12;
5) / → K → C → G = 0.14;
6) / → K → C → D → F = 0.13;
7) / → K → C → Z → F = 0.15;
8) / → K → C → Z = 0.08.

### 4.2.2 Running test

We run 17 tests with increasing load and obtained the results shown in Table 4.2. The load increase was obtained by increasing the rate at which new sessions enter the system. Every run lasts 60 minutes.

In Tsung, the average response time for requests is computed every 10 seconds. That is why we have the highest average and lowest average values in the statistic report.

Table 4.2 shows that the highest average response time is not correlated with the arrival rate, when the load is light, while the lowest average response time is. Moreover, only when the load approaches power saturation (it is 11 sessions/second), then the highest average response time has a correlation with the arrival rate.

| Arrival rate (sessions/second), $\lambda$ | Duration (minute), $\tau$ | Average highest response time, **Wh** (second) | Average lowest response time, ***Wl*** (second) | Average response time, ***W***, (second) |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1.05 | 60 | 0.85 | 0.05 | 0.23 |
| 1.2 | 60 | 2.53 | 0.12 | 0.21 |
| 1.3 | 60 | 0.8 | 0.13 | 0.22 |
| 1.4 | 60 | 0.75 | 0.15 | 0.22 |
| 1.9 | 60 | 1.44 | 0.18 | 0.24 |
| 2.2 | 60 | 3.78 | 0.16 | 0.23 |
| 2.5 | 60 | 1.02 | 0.18 | 0.25 |
| 2.6 | 60 | 0.82 | 0.18 | 0.23 |
| 3.1 | 60 | 1.36 | 0.18 | 0.25 |
| 4 | 60 | 0.79 | 0.2 | 0.26 |
| 4.9 | 60 | 1.66 | 0.21 | 0.33 |
| 5.3 | 60 | 1m38s | 0.19 | 0.33 |
| 5.9 | 60 | 2.68 | 0.17 | 0.29 |
| 6.6 | 20 | 4.95 | 0.19 | 0.31 |
| 9.2 | 60 | 6.53 | 0.17 | 0.68 |
| 10 | 60 | 8.03 | 0.24 | 1.89 |

*Table 4.2 - Performance metrics measured by Tsung*

Figure 4.4 shows the graphs of the lowest average response time and the overall average response time. The lowest arrival rate for heavy and light load varies not significantly, while the overall arrival rate significantly differs for heavy and light loads.

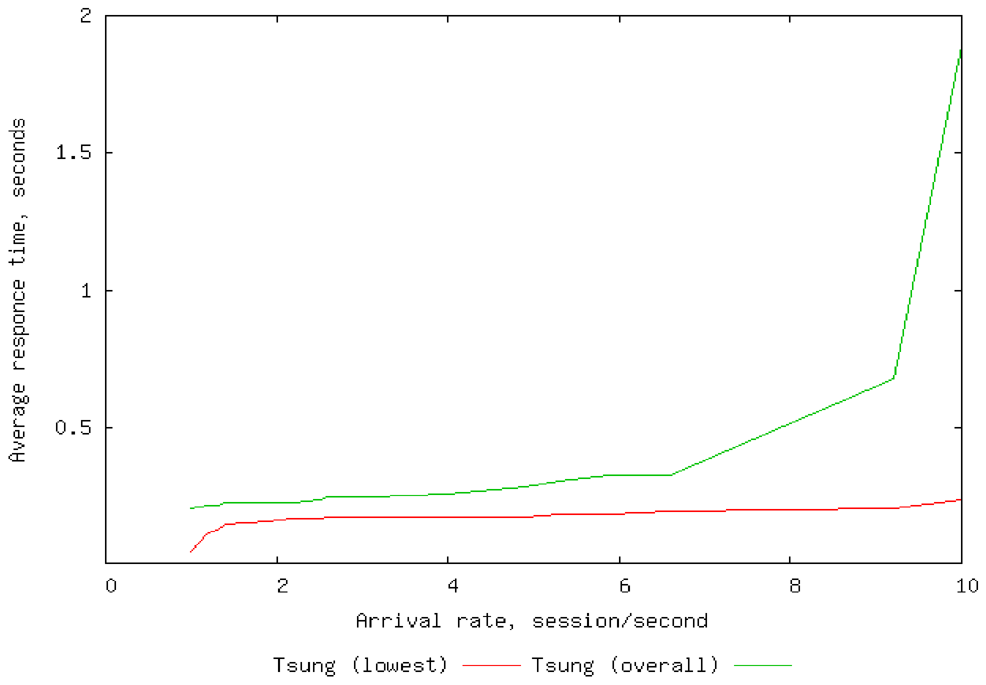


*Figure 4.4 - Average response time measured by Tsung for the increasing arrival rate*

## 4.3 Performance prediction by P0.1

To estimate the performance of the system with P0.1 we need logs and statistical data about service times for different requests sizes.

P0.1 gives the following results after conducting a set of experimental estimations, see Table 4.3.

P0.1 tool executes in several steps.

1) parses the logs with regular expressions (see section 3.3.2);

2) creates a user sessions map and defines transaction types (see section 3.3.2);

3) calculates the probabilities of transfers of users from one state to another (see section 3.1.);

4) estimates service time for every transaction type (see section 3.1.4);

5) calculates the average total service time of the average user session;

6) solves performance model, and outputs the average response time and the utilization of a resource

We predicted the average response times and the utilization for different arrival rates with P0.1, see Table 4.3.

| Experiment with P0.1 | | |
|---|---|---|
| Arrival rate, sessions/second | Response time, second | Utilization |
| 1.05 | 0.11 | 0.1 |
| 1.2 | 0.11 | 0.11 |
| 1.3 | 0.113 | 0.14 |
| 1.4 | 0.114 | 0.15 |
| 1.9 | 0.12 | 0.18 |
| 2.2 | 0.123 | 0.18 |
| 2.5 | 0.133 | 0.27 |
| 2.6 | 0.134 | 0.28 |
| 3 | 0.14 | 0.29 |
| 4 | 0.16 | 0.38 |
| 4.9 | 0.19 | 0.38 |
| 5.3 | 0.2 | 0.43 |
| 5.9 | 0.23 | 0.49 |
| 6.6 | 0.27 | 0.63 |
| 9.2 | 0.77 | 0.87 |
| 10 | 3.35 | 0.97 |

*Table 4.3 - The average response times predicted by P0.1 for different arrival rates*

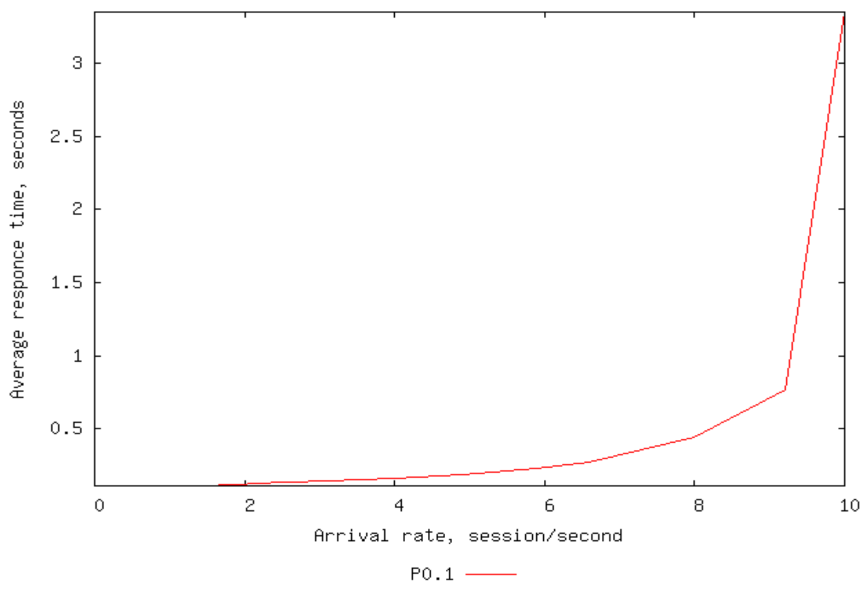


*Figure 4.5 - Average response times predicted by P0.1*

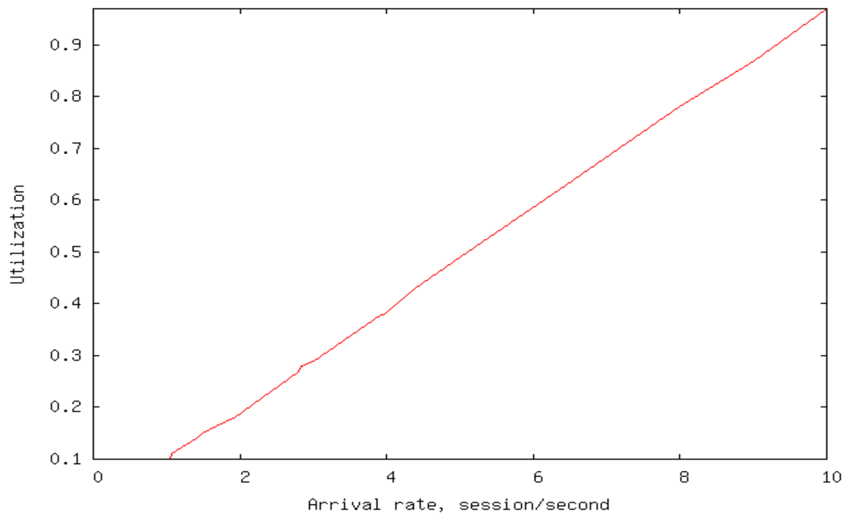


*Figure 4.6 - Server utilization estimated by P0.1*

## 4.4 Discussion

When Tsung estimates the average response time (for requests, page, etc.), then it computes every 10 sec (and reset). That is why in the Tsung report the highest average and the lowest average values of the average response time are presented.

For example, when the arrival rate is 1.4 ses/sec Tsung measures the following values:

– Average response time(lowest) = 150ms;
– Average response time (overall) = 230ms;
– Average response time (highest) = 750ms.

From the Figure 4.7, range of the response times between 150 and 300 ms. P0.1 predicted the average response time as 112 ms for the same workload.

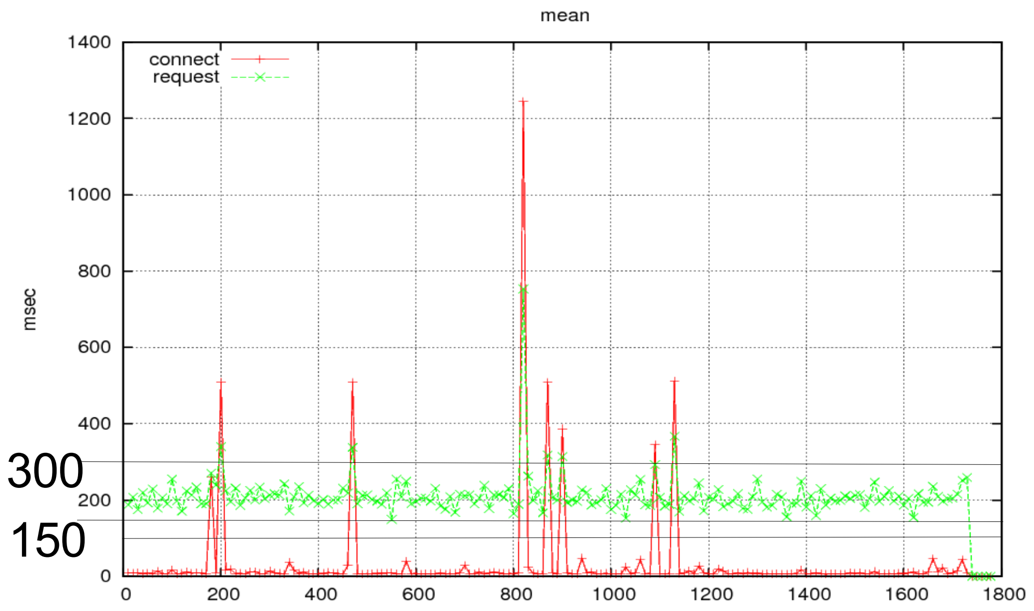


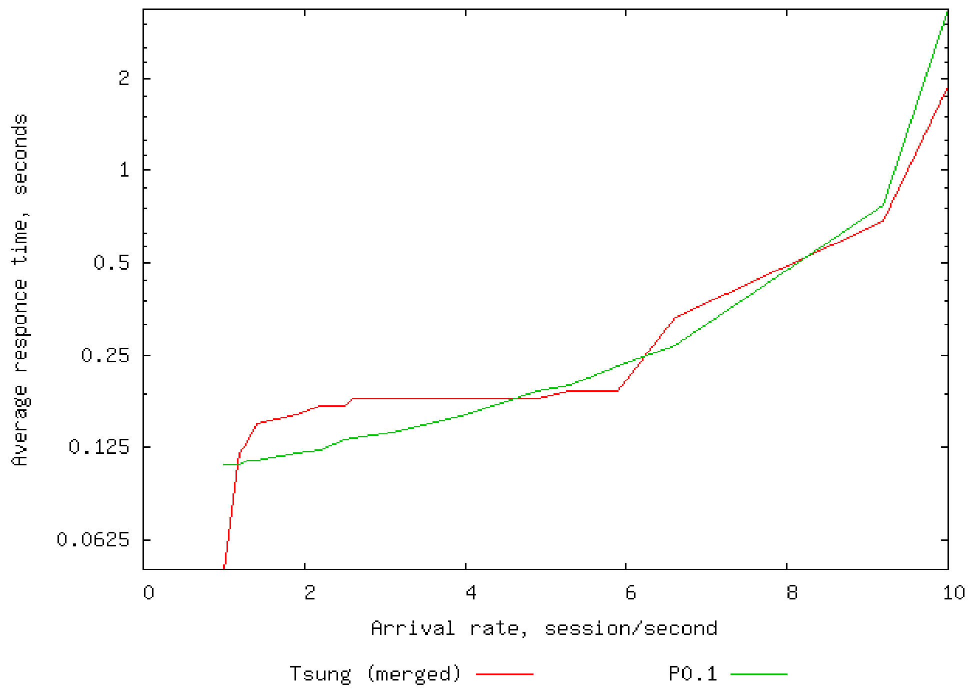
*Figure 4.7 - Average response time measured by Tsung for λ = 1.4, during 30 minutes*

112 ms is closer to 150 ms then to 230 ms. The same pattern is observed in other cases, when we compare the lowest average response times and the overall average response time by Tsung to the average response time predicted by P.01. In other words, with an increasing arrival rate, the average response time predicted by P0.1 is closer to the lowest average response time, then to the overall

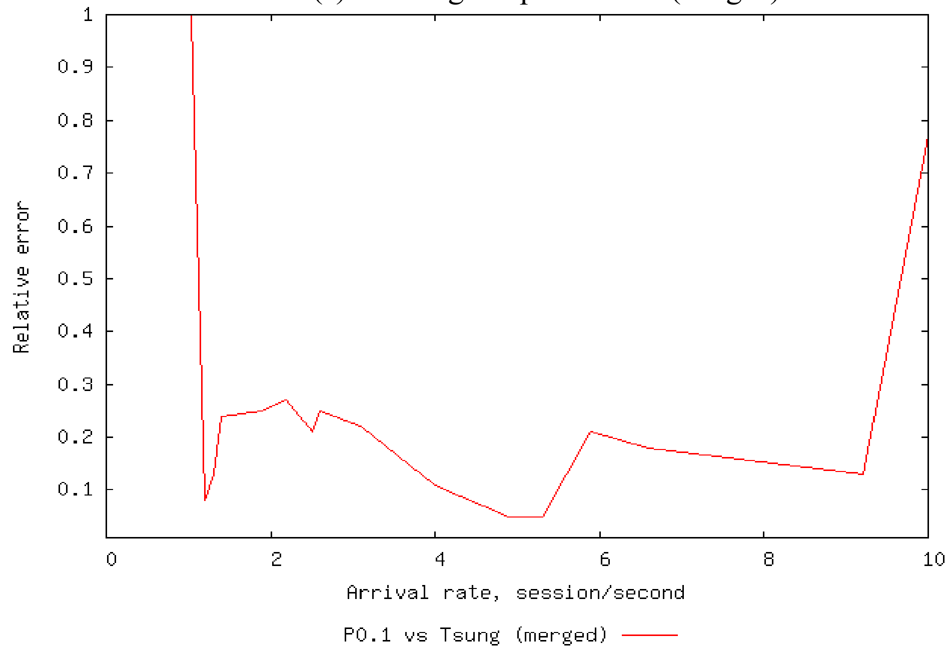
average response time measured by Tsung. Only when workload approaches saturation point, then the predicted value of the average response time by P0.1 is closer to the overall average response time measured by Tsung.

Figure 4.9(a) shows the relationship between the predicted average response times and the overall average response times measured by Tsung. Figure 4.9(b) shows that a relative error between the predicted and the measured response times approaches 40% in average.

However, if we combine and relate the average response times that we compare the lowest average response times, when it is closer to the predicted value with the same load or the overall, when it is closer to the predicted, then we obtain the picture shown in Figure 4.8(a). Figure 4.8(b) shows the relative error in average approaches 20 %. It means that P0.1 can be used to test web applications with a relative error of prediction in the order of 20%.

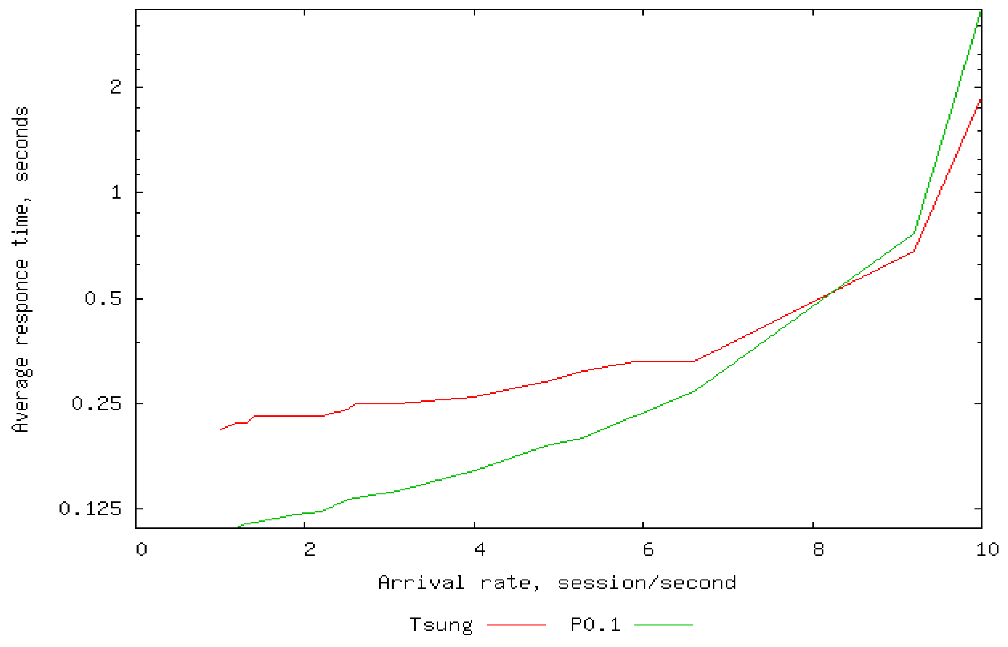


(a) – Average response times (merged)

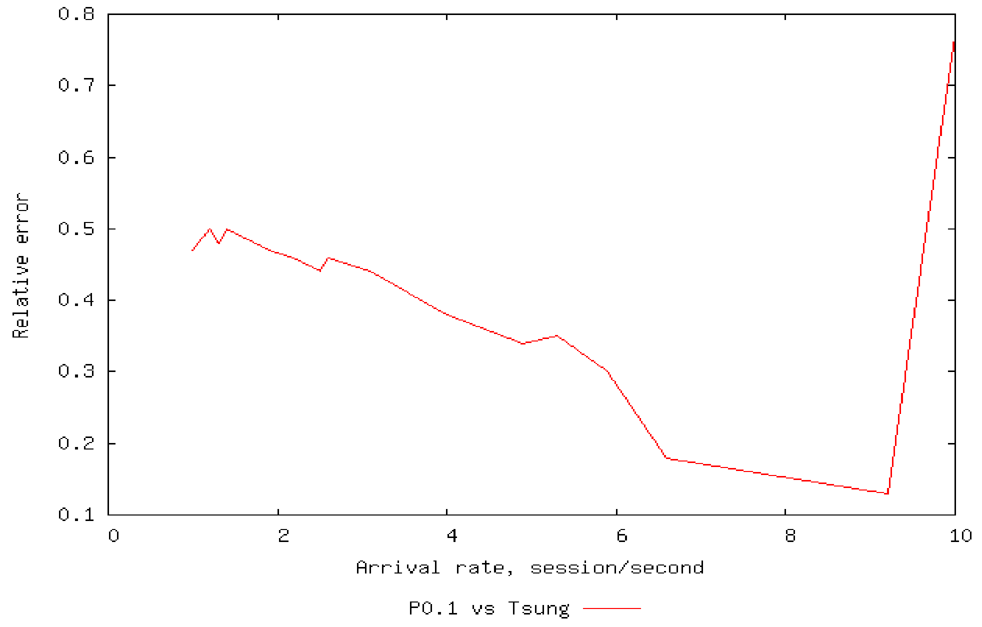


(b) – Relative error

*Figure 4.8 - Comparison of the average response time measured by Tsung (merged the lowest average and the overall) and predicted by P0.1*



(a) – Average response times



(b) – Relative error

*Figure 4.9 - Comparison of average response times measured by Tsung and predicted by P0.1*

# Chapter 5
# Future works

In this section I discuss possible extension to this thesis. Several aspects should be reconsidered and developed.

We found that there are different formants of logs, when we have been analyzing them. This aspect makes their processing very complicated or impossible at all. It would be useful to develop the mechanism of logs processing, that there was an opportunity to define the elements of a log line and their nature. For example, to define the DNS of a user, it is necessary to consider the position of it in a log line, and also the fact that it can be either numerical or literal. There are much more different aspects which have to be taken into account for having a possibility to build a user interaction model based on logs with different structure, not only for a particular structure. For example, if it could be a separate module, which will be responsible for the function, described previously, and will be integrated into the tool.

Another important aspect for improving the developed tool is defining the most significant transaction types. The new issue here is that there are different types of web applications. Sometimes the web applications contain several hundreds or thousands of transaction types. Processing all the transaction types significantly reduces the calculation time, though, if some types of the transactions are significantly rare, then they have a little impact on the system performance. That is why if  we exclude it from statistical data for building a user interaction model, it will not influence significantly estimation results of performance metrics.

Finely, the performance model. Firstly, we propose to separate the transaction types into different classes of transaction types. The transactions are different, but they have something in common that unite them, for example, service time. That is why there is no necessity to define the service time for each transaction type, but more reasonable to do it for each class of the transaction types. Secondly, here we have as a consequence the necessity to refine the performance model in order to take into account different classes of transaction types.

# Chapter 6
# Conclusions

The aim of this thesis was to build a black-box tool which is capable of analyzing the workload (logs) of a web application and predict the performance of the system for different loading conditions. We started to examine the problem by looking at different capacity planning strategies which have proposed before. We found that there is no straight forward solution for predicting the performance of web systems having only workload data. Also we became aware that most of the previous work uses profiling methodologies which are used to provide data such as utilization of the system to measure the service demands. To measure the utilization it is necessary to install additional software. This provided us with the main goal of our thesis: measuring service time with as less effort as possible and creating a performance model capable of estimating the performance.

To measure the service times we exploited the observation that the time spent to serve requests depends on the requests sizes. In order to do so, we use interpolation to approximate the service times for different transaction types.

Since we needed to compute the total service time of the average user session we used a user interaction graph. Calculation of the number of users at each state was not a trivial task, so we applied the Gaussian elimination technique which required some additional effort in order to obtain equations and matrices.

In order to model scheduling of serving jobs, entering the system, we studied: FCFS and PS. I concluded that PS scheduling is the most realistic and it is a good approximation of real systems scheduling strategies.

We designed performance model which calculates the mean response times and utilization of a resource for particular workload. The results can be used to make decisions according to performance of web systems and predict future performance. Although the results are not 100% accurate the results showed that they can be used as a first step to plan the future performance.

# VEEBIRAKENDUSTE TEHINGUTE TÕENÄOSUSLIK TESTIMINE

Magistritöö (30 EAP)
Yuliya Brynzak

## Resümee

IT süsteemid muutuvad oma elutsükli vältel järjest keerulisemaks. Veebirakendusi kasutatakse eriti laialt erinevatel eesmärkidel, sest võrgupõhine juurdepääs informatsioonile on väga mugav. Kuid võrgupõhise juurdepääsu juures tekivad mõned probleemid, mida tuleks silmas pidada. Kasutajad eeldavad prognoositavat jõudlust (nt nõuetekohane reaktsiooniaeg), seega teenusepakkujad peavad teadma, kuidas nende süsteem töötab erinevate koormuste all. Selles teesis loome tõhususe analüütilise mudeli ja töötame välja programmi, mis selle lahendab. Antud programm lubab analüüsida veebirakenduste jõudlust ja vastata järgmistele küsimustele:

1) missugune on keskmine süsteemi reaktsiooniaeg?
2) missugune on süsteemi kasutamine üldiselt?

Parameetrid programmi jaoks nagu keskmine teenindusaeg, uute taotluste keskmine saabumisaeg, keskmine mõtlemisaeg, on saadud testsüsteemi reaalse koormuse logidest. Jõudluse mudel on välja töötatud Queuing Networksi abil, mis lubab analüüsida süsteemi matemaatiliste valemite abil.

# Bibliography

[1] D. A. Menasce,V. A. F. Almeida, and L. W. Dowdy. *Performance by design. Computer Capacity Planning by Example.* Prentice Hall PTR, 2004.

[2] B. Schroeder , A. Wierman , and M. Harchol-Balter . *Open Versus Closed: A Cautionary Tale,* 2006

[3] Q. Zhang, L. Cherkasova, G. Mathews,W. Greene, and E. Smirni. *A capacity Planning Framework for multi-tier Enterprise Service with Real Workloads,* 2007

[4] Yuan Chen, Akhil Sahai, Subu Iyer, and Dejan Milojicic. *Systematically Translating Service Level Objectives into Design and Operational Policies for Multi-Tier Applications.* HPL-2008-16, 2008.

[5] I. Mitriani. *Probabilistic modeling.* Cambridge University Press,1998.

[6] Cristina D. Murta, and Geri N. Dutra. *Modeling HTTP Service Times,* 2004.

[7] Leon Simon. *An introduction to multi-variable mathematics. Synthesis Lectures on Mathematics and Statistics.* San Rafael (Calif.) : Morgan & Claypool Publishers, 2008.

[8] Tsung. http://tsung.erlang-projects.org/

[9] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis, 3rd Edition,* 2001.

[10] D.A. Menasce,V. A. F. Almeida, and L. W. Dowdy. *Performance by design. Computer Capacity Planning by Example.* Prentice Hall PTR, 292, 2004.

[11] John D.C. Little, and Stephen C. Graves. *Little's Law.* Massachusetts Institute of Technology, 2008.

[12] B. Urgaonkar, G. Pacifici, P. Shenoy, M.Spreitzer, and A.Tantawi. *An Analytical Model for Multi-tier Internet Services and Its Applications,* 2005.

[13] I. Mitriani. *Simulation techniques for discrete event systems.* Cambridge University Press. New York, NY, USA,1982.

[14] Flanagan libraries. http://www.ee.ucl.ac.uk/~mflanaga/java/Regression.html

[15] http://www1.cse.wustl.edu/~jain/cse567-08/ftp/k_33o/sld005.htm

[16] Michael J. Fischer, and Michael S. Paterson. *Impossibility of Distributed Consensus with One Faulty Process,*1985.

[17] A. Kamra, V. Misra, and E. Nahum. Yaksha: *A Controller for Managing the Performance of 3-Tiered Websites.* In Proceedings of the 12th IWQoS, 2004.

[18] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[19] D.Villela , P. Pradhan, and D. Rubenstein. *Provisioning Servers in the Application Tier for E-commerce Systems.* Journal ACM Transactions on Internet Technology (TOIT), Volume 7 Issue 1, February 2007.

[20] B.Urgaonkar, P.Shenoy, A.Chandra, and P. Goyal. *Dynamic Provisioning of Multi-tier Internet Applications,* 2005.

[21] JMT tool. http://jmt.sourceforge.net/

[22] B. Schroeder, M. Harchol-Balter, A. Wierman, A. Iyengar, and E. Nahum. *How to Determine a Good Multi-Programming Level for External Scheduling*. 22nd International Conference on Data Engineering (ICDE 06) . Atlanta, Georgia, April 2006

[23] Jeffrey E.F. Friedl. *Mastering Regular Expressions*, 2nd Edition.496, 2002

[24] Wordpress. http://wordpress.org/

[25] Apache HTTP Server. http://www.apache.org/

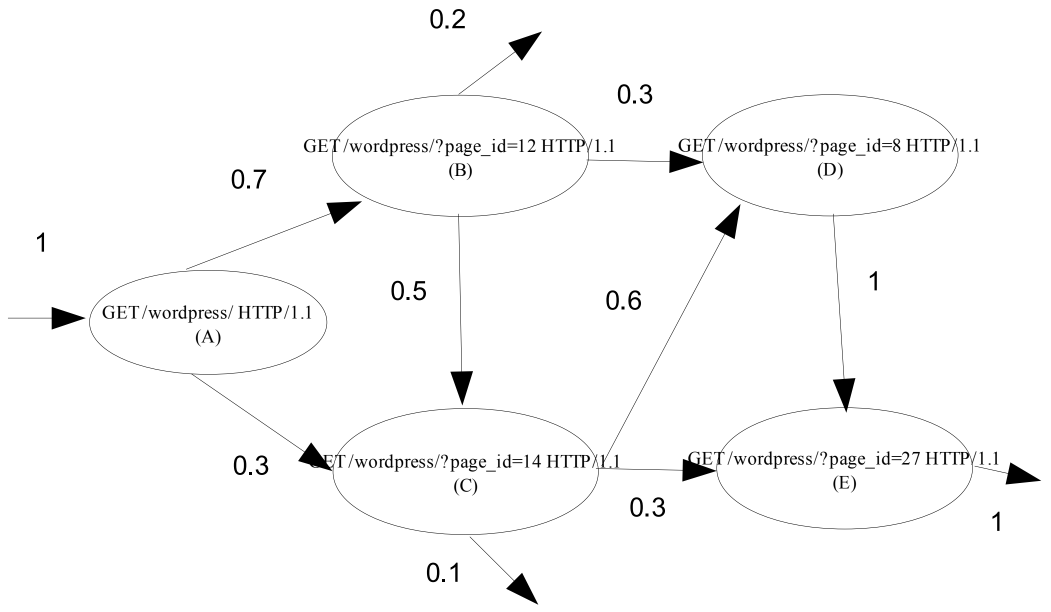
[26] Ben Laurie, and Peter Laurie. *Apache: The Definitive Guide (3rd Edition)*. O'Reilly Media, Inc.,1997.
[27] http://httpd.apache.org/docs/1.3/logs.html

APPENDIX 1. Session parameter from Tsung configuration file

```
  <session name="/HAEG" probability="16" type="ts_http">
        <transaction name="index">
                        <request>
                                <http url="/wordpress" method="GET" version="1.1">
                                </http>
                        </request>
                        <thinktime value="2" random="true"/>
        </transaction>

        <transaction name="H">
                        <request>
                                <httpurl="/wordpress/?page_id=8"method="GET"
version="1.1">
                                </http>
                        </request>
                        <thinktime value="2" random="true"/>
        </transaction>
        <transaction name="A">
                        <request>
                                <httpurl="/wordpress/?
page_id=12"method="GET"versn="1.1">
                                </http>
                        </request>
                        <thinktime value="2" random="true"/>
        </transaction>
        <transaction name="E">
                        <request>
                                <httpurl="/wordpress/?page_id=29"       method="GET"
versn="1.1">
                                </http>
                        </request>
        </transaction>
        <transaction name="G">
                        <request>
                                <http    url="/wordpress/?page_id=31"    method="GET"
versn="1.1">
                                </http>
                        </request>
        </transaction>
```

APPENDIX 2. Example of Customer Behavior Model Graph (CBMG)


0.2
0.3
GET/wordpress/?page_id=12 HTTP/1.1
(B)
GET/wordpress/?page_id=8 HTTP/1.1
(D)
0.7
1
0.5
0.6
1
GET/wordpress/ HTTP/1.1
(A)
0.3
GET/wordpress/?page_id=14 HTTP/1.1
(C)
GET/wordpress/?page_id=27 HTTP/1.1
(E)
0.3
1
0.1

## APPENDIX 3. Pseudo code for obtaining the coefficients and right-hand matrices

```
Input:          <String, <String, Double>>  [map0]  //
Initialization: <int>      [size] = [map0].size() - 1;  //
                double[size][size] [matr1];
                double[size] [matrix2];
                <String, Integer>[legendc];
                <String, Integer>[legendr];
                <int> [indexr] = 0;
                <int> [indexc] = 0;


for each element of [map0]
        <String> get key [pageto]
        <String, Double> get value [pagesfrom];
        if  ( [pageto] != “out”  ) {
                [indexr] = checkIndex ([legendr], [pageto]);
                for each element of [pagesfrom]{
                        <String> get key [k];
                        <Double> get value [v];
                        if ( [k] != “enter” ){
                                [indexc] = checkIndex([legendc], [k]);
                                [matr1][indexr][indexc] = [v];
                        }
                        else {
                                [matr2][indexr] = -[v];
                        }
                }
        }
        else {
                for each element of  [pagesfrom]
                        <String> get key [k];
                        if  (  [k] != “enter” && [legendc].get([k]) == null )
                                [indexc] = [legendc].size();
                                [legendc].put([k], [indexc]);
        }

[matr1] = fillWithMinusOne([matr1], [legendc], [legendr]);


Output: [matr1] , [matr2]
```

***chekIndex()***

```
Input:   <String, Integer> [legend] ;
         <String> [pageto];
Initialization: <int> [index] = 0


if [legend].get[pageto] != null
         [index] = [legend].get([pageto]);
else
         [index] = [legend].size();
         [legend].put([pageto], [index]);

Output: [index];
```

***fillWithMinusOne()***

```
Input:   double[][] [matr1]
         <String, Integer> [legendc];
         <String, Integer> [legendr];

for each element of [legendc]
         <String> get key [page];
         <Integer> get value [indexc];
         <Integer> [indexr] = [legendr].get([page]);
         [matr1][indexr][indexc] = -1;

Output: [matr1]
```