

Web Server Performance Simulation
MEng Individual Project
Outsourcing Report (version 1.0.0.16)

Andrew Ferrier
Supervisor: Peter Harrison

May 21, 2002

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction to this Report | 5 |
| 2 | Introduction to the Project | 7 |
| 2.1 | Aims and Objectives | 7 |
| 2.2 | Knowledge Assumed | 8 |
| 2.3 | Project Website | 8 |
| 3 | Background | 9 |
| 3.1 | What's Important to Clients and Servers | 9 |
| 3.2 | Current State of Subject | 10 |
| 3.3 | Why Use Simulation? | 11 |
| 4 | Specification | 15 |
| 4.1 | General Specification | 15 |
| 4.2 | Input | 16 |
| 4.3 | Simulation and Execution | 18 |
| 4.4 | Output | 19 |
| 4.4.1 | Summarised Output | 19 |
| 4.4.2 | Trace Output | 20 |
| 4.4.3 | During Execution | 20 |
| 4.5 | Extensions and Modifications to Specification | 20 |
| 5 | Testing | 23 |
| 5.1 | Input Tests | 23 |
| 5.2 | Output Tests | 24 |
| 6 | Timetable | 27 |
| A | Errata | 29 |
| B | Bibliography | 31 |
| C | List of Acronyms | 35 |

Chapter 1

Introduction to this Report

In this report¹, a chapter will be dedicated to each of these items:

- Introduce my project, by explaining what my aims for the project are.
- Explain the background to my project: the problems and issues which led to my decision to do this project, the current state of progress in this subject area, and why I intend to use simulation.
- Formulate a specification for the software portion of my individual project — the input, what the software should do, and what form the output should take.
- Describe testing methods that can and should be applied to the software, to ensure that it is written in compliance with the specification and to fulfill the aims of the project.
- Specify an approximate timetable for completion of the project.

The project is not entirely software-based — the software being both an end product in itself and a means to an end, so I will discuss the general aims of the project before going on to specify the software and explain how it will be useful in furthering the aims of the project.

Throughout this report, I have borne in mind the designed target audience: an MEng student such as myself. I have been asked to imagine that I am not doing the project, but merely specifying it for a colleague, when writing this report. I have not tried to actually pretend this in my writing style, but I have tried to keep in mind the idea of someone else implementing these ideas.

¹This report was prepared with L^AT_EX 2_ε

Chapter 2

Introduction to the Project

2.1 Aims and Objectives

The aim of my project is to come to some general conclusions about the best way to configure a web serving system, to maximize performance (primarily latency), minimize cost, and perform other kinds of optimisation.

In order to do this, I will construct software to enable a user to simulate an a web server (or web servers). I will then use this software against some test case web-server systems to try to answer some questions and come to some conclusions.

The primary questions I will be attempting to answer are the following:

- Whether it is best, generally, to use many small interconnected serving machines (a web server ‘farm’), or to use one or only a few large serving machines.
- How many synchronous threads should exist on a single server system for maximum performance.

Of course, these questions will, most likely, not have simple answers — the answers will depend on certain factors. However, I will attempt to discover these factors and explain their effect upon the issues above.

Although ideally I would like to construct my software in as general a manner as possible, time constraints will not allow this. Hence in this outsourcing report I will be specifying a specific set of requirements which the software should follow, which I will generalise further if time allows¹.

The final items which will be produced for the project are:

- Simulation software and ancillary files, as specified in chapter 4.
- A final report documenting the software and attempting to answer the questions above, with the aid of the software.

There are also other questions I would like to answer if time allows. They will be further explored in section 4.5 on page 20.

¹My progress on that will be documented in my final report.

2.2 Knowledge Assumed

This report will assume some knowledge about some background issues. The reader should be familiar with:

- What a web server and client are; the differentiation between them and how they work together to form ‘the web’, on a typical TCP/IP (Transmission Control Protocol/Internet Protocol) network, with the HTTP (Hypertext Transfer Protocol) protocol. However, specific knowledge about the TCP/IP and HTTP protocols is not assumed.
- Web serving from a client’s perspective: what it is like to use the typical web client, in a variety of environments.
- The basic principles of simulation and modelling; the difference between process-based and event-based simulation, etc. See [Fie00] for more information.
- The basic syntax and principles of XML (eXtensible Markup Language). This is necessary to understand the specification for the input file to the software. See [W3C02a] for more information.

2.3 Project Website

This project has a website at: <http://www.new-destiny.co.uk/andrew/project/> that contains various documents and other material related to the project.

Chapter 3

Background

In this chapter I explain the background to my project, by:

- Describing the important issues for clients and servers regarding the web.
- Looking at the current state of progress in the area of web server simulation, and related areas.
- Explaining why I am using simulation.

3.1 What's Important to Clients and Servers

Web Server Performance is an issue that has come to prominence in the last decade or so, since the invention of the web in late 1990 by Tim Berners-Lee at CERN [[W3C02b](#)].

There are two main ways that one can look at web server performance [[MA02](#)]:

- From the user (client)'s perspective.
- From the server's perspective.

From a user's perspective, all that matters about web server performance is the speed with which web pages are retrieved and how they display on their client.

In fact, it has been shown that for client/server systems in general (not just web sites), sub-second response times to user actions are near-essential for users to feel that the system is responding quickly enough not to interrupt their flow of thought [[Nie94](#)]. Even if one doesn't agree, it is obvious that users prefer short response times and it will benefit all concerned if these occur [[Gre](#)].

From a server's perspective, what matters is the ability of the server to handle as many requests as clients demand, synchronously, without unnecessarily delaying responses to the clients (hence causing them to become frustrated), or building up a backlog of clients wishing to use the server.

Obviously ensuring fast response times for clients and sufficient server capacity are not totally distinct problems: in order to have a coherent 'World Wide Web' which works for everyone, performance has to be good all round.

Hence, in this report, and this project, I am interested in two related issues:

- The performance seen by the client *before* the web page is fully downloaded — in other words, the latency between requests and responses, and the bandwidth provided during response.
- Sizing and capacity planning for the web server, in order that the client requirements for performance are fulfilled, and that those running the web server fulfill these requirements at maximal efficiency.

Both of these depend on two general factors: the network in-between the client and the server, and the capabilities of the server hardware and software. Normally the capabilities of the client are irrelevant because even the simplest client will have the ability to receive traffic as fast as it can be sent by the network and the server, and clients typically only retrieve a few files simultaneously.

In this project I will be looking at both of these, but the stronger focus will definitely be on the server. Network performance is a very large topic in itself, and it is worth bearing in mind that even TCP/IP networks carry many different types of traffic, not just HTTP, and hence network performance issues are not just issues with HTTP. This is a subject worthy of investigation all by itself.

Note: When I refer to client-end performance, I will *not* be looking at performance after pages have been retrieved. The ability of web browsers to perform animation, run client-side code, or use proprietary technologies such as Macromedia Flash [MMF02] is not within the remit of this project.

3.2 Current State of Subject

Computer-based Simulation is a fairly well-studied subject, and there are many books, papers, and other sources, which document the area from many perspectives. One of the better ones is [BCINN], a fairly modern and practical book which I have found quite useful. However, there are many others, of which [KWDJ, LK00, ZPK, Jai] are only a few. There are also statistics books specifically geared towards creating computer simulations — for example, [Lew75] has some fairly technical and thorough information on various different kinds of distributions.

It does not appear that much progress has been made in the area of web server simulation, however¹ — one of the reasons why I decided to do this project, and the main reason this section is sadly lacking in volume! Even research in Journals such as the ACM Transaction on Modeling and Computer Simulation [ACM] does not turn up much information.

Web server performance appears to be a subject which is discussed in some circles [MD, Kil], but this rarely seems to stretch to simulation. One example of a simulation that has been attempted is [Hon98]. However, this source does not contain much information on the simulation itself or how it was implemented — certainly not enough to compare it to what I intend to do — and the simulation does not appear to be freely available.

Progress is being made in related areas. For example, queuing theory has been used as a tool to model web servers [Slo95], but as I will explain below, this

¹Ironically and ‘unhelpfully’ (at least when searching for information!), there seem to have been many people who have tried *web-based* simulation — but not related to actual simulation of web systems themselves.

method can have disadvantages. Papers have also been published on profiling web usage analysis [MS], which is a subject area which can have an impact on web server sizing, in particular for large projects. This book is quite useful if one is interested in profiling web usage, which of course one should be if setting up a web server, but does not help with the direct problem of how to size the web server — in a way, knowing about the profiles of your users helps with two other problems: what to set up your network links to, and what to put on the site itself.

Work has been published on engineering software for the web (i.e. how to structure programmatic web-sites) [MD]. This is once again useful to those attempting to size web servers as it helps them to understand what programmatic demands sites will place on their servers, but it is not of direct relevance to web server modelling.

Progress is also being made in the area of data mining and retrieval on the web — [LZ00] is but one example. These are traditional subject areas which have been around for a while and are now being investigated in the context of the web. [ACM00] contains information on servers farming out search tasks to child servers, for example. However, this has more to do with search engines, web indexes, and other related technologies, then the sizing and creation of web server(s).

So, although sadly there is not much to document in the area of web server simulation itself, this was the main reason that I felt that it would be useful to try to write a web server simulation myself. Whilst I cannot hope to make it in any way definitive or complete, hopefully I can document the improvements that would have to be made to further improve it.

3.3 Why Use Simulation?

This section will describe why I am using simulation, as opposed to other methods of analysis, to explore the performance of web servers.

There are four potential strategies which can be used when deciding how to size web servers (of which only the first two tend to be used in practice):

1. Use rules-of-thumb; calculate using approximate equations but always overestimate to make sure one has enough capacity. This is the approach normally taken in industry, and be easily subject to obfuscation, since it is based on empirical measurements. This observation is based mainly upon personal experience.
2. Use real measurement; set up a real system, with a real web server and some real clients, attached with a network, and install software on them to make many simultaneous requests for files. This does to a certain extent help with deciding the load that a server can handle, and is sometimes used for larger projects. However it has several disadvantages:
 - Instead of having many thousands (say) of distinct clients, one will typically have tens or maybe hundreds, making many more requests than they typically would in order to produce the same demand. However one has to be absolutely certain that these clients can handle the load of making that many requests. A typical client only handles

a few requests simultaneously, which is trivial. If the client handles more than this, it may become bogged down in network code and hence not produce realistic timings.

- Typically such machines are connected together on a LAN (Local Area Network). This is very unrealistic of many web servers: although it may reliably test the load on a server designed only for a small intranet, any server which is serving requests to clients which are widely geographically spread will have much higher latencies on requests and replies because packets will have to travel through many switches and routes, possibly encountering delays along the way, in both directions. It will hence increase transaction time for web requests, meaning that server resources are used up for a lot longer than on a local LAN.
- Sometimes it is not possible to alter a running system without being fairly sure it will achieve the desired results, as it is too important — obtaining duplicate hardware may not be an option.

Some of these problems can be overcome: for example, it can be arranged for geographically spread clients to test the server. However, whatever one does, all real-world testing is very expensive and time-consuming, and becomes more so the more realistic one attempts to make it, in general [SAS99]. Moreover, for the reasons explained above, they may well produce misleading and hence possibly dangerous (if the server sizing procedure is important) results.

3. Use queueing theory to investigate the system. This is very good in theory; however the problem is that it is very mathematically demanding, requiring quite a lot of mathematical knowledge, which is typically not available to those setting up web serving systems, and more importantly, most non-trivial systems cannot be solved exactly or easily. Sadly, this makes less useful for most real-world systems and it is not generally used.
4. Use simulation software to model the system. This is the approach I will take with this project, and I will explore this further below.

Simulation software has several advantages over the set-it-up-and-see approach, and I will outline those here in the context of my project:

- One does not require the expense and inconvenience of setting up an actual system: this can take a long time [Fie00, SAS99] and be very resource-intensive. In many cases, it simply would not be done — simulation thus allows one to explore new possibilities.
- Simulation is a lot less risky than altering an already-existing system: simply set up a duplicate system in the simulation, then alter some of its attributes.
- It can answer many different questions, easily and quickly, e.g. What would happen if the attributes of this server were changed? What about if I double the number of servers? What is the mean value for this wait time? Etc. . .

However, simulation software also has some disadvantages:

- It does not produce exact solutions to most problems; solutions are at best approximate. This can be overcome by using a more detailed simulation system, which will provide more accurate (though still not exact answers).
- Dependent on the simulation software and hardware, determining solutions of sufficient accuracy can be very time-consuming and resource intensive, more so than running the corresponding real system.

I will try to minimise the effects of these disadvantages in my software design by choosing carefully which parts of the software to simulate.

Chapter 4

Specification

This chapter specifies the software I intend to write. In order to specify the software I will describe the input to the software should be, the behaviour of the software, and what the output should be.

I have deliberately tried to avoid discussing implementation issues. Of course, it is naïve to think that design and implementation are completely separable and sometimes it is debatable what is a design issue and what is an implementation issue, but I have tried to separate them as much as possible, and I left as much scope for the implementor as possible. In particular I have avoided discussing the look and feel of the software (for example, whether it is a console-based application or a GUI (Graphical User Interface) application).

Bear in mind that the specification I lay out here is a minimum specification. The software can go further than this; indeed, it is desirable that if time allows, more flexibility or features can be built into the software. Obviously, if this is done, then it would be preferred if it can be done with backwards compatibility with the specification, but whether this is achieved or not, extensions must be documented. Extensions are explored further in section 4.5 on page 20.

4.1 General Specification

The simulation software should simulate an arbitrary client-server system based upon the WWW (World Wide Web) model, i.e. with clients requesting files from servers via the HTTP protocol.

The simulation program must be written so as to accept input as specified in section 4.2, execute as per the descriptions in section 4.3, and produce output as per section 4.4. Beyond this, no requirements are specified here — in particular, it is left open to the implementor which language to use, how to construct the simulation system and whether to use event-based or process-based simulation.

The basic simulation system as specified here simulates three types of ‘system objects’ — clients, network nodes, and servers. They interact with each other in the simulation and various parameters from them are measured and should be output by the simulation software.

4.2 Input

The input to the software will consist of a single XML [W3C02a] file which will specify the complete system. The name of this file shall be specified by the user to the software. The XML file must be valid according to the XML 1.0 specification [W3C02a].

The XML file will have the following structure:

- **<system>**. This tag represents a simulation system. There must be only one of these tags in the file, and all the other tags I will go on to specify must be enclosed within the **<system>** opening and closing tags. The tag is provided only for future-proofing, in case one wanted to develop the software to allow more than one simulation system to be specified in the same XML file. This feature does not have to be provided in the software here but can be if time allows.

There are then various tags which can appear zero or more times enclosed within the **<system>** tag, which specify objects in the simulation system (these objects are known henceforth as system objects). These tags are **<client>**, **<server>**, and **<networknode>**, and their use is fairly self-explanatory from their names. They all have the following common attributes, which I will specify here:

- **name**. A name which identifies the system object. This must be a unique identifier within the system (not just within this type of object) to enable inter-system-object connections to be specified (see the common attribute **connectto** below). If it is not unique, the software can report it as an error. This parameter is a non-zero-length string. This attribute is compulsory.
- **instances**. The number of instances of this system object within the system. This is provided so that the user can conveniently, say, create 100 clients without having to specify 100 client tags. As multiple instances are created, the system can alter attributes if appropriate (for example, if 100 instances of the Network Node *nna* are created, and a connection is specified to *n nb* — see the **<connectto>** tag below for more information — of which 100 instances are also created, each instance of *nna* should connect to the corresponding instance of *n nb*. More complex cases are left to the implementor's discretion but should be documented). This attribute is an integer which is greater than 0. The default if this attribute is not specified is 1.

Before I go on to describe the tags **<client>**, **<server>**, and **<networknode>**, I will describe the tag **<distribution>**, which is used within some of them and describes a statistical distribution, and the tag **<connectto>**, which is used for creating inter-object connections. The tag **<distribution>** has the following attributes:

- **type**. This describes the type of the distribution. It can be:
 - **constant**. This represents a distribution which simply always returns a constant value, i.e. has a probability 1 of that value. The value is specified by the attribute **value**.

- **uniform**. Represents a uniform distribution between the values specified by the **lbound** attribute and the **ubound** attribute.
 - **poisson**. Represents a Poisson distribution with the mean specified by the **mean** attribute.
 - **experimental**. Represents a experimental distribution. This enables the program to have distributions specified which are drawn from real experiments. An external file must be specified with the **filename** attribute, and this file must contain numerical values, separated by whitespace. Anything else in the file is illegal. Each time the distribution is used, a value will be drawn from this set strictly randomly.
- **value**. For a constant distribution, specifies the constant value. Must be a real number greater than zero. Compulsory for a constant distribution, illegal for all others.
 - **lbound**. The lower bound for a uniform distribution. A real number greater than zero. Compulsory for a uniform distribution, illegal for all others.
 - **ubound**. The upper bound for a uniform distribution. A real number greater than zero. Compulsory for a uniform distribution, illegal for all others.
 - **mean**. The mean value for a Poisson distribution. A real number greater than zero. Compulsory for a Poisson distribution, illegal for all others.
 - **filename**. Specifies the filename for the input file for an experimental distribution. This filename must either be full, or relative to the XML input file.

Other statistical distributions can be provided at the discretion of the implementor if time allows.

The tag `<connecto>` can appear zero or more times inside the tags `<client>`, `<server>`, and `<networknode>`, and represents a connection to another system object. It has one attribute:

- **connectname**. The unique system name referring to the object that one wants to connect to. Creates a one-way network connection (if two-way is desired, one can create a connection each way — this is for flexibility). Must refer to another system object.

I will now go on to describe the tags `<client>`, `<server>`, and `<networknode>` individually, with their own unshared attributes:

- **<client>**. This tag represents one or more instances of a client system. The tag must have one and only one `<distribution>` sub-tag, which represents the inter-event time distribution with which the client makes HTTP requests (in virtual seconds). Requests can be synchronous: a client can make one request and then make another whilst the first is outstanding.

- **<server>**. This tag represents one or more instances of a server system. The tag must have one and only one **<distribution>** sub-tag, which represents the service time for an incoming HTTP request (in virtual seconds). The server object has the following attributes of its own:
 - **threads**. The number of synchronous serving threads in this server object. For more information on how this affects serving, see the section 4.3, Simulation, below.
- **<networknode>**. This tag represents one or more instances of a network node. This tag must have one and only one **<distribution>** sub-tag, which represents the distribution of a packet's wait time in the node before being sent on (in virtual seconds). Packets are dealt with in a FIFO (First In First Out) manner and are 'magically' routed (see Section 4.3, Simulation, below).

A DTD (Document Type Definition)¹ for this XML file should be provided with the software to assist those with generalised XML editors to create an input file for the software which can be checked.

Also, some sample input files (with documentation explaining the expected output) should be provided with the program to enable the user to test the program's correct running. For an example of a simple test input file, see section 5.2 on page 24.

4.3 Simulation and Execution

The simulation must be set up as described in the XML file, so that all system objects behave at all times as specified there.

The clients must generate HTTP requests according to their distributions, and send them out onto the network. The network nodes must queue incoming packets and send them onwards, delaying them by their delay distributions. Servers must queue incoming requests, serve them according to their service time distributions, and send replies back to the same client which made the request — the replies are dealt with by the network nodes on the same basis as requests. Clients do not queue incoming replies.

Servers with more than one thread execute the threads in parallel, in time. There is only one input queue for the server, and the threads retrieve the next request from the head of the queue when they are not busy.

All time is measured in virtual seconds, starting at 0 when the simulation is started, and the system should be executed 'as fast as possible' (which may be slower or faster than real-time depending on the performance of the implementation).

Actual requests and replies do not need to consist of anything².

There is no routing information provided in the XML specification for the system. The routing method used will be what I call 'magic'. That is to say, each node will simply know the shortest path back to each and every destination. Exactly how this is done is an implementation issue, but it would almost

¹Or XML Schema; the decision on which to provide is left as an implementation issue.

²This may be modified if time allows to represent different types of queries: those which are more and less complex.

certainly be pre-calculated. This nicely simulates a real system where routing information is propagated via complex algorithms: it is simply beyond the initial scope of this project to simulate such algorithms.

If there is no possible route from the source to the destination, the software should output a warning and ‘drop’ the packet.

4.4 Output

4.4.1 Summarised Output

The following items must be output from the software:

- The mean, minimum, and maximum response time for an HTTP query for each client.
- The mean, minimum, and maximum wait time for a network packet for each network node.
- The mean, minimum, and maximum interarrival times for HTTP requests for each server.
- The mean load on each server: i.e. how much of the time is each server doing work.

These items must be output in a machine-readable structured manner, not merely printed onto the screen, so that they can be loaded into another piece of software for analysis. It is desirable for the user to be able to specify a variety of different formats that they want this data to be output in. At the very least, the option to output a CSV (Comma Separated Variables) file should be provided. This file should contain data in the following format:

- A header line with titles for each field in the rows to come.
- Each row from then on should contain the name of an object (i.e. a client, network node, or server), followed by the statistics specified above.

An example of the type of output expected is shown in Figure 4.1 on Page 19.

```
"Object Name", "Minimum Time", "Maximum Time",  
  "Mean Time", "Mean Load (if applicable)"  
"client1", 0.4, 0.8, 0.6  
"client2", 0.3, 0.4, 0.32  
"networknode1", 0.6, 0.8, 0.7  
"server1", 0.1, 0.2, 0.16, 0.03
```

Figure 4.1: Example of output CSV file

If it is necessary to split this data into multiple CSV files (because fields vary between different types of objects), that should be done.

4.4.2 Trace Output

As well as the above summarised output (which should always be produced), the software should produce a trace output. Whether this trace output is produced or not should be a user-configurable option.

The trace output should contain information on every discrete event that happens in the simulation system. Exactly what information is outputted is left as an implementation issue, but at least the following items should be included for each event:

- The unique name(s) of the system objects involved.
- A descriptive name describing the event.
- The virtual time at which that event occurred.

The trace output should be sorted strictly according to increasing virtual time.

The trace output should ideally be available in a variety of different user-specified formats, but must at least be available in CSV.

4.4.3 During Execution

The system should show at least the following parameters whilst it is executing:

- The running mean response time for each client over the last 30 virtual seconds, 5 virtual minutes, and 10 virtual minutes.
- The running mean service time for each server over the same time periods.

If there is not enough screen real-estate for these items, then they should be averaged across all clients or servers, as appropriate, and displayed.

4.5 Extensions and Modifications to Specification

There are some additions or modifications to the above software specification which it would be nice if the software incorporated. These should be added as time allows. They are not fully specified here (although they are briefly described, one-by-one), but should be documented in the final report if they are implemented. Other modifications and additions can also be made to the specification as appropriate. They should also be documented.

- The software could use more complex methods to model the server by allowing multiple distributions to represent different parts of serving a request, instead of simply using one distribution.
- The client side of the HTTP transaction could be modelled to a greater extent; a virtual 'web site' could be specified, with clients requested pages from that site in a typical manner. This would allow for greater realism, as, for example, servers typically cache often-used pages.

- Other server-side system objects could be modelled, apart from web servers. One example would a database system: the interaction between the web server and the database system would be simulated also.
- The input file could allow for specification of more than one system.
- Support for more statistical distributions could be built into the system.

Chapter 5

Testing

This chapter consists of various tests which should or could be applied to the final software to ensure that it meets the requirements specified in the specification. These tests are for the minimum specification in the previous chapter only. They are not designed to test any extensions to the software — though these should be thoroughly tested too.

5.1 Input Tests

- The software should never crash. In the case of invalid input, it is acceptable for the software to abort with an error. There are two different testing methods which should be used in order to determine whether the software is likely to crash:
 - Unusual input cases should be deliberately created in an attempt to crash the software. Three types should be created:
 1. Those which are invalid XML. An example is shown in figure 5.1 on page 23.
 2. Those which are valid XML but are not valid for the program specified in this report. An example is shown in figure 5.2 on page 24.
 3. Those which are valid XML and are valid for the program but contain unusual or atypical simulation cases. An example is shown in figure 5.3 on page 24.

More lengthy examples of test cases will not be given here but an infinite number could be created and the software should be tested with a representative number, which cover all the appropriate boundary conditions.

```
<system>  
</system
```

Figure 5.1: Example of invalid XML input

```
<fruit>
  <banana>
</banana>
</fruit>
```

Figure 5.2: Example of valid XML input, but not for this software

```
<system>
  <networknode name="networknode1" instances="2">
    <distribution type="constant" value="2">
  </networknode>
</system>
```

Figure 5.3: Example of valid program input

- The software should be fed entirely junk input. One example of a program which does this automatically is [\[FUZ\]](#). This program should be used if appropriate and the software should pass its tests.

All of the above tests are black-box tests because they do not require looking at the internals of the software.

- The software should create a simulation system which accurately represents what the input XML file specifies. In order to test this, one will have to perform white-box testing: outputting diagnostic information about what simulation objects the software is creating, and their attributes. This diagnostic should be complete insofar as it will output information about all objects in the system and their attributes. It should be designed so that one could in theory re-create the simulation system XML specification from it.

The diagnostic output should be off by default, but one should be able to turn it on with a binary flag somewhere. The diagnostic information should be distinct from the trace output described in section [4.4.2](#).

5.2 Output Tests

- All output that the software creates should be syntactically valid. For various different input test cases, all CSV output should conform to the conditions laid out in the specification. In particular, there should be one and only one header row, and there should be a row (record) for each object in the simulation system, with the correct number of fields following it.
- All output that the software creates should be semantically valid. Various different input test cases should be given to the software, and the following output tests should then be performed:
 - Each object name should reflect the name of an object in the system. Each object name in the system should appear once and only once

```
<system>
  <client name="client1" instances="1">
    <distribution="constant" value="2">
      <connectto connectname="server1">
    </client>
  <server name="server1" instances="1">
    <distribution="constant" value="1">
      <connectto connectname="client1">
    </server>
</system>
```

Figure 5.4: Simple test case which can also be analytically solved

in the output.

- For all response, inter-arrival, and wait times, the minimum time should be less than or equal to the mean time, and the mean time should be less than or equal to the maximum time. Also, all of these times should be greater than or equal to zero.
- Server loading values should be between zero and one (in other words, the percentage load should lie between 0% and 100%).
- All output that the software creates should be correct: that is to say, it should simulate the system correctly and produce the correct answers. One can never be certain that this will always be the case, but one can test by running test cases through the software. The output should match up with the analytic queueing theory analysis of the same test case.

One example of an input file which specifies a simple test case is shown in 5.4 on 25. In this simple case, for example, the server utilisation should be exactly 50%.

Chapter 6

Timetable

This chapter contains a timetable for completion of the project. It is given as a set of goals, with associated dates. These goals are fairly high-level and the dates should be taken as very approximate and subject to change.

31st January Have a working system with simulation capability up-and-running (not necessarily the full specification).

4th February Material for project review (with second marker) complete. Includes simulation system as per 31st January and an outline for the final report.

14th February DTD or XML Schema matching minimum specification should be complete.

28th February Have full input capability as specified by this report working.

22nd March Have full output capability as specified by this report working. Start work on any project extensions.

27th March–3rd April Exam work: no project work.

17th May Have early draft of report available and show to supervisors.

31st May Stop all work on software — tidy up loose ends and complete or remove any extensions.

7th June Show final draft of project report and final software to supervisor. Make any last-minute changes.

13th June Submit Project Report.

17th–19th June Project Presentation.

Appendix A

Errata

Since the officially-submitted version (1.0), changes were made to the following items:

- Version 1.0.0.16** Changed a few compile options.
- Version 1.0.0.15** Change to PDF initial bookmark style.
- Version 1.0.0.14** Copyright notice added.
- Version 1.0.0.13** Changes to bibliography and a few other minor items.
- Version 1.0.0.12** Change to internal glossary style.
- Version 1.0.0.11** Bibliography style changed, as I start to use BibTeX.
- Version 1.0.0.10** Accidental addition of a glossary rectified.
- Version 1.0.0.9** Method for doing acronyms and the glossary changed to comply with that used in the final report and the standalone user guide.
- Version 1.0.0.8** The layout of the pages was changed slightly again.
- Version 1.0.0.7** The layout of the pages was changed slightly again.
- Version 1.0.0.6** The layout of the pages was changed slightly.
- Version 1.0.0.5** Project website location changed, URLs formatted differently.
- Version 1.0.0.4** Uli Harder removed as supervisor.
- Version 1.0.0.3** Style of this errata section changed slightly.
- Version 1.0.0.2** This errata section made into an appendix.
- Version 1.0.0.1** Minor corrections to the specification chapter (Chapter 4).

Appendix B

Bibliography

- [ACM] ACM. *ACM Transactions on Modeling and Computer Simulation*. One of the major academic journals in the field of computer simulation.
URL <http://www.acm.org/tomacs/>
- [ACM00] ACM. *ACM Digital Libraries* (ACM Press, 2000). ISBN 158113231X. See in particular the paper ‘Server Selection on the World Wide Web’.
- [BCINN] Banks, J., Carson II, J. S., Nelson, B. L., and Nicol, D. M. *Discrete-Event System Simulation* (Prentice Hall). ISBN 0130887021. A good, detailed, up-to-date book.
- [Fie00] Field, T. *Simulation and Modelling* (2000). Lecture notes from Simulation and Modelling course, lectured by Tony Field, 1999/2000 academic year.
URL <http://www.doc.ic.ac.uk/~ajf/Teaching/Simulation.html>
- [FUZ] *Fuzz — A program for testing other programs by feeding them random input..*
URL <https://sourceforge.net/projects/fuzz>
- [Gre] Greenspun, P. *Philip and Alex’s Guide to Web Publishing*, chap. 5: Learn to Program HTML (Hypertext Markup Language) in 21 Minutes (Morgan Kaufmann). ISBN 1558605347. Or available online.
URL <http://www.arsdigita.com/books/panda/html>
- [Hon98] Hong, D. P. *Constructing Web Server Performance Models* (1998).
URL <http://www1.ics.uci.edu/pub/duke/opnet98-web.pdf>
- [Jai] Jain, R. *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling* (John Wiley & Sons). ISBN 0471503363.
- [Kil] Killelea, P. *Web Performance Tuning: Speeding Up the Web* (O’Reilly). ISBN 1565923790. Has a limited amount of information which is relevant to this project.

- [KWDJ] Kuhl, F., Weatherly, R., Dahmann, J., and Jones, A. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture* (Prentice Hall). ISBN 0130225118.
- [Lew75] Lewis, T. *Distribution Sampling for Computer Simulation* (Lexington Books, 1975). ISBN 0669971391.
- [LK00] Law, A. M. and Kelton, D. W. *Simulation Modeling and Analysis* (McGraw-Hill Higher Education, 2000). ISBN 0070592926.
- [LZ00] Lu, H. and Zhou, A., eds. *WAIM (Web-Age Information Management) Conference* (Springer Verlag, 2000). ISBN 3540676279.
- [MA02] Menasce, D. E. and Almeida, V. A. *Capacity Planning for Web Services*, pp. 23–63 (Prentice Hall, Upper Saddle River, NJ 07458, 2002). ISBN 0130659037. A ‘conventional’ book on web capacity planning based on analysis and theory. Makes limited reference to web simulation.
- [MD] Murugesan, S. and Deshpande, Y., eds. *Web Engineering: Managing Diversity and Complexity of Web Application Development* (Springer Verlag). ISBN 3540421300.
- [MMF02] *Macromedia Flash* (2002).
URL <http://www.macromedia.com/software/flash/>
- [MS] Masand, B. and Spiliopoulou, M., eds. *Web Usage Analysis and User Profiling*. WEBKDD’99 Workshop (Springer Verlag). ISBN 3540678182.
- [Nie94] Nielsen, J. *Response Time Overview* (1994). Excerpt from the book *Usability Engineering* by the same author.
URL <http://www.useit.com/papers/responsetime.html>
- [SAS99] *Web Server Simulation* (1999). A Powerpoint Presentation on Web Server Models. Most of it is not relevant to this project, but they state ‘Experimenting with actual server systems can be more expensive than simulation software’ — the premise behind my project.
URL http://www.eos.ncsu.edu/eos/info/ie/ie441_info/arena/Semester%20Projects%20441/Fall_1999/Fall1999Presentations/Group1.ppt
- [Slo95] Slothouber, L. P. *A Model of Web Server Performance* (1995).
URL <http://www.geocities.com/webserverperformance/modelpaper.html>
- [W3C02a] W3C. *XML (eXtensible Markup Language)* (2002).
URL <http://www.w3.org/XML/>
- [W3C02b] W3C. *Who’s Who at the World Wide Web Consortium* (2002).
URL <http://www.w3.org/People/all#timbl>

-
- [ZPK] Zeigler, B. P., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems* (Academic Press). ISBN 0127784551.

Appendix C

List of Acronyms

| | | |
|--------------|---|--|
| CSV | Comma Separated Variables | A record-based file where each new field is delimited by a comma, and each record is delimited by a new line. They can be used often for exchanging information between spreadsheets and similar programs. |
| DTD..... | Document Type Definition | A file specifying the structure of an SGML file, of which XML is a subset. See [?] for more information. |
| FIFO | First In First Out | A queueing discipline. |
| GUI..... | Graphical User Interface | A user interface using graphics and commonly ‘windows’, such as Microsoft Windows or XWindows. |
| HTML | Hypertext Markup Language | The markup language used to create the typical web page. |
| HTTP | Hypertext Transfer Protocol | The protocol used on top of TCP/IP to transfer web data — typically but not necessarily HTML pages. |
| LAN | Local Area Network | A network typically with no more than a few hundred hosts, distributed over a small (local) area, at most typically a building or two. |
| SGML | Standard Generalized Markup Language | The more powerful and more complex precursor to XML. |
| TCP/IP | Transmission Control Protocol/Internet Protocol | The general transfer protocol used on internets and on the Internet. |
| WWW | World Wide Web | The interconnected hypertext web created by the worldwide network of HTTP servers, connected with TCP/IP. |
| XML..... | eXtensible Markup Language | A generalised markup language. See [W3C02a] for more information. |