`WebAppLoader`: A Simulation Tool Set for Evaluating Web Application Performance

Katinka Wolter
Humboldt-Universität zu Berlin,
Institut für Informatik,
Unter den Linden 6,
10099 Berlin, Germany,
wolter@informatik.hu-berlin.de

Kristian Kasprowicz
Technische Universität Berlin,
Institut für Mikroelektronik und
Technische Informatik,
Einsteinufer 17
10587 Berlin

2nd March 2003

Abstract

In this paper we present `WebAppLoader`, a set of tools that allows for analysing web application performance as it is perceived by the user. This is done by creating 'virtual users' through tracing of web sessions and then simulating groups of virtual users by simulating sets of replications of earlier traced samples. The groups are defined based on a trace, and the number of users performing a trace as well as their start times can be adapted. Measurements of the transaction times broken up into several segments are taken on the client side.

The resulting tool suite has three main components, one for tracking a web session, one for simulating repetitions of that trace and one for taking and evaluating measurements.

1 Introduction

As the use of the internet has become an integral part of every day's life its well-performing has become essential. Especially e-commerce and various kinds of web applications have become a crucial element in many people's daily life and consequently performance and reliability of web applications has become an important issue.

The work presented in this paper has been initiated by Novedia AG [Nov]. Novedia provides its customers with web solutions including the selection of all necessary equipment. In order to find out how many how powerful servers a customer will need, Novedia wants to have a tool for testing the performance of a prototype system. The customers of Novedia typically know estimates of how many visitors they expect to their sites. The objective in building `WebAppLoader` was to be able to answer the question whether a web application will perform well on a given set of computers under the expected load generated by possibly large numbers of transactions of different types.

`WebAppLoader` has several components, which together serve the purpose outlined above. The components are a recording tool for tracking and writing traces of web application sessions, a simulation tool for executing several instances of sets of the recorded traces and a simple evaluation tool for taking measurements and for showing results of the simulation runs.

The request sequences are generated by manually carrying out each sequence once and tracing it with the recording tool. A trace is then stored in XML format. One or more of these sequences are needed as profiles of a 'virtual user' for the simulator. The XML files can be modified easily in an editor. Modifying the XML file of a user session one can for instance speed up the transaction or vary the time different users spend looking at a particular page. In consequence, different virtual users follow the same path at different pace.

To make a simulation of virtual users be more realistic, several groups of users can be defined and included in the simulation simultaneously. In order to load a web server more and more over time the group sizes can increase. They can do so to a different degree and at different times for each group. As a result, `WebAppLoader` becomes a very flexible tool to generate traffic and study web server responsiveness.

There is a vast amount of related work. A number of studies on how to characterize internet traffic as well as web application traffic have been carried out by taking measurements of real traffic or creating traffic with tools such as SURGE [BC98] and WAGON [LNJV99]. The Profit tool, presented in [PJA$^+$00], takes a different angle. It measures server side response times and breaks them down into response times of the different types of servers (web server, application server, data base server) and various transactions carried out on those servers.

A widely used tool for the analysis of web application performance is httperf [MJ98] developed at Hewlett-Packard Comp. httperf is very similar to

WebAppLoader in that it also sends requests as they are specified instead of creating request sequences analytically to match given traces, as for instance SURGE does.

The tool presented in this paper is more powerful than httperf in several ways: the simulator can issue transactions not only to one given address, but it can perform full sessions consisting of several transactions specified each by a web address and a think time. It can represent not only one user, but a group of users. During a simulation run the size of groups can increase, in order to increase the server load over time. A simulation can consist of several groups, possibly using different user profiles.

The remainder of this paper is organized as follows: a short section displays the architecture of WebAppLoader, then we will present the components of the tool set in the sequence they are used; in Section 3 we present the tracing component, in Section 4 the simulation tool is described, in Section 5 results as they are obtained with the tool are presented and in Section 6 we conclude this paper.

2 The architecture of WebAppLoader

WebAppLoader is built from several components as shown in Figure 1. These components are a tracing tool, a simulator, a logger and an evaluation tool.

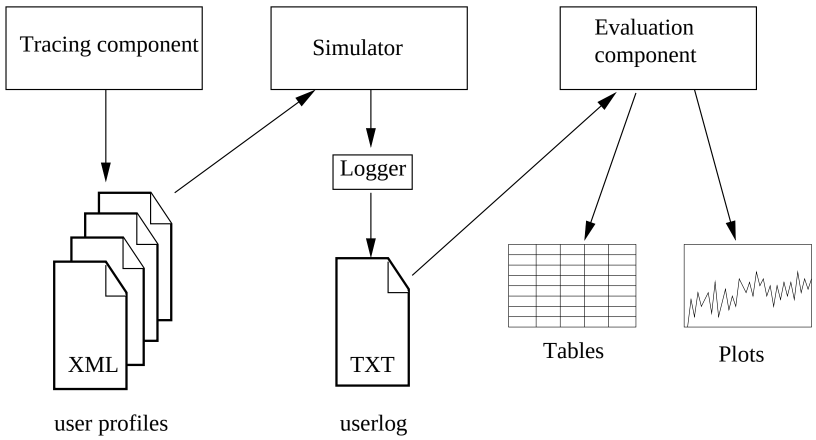

Figure 1: The architecture of the tool suite

The tracing component records a user session and creates an XML trace file for each user session. If necessary, the XML file can be edited with regular text editors outside `WebAppLoader`.

The simulator then takes the XML trace files as input and generates virtual users. Note that we implemented `WebAppLoader` in Java. Each virtual user is represented by a thread in the Java simulator executing a user profile.

The logger logs all output of the simulator. It collects measurements for various metrics, and writes the collected data to a text file. This text file is used by the evaluation component of the tool to generate statistics.

The most interesting component from a software architecture perspective is the tracing tool. The architecture of the tracing tool is shown in Figure 2. The tracing tool implements a proxy, which is plugged between the web browser and the Internet. Each request that is sent to the web application

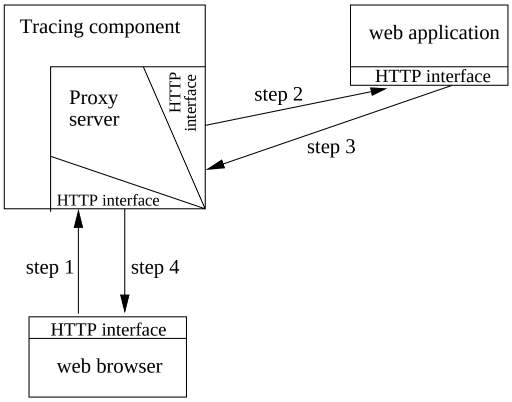

Figure 2: The architecture of the tracing tool

first passes the proxy. The request is scanned and time stamped to generate the above mentioned XML files.

We now explain the functionalities offered by the various components of `WebAppLoader` in detail.

3 The tracing component

The first step in testing a web server's performance with `WebAppLoader` is to create virtual user sessions as they will be performed by real users. A web application can offer a number of different sessions, which real users might carry out. If later all sessions shall be simulated then each of those sessions is carried out once by the tool user and meanwhile it is being traced. This is done by choosing the component *Benutzerprofil*[1] from the main menu.

Figure 3: Tracing a web session

Any web browser can be chosen for creating the traces. In order to make the web browser communicate with the tracing component of `WebAppLoader` the web browser has to be configured so that it will use the `WebAppLoader` proxy over port 8081. The proxy in `WebAppLoader` is started when the tracing component of `WebAppLoader` is active.

Figure 3 shows the interface of the tracing tool. As soon as the button *Aufzeichnen* (start recording) is pushed the tool starts tracing what web page is being loaded and how much time is spent by the test person looking at each individual page. Pushing *Aufzeichnen beenden* ends the session. The session can then be saved to a file in XML format. This file can later be edited to add some attributes that might be beneficial for simulating, or

[1] The tool set is property of Novedia and therefore most of the labeling is in German.

for changing recorded values, if it is desired. We will describe this in more detail in Section 4.

A	/htmlfiles/
B	/htmlfiles/teaching/
C	/htmlfiles/teaching/seminarWS02/
D1	/htmlfiles/teaching/seminarWS02/restart1.pdf
D2	/htmlfiles/teaching/seminarWS02/kotzAcampus.pdf
E	/
F	/htmlfiles/seminarWS02.html
G	/cgi-bin/hello

Table 1: The web session traces

For illustrating how to use the tool set we have traced three very simple web sessions. All our experiments are carried out on an HP omnibook xt6200, 1.6 GHz laptop with 256MB RAM and 20GB disk. We decided to install the Apache V1.3 web server and do all experiments locally on this machine, showing how the server performance can be measured, without dealing with network delays. We note that the pages we load are just small and for the purpose of illustration.

	URL1	URL2	URL3	URL4
session 1: seminar1				
URL sequence	A	B	C	D1
think time (sec)	16	10	15	10
no. of requests	6	1	7	1
session 2: seminar2				
URL sequence	E	A	F	D2
think time (sec)	6	4	25	35
no. of requests	2	6	1	1
session 3: helloworld				
URL sequence	G	E		
think time (sec)	22	1		
no. of requests	1	1		

Table 2: The web session traces

Two of our web sessions consist of loading a HTML page and a paper, respectively. The third one is a simple CGI-script designed to load the server more and more as it is executed by more virtual users. The script performs some simple mathematical computations.

Table 2 lists the three recorded sessions and Table 1 gives the abbreviated web locations. The think time, which is the time spent on each of the pages is listed in Table 2 in the row following the transaction sequence.

For example, the first session consists in first loading page A, where the URL abbreviated by A can be found in Table 1. The user spends 16 seconds looking at page A. Page A is loaded by issuing 6 different requests for loading the text and the included graphics. Following after page A is page B then C and D1, respectively. Pages D1 and D2 are different documents in the same location.

Some of the visited pages need several requests since they include a number of graphics that are loaded separately. The number of requests needed to load a page is given in the third row for each session in Table 2. For the trace *seminar1* the XML file can be found in Appendix A.

In the next section we will show how to use traces for creating a set of virtual users that put load on the web server. We will use the three traces to compose a mixture of groups of potential users.

4 The simulation component

The simulation component executes given XML files in as many instances as the user of the tool specifies. We now shortly describe the structure of the XML trace files, an example of which is given in Appendix A. An XML trace file holds some header definition, e.g. the document type definition (DTD) used. Then follows a

```
<browsertrace> trace <\browsertrace>
```

statement. The `browsertrace` can have an attribute called *ActiveCookieBrowser*, which indicates that each user using the profile will be treated as a new user, not sending any cookies and hence receiving new ones from the web server. The default value is *SimplePlaybackBrowser* that will keep all cookies and reuse them for new virtual users.

A `browsertrace` consists of several

```
<page> page <\page>
```

statements. A page can have two attributes, *thinktime*, the time that is spent looking at this page and *thinktimemode*, which can be either *fixed* or *random*. *Thinktimemode fixed* indicates that each virtual user in the simulation should spend this exact time on a page, while *thinktimemode random* means that the time each virtual user spends on a page is to be chosen randomly.

Each page then is composed of a number of

```
<request> page element <\request>
```

statements that load the different elements of a web page. Usually the elements are text, pictures, or icons. A `request` can have the attributes *GET* and *POST*, where the former means that a page is to be loaded, whereas the latter indicates posting of a page to some location. The default value is *GET*.

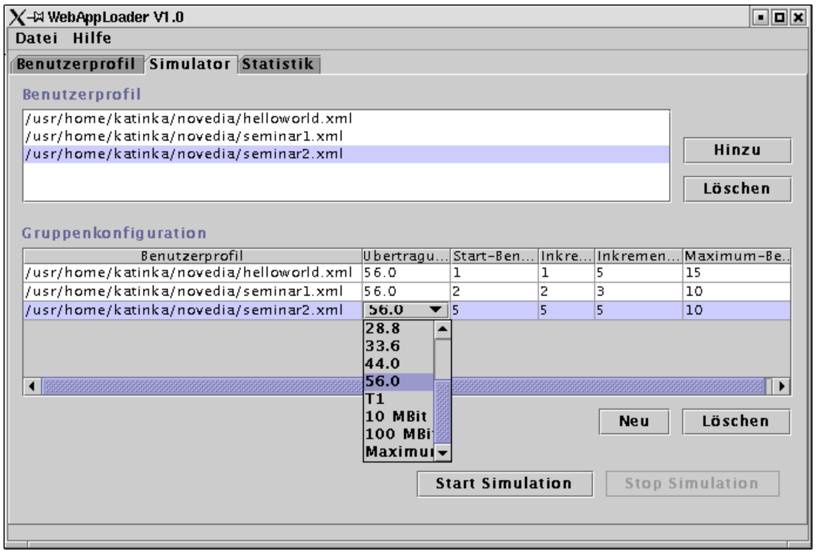

Figure 4: The simulation tool

In addition to the attributes given in the XML file representing the session, the simulation tool accepts a set of parameters that are specified in the

Profile	connection speed	initial #users	inc. users	inc. time interval	max. users
helloworld.xml	56Kbps	1	1	5 seconds	15
seminar1.xml	56Kbps	2	2	3 seconds	10
seminar2.xml	56Kbps	5	5	5 seconds	10

Table 3: The parameters for the simulation

simulation interface for each group of users. A group is a set of virtual users using the same profile, possibly running concurrently. In our tool implementation each virtual user is assigned a thread. This allows us to run several hundreds of virtual users concurrently. For each group we set the speed of the network connection, that can range from a slow simple modem connection up to a very fast Ethernet connection. We chose in our experiments the speed of a good modem connection, 56Kbit/sec. The interface has fields for the initial number of users of each group, the number of users that is added each time the group size increases and the time interval in seconds after which the group size is supposed to increase. Pushing the button *start simulation* will start a simulation run, pushing the button *stop simulation* does end it.

We specify a scenario in Figure 4 and Table 3. We generated traffic for approximately 1 hour and 20 minutes. A logger attached to the simulator takes a number of time stamps and writes them to an output file which will be studied in the next section.

5 Results

In this section we first look at what measurements are being taken by the tool and then we analyze the data to show the kind of questions that can be answered by using `WebAppLoader`. The tool uses 5 different time stamps ($t_0, \ldots, t_4$), as shown in Figure 5. The first measurement point is the initiation time of a transaction, the second is the time at which a connection with the server has been established, the third is when a request is sent out completely, the fourth measurement point is after receipt of the first byte of the answer from the server and the last measurement is taken when the answer has been obtained completely. From these measurement points we can derive time metrics as shown in Table 4.

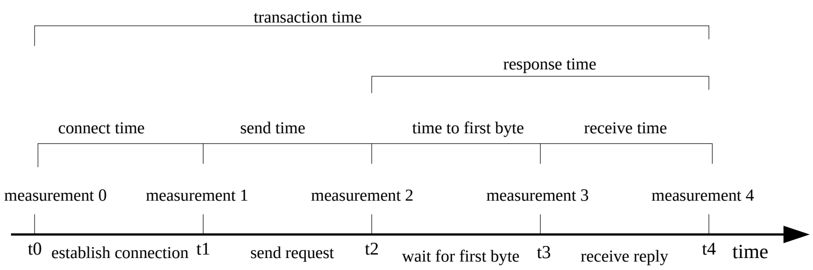

Figure 5: Measurement points

The transaction time is the total sum of all measured times and the response time is the time to first byte (TTFB) plus the receive time. Ignoring the network delay, the response time corresponds to the time the server needs to process and transmit a page. All measurements are taken in the proxy,

connect time (CT)	t1 - t0
send time (ST)	t2 - t1
time to first byte (TTFB)	t3 - t2
receive time (RT)	t4 - t3
response time (ReT)	t4 - t2
transaction time (TT)	t4 - t1

Table 4: The measurement intervals

we do not instrument either the server or the network.

Figure 6 shows the results given by `WebAppLoader` in the menu *Statistik* after loading the file *userlog.txt*. Clicking on *root* will provide the list of groups that have been included in the simulation run. In the lower part of the window the average number of transactions per second and the average number of bytes transmitted per second is shown. As the user chooses a group, the pages loaded for that group are shown in the right most column (see Figure 7 on the left).

In the bottom part of the window the average, minimum and maximum transaction time in milliseconds for that page are shown, as well as the average bytes loaded per second while requesting that page.

As one clicks on page i in the right section of the upper part of the window (shown in Figure 7 on the right side), the lower part of the window shows the

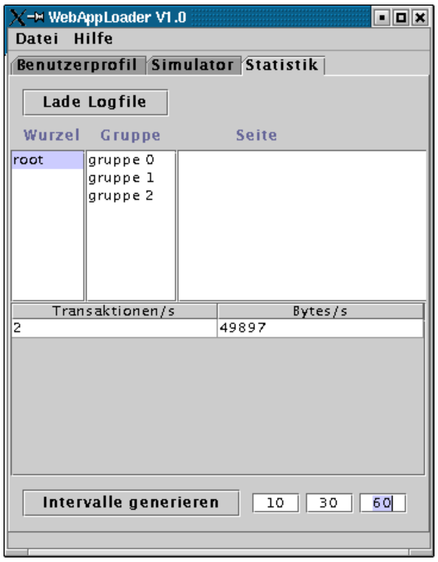

Figure 6: First level results output

list of requests that are sent out for loading the page. For each request the average, maximum and minimum transaction time are shown as well as the average number of bytes transmitted and the status code. The status code field indicates what percentage of requests achieved HTTP status code 200, which means successful completion. Other values of the status code indicate that the pages have not been found, or have still been cached, so they were not reloaded. For the purpose of analyzing the web server performance we only consider requests with status code 200.

On the bottom of the *Statistik* window there is a button *Intervalle generieren*. Pushing this button generates averages over time intervals in seconds as specified in the three fields next to the button. The size of the time intervals can be arbitrarily chosen. We have computed 10-, 30-, and 60-second averages. In the discussion below we always use the data from the 60-second averages if not stated otherwise.

From the transaction times and loaded bytes per second we can derive which pages are big in size (these take long at high rate of bytes per second). Using our tool, we answer some questions about the reasons of delays by looking at the output file `WebAppLoader` generates. The answers we give are specific for our scenario, but they show the kind of answers that can be obtained by

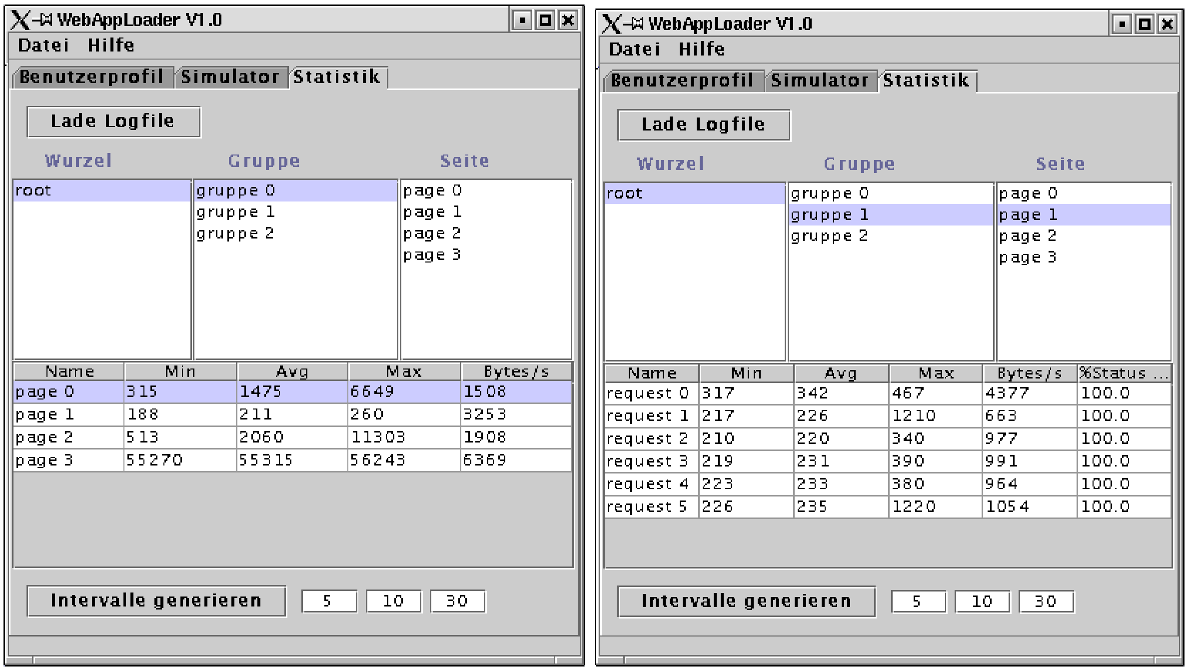

Figure 7: More detailed results output

using `WebAppLoader` in an arbitrary setting to study a system's behavior.

The output file gives all measurements (see Table 5) for all groups, individual virtual users in a group, pages per virtual user and requests per page. All plots in this paper are generated using the output file of `WebAppLoader`.

Figure 8 shows the observations for the average transaction time, the receive time, the time to the first byte, the average number of bytes transmitted per second and the average number of active virtual users. All averages are taken over 60-second intervals, while all times are measured in milliseconds. To make the curves all fit into one plot some of the data has been scaled as labeled in the legend. E.g. the curve showing the number of bytes transmitted has to be multiplied by 10 to obtain the real numbers, while the number of active users has to be divided by 250.

Figures 9 and 10 show selected curves taken from Figure 8 for easier interpretation.

The curves in Figure 9 on the left indicate at least trends: i) the transaction

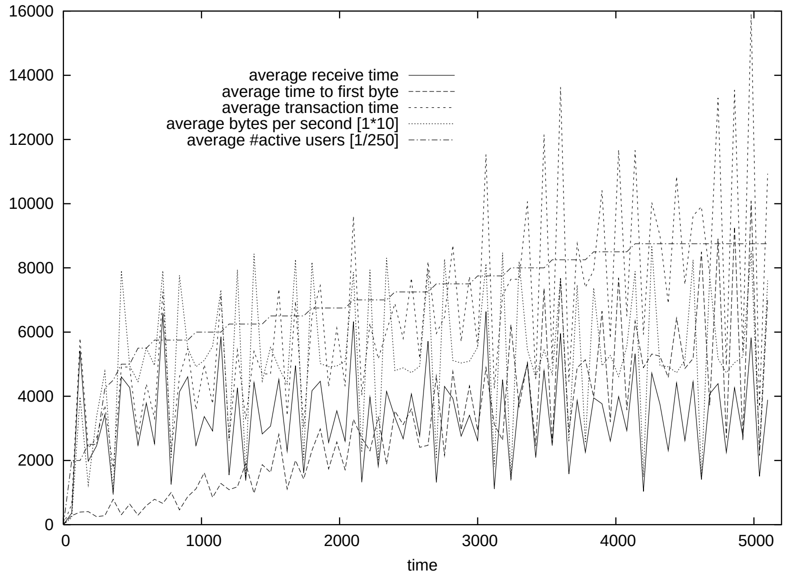

Figure 8: Averages over 60 second intervals

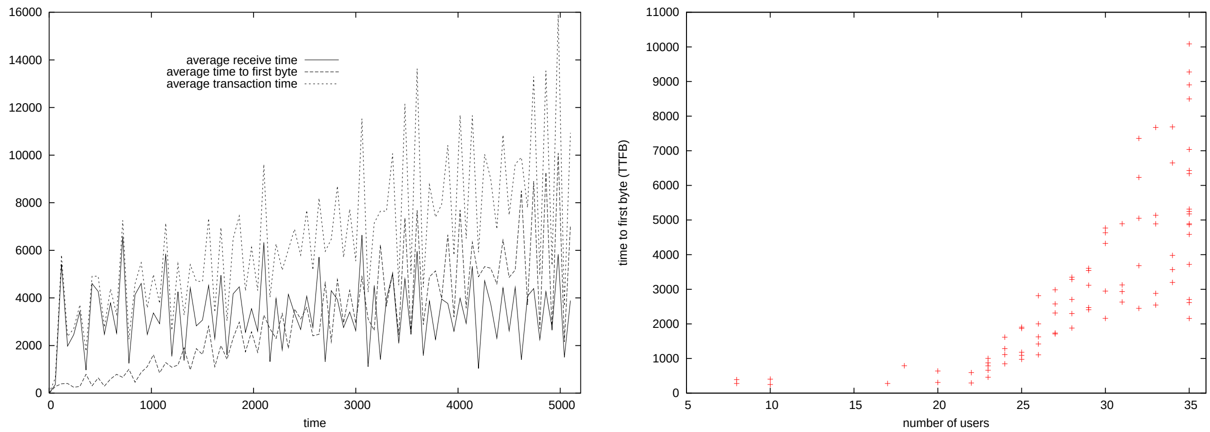

Figure 9: Addition of time periods in 60 second intervals and average TTFB versus number of users

time increases with time, ii) the receive time remains constant over time, and iii) the time to the first byte increases over time. Note, that the transaction time is the sum of the time to the first byte and the receive time, since all other contributing metrics equal zero in our measurements. So the trend in the transaction time is dominated by the trend in the time to the first byte.

We also see from Figure 8 that the number of virtual users increases over time, and an obvious conclusion is that the time to the first byte increases, because more and more virtual users execute the CGI-script, carrying out computation on the web server. Figure 9 on the right side therefore shows the TTFB versus the number of virtual users in the original data. Note, that the initial number of virtual users was 8, which very soon increased to 20 and more. Obviously, there is a strong correlation between the two metrics, which we do not try to characterize formally in this paper.

In order to make sure, that the delay is not due to slower file transmission, Figure 10 shows the average number of transmitted bytes per second and the average number of virtual users. After an initial increase the average number of transmitted bytes shows no trend over time, while the number of virtual users increases.

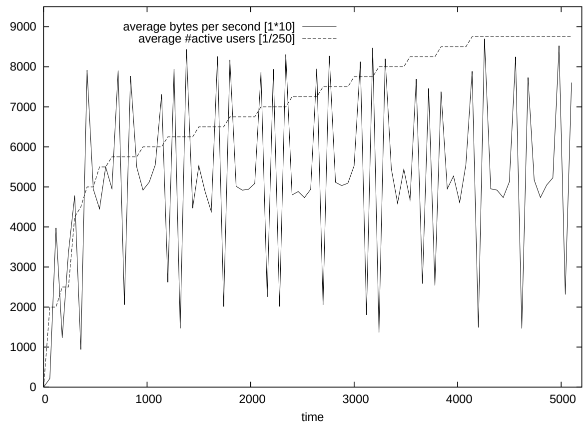

Figure 10: Transmitted bytes per second and number of users in 60 second intervals

It can be concluded from the above that the bottleneck in this web application is the computing power of the web server. If we consider a response time of up to 8 seconds acceptable, this corresponds to a TTFB of roughly 4 seconds (see Figure 9 on the left), then the system can deal with up to

28 users (see Figure 9 on the right). If we want to allow for more users, we should add a new server, possibly an application server to do the computations. If estimated network delays, as they will occur to real users, should be taken into consideration, the acceptable transaction time and the number of users will be even less. We do not carry out sound analysis in this paper but merely demonstrate how `WebAppLoader` can help gaining the desired insights for planning and dimensioning of web applications.

6 Conclusions

In this paper `WebAppLoader`, a tool for planning and dimensioning web services, has been presented. The interesting features of the tool are the sophisticated means by which user sessions can be specified and mixed. It has been shown that the tool set allows for *a priori* quantifying the capacity of a web application to be newly installed as well as for finding bottlenecks in the performance of a web application. In future work data analysis components should be added to extract automatically the important information from the measurements.

References

[BC98] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS Performance Evaluation Review, Proc. of the ACM SIGMETRICS joint Intl. Conf. on Measurement and Modeling of Computer Systems*, volume 26(1), pages 151–160, June 1998.

[LNJV99] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. *System Performance Evaluation: Methodologies and Applications*, chapter Web Server Benchmarking and Web Traffic Modeling. CRC Press, 1999.

[MJ98] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *First Workshop on Internet Server Performance*, pages 59–67, Madison, WI, USA, June 1998. ACM.

[Nov] http://www.novedia.de.

[PJA+00] G. T. Paixão, W. Meira Jr., V. A. F. Almeida, D. A. Menasce, and A. M. Pereira. Design and Implementation if a Tool for Measuring the Performance of Complex E-Commerce Sites. In *Proc. 11th Int. Conf. on Computer Performance Evaluation; Modelling Techniques and Tools*, LNCS 1786, pages 309–323, Schaumburg, IL, USA, March 2000.

A The seminar1.xml user profile


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE browsertrace SYSTEM "newxml.dtd">
<browsertrace browsertype="de.novedia.webapploader.ActiveCookieBrowser">
    <page thinktime="16">
        <request method="GET" url="/htmlfiles/" version="HTTP/1.0">
            <header>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*</accept>
                <accept-encoding>gzip</accept-encoding>
                <accept-language>en</accept-language>
                <accept-charset>iso-8859-1,*,utf-8</accept-charset>
            </header>
        </request>
        <request method="GET" url="/icons/blank.gif" version="HTTP/1.0">
            <header>
                <referer>http://localhost:8080/htmlfiles/</referer>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png</accept>
                <accept-encoding>gzip</accept-encoding>
                <accept-language>en</accept-language>
                <accept-charset>iso-8859-1,*,utf-8</accept-charset>
            </header>
        </request>
        <request method="GET" url="/icons/back.gif" version="HTTP/1.0">
            <header>
                  .
                  .
            </header>
        </request>
        <request method="GET" url="/icons/text.gif" version="HTTP/1.0">
            <header>
                  .
                  .
            </header>
        </request>
        <request method="GET" url="/icons/folder.gif" version="HTTP/1.0">
            <header>
                  .
                  .
            </header>
```

```
        </request>
        <request method="GET" url="/icons/unknown.gif" version="HTTP/1.0">
            <header>
                  .
                  .
            </header>
        </request>
    </page>
    <page thinktime="10">
        <request method="GET" url="/htmlfiles/teaching/" version="HTTP/1.0">
            <header>
                <referer>http://localhost:8080/htmlfiles/</referer>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*</accept>
                <accept-encoding>gzip</accept-encoding>
                <accept-language>en</accept-language>
                <accept-charset>iso-8859-1,*,utf-8</accept-charset>
            </header>
        </request>
    </page>
    <page thinktime="15">
        <request method="GET" url="/htmlfiles/teaching/seminarWS02/" version="HTTP/1.0">
            <header>
                <referer>http://localhost:8080/htmlfiles/teaching/</referer>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*</accept>
                <accept-encoding>gzip</accept-encoding>
                <accept-language>en</accept-language>
                <accept-charset>iso-8859-1,*,utf-8</accept-charset>
            </header>
        </request>
        <request method="GET" url="/icons/layout.gif" version="HTTP/1.0">
            <header>
                <referer>http://localhost:8080/htmlfiles/teaching/seminarWS02/</referer>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png</accept>
                <accept-encoding>gzip</accept-encoding>
                <accept-language>en</accept-language>
                <accept-charset>iso-8859-1,*,utf-8</accept-charset>
            </header>
        </request>
              .
              .
              .
              .
  </page>
  <page thinktime="10">
        <request method="GET" url="/htmlfiles/teaching/seminarWS02/restart1.pdf" version="HTTP/1.0">
            <header>
                <referer>http://localhost:8080/htmlfiles/teaching/seminarWS02/</referer>
                <user-agent>Mozilla/4.77 [en] (X11; U; Linux 2.4.19 i686)</user-agent>
                <host>localhost:8080</host>
                <accept>image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*</accept>
```

```
                &lt;accept-encoding&gt;gzip&lt;/accept-encoding&gt;
                &lt;accept-language&gt;en&lt;/accept-language&gt;
                &lt;accept-charset&gt;iso-8859-1,*,utf-8&lt;/accept-charset&gt;
            &lt;/header&gt;
        &lt;/request&gt;
    &lt;/page&gt;
&lt;/browsertrace&gt;
```