Performance Measurement of a Web Application

CS 681 Assignment 1. Released: January 12, 2026. Due: Feb 1st, 2026

Assignment is to be done by TWO team members.

Overall Goal of the Assignment

The high level goal of the assignment is for learners to “experience” and understand the performance of an actual queuing system first-hand, by direct measurement of metrics, and experimenting with how these metrics change with various parameters.

The example we will use is of a Web Server, which was discussed in class as a typical queuing system. The resources a Web server uses are socket connections, threads, CPU, network, disk, memory and so on. Each resource forms a queueing system of its own and impacts overall performance. You will try your best to understand these impacts, and *relate* what you are learning in class to what you observe in the experiments.

Detailed Specification

For this assignment you will need TWO machines.

One one machine, you must install a Web server (e.g. apache, or any other web server of your choice). You will also be given a php script to run on this Web server, so you must load the PHP module and make PHP work with this Web server.

On the other machine, you will install two *Web load generators.* Load generators are programs which emulate users of a Web site by sending certain specified URLs to the Web server, reading the response, and as a result noting down performance metrics such as response time, throughput, number of requests lost, timed out, etc.

The two load generators to be installed are: httperf, and Tsung. Httperf is capable of generating an “open” load - i.e. open arrivals specified by an arrival rate, while Tsung generates a “closed load” - i.e. it creates multiple virtual users who are in a request-response loop with the server.

Both these tools will take specifications in their own formats, of which URLs are to be sent by the tool, and the load quantifier. For httperf, you have to specify an arrival rate, and for Tsung, a number of users and *think time.* In both cases, a load test duration is specified. To get sensible results, you must run a load test at each “load level” for at least 5 minutes.

Each of these tools will output "client-side" performance metrics such as response time, throughput, requests refused, timed out, etc.

For this assignment, you are also required to measure some “server-side” performance metrics. Such metrics can be measured using Linux utilities such as top and vmstat while the load generation is going on.

The Load Test

Your "loadtest" will generally proceed as follows:

- Start Web server. ENSURE THAT NO OTHER PROCESS IS RUNNING ON THE SAME MACHINE THAT MAY AFFECT PERFORMANCE.
- Start load (e.g. httperf on the client machine) - here also, it is best that client is not running other major processes. Run the load for a few minutes. For this assignment I suggest setting the timeout to something very large (we won't measure timeouts).
- As soon as you start load, start server side performance measurement (top, or vmstat that writes out measurement snapshot periodically into a file). Wait for a few minutes (load generation should be going on, 4-5 mins should be enough) and then stop the server side performance measurement
- Stop the load generation.

These steps are important because you must measure the server only when a steady load of the rate that you specified is actually coming to the server. If you start measurements before load starts, then you may get nearly zero CPU utilization. The same problem will happen if you continue taking measurements after load generation has stopped. So it is critical that the average CPU utilization is calculated only for the phase when the server is busy serving requests at the rate offered to it.

For generating performance curves, you must repeat the load test at various load levels.

Server side performance measurement

Also, it is very important to take an average of snapshots only in the duration that the server is ‘warmed up’ and also before it starts ‘cooing down’. You can do this by plotting the time series of the utilization snapshots and just ‘eyeball’ some the duration which seemed after warm up and before cool down. Take an average of only this period.

You have to ensure that your server side configuration is sensible. Ensure that Apache has enough threads so that threads are never the bottleneck. (About 50 threads per core is a safe choice, but confirm this.) Also, adjust the loop count in the PHP script so that the time taken without any load is 30-50 milliseconds (or even more if required). This helps in keeping the service rate of the server low, and thus it can be stressed by generating load from a basic machine. (High performance servers are hard to stress, and you will find the assignment difficult to do.)

It will be easiest if you run this assignment on a single-core server (find out the command by which you can disable cores). If you have a multi-core server, ensure again that the number of Web server threads is enough, and remember this fact when you make plots and compare with theory (multiple server queueing system). Also you may need to increase your service time even further, so as to again ensure that overall service rate is low.

Plots of performance metrics

We will study the behaviour of performance metrics by mainly varying the request arrival rate for open load, and the number of users, for closed load.

For generating sensible plots, you should generate “points” corresponding to a very low CPU utilization (~5%) and go smoothly up to 100% system utilization. At least 10 points evenly spaced points should be generated. This means you have to do at least ten “runs” with varying load (request arrival rate or number of users). Ensure that the points are evenly spaced out - one hint is that your throughput curve should be such that it increases and then you can *see* it flattening out (or starts dropping). Most of the graph should have points *before* it flattens out, but there should be enough points for us to see throughput flattening/dropping.

Open Load

Collect data to plot the following curves. In each of the following curves the metric (the first quantity) should be on Y and the second quantity should be on X

- Response time vs request arrival rate
- Throughput vs request arrival rate
- Server CPU utilization vs request arrival rate
- Fraction of requests dropped (basically, unsuccessful) vs request arrival rate

Based on the plots, make the following observations & calculations

- For each plot comment on its "nature" (Increasing? Decreasing? Linear? Sub-linear? Exponential? What is the min? Is it flattening out? Increasing then dropping? Etc...).
- Explain intuitively why you see this type of plot (E.g. why is it increasing?). The explanation in most cases will be obvious and can be brief. In some cases if you have no explanation simply state so.
- What is the maximum load in terms of request rate supported by this server?
- Is your utilization graph roughly linear? Use it (with Utilization Law) to estimate the *service time* of the request on the Web server CPU. Remember to account for multiple CPUs.

- Does your response time vs load curve have a shape in which there is graceful increase initially, and then a sharp increase? At what point approximately does this sharp increase occur? Find this point in terms of request rate and utilization.
- Use Little's Law to estimate the average number of requests at the server. (For bonus points, explore whether there is a way to verify this number by direct measurement)

Closed Load

Collect data to plot the following curves. You will have to choose a **think time** value - choose one such that the maximum number of users that the server can "support" is at least in the 100s. Most likely a 6 second think time should work, but confirm this.

In each of the following curves the metric (the first quantity) should be on Y and the second quantity should be on X

- Response time vs number of users
- Throughput vs number of users
- Server CPU utilization vs number of users
- Response time vs throughput
- Server Utilization vs throughput
- Fraction of requests dropped (basically, unsuccessful) vs number of users

Based on the plots, make the following observations & calculations

- For each plot comment on its "nature" (Increasing? Decreasing? Linear? Sub-linear? Exponential? What is the min? Is it flattening out? Increasing then dropping? Etc...).
- Explain intuitively why you see this type of plot (E.g. why is it increasing?). The explanation in most cases will be obvious and can be brief. In some cases if you have no explanation simply state so.
- What is the maximum load in terms of *number of users* supported by this server?
- Does your response time vs load curve have a shape in which there is a very slight increase initially, and then a sharper increase, after which the plot looks linear (is the asymptote a line?) What is the average slope of this line (asymptote)?
- Apply utilization low to the utilization vs throughput plot - do you get the same service time as for open load?
- Apply the closed system's Little's Law and see if it is verified at all points.
    - One way to do this is to express Think Time in terms of the Little's law, and check whether it is equal to the configured Think Time.
    - If it is not equal, what is your explanation?

Submission Instructions and Deadlines

Submit a report in the form of **a presentation (in pdf) only.** Do not write a "document", it is not required. Explain your setup (h/w, s/w details) and then add the graphs and observations and other calculations. Add any overall summary and conclusions.

Observations, inferences and conclusions are very important. Do not under-emphasize them.

Deadlines:

1. Intermediate submission (Optional, for feedback only): Open Load Analysis, Thursday, January 22nd, 11:59pm. (Upload link TBD on Bodhitree)
2. Final Submission: Open and Closed Load Analysis (slides in a single pdf): **DEADLINE: 11:59pm, Sunday, Feb 1st.**

Appendix: Details about httperf and tsung

httperf basic usage:

Home Page: http://www.hpl.hp.com/research/linux/httperf/

https://github.com/httperf/httperf

Installation:

```
wget https://github.com/rtCamp/httperf/archive/master.zip
unzip master.zip
cd httperf-master
autoreconf -i
mkdir build
cd build
../configure
make
make install
```

Usage:

```
httperf --hog --server <server ip/hostname> --uri /<url of the php> --num-conns 200 --num-calls 5 --rate 150
```

Link of documentation:

http://www.hpl.hp.com/research/linux/httperf/httperf-man-0.9.pdf

tsung basic usage:

Installation:

```
sudo apt-get install build-essential debhelper \
erlang-nox erlang-dev \
python-matplotlib gnuplot \
libtemplate-perl
wget https://github.com/processone/tsung/archive/v1.5.0.tar.gz
tar -xvzf v1.5.0.tar.gz
cd tsung-1.5.0
./configure
make
make deb
cd ..
sudo dpkg -i tsung_1.5.0-1_all.deb
```

Usage:

1) Running Tsung:

   Go to the directory where the configuration xml file is stored
   tsung -f <configuration xml file> -l . -m result.log start

   (Default location of log file is : /home/$USER/.tsung/log/20150111-2331/tsubg.log)

2) Report Generation:

   Go to the directory where log is generated
   /usr/lib/tsung/bin/tsung_stats.pl --stats result.log

Link of Documentation:

http://tsung.erlang-projects.org/user_manual/index.html

Sample configuration file:

```
<?xml version="1.0"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd">
<tsung loglevel="notice" version="1.0">

<clients>
        <client host="localhost" maxusers="200" use_controller_vm="true"/>
</clients>

<servers>
        <server host="server-ip" port="80" type="tcp"/>
</servers>

<load duration="120" unit="second">
<arrivalphase phase="1" duration="120" unit="second">
        <users maxnumber="200" arrivalrate="200" unit="second"/>
</arrivalphase>
</load>

<options>
<option type="ts_http" name="user_agent">
        <user_agent probability="20">Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:28.0)
Gecko/20100101 Firefox/28.0</user_agent>
</option>
<option name="thinktime" value="4" random="true" override="true"/>
</options>

<sessions>

        <session probability="100" name="login" type="ts_http">

        <for from="1" to="2000000" var="i">


        <transaction name="loginURL">
        <request subst="true">
                <http url="Login URL goes here" version="1.1">
                <http_header name="Cache-Control" value="no-cache"/></http>
        </request>
        </transaction>
```

```
        <thinktime value="4" random="true"/>

        </for>

        </session>
</sessions>
</tsung>
```

Note for tsung:
load duration: This is total load duration.
arrival phase duration: This should be same as total load duration if we running a single phase.
maxnumber: This value should be same as maxuser (total number of users) inside <client> tag.
arrivalrate: This is user creation rate (not same as httperf arrival rate). Here maxnumber=100 and arrival rate=50/sec means 50 users will be created per second, so 100 users will be created in two seconds.
For loop is necessary to keep users running in loop for specified duration. Otherwise each user will generate a request and then exit from the system.

For experiment result part please refer to auto-generated report.html page. In “Main Statistics” section “request” row and “Mean” column gives you the average response time. The “Highest Rate” column is not the actual average throughput as tsung reports highest throughput at 10sec time interval. In an alternative way you can get total number of successful requests (HTTP Code 200) and divide this value by load duration.

Server side script:

```
<html>
<body>
<?php
$time1 = microtime(true);
#echo $time1."<br/><br/>";
$val=400000;
for($i=1;$i<=$val;$i++);
$time2 = microtime(true);
#echo $time2."<br/><br/>";
$time=$time2-$time1;
echo "Time taken = $time sec";
?>
</html>
</body>
```

You can run the above php script in server. Save it in location /var/www as test.php
You can modify the value of $val depending on your requirement.

If you do not have two laptops (one client, one server) to do the above experiment, we can arrange something in the lab. Please let us know.