Distributed Systems
Performance Tutorial

Cristian Klein, Ewnetu Bayuh, Francisco Hernandez-Rodriguez
2014-09-30

This tutorial will allow you to more intuitively discover the concepts introduced during the lecture part. The exercises will help you understand the factors that affect throughput and response time, so as to take better decisions in designing a distributed system or improving an existing one. We kindly ask you to fill in the tables with measurements and give short, one-sentence answers to the questions asked. This serves both for yourself as a self-evaluation and for us to evaluate the effectiveness of the course and allow for early discovery of potential unclarity in our presentation.

Note that the tutorial has been designed and tested for Linux. The tutorial is divided in three parts: setting up, throughput and response time.

1 Setting up

1. Download the tutorial kit located at:
   `http://www8.cs.umu.se/~cklein/teaching/performance-tutorial.tar.gz`

2. Decompress it in a convenient location, for example, your home directory.
   `tar -xzvf performance-tutorial.tar.gz`

3. Study and **adapt** the files in the decompressed apache directory, especially:
   - `apache2.conf` (you need to change the `ServerRoot`, `DocumentRoot` and `Directory` line)
   - `htroot/heavy.php`

4. Start the Apache web server:
   `apache2 -d apache/`

   If you get a "command not found" error, try using the following command instead:
   `/usr/lib/apache2/mpm-prefork/apache2 -d apache/`

5. Test that the web server is working correctly. In your browser open the following URL:
   `http://localhost:8080/heavy.php?n=1000`

   You should get the message "done". The PHP script that you just invoked simulates a computationally heavy service. The amount of computing power it requires is roughly proportional to the n parameter. We will be able to use this to see what happens with the performance of the system as the service time varies.

6. Test and **understand** the workload generator:

`./httpmon --help`

Type the following command **on a single line**. Since you are going to reuse this command quite often (e.g., from your shell's history), make sure that you typed **all parameters correctly**.

```
./httpmon --url "http://localhost:8080/heavy.php?n=1000"
   --thinktime 1 --concurrency 1 --interval 10
```

It accepts, among others, the following parameters:

- target URL;
- think-time of users, i.e., the amount of time that elapses between consecutive requests of a user;
- number of users (called concurrency);
- reporting interval.

Every 10 seconds, as configured using the interval option, it reports the following:

- time (in Unix timestamp[1]) at which the report was issued;
- latency: minimum, first quartile[2], median, third quartile, maximum and average;
- 95th-percentile[3] latency;
- 99th-percentile latency;
- throughput.

If you are unfamiliar with any of these statistics-related terms, please read the linked Wikipedia articles. Note that the first report might look worse than the rest. This is due to the fact that the system might be "cold" and it needs a few requests to "warm" up. In the rest of the tutorial, you should ignore this first report, as it does not represent the steady-state behavior of the system and focus on the second or third report instead. Press CTRL+C to stop the workload generator.

7. Kill the server. We shall restart it later with new settings:
   `killall apache2`

2 Throughput

Let us first focus on throughput. For now, ignore the latency reports. To efficiently complete this tutorial, we recommend having 3 terminal windows open side-by-side. We shall call them TermA, TermB and TermC.

Your machine has 4 CPU cores. To emulate a machine with fewer CPU cores, we shall use the `taskset` command to restrict what cores `apache` may use: `taskset -c 0-1` means

[1] `https://en.wikipedia.org/wiki/Unix_timestamp`
[2] `https://en.wikipedia.org/wiki/Quartile`
[3] `https://en.wikipedia.org/wiki/Percentile`

that the following command may only use the first and second core. You can read more about `taskset` in its manpage[4].

1. TermA: launch the `top` command and press the key 1. This will show the status of all CPUs.

2. TermB: launch the apache web server on a single processor as follows:
   `taskset -c 0 apache2 -d apache/`

   Normally, it should daemonize.

3. TermC: run the workload generator with an increasing number of users and record the obtained values in the table below.
   Q1. What can you observe regarding the throughput? How does it vary depending on the number of users?
   Q2. What about the CPU usage? For what CPU usage is the maximum throughput obtained?

4. TermB: kill the apache web server and re-launch it on 2 CPUs as follows:
   `taskset -c 0-1 apache2 -d apache/`

5. TermC: repeat step 3

6. TermB: kill the apache web server and re-launch it on 4 CPUs as follows: `taskset -c 0-3 apache2 -d apache/`

7. TermC: repeat step 3
   Q3. What can you say regarding the throughput? How does the maximum throughput vary with the number of CPUs?

8. TermB: launch apache on a single CPU as done at step 2.

9. TermC: test the throughput of the "light" service, i.e., repeat step 3 with the following URL:
   `http://localhost:8080/heavy.php?n=100`

   Notice the 10-fold reduction in required computing power, controlled by the n parameter.
   Q4. What do you observe?

# users	1	10	20	50	100	150	200
Throughput (1 CPU)							
Throughput (2 CPUs)							
Throughput (4 CPUs)							
Throughput (1 CPU, light service)							

[4] `http://linux.die.net/man/1/taskset`

3 Response Time (or Latency)

Let us now direct our attention to latency. For latency, we are not only interested in the average, but also the distribution. Hopefully, by now you are familiar with the experimental methodology, so we expect you to type the correct commands yourself. If not, do not hesitate to ask for our help. Fill in the following table for 1 CPU and the "heavy" ($n = 1000$) service:

	latency			
# users	average	95%	99%	CPU usage
1 (base latency)				
2				
5				
10				
20				
30				
50				

Q5. How does average latency vary with the number of users?

Q6. What about 95% or 99% latency, how many times is it larger than average latency?

Q7. What about the CPU utilization? When would you consider the system "saturated", i.e., unable to serve users in a timely manner (compare to the base latency)?

Feel free to come up with your own experiments and even to draw some graphs like the ones you saw during the lecture. As a token of appreciation for this last effort, you will receive the **diligent** award.

Answer Sheet Name: _____