Laboratory 2: Server benchmarking

Introduction

The laboratory consists of following parts:

1. Building, running and basic testing of a simple HTTP server provided by the lecturer
2. Benchmarking the provided server
3. Modifying the HTTP server so it does no computation when it approaches overload
4. Benchmarking the provided server again

Building and running the example HTTP server

1. Login to one of the servers

```
ssh <username>@<username>vm101.rtb-lab.pl
```

2. Checkout the github repo with the labs

```
git clone https://github.com/RTBHOUSE/mimuw-lab.git
```

2. Build the maven project with the HTTP server

```
sudo docker run --rm -it -v $(pwd)/mimuw-lab:/mimuw-lab
maven:3.8.4-openjdk-17-slim sh -c "cd /mimuw-lab/lab02 && mvn install"
```

3. Run the HTTP server

Start the HTTP server like this:

```
sudo docker run --network=host --rm -it -v $(pwd)/mimuw-lab:/mimuw-lab
openjdk:17.0.2-buster sh -c "java -jar
/mimuw-lab/lab02/worker/target/worker-1.0-SNAPSHOT-shaded.jar"
```

vm101 will act as the server host in the exercises.

Now login to another vm:

```
ssh <username>@<username>vm102.rtb-lab.pl
```

vm102 will act as the client host in the exercises.

You should be able to make a request to the server and get a response:

```
curl 'http://<username>vm101.rtb-lab.pl:8000/?n=2500000'
```

1. Benchmarking the server

The tests use this benchmarking tool: https://github.com/tsenart/vegeta

Look through the homepage for the tool to get some idea what it does and how it works.

Then, perform a sequence of tests like the one below, increasing the rate parameter successively:

```
sudo docker run -it --rm peterevans/vegeta sh -c "echo 'GET
http://<username>vm101.rtb-lab.pl:8000/?n=2500000' | vegeta attack
-rate=10 -duration=30s | vegeta report"
```

Figure out what rates to test so that you get 10-20 test points and that the last few test points report the same throughput or decreasing throughput (indicating this is the maximum throughput achievable for this server on this workload).

Note down the rates, the mean latency and the 99-percentile latency into a spreadsheet, so that you end up with a table like this:

Rate	Mean latency	p99 latency
5	100ms	200ms
10	200ms	400ms
…	…	…

Use the table to create plots:

Plot 1: rate on X axis vs mean latency on Y axis
Plot 2: rate on X axis vs p99 latency on Y axis

2. Implement load shedding

The server does its core computation using a dedicated thread pool. Your goal is to implement load shedding in the server. The server should shed load only when under heavy load. One way of doing it is this:

1. Using another thread, periodically (every Xms) check the utilisation of the thread pool
2. Keep N most recent samples of the utilisation
3. When processing the request, before starting to do work, check the average of the N most recent samples. If it is higher than some threshold U%, send a dummy HTTP response without doing the normal computation work (without running the dummy sqrt() load).

Java has a ScheduledExecutorService class that allows easy execution of periodic tasks and the Executors class has helpful helper methods for creating executor services. Call scheduleWithFixedDelay on the ScheduledExecutorService to execute something every X ms. You will need some data structure for storing the samples and somehow synchronise its updating (from the scheduled executor) and its reading (from the logic that will decide if process request as usually or return a dummy HTTP response). Try to pick sensible values for X (how often to update the utilisation), N (how many utilisation samples to keep) and for U% (at what average utilisation to start shedding).

3. Benchmarking the server a second time

Repeat the tests from point 1 and create a second set of plots. Make sure that load is only shedded when the server is under some load.