the URL for your git repo.

https://github.com/jerryyummy/distributed\_system\_assignment1

https://github.com/jerryyummy/assignment1\_server

a 1-2 page description of your client design. Include major classes, packages, relationships, whatever you need to convey concisely how your client works.

Major classes:

1.ThreadGroupHandler(int threadGroupSize, int numThreadGroups, int delaySeconds, String IPAddr)

It will create a countdownlatch to generate all threadgroup, and every group will revoke multiple mainthread

2.MainThread(int threadGroupSize, int delaySeconds, String IPAddr,int id,CountDownLatch latch,BufferedWriter out)

It will create a countdownlatch to genrate all threads, and every thread will call clientapi to make requests

3.ClientAPI(String IPAddr,CountDownLatch latch,int threadGroupId, int threadId)

It has get() and post(), to make a request to aa certain address

Pakeages:

org.apache.http.client

apache.http.impl.client

java.util.concurrent.\*;

how it works:

**Firstly, the Client class initialize with an ip address, which is the url for the website we want to request. And it has get() method and post() method to create a request. I use the CloseableHttpClient class to create HttpClients.createDefault(); the use HttpGet get an instance of Get() request.and the client will excute this instance , which will returns the response. From the response, we can get the status code, response body. By the way, the Post() uses the same process.**

**When I want to create multi-thread program, I create a class called** ThreadGroupHandler **to create multipul thread groups with a countdownlatch to record when all groups are finished, every group has a class named** MainThread **and receive parameters, this class will create a countdownlatch, when one thread finishes the task, use llatch.countdown(), when the latch equals to 0, which means this group has done. The timestamp will record how many seconds it takes, and the total time, throughtout. Use Thread.sleep(delaySeconds \* 1000) to set a timeout when a thread finishes 2000 times requests. When all groups are finished, the first countdownlatch.awit() will pass, and output the total time and throughtput**

Client (Part 1) - A Plot show the throughput for the tests comparing the two servers. This should also include a screen shot of your output window with your wall time and throughput for each of the 6 tests.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Java rps | Go rps | Java time | Go time |
| 10,10,2 | 28 | 69 | 712872 ms | 288182 ms |
| 10,20,2 | 51 | 77 | 781610 ms | 516278 ms |
| 10,30,2 | 72 | 97 | 829687 ms | 617275 ms |

a plot comparing the throughout

As we can see the go server’s throughout is higher than java, and it costs less time.

Client (Part 2) - run the client as per Part 1, showing the output window for each run with the specified performance statistics listed at the end, and a plot comparing the two servers.

Go:

A screenshot of a computer

Description automatically generated

**mean response time: 251**

**median response time: 201**

**99% response time: 173**

**min response time: 159**

**max response time: 4566**

A screenshot of a computer

Description automatically generated

mean response time: 388

median response time: 219

99% response time: 175

min response time: 162

max response time: 7025

A screenshot of a computer

Description automatically generated

**mean response time: 534**

**median response time: 263**

**99% response time: 182**

**min response time: 163**

**max response time: 7883**

java:

A screenshot of a computer

Description automatically generated

**mean response time: 340**

**median response time: 210**

**99% response time: 173**

**min response time: 160**

**max response time: 2267**

A screenshot of a computer

Description automatically generated

**mean response time: 365**

**median response time: 233**

**99% response time: 178**

**min response time: 161**

**max response time: 3196**

A screenshot of a computer

Description automatically generated

**mean response time: 369**

**median response time: 235**

**99% response time: 177**

**min response time: 162**

**max response time: 2369**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Mean time** | **Median time** | **99% time** | **Min time** | **Max time** |
| **go** | **534** | **263** | **182** | **163** | **7883** |
| **java** | **369** | **235** | **177** | **162** | **2369** |

The plot of your throughput over time for a single test

A graph showing a graph

Description automatically generated

Java

A barcode with a graph

Description automatically generated

go