**Cloud Job Scheduler**

Kelly Flett (45350043)

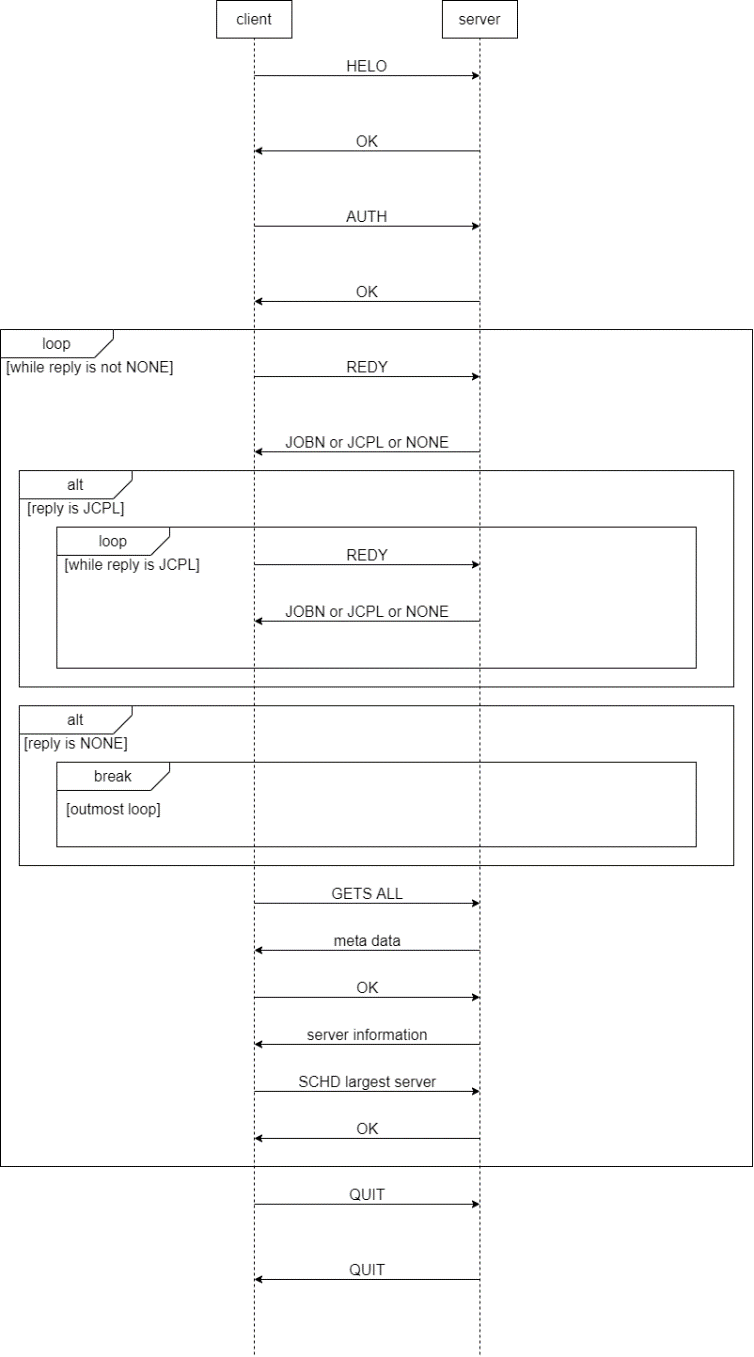
Scott Lin (45985995)

Jaime Sun (45662398)

**Introduction:**

The goal of this project is to develop a client-side simulator that acts as a simple job dispatcher for distributed systems. For the first stage we are tasked with implementing the ds-simulation protocol such that it successfully connects to the server-side simulator provided and will receive and schedule jobs. We are also required to create a simple job dispatcher that sends all jobs to the first entry of largest server type. The project is stored in a Github repository where all team members can access and make changes to the various components easily. This second stage requires each team member to individually implement at least one more scheduling algorithm. To verify that our code is correct it is tested under a set of conditions using the test1.sh file.

**System Overview:**

****

Detailed steps of the protocol

1. C(lient) sends HELO to S(erver).
2. S replies with OK.
3. C sends AUTH with authentication information to S.
4. S replies with OK.
5. C sends REDY.
6. S sends one of the following:

* JOBN: a normal job for scheduling for the first time.
* JCPL: information on the latest job completion.
* NONE: when there are no more jobs to schedule.

1. C takes one of the following actions upon receiving a message from S:

* Go to step 5 for JCPL.
* Go to step 16 for NONE.
* Go to step 8 for JOBN.

1. C indexes current job.
2. C sends GETS ALL.
3. S sends DATA (the preparation message that indicates how many bytes to expect) for GETS ALL.
4. C sends OK for DATA.
5. S sends information of all servers.
6. C indexes all servers and finds the one with the largest core count.
7. C sends SCHD for current job to largest server.
8. S replies with OK.
9. C sends QUIT to S.
10. S sends QUIT to C and exits.
11. C exits.

**Design:**

The client consists of 3 classes:

* The allToLargest class
* The Servers class
* The Jobs class

The Servers and Jobs classes contain attributes in the form of Java fields for simpler referencing in the allToLargest class. allToLargest is where the exchange between client and server happens as well as the largest server algorithm. Our design philosophy is structured and comprehensive. We first make a working draft of the program then work to make it more efficient and modular until we have optimized our code as much as possible. We decided to utilize Java as our coding language as we are the more comfortable and experienced with it than other languages.

An important consideration and constraint of the project is that it must be run in Ubuntu, which we are all running on virtual machines on our devices. Utilizing Github allows us to easily store and update our project while keeping records of each team members contributions.

As mentioned, the Servers class contains variables that are used to help the allToLargest class. It stores the server’s name, id, state, start time, cores, memory, and disk. To identify the biggest server the allToLargest class uses a for loop to compare the number of cores used in each server until it reaches the end of the list and is left with the result being the biggest server. The Jobs class stores all the job info required to schedule jobs, which is utilized in the allToLargest class. The allToLargest class itself communicates with the server and contains all the relevant methods and algorithms for job scheduling and the other requirements of the project.

**Implementation:**

Jamie was responsible for the majority of the allToLargest class and Scott made the Servers and Jobs classes. Kelly was primarily focused on the reporting and testing. The majority of the code was developed using Visual Studio Code with Java code.

The first stage of implementation was establishing communication between the ds-sim server and our client and printing the acknowledgements of the 3-way handshake as output. This involved methods for both sending and receiving ‘HELO’ and ‘AUTH’ messages to and from the server. The next step was to create methods to manage and schedule jobs to be sent to the largest server. After the initial handshake there is a while loop that runs until no more jobs are available. In the loop the client receives the current job from the server, and it is stored in an array with the 0-index housing the reply from the server. Job attributes are stored in the 1-5 indices using the Jobs class and are retrieved by the extractJobInfo method. Next the biggest server is identified using the populateServerList method. This gathers all the available servers sent from the server that may be used and the biggest server is discovered from those using the findBiggestServer method. This method loops through the available servers and compares core counts until it is left with the largest as a result. Server information is stored in an array list, with each node housing an array with indices 0-6 each containing an attribute of 1 server (each node is basically a server). Once the biggest server is identified, a biggest server message is created, and the job is scheduled on that server. The server will then either send information on the last job completed, a new job for scheduling or ‘NONE’ which indicates there are no more jobs left to be scheduled. When there are no more jobs to be scheduled the loop ends and the client initiates a quit request with the server.

**References:**

Git Repository: https://github.com/JSFun9888/job-scheduler-assignment