Project title

Group members:

Yitong Yan 44986629

Jiajun Chen 44509502

Weixing Yin 44432593

# Introduction

Our project aims to build a simulator that provides task scheduling for distributed systems to support the job assignment and optimize the task processing speed of distributed systems such as cloud services or data centers. We hope that the simulator, after the completion of this project, can have different algorithms to choose from to optimize the task processing in the task queue to different degrees, so as to meet the requirements of different services, such as fastest processing speed or more cost-effective task handling, etc.

However, due to the relatively complex content of the project, we divided the implementation of the simulation system and the implementation of the algorithm into two stages. In the stage 1, a “vanilla” version of the system will be established. Our goal is to enable normal communications between the emulator's client and server, and to find the most capable server as the simplest algorithm to handle all tasks.

# Overview

First, since our simulation project treats each job submitted by users as an event, discrete event simulation is a suitable method for the project. As mentioned before, the simulation system in our project is composed of two main components: client-side simulator and server-side simulator. Besides, there is an XML configuration file that contains the tasks (jobs) to be processed and servers’ information in the distributed system. The server simulator generates two independent files of task information and server information by reading the information in the XML file, and then sends the server information file to the client simulator for its understanding of the server information. After that, the client-side simulator as the program of scheduling task queue will make decisions about the arrangement of tasks and servers, and finally return the decisions to the server simulator for execution. The communication protocol in stage 1 between client and server simulator is shown in Figure 1.

Since we only implement the simple “the largest server” algorithm in this stage, the client only compares the server once as soon as it gets the “server.xml” file, and return the most capable server. Besides, it should be emphasized that this protocol may be changed in the next stage to fit the more complicated algorithms.

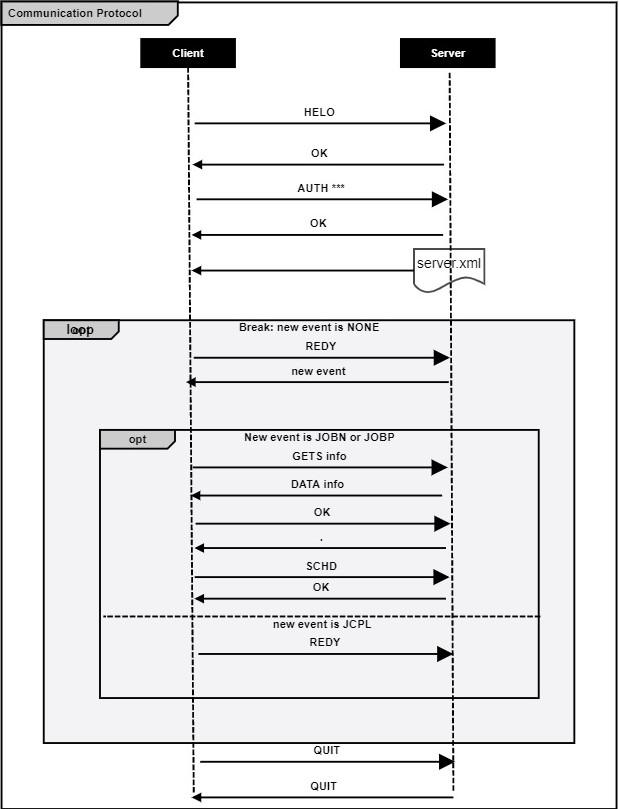


Figure 1. simple communication protocol in stage 1

# Design

In terms of designing the client-side simulator, since our programming ability is not enough to directly complete the entire project, we try to complete the goal step by step. It means that we will start with the simplest functions to implement the communication between client and server. After that, we shall try to integrate several important modules in stage 1 to form the required simulation program.

First of all, we regard the client as the class, think about what functions are needed, and put the communication process with the server in the main function. After making this decision, the code structure can be determined.

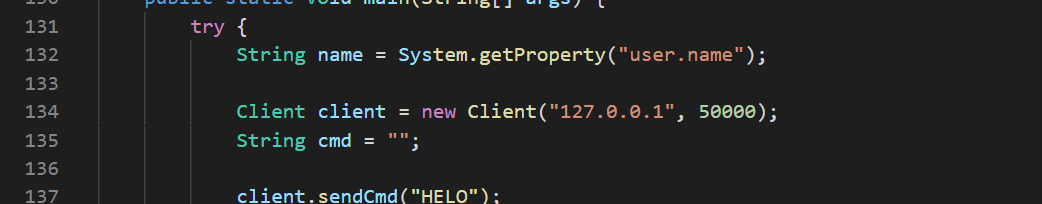
Then, after discussion, we believe that among these functions of the client, it is easier to implement the functions about establishing a connection with the server and receiving and sending messages between them. However, there are two important and complicated functions that need to be implemented. The first one is to read the received “server.xml” file and find the largest server type. The second one is to read the data sent by the server afterwards.

We follow the ideas above to implement the client simulator. The specific implementation process is described in the next section. However, we also reflected on some shortcomings in this design in order to write better code in stage 2. First, we only implemented the process of accepting and processing jobs under normal circumstances, but we have not considered clearly how to deal with the situation of receiving other events from the server. Secondly, the function of finding the largest server type still needs to be modified, because we now only compare a parameter to find it. However, if we want to try more complex algorithms, this function is not up to them. Therefore, we will improve these areas in stage 2 and strive to design a more perfect client-side simulator.

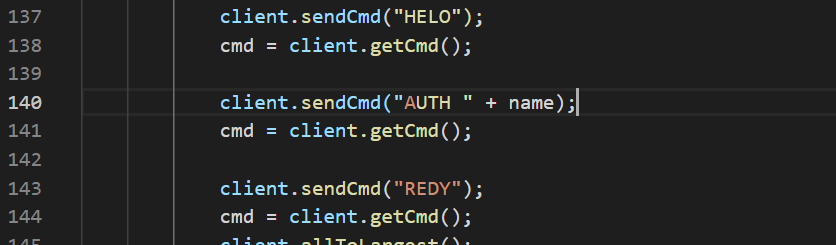
# Implementation

In Client.java:

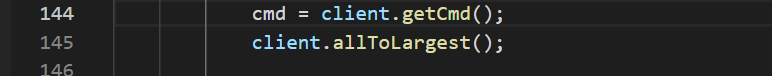
Line 132-134 Connect Server.



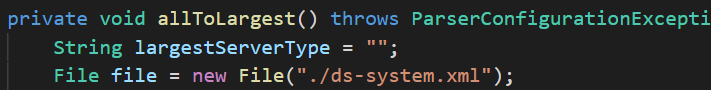
Line 137-144 Exchange command between server and client.



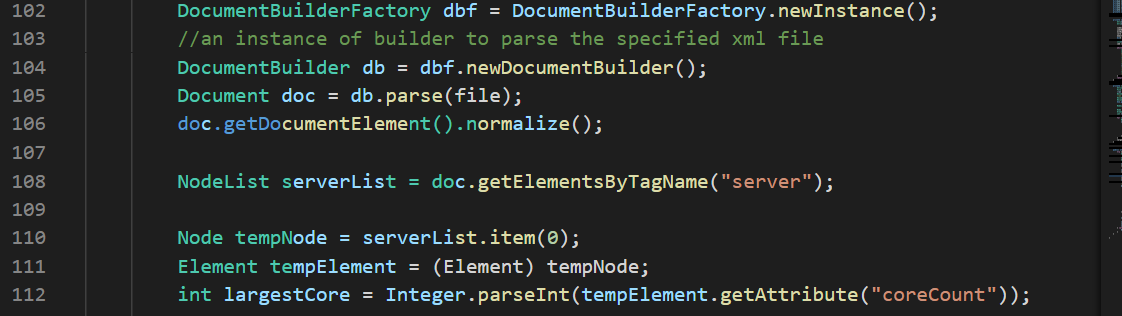
Line 145 Call allToLargest().



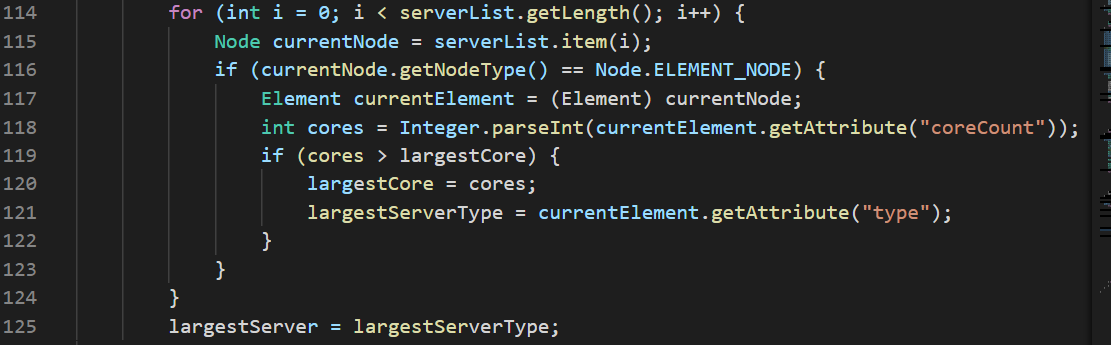
-> Line 100 Read the xml file



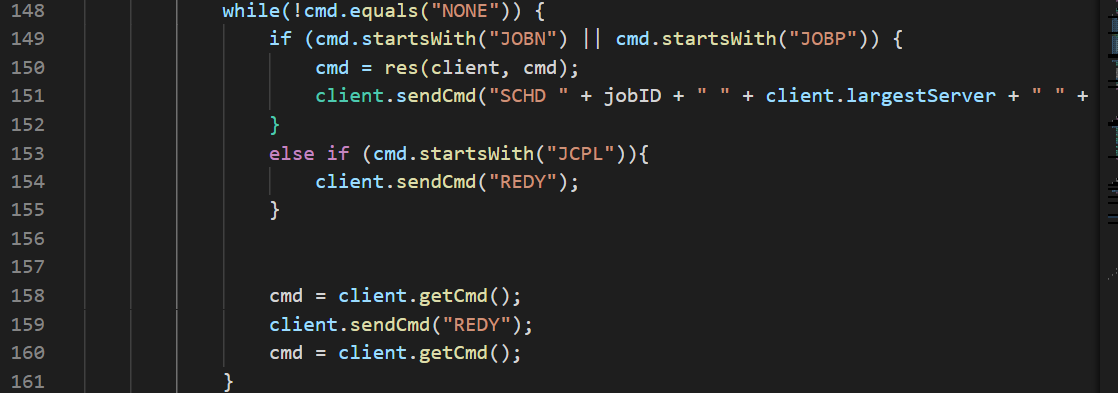
-> Line 102-112 Parse the xml file



-> Line 114-125 Find the largest server.



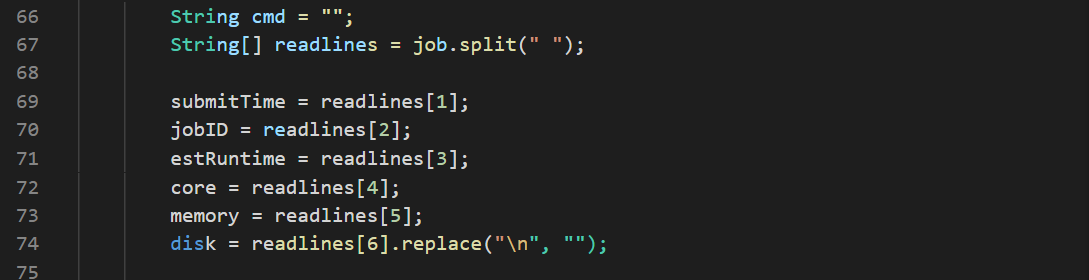
Line 148-161 Scedule job by giving the commands.



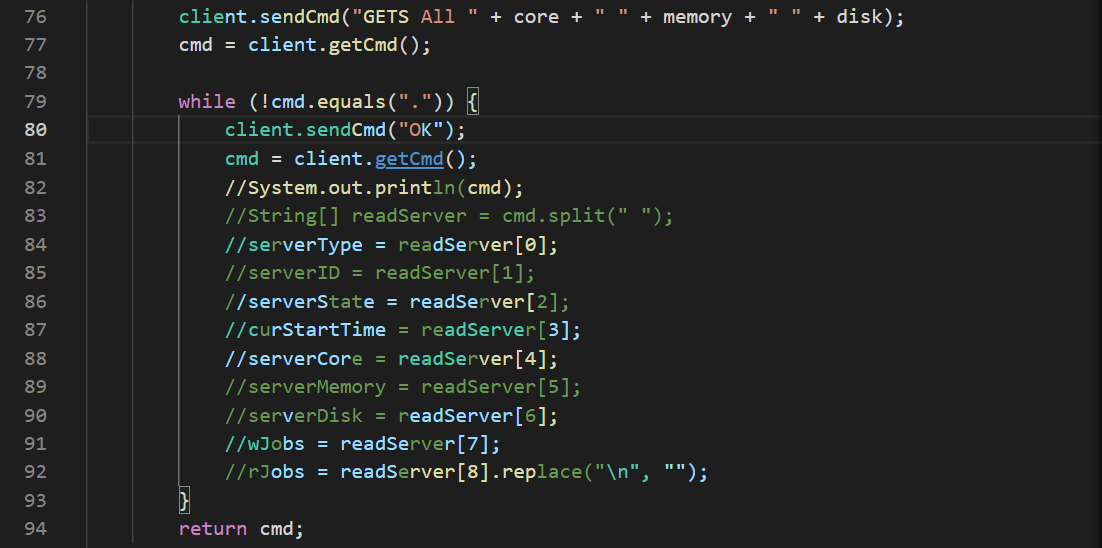
Line 150 Call res(client, cmd).



-> Line 66-74 Determine the attribute of job.



-> Line 76-94 RESC.



Line 164 Quit.

**GITHUB HISTORY**

**URL:** [**https://github.com/1084597302/Stage1**](https://github.com/1084597302/Stage1)