Stage 1 Report

Job Scheduler for Distributed Systems

Group 23 Members:

Nour Salama (45243336)
Chenlin Zhu (42055563)
Runde Jia (44434065)

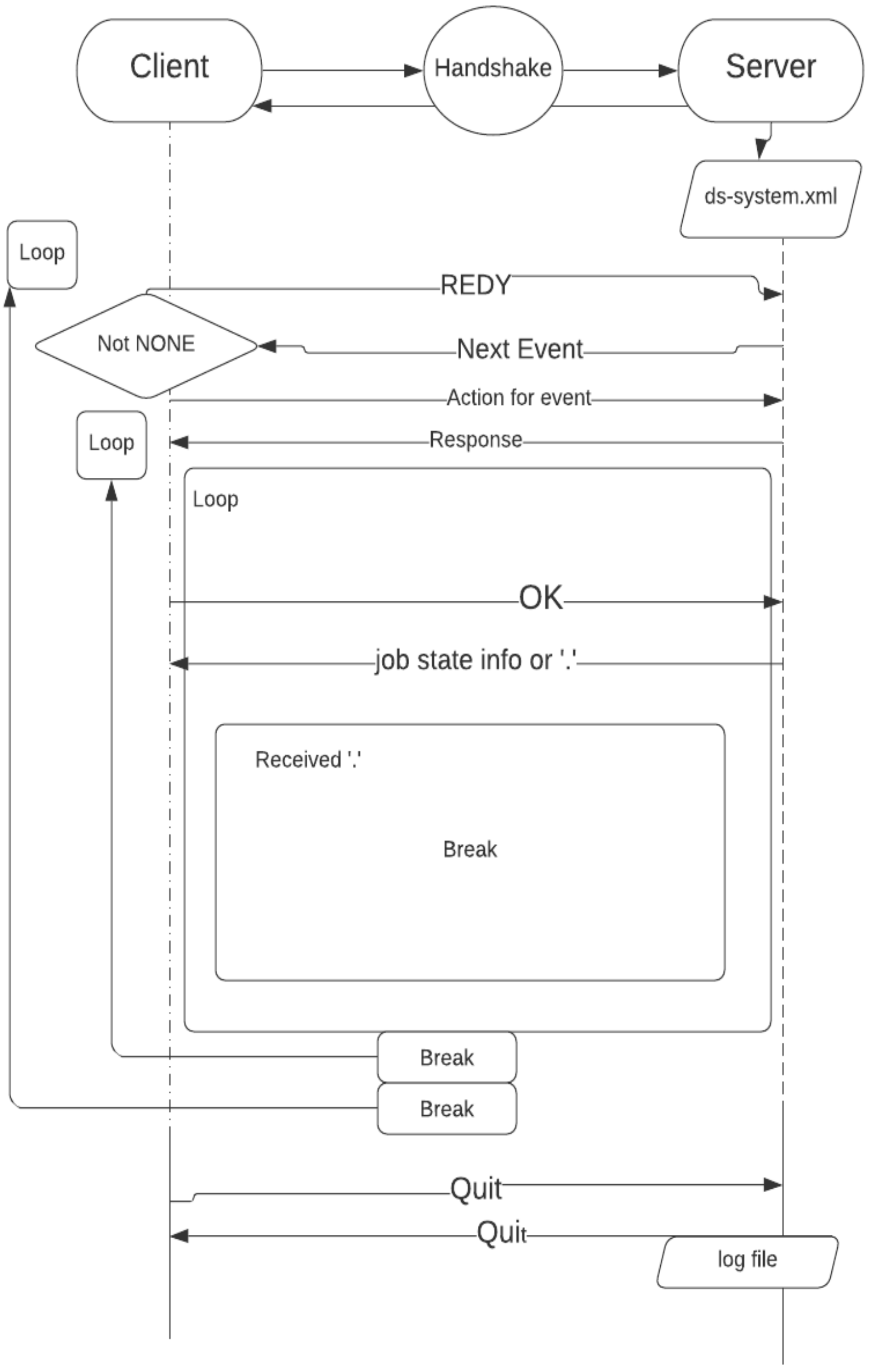
Introduction

The goal of the project is to create a client-side simulator that is able to manage and receive data from a server-side simulator and subsequently schedule those jobs. This is referred to as the ds-sim protocol. The allocation of the jobs depends on a range of resources such as response time, resource utilisation and capital/operational cost, and deadline and budget constraints [1]. Stage 1 is particularly aimed at designing and implementing the vanilla version of the client-side simulator by implementing the ds-sim protocol and a simple job dispatcher with an algorithm called 'allToLargest'. This dispatching procedure sends all the jobs to the first one of the largest servers based on the largest number of CPU cores. Ultimately, once the built simulator establishes the connection with the server-side simulator then it will accept the jobs that it receives and assign the jobs to the server type that is the largest in the available list of servers [2].

System overview

The system consists of both a client-side simulator and server-side simulator which are continuously communicating with each other to receive and schedule tasks. The client requires the IP address and port number of the server in order to establish a connection. The IP address is 127.0.0.1 and the port number is 50000. Once the server accepts the connection request it will send jobs to the client and the client will then be able to go through the information and hand over jobs. This is done by the client in a way to ensure optimal performance [3].

The diagram below illustrates the workflow of the system.



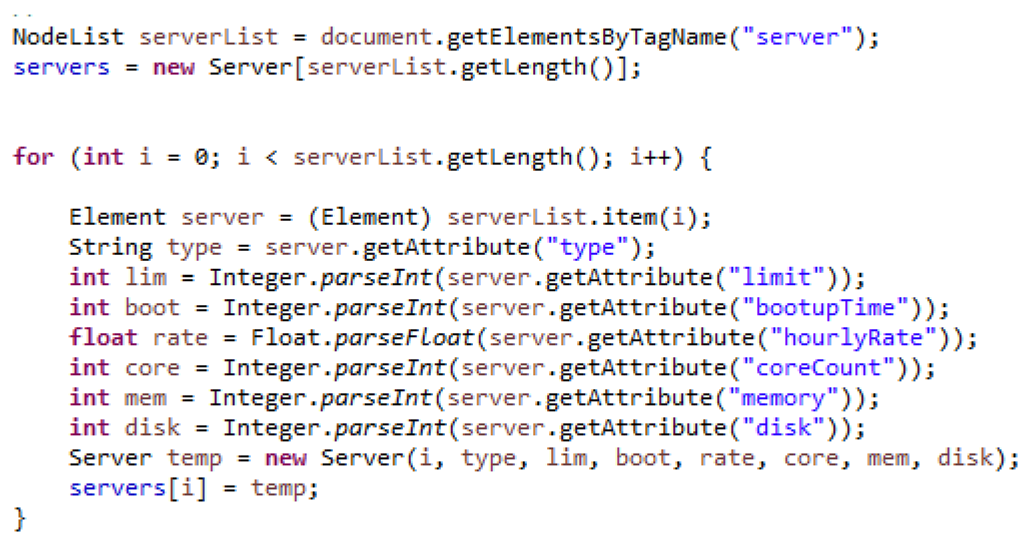
Design

With the development of computers, scheduling in computer processors has gained great development. This is the most common goal to minimize task completion time as a benchmark. The basic principles are the same as those for scheduling activities in the machine in production.

In a distributed system, multiprocessor scheduling considers multiple processors of the same capacity in parallel. In addition, data sources are thought to be centralized and connected through high-speed channels between processors, where activities can quickly exchange messages.Scheduling algorithms can expect input data to be exact, indeterminate, or random.When the input data is believed to be accurate, the algorithm is developed to work better in conditions where the input data about the job and the resource reflect the actual execution behavior.Input data is considered to be random when the expectation follows some generalized probability distribution.If the input data is assumed to be uncertain, the algorithm can use a variety of mechanisms to deal with various uncertainties, including application requirements and resource availability uncertainties.Schedulers typically assume that application communication and computing costs are known. Therefore, we believe that the application specification may introduce uncertainty regarding its communication and computing requirements as well as other possible application requirements such as memory, storage, and so on. The scheduler's target system is associated with the computing architecture of the system on which the scheduling job will run.Therefore, another factor that influences the scheduler design is the information about the target system for making decisions.The two main characteristics considered during scheduling are resource processing capacity and link bandwidth values of interconnect processing resources.

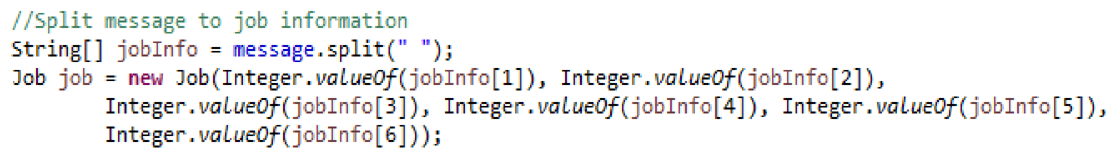
The client-side simulator should request for the server state information first after receiving server authentication information. Server will send all the server state information if there is no error, by recognizing all the servers’ coreCount we may be able to find the largest server here. In this way, we can hand over the job to the largest server, so that the work can be completed in an orderly and efficient manner.

Implementation



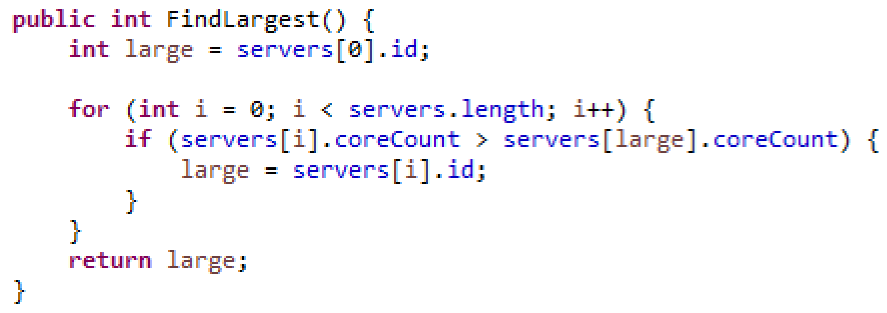
```
NodeList serverList = document.getElementsByTagName("server");
servers = new Server[serverList.getLength()];


for (int i = 0; i < serverList.getLength(); i++) {

    Element server = (Element) serverList.item(i);
    String type = server.getAttribute("type");
    int lim = Integer.parseInt(server.getAttribute("limit"));
    int boot = Integer.parseInt(server.getAttribute("bootupTime"));
    float rate = Float.parseFloat(server.getAttribute("hourlyRate"));
    int core = Integer.parseInt(server.getAttribute("coreCount"));
    int mem = Integer.parseInt(server.getAttribute("memory"));
    int disk = Integer.parseInt(server.getAttribute("disk"));
    Server temp = new Server(i, type, lim, boot, rate, core, mem, disk);
    servers[i] = temp;
}
```

Our group stored servers in a NodeList. NodeList can be traversed more easily than any other data structures. Variables like type, lim, boot are extracted from “ds-system.xml”. We created a class “Server” to represent received server information.



```
//Split message to job information
String[] jobInfo = message.split(" ");
Job job = new Job(Integer.valueOf(jobInfo[1]), Integer.valueOf(jobInfo[2]),
        Integer.valueOf(jobInfo[3]), Integer.valueOf(jobInfo[4]), Integer.valueOf(jobInfo[5]),
        Integer.valueOf(jobInfo[6]));
```

Used to split messages to the job information. Store every job attribute to the String array.



```
public int FindLargest() {
    int large = servers[0].id;

    for (int i = 0; i < servers.length; i++) {
        if (servers[i].coreCount > servers[large].coreCount) {
            large = servers[i].id;
        }
    }
    return large;
}
```

Algorithm for finding the first largest server. We compare server cores to the first largest server.

Variables in “Client” Class

Name	Accessibility	Type	Description
socket	private	socket	Used to connect server
BufferedReader	private	BufferedReader	Used to input stream

DataOutputStream	private	DataOutputStream	Used to output stream
serverList	private	ArrayList	Used to store servers
servers	private	Array	Used to store temporary server
large	private	int	First largest server ID
message	private	String	Messages sent/received by client
completed	private	Boolean	Flag to stand for action completed

Conclusion

By implementing the “vanilla” version of client-side simulator with a simple job dispatcher, we have understood the basic communication between client and server. Communication follows strict protocol. Building the client side has broadened our knowledge of distributed systems.

Reference List

[1] Y.C. Lee, Y.K. Kim and J. King, ‘ds-sim: A Distributed System Simulator User Guide’, Feb 2021.

[2] P. Lin & P. Liu, ‘Job Scheduling Techniques for Distributed Systems with Temporal Constraints’, *Advances in Grid and Pervasive Computing*, pp. 280-289, 2010.

[3] J. Leeuwen, ‘The Client/Server Model in Distributed Computing’ *CWI*, vol. 1, no. 3, pp. 193-218, 2010.

Bitbucket Repository Link:
<https://NourSalama1@bitbucket.org/NourSalama1/ds-jobscheduler.git>