Project report: Stage 1
COMP3100 Distributed Systems, S1, 2022
SID: 47478969, Name: Saad Alkhudaydi

Contents

1 Introduction

Distributed systems are a collection of various independent systems that appear to the user as a single coherent system. This type of layout is optimal for resource allocation, especially for job scheduling as it increases fault tolerance and the system's availability.[1]
At this stage of the project, the focus is to implement a simple client-side simulator that communicates with a simulated distributed server (ds-server), reads input, responds accordingly to the given input, and schedules jobs received from the server. The main purpose of this project is to demonstrate how distributed systems function in a realistic scenario. In this report, I aim to demonstrate the design and implementation of a simple job dispatcher that schedules jobs given from the ds-server. The client-side simulator will first initiate a connection to the ds-server and give authentication credentials to the server (handshake). Then the client-side simulator will signal to the ds-server that it's ready to begin scheduling jobs. The job scheduler implemented will work in a Largest-round-robin fashion. It will detect the largest server from a list provided by the ds-server and schedule all incoming jobs to that server

2 System overview

2.1 Ds-server

The ds-server is a distributed system simulator[2]. it is used in this project to simulate users, job submissions, job execution and servers ranging from small servers with low core count to large servers with high core count. The ds-server presents an option set of commands that the client-side simulator can utilise to initiate the handshake and schedule jobs such as

- HELO: Initiates a connection with ds-server
- AUTH: Authentication information of the client
- REDY: Signal to the server that the client is ready for a job submission
- GETS: Request information on servers
- SCHD: Initiate a scheduling decision to the ds-server

2.2 Handshake

The system consists of a client-side simulator and a server-side simulator. The client-side simulator communicates with the ds-server to receive server information and job submissions. It allocates them to the first largest server in the server-side simulator using Largest-round robin Algorithm(LRR). The client-side simulator first initiates the connection to the ds-server by using the command (HELO). Then the ds-server responds to the Client by sending (OK). After that the client sends authentication credentials to the ds-server using the command (AUTH) then the server responds by sending (OK). This process is known as a handshake. The workflow of the handshake can be seen in the figure below.

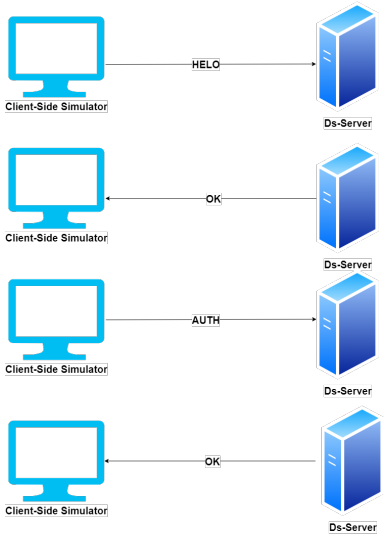

Figure 1: Handshake

2.3 Storing Server Data

After the client-side simulator establishes a connection to the ds-server (handshake), it signals to the server that it's ready to begin scheduling jobs. The server responds by either

- Sending a job submission to the client via (JOBN)
- Sending information on the latest job completion via (JCPL)

upon receiving a job submission, the client-side simulator requests current server information and stores them in a hash table format with the key being the server type, e.g.(xlarge,small...etc) and the value being an array list of the server information. The workflow can be seen in figure 2

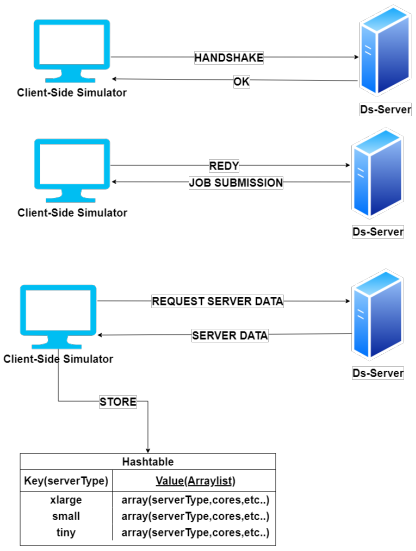

Figure 2: Storing Server Data

then the client-side simulator determines the largest server types by looking up the largest core count of each type.

2.4 Job Scheduler

The client-side simulator sends a scheduling decision to the ds-server and repeats this process until all jobs have successfully been scheduled. The ds-server then sends ”NONE”, indicating that no more jobs are available to schedule. The client responds by sending ”QUIT” to the ds-server and closing the connection.

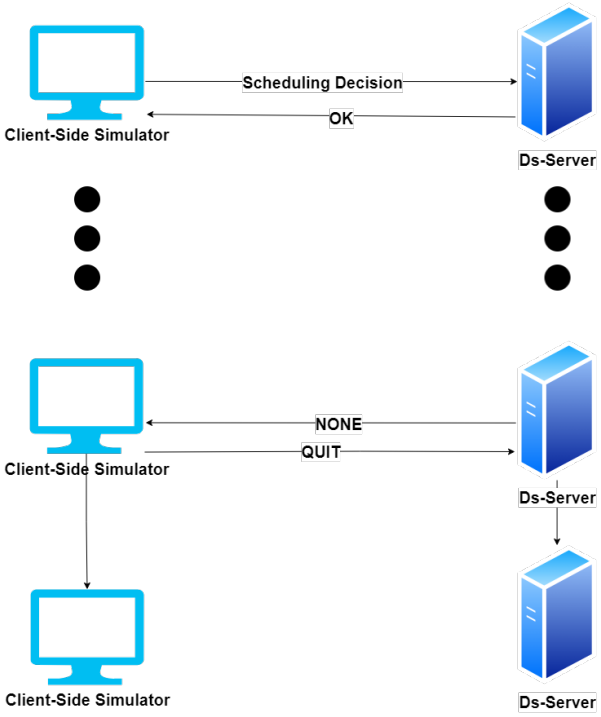

Figure 3: Scheduling

3 Design

3.1 Design philosophy

The design of the client-side simulator focuses on simplicity, modularity and efficiency. it implements the ds-server simulation protocols such as (GETS,SCHD) and a Largest-round robin algorithm (LRR) for scheduling jobs. The LRR algorithm was chosen for its simplicity and ease of implementation. While it may not be the most efficient in terms of resource utilisation, it provides a good starting point for implementing core functionalities to the Client-side simulator.

3.2 Considerations and constraints

The design considers various constraints, such as

- Network communication: The client-side simulator must establish a reliable connection with the ds-server using socket programming
- Responsiveness: The client-side simulator should handle all incoming messages and make informed scheduling decisions
- Scalability: The client-side simulator should be able to work with different server configurations
- Efficiency: The simulator should make optimal use of memory by implementing hash tables

3.3 Functionality of the Client-side simulator

The functionality of the Client-side simulator is to act as a job scheduler for the ds-server via these steps

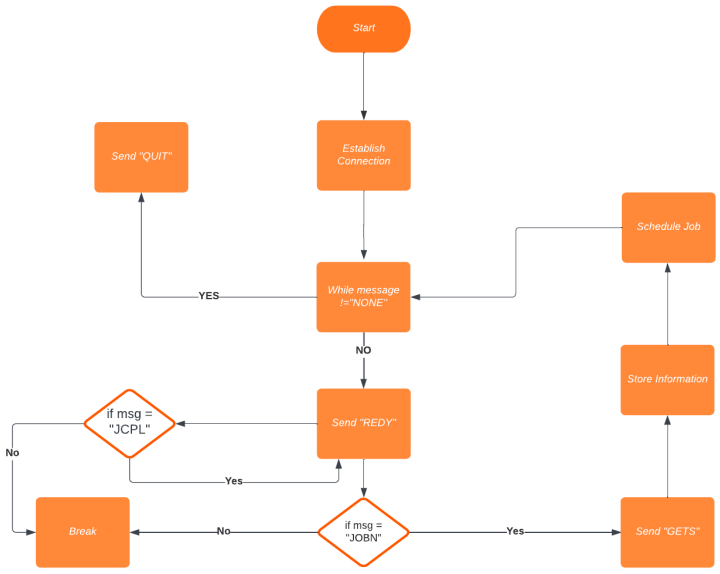

Figure 4: Functionality Flowchart

1. Establish a connection with ds-server (Handshake)
2. While the last message received from the server was not (NONE), signal to the ds-server that the client is ready to receive job submissions (REDY)
3. If the message received is (JOBN), then get server information (GETS)
4. If the message received is (JCPL), then go back to step 2
5. Schedule a job

4 Implementation

The Client-side simulator is implemented using java and uses various technologies, techniques, data structures and software libraries. The source code is divided into four classes, each responsible for a specific aspect of the simulator.

4.1 Client Class

The main component of the client-side simulator is the Client class[3]. The Client class is responsible for communications with the ds-server, other classes and the scheduling logic. This class is responsible for initialising and managing other classes. The main method is located in this class which starts the Client-side simulator. Furthermore, this method stores server information using a hash table, as illustrated in figure 2.

4.2 Job Class

The job class represents the jobs submitted by the ds-server and organises them into objects that the Client class can access. Each object specifies information on the job submitted, such as Job id, cores required , memory required, disk space required. This class provides setters and getters, which are tools for accessing and modifying the content of a job submission.

4.3 Schedule Class

The Schedule class manages the LRR scheduling algorithm by determining the first server in the list provided by the ds-server. This class also takes input from other classes such as Job, ServerInfo to calculate the smallest index of the largest server type.

4.4 ServerInfo Class

While the Client class stores the given input by the ds-server, the ServerInfo class represents each server information provided by the ds-server. More importantly, the ServerInfo class takes input given from the main class (Client.class) and organises them into objects that later can be accessed.

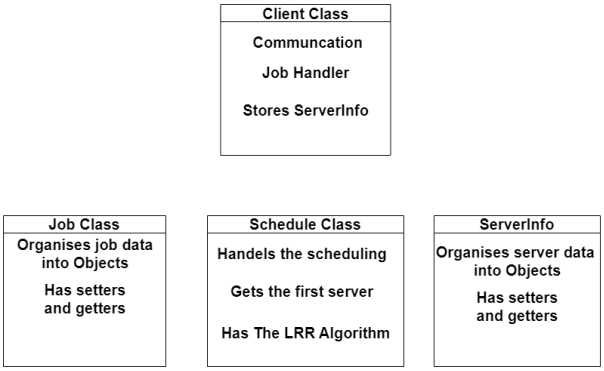

Figure 5: Classes

4.5 Technologies Used

The Client-side simulator was built in a Linux Ubuntu using a Virtual Machine (VMware). The base operating system that used VMware is Windows 10. The text editor used to create the source code is Visual Studio Code. Various libraries were used to create the code such as

- BufferedReader: Used to read input from the ds-server
- DataOutPutStream: Used to write commands to the ds-server such as (GETS,SCHD)
- IOExcepation: to handle exceptions that occurred when running the source code
- InputStreamReader: Used in conjunction with bufferedReader to write to the ds-server
- Socket: Used for socket programming and managing the connection
- UknownHostException: same as IOExcepation used for exception handling
- Arraylist: Used in conjunction with HashMap to store server information[4]
- HashMap: Used to store server information in a hashtable format where the key is the ServerType and the value is an Array list that contains the server information[5]. (see figure 2)

References

[1] M. Van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, pp. 967–1009, 2016.

[2] ds server, "GitHub Repository of ds-server used." https://github.com/distsys-MQ/ds-sim.

[3] S. Alkhudaydi, "GitHub Repository." https://github.com/Saadkhudaydi/COMP_3100_Project.

[4] W3school, "Arraylist." https://www.w3schools.com/java/java_arraylist.asp.

[5] W3school, "HashMap." https://www.w3schools.com/java/java_hashmap.asp.