Project Report: Stage 1
COMP3100 Distributed Systems, S1, 2022
SID: 45961735, Name: James Coulson

1 Introduction

This project is aiming to develop a job scheduling solution for distributed systems. Rather than using a live server, a server side simulator called enabled by ds-sim. ds-sim employs a server-client model and the server side simulation has already been provided therefore the goal is to develop a client that is a simple job dispatcher which satisfies two goals (for Stage 1):

- Connects to the server simulator, receives jobs and schedules received jobs.
- The dispatcher scheduler using Largest Round Robin(LRR) according to the server with the largest amount of CPU cores.

LRR assigns jobs to the largest servers descending according to the amount of CPU cores. For example, the first job gets assigned to the largest server, the second job gets assigned to the second largest server and so on until the last server. Once the job gets assigned to the smallest server, the next job gets assigned to the largest server and repeating the pattern until all jobs are assigned.
ds-sim aims to simulator a client-server model. There are two parts to the program, ds-server and ds-client. In basic terms, the ds-server generates a generic job that requires computer processing. This then gets sent to ds-client in order for the client make a decision on the scheduling of the job based on the design and algorithm that has been applied by this project before sending it back to the server. The job then gets run by ds-server according to the decisions made by ds-client including the allocation of servers and resources.
The goal of Stage 1 is to present a working vanilla client to communicate with the server simulation using the sample configurations provided by ds-sim. The simulation should not require extra configuration and should be able to run using a script provided at the time of testing.
Stage 2 of the project explores other scheduling algorithms and aims to benchmark each against some baseline algorithms which include First-Fit, Best-Fit and Worst-Fit. However with this said, this report pertains mainly to Stage 1.

2 System Overview

This section focuses on outlining ds-sim to gain an understanding how it works in order to deliver a working ds-client as per the guidelines.
As stated in the previous section, ds-sim uses a Client/Server model which is a widely adopted model within distributed systems. ds-sim has two main parts a ds-client and ds-server. The ds-server simulates both users submitting jobs to the servers and the servers themselves. The ds-client aspect of the simulation which is the part that is being designed acts as a scheduler which exists to assign jobs to each server according to the implementation that the developer chooses.
The system uses sockets which the ds-client communicates with ds-server. The ds-server reads a configuration file (For example, ds-config01.xml) which contains the server information including:

- Server Type
- Amount of servers with that type
- Bootup Time
- Hourly Rate (cost of running the server)
- Amount of Cores
- Memory
- Disk space

It also contains job generation information and the amount and workload of each job

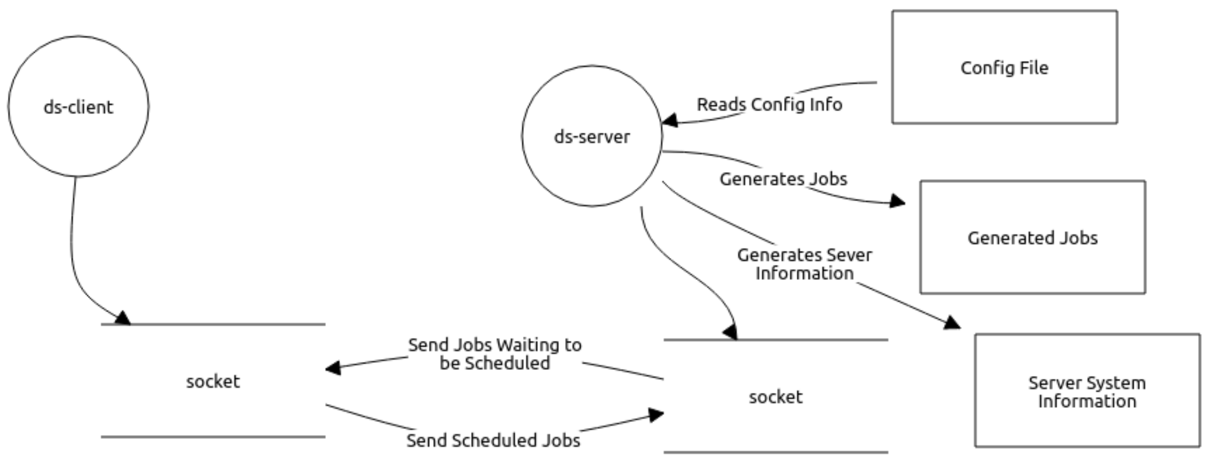

Figure 1: Figure 1. System Diagram

3 Design

The design of the system has a focus on being able to scale to different algorithms as what is going to be required in stage 2 of the assignment. It is important to have a solid building block with the first single algorithm (LRR) in this stage. Designing an effective system that can easily be changed should be the top priority. The most accepted way to do this in Java is to use classes to break up certain functions of the program into smaller chunks which then can be changed without effecting the entirety of the code.
The main functions of the client should be first establishing and connecting through a socket to the server. Then there must be a handshake to establish the connection. Once the handshake is successful, then the client will have to read the server configuration information and being able to read which is the largest server to allocate the jobs to. Once the client has done that, then it needs to allocate the jobs according to the LRR algorithm and be able to handle any other problems that might pop up. It then must quit the connection once all the jobs have been allocated.
The client can either read the server information from the ds-system.xml file that gets generated whenever the simulation is run or through the GETS command. Each of the apporaches has benefits. One benefit of using GETS is that there is no other libraries required in order to parse the information, it can be done within Java however using the system file will yield a cleaner output and it is easier for debugging purposes as the input can easily be imported and compared to ensure the correct parsing of information.
There must also be a implementation of the LRR algorithm in order to schedule the jobs. The requirements of LRR require again the client being able to read server information in order to find the size of each of the servers and be able to allocate the jobs accordingly. It is also required to handle any situation that might arise such as unavailable servers. These need to be handled and then the jobs should then be assigned.

4 Implmentation

The implementation of the LRR algorithm starts with the importing of several different libraries which are:

- java.io: for basic input and output streams
- java.net: for java networking features such as sockets
- java.io.File: to read the ds-system.xml file and parse it into usable information
- javax.xml.parsers.DocumentBuilder and DocumentBuilderFactory: specifically for use when working with xml files
- org.w3c.dom: for document reading features and NodeLists to use with the server list.

The program then begins to utilise a socket available in java.net. For the purposes of simulation, the IP address that gets used is the localhost loopback and the default ds-sim port of 50000. This implementation then uses a bufferedReader to read the messages that the server sends and uses DataOutputStream to send messages to the server.
The server then sends a handshake in the form of HELO. This is to ensure the connection is successful. Once the HELO message is sent the client then receives the servers reply before sending an AUTH message and

once again receiving a reply. The client then waits for the OK message from the server before reading the ds-system.xml in order to gain the server information. The OK message must be received before the file is read otherwise the file will not be available and the client will gain incorrect information. Once the file is open, the client reads and grabs any elements that are labeled servers before butting them in a node list. A loop is then used to iterate through the servers and find the maximum amount of cores in order to store it for later use. It also splits each of the server's information into a two-dimensional array in order to all information that's available in the xml file.
The client then uses a while loop to check if the server message is set to NONE. If it is NONE, then there are no jobs to allocate and the client proceeds to exit by issuing a QUIT command and then closing the network socket. If the message is not NONE, then there will be jobs to queue and the client send a REDY message to indicate it is ready to receive information and allocate jobs. Once REDY has been sent the server will send a JOBN message which contains the information for the current job. This then gets stored. If the message is JCPL instead of JOBN the client will handle it and stay in a ready state until the server is ready to issue a new job. After this check the client then checks again if the message is NONE before scheduling the job.
The allocation function works by first checking which servers are available for the upcoming jobs which then gets placed into an array. If there are no servers in the available state it then goes on to pass the job onto the largest server before returning. If there are, the information gets parsed to create a capable server list before iterating through each of them and finding which one has the largest core count and then assigning the job to that server.

References

Project repository avaiable from: `https://github.com/jamespandaz/DSJobScheduler`