COMP3100 Assignment - 1

Group 46 : Afroja Rowson Akter (45792895)
Hubert Hartan (45543003)
Jayakrithi Shivakumar (45630216)

Abstract

This report details an algorithm implemented for job scheduling in distributed systems. It includes a brief introduction of the project, an overview of ds-simulator followed by the design and implementation of the algorithm.

Introduction

Job scheduling is fundamental to carry out effective operations in Distributed Systems, and this project aims to create a job scheduler that is run with the use of an open source distributed systems simulator called the ds-sim.
The stage 1 of the project requires to design a client side script which acts as a job scheduler that assigns all the jobs to the first one of the largest servers. The server capacity is determined by the number of cores.

System Overview

This system adopts a client server model where the client acts as the job scheduler and the server simulates job execution and job submissions. The ds-server runs a user specified configuration file and pairs up with the client via a socket as the communication channel. Initial three-way handshake is established between the server and the client along with authentication. Upon successful handshake, ds-system.xml file which contains the information on the available resources to simulate, is generated and parsed in the client side script which stores all the resource information. This information is fed into the *All_To_Largest* algorithm which compares the core count of all the servers and returns the largest server.
Upon generating the largest server information, the client is ready to receive jobs from the server. All the jobs received are assigned to the largest server and a scheduled message is sent back to the server. Once all jobs are scheduled, a ‘NONE’ message is sent to the client and the connection is terminated with a ‘QUIT’ message.

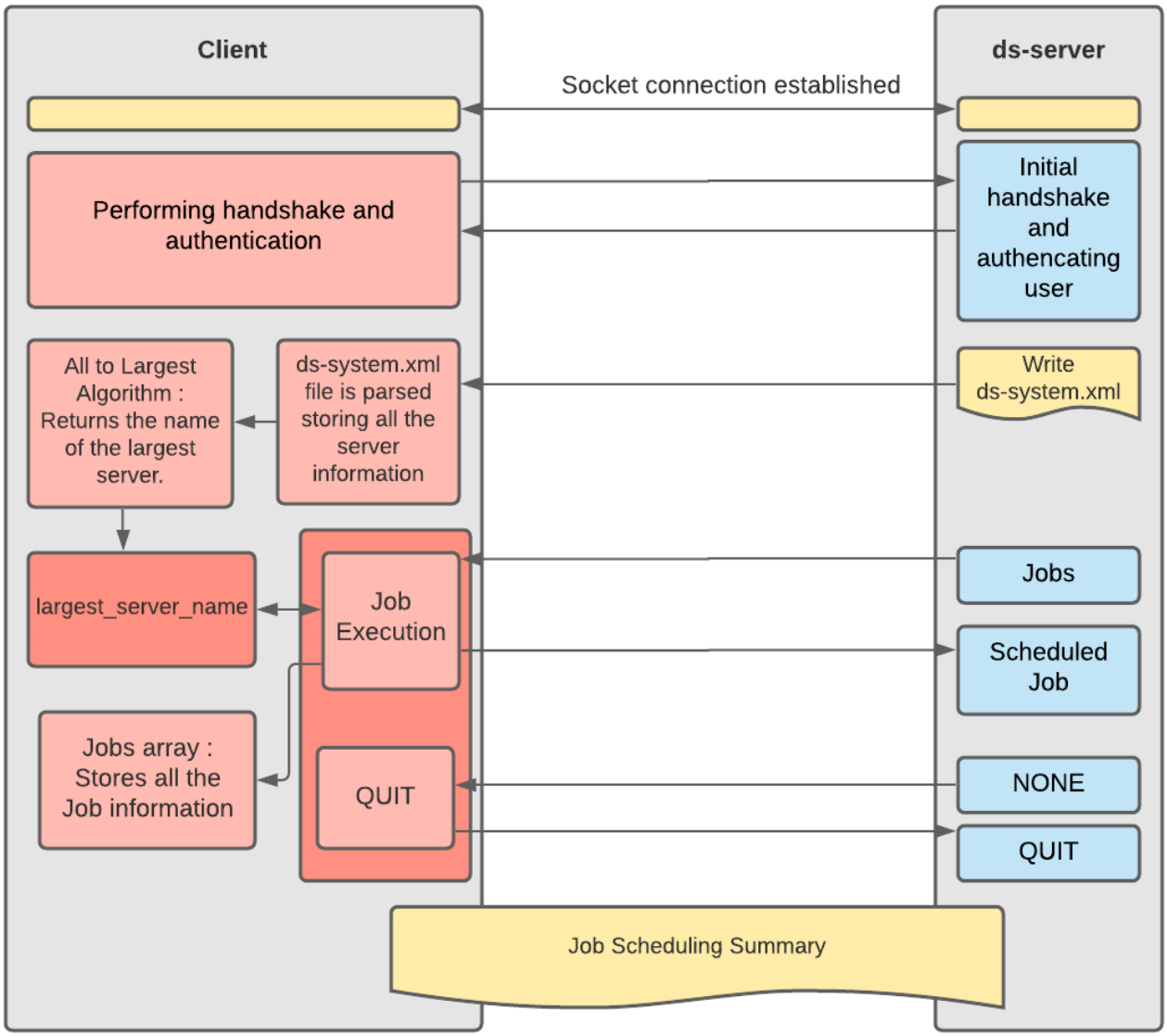

Figure 1 : System Overview Diagram - Depicts the workflow of the system

Design

The design is an important aspect of the program as a whole. The program is philosophically designed to prioritise functionality of one algorithm for job scheduling which is executed along with a user defined config setup provided in the form of sample_config.xml files.

Design Considerations:

There are some aspects that were taken into consideration when we were designing the program:

- The program will be improved further down the line and can include different scheduling algorithms.
- The transition from the current use of 2-Dimensional integer arrays to the use of a ServerInfo class along with more functionalities will be implemented.

Design Constraints:

These are some constraints which would limit the overall functionality of the program:

- The entirety of the program must be written in Java
- The project specifications specified that the program is designed to read the server information from .xml files, thereby constraining the program to only function by reading said file format.
- In order for the program to function properly it requires the communication protocol between it and the server to run smoothly. The server must receive the standard messages from the program (e.g. “HELO”, “REDY”, etc) and not just any random message.
- The program can only be run on a Ubuntu machine. This can be bypassed by installing a Ubuntu environment as a virtual machine.

Design Features :

- Effective memory management is in use, this allows the program to not consume memory unnecessarily.
- Use of 2D arrays to store server information enables a structured pattern to store data in tabular format therefore making it easy to access any attribute value.
- Regular messages are received on the client end, therefore making it easier to debug and track the progress of the program
- The program is expandable within the same class, making it easy to implement other algorithms and using them in the job scheduler

Implementation

The implementation is done in the Dsclient.java class.

Dsclient class

The Dsclient class is used to establish connections and simulate the job scheduling algorithm. The class is defined by a constructor which accepts arguments required for establishing connection, and includes various global variables, helper methods and a main method to simulate the process.

Libraries Used

- File
- DocumentBuilderFactory
- Document
- NodeList
- BufferedReader
- DataOutputStream
- Socket
- StringBuilder

Global Variables

Client messages (String messages sent from the client to the server) defined in the communication protocol are declared as global variables to facilitate reliable and flexible code practice, therefore preventing redundancy. Other variables which determine connection (**Socket instance**) and facilitate exchange of messages(**BufferedReader instance, DataOutputStream instance**[1]) between the two endpoints are also declared as global variables, therefore offering the flexibility to easily configure the communication channel (i.e, server name , port number).

Other Methods

Method Definition	Return Type	Description
performAuth()	boolean	● Performs authentication process (HELO-Send User Info- REDY-OK). ● method returns true on successful connection.
readXML()	int [ ] [ ]	● Read the ds-system.xml file received from the server. ● Details of the servers stored in a 2-Dimensional array.
getMsg()	String	● Uses an object of type **StringBuilder.** ● Reads and stores the message from the server in

		the object and returns it as a **String.**
messageToServer (String message)	void	• Sends a message to the server through **DataOutputStream** component.
scheduleJobs(int index, String[ ] types, String [ ] jobs)	String	• Assigns jobs to the largest server after grabbing it's position
getsInfor(String [ ] jobs, BufferedReader din)	String	• Modifies string to store the job for later use.
allToLargest(Int [ ] [ ] servers, String [ ] types)	void	• Core function for implementing the **All_To_Largest Algorithm.** • Finds the largest server among all servers • Comparison criteria - number of cores • Hence, the client can successfully assign jobs to the largest server based on the position of the largest server obtained.
main(String [ ] args)	void	• Environment is set up using a constructor. • Connection is established and authentication successful via **performAuth().** • Server info obtained. • **All_To_Largest** Algorithm is implemented, hence **allToLargest** function is called to obtain the largest function. • Due to the variety of jobs, **switch** statements are used, hence ensuring the capability to handle and react to different job types. • Once read "NONE", the client must STOP.

References

Java Socket Library Documentation:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/net/Socket.html

Java BufferedReader Library Documentation:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/BufferedReader.html

GitHub Repository Link:
https://github.com/jayakrithi/COMP3100-Group46