Stage 1: Client-Server Simulator and Simple Job Scheduler

Project Title

Basic job scheduler for distributed systems.

Group Members (Group 54)

Chris Nance: 43254748

Tapasvi Muthavarapu: 44912145

Introduction

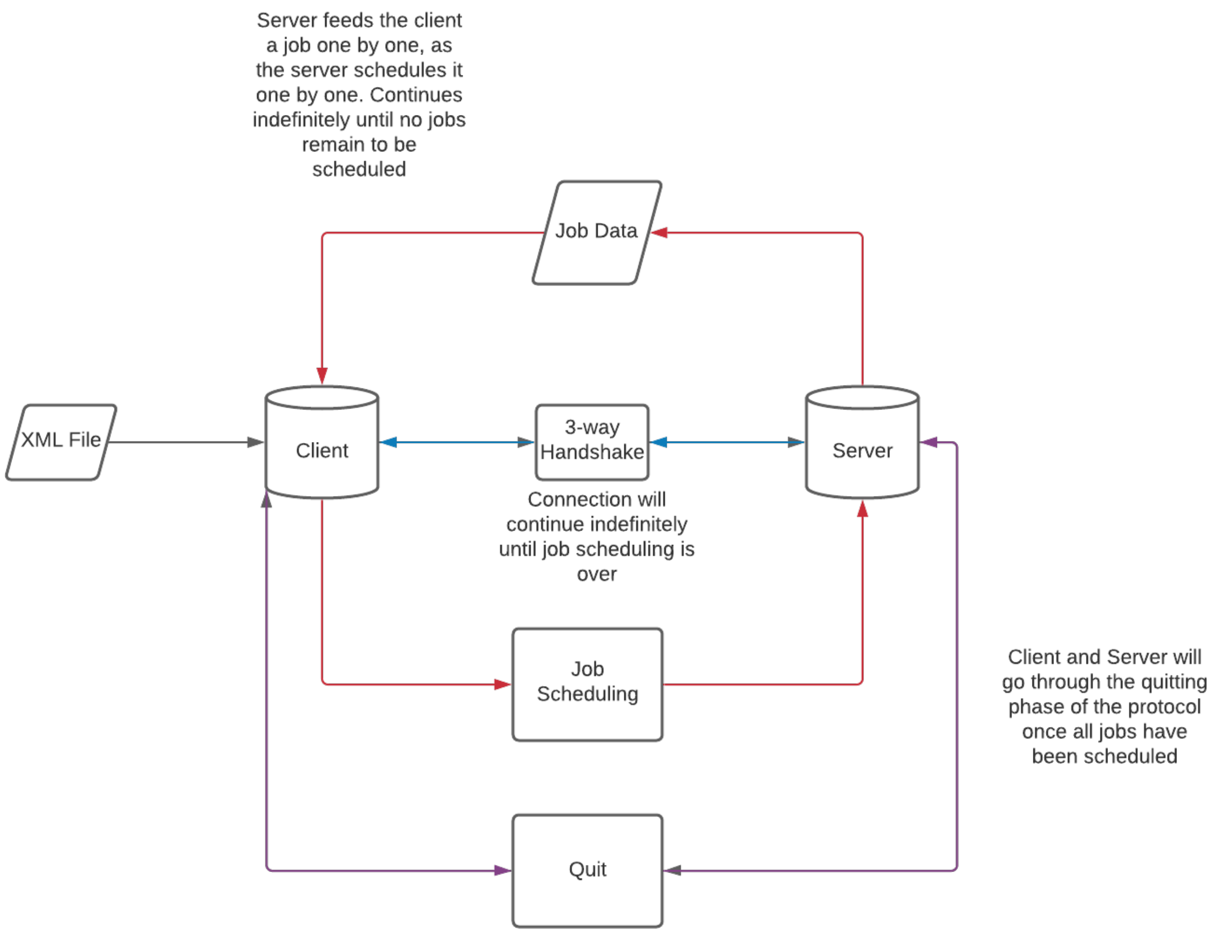
This project deals with a distributed systems simulator, and its behaviour with respect to a client-side end user with the purpose of scheduling jobs to specific servers defined by the simulator. The main goal of this project was to develop the client side end with the ability to communicate, respond to and take action in response to commands from the server side simulation, and thus take on the job scheduling role as defined above.

The program developed in this project closely follows the ds-sim protocol as outlined in the ds-sim user guide, with minor simplifications and variations to account for the scope defined in the assignment specs. Although the client appropriately schedules the jobs to the largest server provided by the server (as defined by the number of CPU cores), this function is not provided by a specific algorithm *allToLargest*, an oversight which may be a violation of the assignment specifications. The functionality however remains the same.

System Overview

The client-server connection is established initially via a handshake, with the IP and port set to 127.0.0.1 and 50000, respectively, which is what the server is attuned to listen for (50000 being the default). Communication between the client and the server begins in earnest after this process. After the handshake the client parses the relevant XML file that contains the data on the potential servers that can be scheduled to, and stores this data for its own use. This accounts for steps 1 to 5 of the ds-sim protocol.

Steps 6 to 11 of the protocol are carried out within a loop that the client goes through, as it will indefinitely listen for jobs to schedule from the server until it receives the appropriate command signalling that there are no more jobs available to be scheduled. Steps 12 to 14 are then carried out, which is just the process of having both the server and client confirm to each other that they are ending communication and exiting the simulation. Below is a flowchart giving a basic outline of how the system works:


Server feeds the client a job one by one, as the server schedules it one by one. Continues indefinitely until no jobs remain to be scheduled
Job Data
XML File
Client
3-way Handshake
Server
Connection will continue indefinitely until job scheduling is over
Job Scheduling
Client and Server will go through the quitting phase of the protocol once all jobs have been scheduled
Quit

Design: Design Philosophy - Constraints and Considerations

The approach to this project was an incremental one. Many of the necessary functions were quite simple on paper (such as sending and receiving messages to and from the server, as well as the quit function) and were addressed first and updated constantly over time so as to work with the more complicated methods that were later implemented. Each method was tested individually using a separate Server program that was created to facilitate testing. The job scheduling (Steps 6 – 10 of the protocol) were the most daunting and thus most time was spent ironing that out so that it would work. It was always the intention to address that aspect of the project last as having the other basic general functionalities working would help provide context and perspective to the project as a whole, making addressing the hardest individual aspect of the project significantly easier (which it did).

The first constraint and consideration to take note of is how the server side was written in C. This didn't provide to many issues with the exception of noting C's handling of newline characters. This was fixed by ensuring that every message (aka string) that was sent to the server had a newline appended to the end of it ("\n").

The second constraint was an often encountered *OutOfMemoryError* exception that seemed to arise after having run the test files or just simply the ds-server with one of the configuration files enough times. Previous iterations of the client program I had saved and marked as safe (ones that previously got through the protocol even if it didn't schedule properly) would end up encountering this error once a newer version had encountered it, so it seemed to be an issue with the running system or the laptop. This issue was fixed (or band-aided) by using a different laptop that had Ubuntu as its running system, and possibly by increasing the available java heap size allocated to the program through the terminal when running the program.

Design: Functionalities

Our client is defined within the *SocketClient.java* class which encompasses the entire program. The constructor *SocketClient* initialises its input parameters as well as various class variables; *Socket*, *DataOutputStream* and *DataInputStream*, all of which are used constantly throughout the entire running of the program.

In our main method, we can divide the code up in to three parts (in theory each could have been abstracted out of main into its own method): The handshake protocol, which follows our initialization of our *SocketClient* object. Here our *Send* and *Receive* methods are first called. The former takes in a String as a parameter and send it to the server in byte form. The latter receives data in byte form, which is then converted to a string and returned as a String output.

The client then proceeds to extract the data from the provided XML file that are relevant to the available servers that can be scheduled to, and creates a Server object for each available server and stores this inside of an *ArrayList allServers*. We only make use of two attributes from our server object, but I found it simpler and more understandable to have our Server object to include every attribute found inside of the XML file. The *readXML* method also stores the index of the location of the largest server found in *ArrayList allServers* and stores this in an Int class variable *largestServer* to be accessed later.

The remaining code does not make use of any new methods or objects, but it involves a moderate level of string manipulation, which is needed to get through the remaining steps of the ds-sim protocol. The code inside the main method was carefully structures to look as if it was closely following the ds-sim protocol.

<u>**Implementation**</u>*

In general, the supplementary functions that aided the protocol were abstracted out of main and defined inside of their own methods. This provided an easy way to view the protocol, written in code, in a more simple manner. The data structures also worked better as class variables, the ones which were used included *ArrayList* and a *Server* object defined by an inner class. *NodeList* was used inside the *readXML* method that helped parse the XML file.

The most relevant software libraries that were used are as follows:

- *import java.io.DataInputStream;*
- *import java.io.DataOutputStream;*
- *import java.net.Socket;*
- *import java.io.File;*
- *import javax.xml.parsers.DocumentBuilder;*
- *import javax.xml.parsers.DocumentBuilderFactory;*
- *import org.w3c.dom.Document;*
- *import org.w3c.dom.Element;*
- *import org.w3c.dom.NodeList;*

Components/Functions:

- *public SocketClient(String IP, int port)*
    - This is the constructor for SocketClient.java, with the IP and port number servers as parameters and passed through in the main method.
- *public void Send(String s)*
    -
- *public void Send(String s)*
    - Used to send out messages to the server.
- *public String receive()*
    - Used to receive messages sent to the client from the server, with the message returned in the form of a String.
    -

- *public void readXML(String XMLFile)*
  - Parses the XML file identified in the parameter and populated the various class variables and objects with the data.
- *public static void main(String args[])*
  - Contains all the work involving the protocol, which involves a mix of calling the methods mentioned above and its own work facilitating the progress of the protocol.

References

- Flowchart was created using Lucid: https://lucid.app/
- ds-sim user guide.
- Just in case: Almost the entire code under the readXML() method was derived from the example code provided here: https://www.javatpoint.com/how-to-read-xml-file-in-java
- https://github.com/Abstractvice/COMP3100_Project
  - Tried changing the username to my actual name but apparently it seems I'm unable to without another account!