**---------------------------------------------------------------------------**

**Simple Job Dispatcher for Distributed Systems**

**Group 27**

**---------------------------------------------------------------------------**

**Members**

|  |  |  |
| --- | --- | --- |
| Name | Student ID | GitHub Name |
| Prithivi Sharma | 45705704 | PrithiviSharma11012001 |
| Benjamin Ahamed | 45934029 | BenAhamed |
| Edoardo Busano | 45757100 | edoardobusano (and edobusy) |

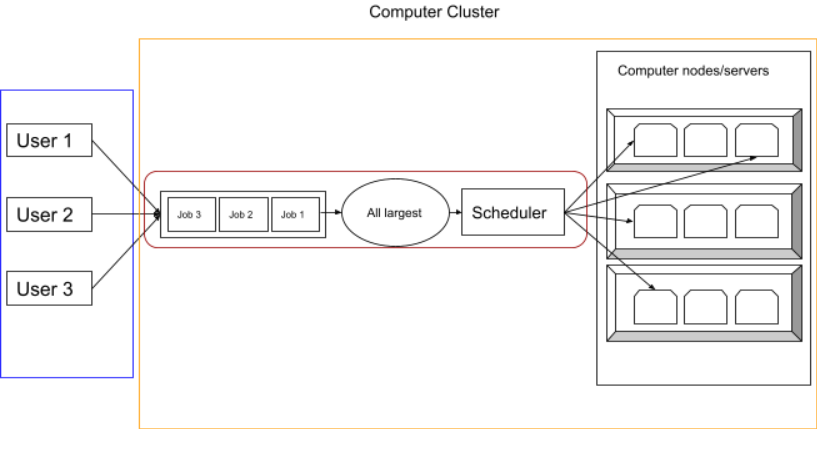
**Introduction**

This project is centred around developing a scheduler for jobs for distributed systems. The first stage of this project focuses on allocating tasks on a server by clients. This is firstly done through a client-server connection, allowing the client to schedule jobs to the server simulator of the distributed system. Through this connection, the main task that is done in this stage is to find the largest server that is present within the distributed systems configuration file and sending all jobs from the client to that server. This is done with the use of a function called “allToLargest” created on the client-side where the largest server within the configuration file is determined. Subsequently, the client-side simulator uses this information for transferring all jobs of the client to that server.

All in all, the main goal of this stage is to be able to create a stable connection between the client-side simulator and the server-side simulator and be able to schedule jobs to the largest server within the configuration file that the client-side simulator is connected to.

**System Overview**

The main system used for this project is classified as Ds-sim. It is an open-source distributed system simulator which consists of two elements used to connect a client and server through simulations, both known as ds-client and ds-server. The main purpose of ds-client within this project is to schedule jobs for the nodes which are within ds-server. Furthermore, the ds-client’s responsibilities also include finding the largest server within the configuration file which ds-client is connected to and schedule all jobs to that server. The ds-server contains virtual servers whose specifications are saved in an XML file. Ds-server’s main responsibility is to ensure to run the scheduled job from the users on the virtual servers it has which are fabricated and are a part of a simulation. The ds-server is also responsible for sending back any error messages that occur and to acknowledge a completed job.

**System Diagram**

**Design**

The design process for the client-side simulator took a long time, as we understood the ds-simulation better and better during the practicals.

The only constraint clearly stated concerned the coding language, which needed to be Java.

The client functionalities were designed following the ds-sim user guide. After understanding the ds-sim workflow, the next step was turning the client-side communication into pseudo-code, that would later be replaced with actual code. The client class has a constructor, where the code for connecting server and client is placed. The other important method for the client is the start method, in which all the messages are sent and received. Instead of having a large block of code within the start method, each action is turned into a method with an appropriate name, resulting in an overall more polished work.

The data received from the server-side simulator about the servers available for job scheduling is saved and stored in custom Server objects, with a very simple structure, as for Stage 1 only the number of cores and the server type are necessary for scheduling jobs.

The main algorithm allToLargest is packed into one method containing the loop that schedules all the jobs provided following the logic explained in the assignment document.

**Implementation**

Our software and simulation are built on the technologies surrounding sockets. Socket programming allows the connection between two nodes on a network be that the same internal network, local network or even across the internet. It works by using what are called sockets and IP. sockets are a virtual port on a computer from which it can send and receive input. Its address is determined by its IP and socket number. This means another node with this information can communicate with the other node using this address information.

XML (Extensible Markup Language) is a markup language created by the World Wide Web Consortium and is used to encode information that is easy for both a human and computer to understand. This technology is essential for our program and simulation to function as we access XML file that contains information on available servers.

Software used to run the simulation and test the program we created included:

* Virtual Box : Is a piece of software created by the company oracle that allows a user to create a virtual computer. A virtual computer effectively  works as any other computer but allows the use of different OSs.
* Ubuntu desktop : ubuntu is an operating system built by the company Linux. The simulation was run on this operating system as it is much more flexible than windows and less resource intensive.
* Java: Java is an object oriented programming language developed by Oracle. It is what our client is written in and is used to to compile and run our application.

As previously mentioned java was used in the creation of our program, which involved the use of many different libraries/packages within the java ecosystem. Libraries and packages used include:

* Java.net.\*: is a package that contains multiple libraries all useful for establishing connections between applications. This enables our program to establish a connection with the ds - server and enables the sending of information back and forth.
* Java.io: Is the package that actually enables the sending of data back and forth between the client and server using the connection created by Java.net.\* objects. Information sent is encoded as bytes not strings.
* The java.xml package: This library contains methods and objects that allow you to parse through xml files. This is what enables our program to establish which server is the largest by reading the xml file

The data structure that is used in our program revolves mainly around Arrays as information from the XML file is parsed into an Array to allow for easy access and data manipulation. The Array named servers stores useful information of each server with the server at array[0] being the first server in the XML file. The Array server is then parsed through using the method findLargestServer() which returns the server ID of the largest server.

The method used to return the largest server is a relatively simple technique which involves extracting the cpu core count of each server contained within the XML file.

Each function is commented with a description of the functions function.

List of functions:

Write(), read() and find LargestServer() – Ben

readFile(), quit(), server class – Prithivi

allToLargest(), constructor of class – Edoardo

References

The GitHub repository URL that contains our code

* https://github.com/edoardobusano/COMP3100\_Stage1