# Distributed Cloud Scheduler

Anthony Allan (45634963)
Chaz Lambrechtsen (45426317)
Michael Thygesen (45207275)

## Introduction

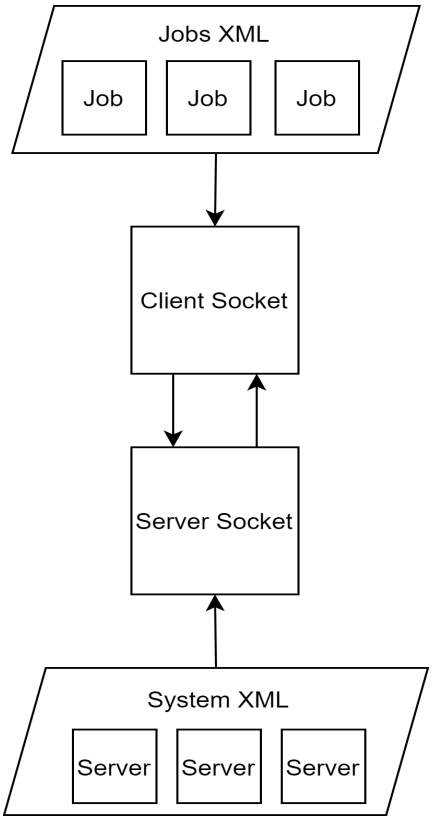
Stage 1 of this project is to design and develop a client-side simulator for basic scheduling and dispatching of jobs. The jobs are provided to the client via the ds-sim protocol in plain text from the server-side simulator. The client will then dispatch the jobs to the largest available server.

This client will follow the structure and communication format as shown in ds-sim user guide [2] section 6 “ds-sim communication protocol”. The client will behave similarly to the given pre-compiled ds-sim-client [3]. Actual implementation can be found by following reference [1] - a link to the GitHub repository for this project.

## System overview

This system will utilise both a client and server, the client will be responsible for scheduling jobs received by communicating with the server. The job scheduler should schedule jobs to multiple simulated servers provided in the form of an XML file by the server.

The diagram on the right shows a high level data flow diagram of the server and client communicating to each other and getting input from System xml file and Jobs xml file respectively.



# Design

**Design Philosophy**

As a group we had no specific design methodology in approaching this project. We completed the project by completing issues & tasks as they appeared. That being said, we made use of the fact that Java is an Object Oriented Programming Language (OOP). Essentially, “objects” contain data (fields) and code (methods). We used this paradigm for the storing, reading, and writing of available Servers (type, count, cores etc). This will be covered more thoroughly later in the document when we discuss “*Server.java*”.

**Constraints include:**

- Time to complete the project, funding available, forced to use Ubuntu via VM.
- Must use ds-sim protocol for communication
- Must use XML files as input

**Considerations include:**

- Each group member has a different level of proficiency & understanding in regards to socket programming, networking, and Ubuntu.
- Workloads of each group member should be considered, meetings should be organised at times that are convenient for all members.

**Component Functionality:**

There are two major components of our system - Client & Server. Our Client communicates with the ds-server, and uses the Server class definition to build a collection (linkedList) of Server objects. The methods in our Client are discussed in a later section of this document. The methods in our Server class are just simple setters/getters, including a default constructor to instantiate an instance of said class.

# Implementation

The libraries we used for this project are:

java.net.*
java.util: arrayList & concurrent.TimeUnit
java.xml.parsers: DocumentBuilder & DocumentBuilderFactory
org.w3c.dom: Document, Element & NodeList

The java.net* package is a ServerSocket class, it is used by our Client to obtain a port and listen to requests. Essentially this is what allows our Client to communicate with the DS-Server.

The arrayList package is used for storing the data about the servers. We parse the information about each server from the XML file to a Server Object (a class we have defined) which is then held in an arrayList of Server Objects.

The concurrent.TimeUnit package is used as a means to wait a certain period of time before trying to connect to DS-Server in the event there are connection difficulties.

The xml.parsers packages are what allow us to open and read the information about the servers from the XML files.

The org.w3c.dom packages are used for file input.

As mentioned earlier, the main data structure being used is a LinkedList. This is a linear data structure that is mutable and allows for the addition and deletion of elements with relative ease. This ArrayList holds a collection of Server Objects. We use this ArrayList of Server Objects to determine which Server is the largest by iterating through the list and comparing the number of cores.

The java class "*Server.java*" was developed & defined by our group. The class contains fields for: the type of server (String), the limit/number of available servers (int), boot-time (int), hourly Rate (float), core-count (int), memory (int), and disk-size (int). We also getters that return this information.

The components & functions of our project are as follows:

```
public Client(String address, int port) throws IOException {}
```

This is the constructor for Client.java, its input parameters are the address and port.

```
private void start (){}
```

This function is called once we have established a connection with the Server.

```
private String toLargest(String job, Server s){}
```

The string this method returns is a command that sends the request to the largest server. The input parameters are a string (job) from the ds-server, and a Server (s) the largest server pulled from ds-system.xml.

```
private int findLargest(ArrayList<Server> s){}
```

This function returns the index of the Server (from our Server ArrayList) with the highest core count.

```
private void sendMessage (String outStr) {}
```

This function takes a string and sends it as a message to ds-server.

```
private String readMessage () {}
```

This function reads incoming messages from ds-server, and returns it as a string.

```
private static void connect(String address, int port) {}
```

This function establishes a connection to ds-server.

```
public static void main(String args[]) throws IOException {}
```

The java main method is the entry point of any java program.

```
public static ArrayList<Server> readXML(String fileName){}
```

This function returns an arrayList of Server Objects through reading an XML file (the input parameter). The XML file ds-system.xml, is produced by ds-server. It's contents vary depending on which config file is used by ds-server.

# References

[1] https://github.com/CazDev/Distributed-Cloud-Scheduler
[2] https://github.com/distsys-MQ/ds-sim/blob/master/docs/ (ds-sim user guide)
[3] https://github.com/distsys-MQ/ds-sim/blob/master/src/pre-compiled/ds-client