Stage 2: Baseline algorithms – Design Document

1. **Project Title: Autonomous job scheduler**

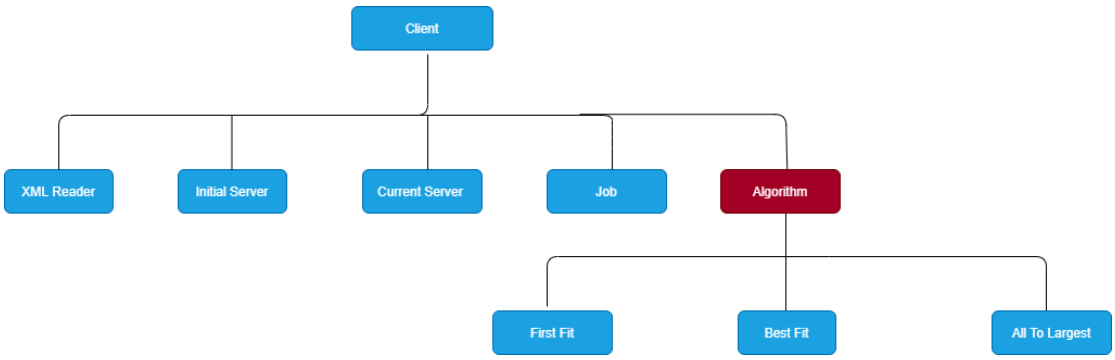
2. **Group Members of Group 42**

Student ID	Student Name	Baseline Algorithm
45141916	Jiahui Lin	First Fit
44129866	Fei Huang	Best Fit

3. **Introduction**

This stage designed to perform the job scheduling procedures to allocate jobs to their target servers upon the server's resource capacity and the chosen baseline algorithm. Each baseline algorithm has its own unique selection method to find an ideal server for each job. For this task, each member of the group needs to implement their assigned algorithm, and the output of their algorithm should be the same as the output generated by default client simulator (ds-client).

4. **Design Consideration and Preliminaries:**



This stage is based on the previous stage with three newly introduced baseline algorithms, First Fit, Best Fit and Worst Fit (no implemented).

Assumption: There must be a server with the initial resource which capable of running all different kinds of jobs.

XML Reader Class: Extracts and read all the server types with their initial resource capacities from System.xml. This information is stored inside of XML reader class and is needed when all the servers with their current resource capacity are not capable of running a task. In this case, we can use this information to schedule this job to servers upon their initial resource capacity rather than their current capacity.

Jobs (Array): An Array is created to store each job information.

Current server (Array List): When the **client** sends a **RESC** message with job information, the **ds-server** will return the servers with their current available

resource capacity, and then this information is fetched by the client and stored it in an Array List. When a baseline algorithm is performed, the client will loop over this list to find its target server.

Algorithm interface: The client simulator is capable of performing a specific algorithm one at a time, depends on the which argument (" -a bf" / "-a ff") is inserted along with command to run client simulator at the terminal. If not argument or unrecognized argument is inserted, the simulator would run the AllToLargest algorithm as the default.

Note: Our group decided to have two different implementations, and each implementation is only capable of running their designated baseline algorithm, either best fit or first fit.

5. Algorithm Description:

Our group only have two members, so only two algorithms are implemented for this project. 'First Fit Algorithm' is designed and implemented by Jiahui Lin and 'Best Fit Algorithm' is designed and implemented by Fei Huang.

First Fit: Jiahui Lin

The algorithm performed by the following steps:

1. Fetches the initial server list and store into the An Array List
   Note: When the client first time sends **RESC All** message, the server will return the servers with their initial resources.

```
    while (!msg.equals(DOT)) {
        client.sendMsg(OK);

        msg = client.getMsg();
        // store the initial server list
        if (!msg.equals(DOT)&& isInitial == true) {
            initialServerList = assembleServerList(initialServerList, msg);
        }
        if (!msg.equals(DOT)) {
            serverList = assembleServerList(serverList, msg);
        }
    }
    isInitial = false;
    return msg;
}
```

initialServerList: Stores the initial server information which returns by the **ds-server** when **RESC All** message is sent at the first time.
IsInitial: Boolean value that determines if this is the first time that **ds-server** returns the server information.
serverList: Stores the current server information returns by ds-server every time the client sends the RESC message.

2. Sort the initial server list according to their resource capacity, sort the current server list according to the ordering in sorted initial server list.

```
private static HashMap < String, String > allToFirstFit(ArrayList < HashMap < String, String >> ini
    ArrayList < HashMap < String, String >> sortedList = new ArrayList < HashMap < String, String >

    // sort the initial server list
    initialServerList.sort(Comparator.comparing(m ->Integer.parseInt(m.get(CPU_CORES)),
                                        Comparator.nullsLast(Comparator.naturalOrder())));

    // sort the current serverList according to orginial list
    for(int i = 0; i < initialServerList.size();i++){
        String initialServerType = initialServerList.get(i).get(SERVER_TYPE);
        int initialServerId = Integer.parseInt(initialServerList.get(i).get(SERVER_ID));
        for(int j = 0; j < serverList.size(); j++){
            String serverType = serverList.get(j).get(SERVER_TYPE);
            int serverId = Integer.parseInt(serverList.get(j).get(SERVER_ID));
            if(serverType.equals(initialServerType) && initialServerId==serverId){
                    sortedList.add(serverList.get(j));
                    break;
            }
       }
    }
```

InitialServerList.sort(): sort the initial server list from smallest to largest by their resource capacity. In our case, the larger server contains larger number of CPU core, larger amount of disk space and memory, thus there is no need to comparing all three parameters, anyone of these parameter is enough determine the ordering of the servers.
SortedList: Stores sorted server list according to the ordering of initial server list.

3. Loop through the sorted list to find a capable server, if no found, schedule the job to the first capable active server.

```
    firstFitServer =new HashMap < String, String > ();

    // find a server of current serverlist with capable resources
    for(int i = 0; i < sortedList.size(); i++){

      int serverCoreSize = Integer.parseInt(sortedList.get(i).get(CPU_CORES));
      int serverDisk = Integer.parseInt(sortedList.get(i).get(DISK_SPACE));
      int serverMemory = Integer.parseInt(sortedList.get(i).get(MEMORY));
      int serverState = Integer.parseInt(sortedList.get(i).get(SERVER_STATE));

      if(cpuCores <=serverCoreSize && memory <=serverMemory && disk <= serverDisk && serverState!=4 ){
            firstFitServer = sortedList.get(i);
            break;
       }
    }

// if all the current servers are not capable, find the server that is initially capable
if(firstFitServer.get(CPU_CORES)== null){
outerLoop:
for(int i = 0; i < sortedList.size(); i++){

     int serverState = Integer.parseInt(sortedList.get(i).get(SERVER_STATE));
      if(serverState ==1 || serverState ==3){
        String serverType = sortedList.get(i).get(SERVER_TYPE);
        int serverId = Integer.parseInt(sortedList.get(i).get(SERVER_ID));

       for(int j = 0; j < initialServerList.size(); j++){

          String initialServerType = initialServerList.get(j).get(SERVER_TYPE);
          int initialServerId = Integer.parseInt(initialServerList.get(j).get(SERVER_ID));
          int initialCoreSize = Integer.parseInt(initialServerList.get(j).get(CPU_CORES));

          int initialDisk = Integer.parseInt(initialServerList.get(j).get(DISK_SPACE));
          int initialMemory = Integer.parseInt(initialServerList.get(j).get(MEMORY));

           if(serverType.equals(initialServerType) && initialServerId==serverId
              && initialCoreSize>= cpuCores && initialDisk >= disk && initialMemory >= memory ){
              firstFitServer = sortedList.get(i);
               break outerLoop;
            }
       }
```

FirstFitServer: Store ideal first fit server.
Memory, Disk, CoreSize are the job details that gets pass to the function as parameters.
ServerCoreSize = server core size.
ServerDisk = server disk space.
ServerMemory = server memory.

For loop: Looping over the sorted server to find first ideal server.

If there is no server available or servers no having enough resource capacity to schedule the job.
outerLoop : nested for loop break point
First For Loop: loop over the sorted server list to find active server.
Second For Loop: If an active server is found, store its server type and server Id, then loop over the initial list to find the corresponding initial resource for this server. Comparing job required resources with active server's initial resource, terminate the loop once a capable server is found.

Best Fit: Fei Huang