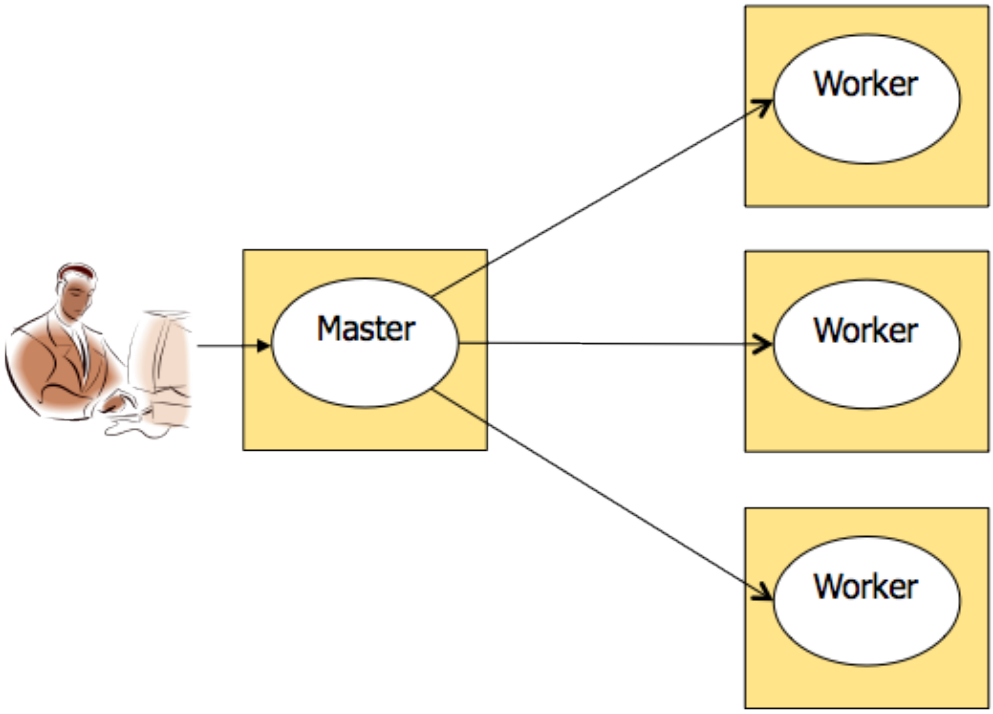
#Distributed Systems
COMP90015 2015 SM1
Project 2 - Job Management System

Synopsis

The purpose of this assignment is to develop a distributed and generic job management solution. Each user will submit jobs from a *Master* node (e.g. their laptop) and the job management system should dispatch them to dedicated remote *Worker* machines for execution. Eventually, the results of a job execution (i.e. an output file) should be returned to the Master.



Job

A Job is a pair of (i) a runnable JAR file and (ii) an input file (e.g. a text file). The JAR file is executed with the following syntax:

```
java –jar <Jar file name>  <input file name>  <output file name>
```

for example:

```
java –jar job.jar input.txt output.txt
```

As an example, a job could consist of an runnable jar file, which multiplies two matrices provided in the input file, and outputs the result in the specified output file.

The JAR file implements the logic which generates the output based on the input. Note that the jobs are specified by the end user and your application can not have any assumptions about the logic they implement. The only assumption you can have is that the jobs run in the above format. Jobs can also be faulty and can fail at any time and your application should expect that.

Master

The master process can be hosted on the end user's machine. It is responsible for receiving jobs and making connections with Worker nodes, sending the jobs, reporting their statuses and fetching back the results.

Master Initialisation: When started, the master process should accept a list of addresses and ports designating all workers. It is up to you to decide what format to use - e.g. CSV, JSON or a text file.

Master User Interface: The Master provides user interface (graphical or command line) through which users can:

- Submit jobs by specifying the jar and input files;
- Monitor the status of an already submitted jobs - e.g. RUNNING, FINISHED, DISCONNECTED or FAILED;
- Get the output/result file when the job completes. If the job fails, the standard output and error should be presented to the user in the Master node instead;
- List all worker nodes and their statuses - e.g. RUNNING, DOWN;
- Add a new Worker node by specifying an address and a port.

Communication with the Workers: The Master connects to each Worker in a client-server fashion using Secure Socket Layer (SSL). Through this channel of communication, the Master implements the aforementioned functionalities. The exact format of communication (e.g. the format of exchanged JSON messages) is up to you. Master will connect with Worker using an SSL connection, and must have appropriate certificates signed by a trusted CA, preinstalled. You will need to generate and distribute the required certificates with Master and Worker code. This will ensure that only authenticated Masters can connect to a given Worker.

Workload Distribution: the Master distributes the jobs to the Workers using a simple Round Robin algorithm.

Deployment: You can execute the master locally from your laptop. You can run it directly from your IDE or build a runnable jar file and start it in the terminal.

Worker

A Worker is a process which can receive multiple jobs from a Master and executes them locally. Workers must be able to report the jobs statuses to the Master (executing, failed etc.).

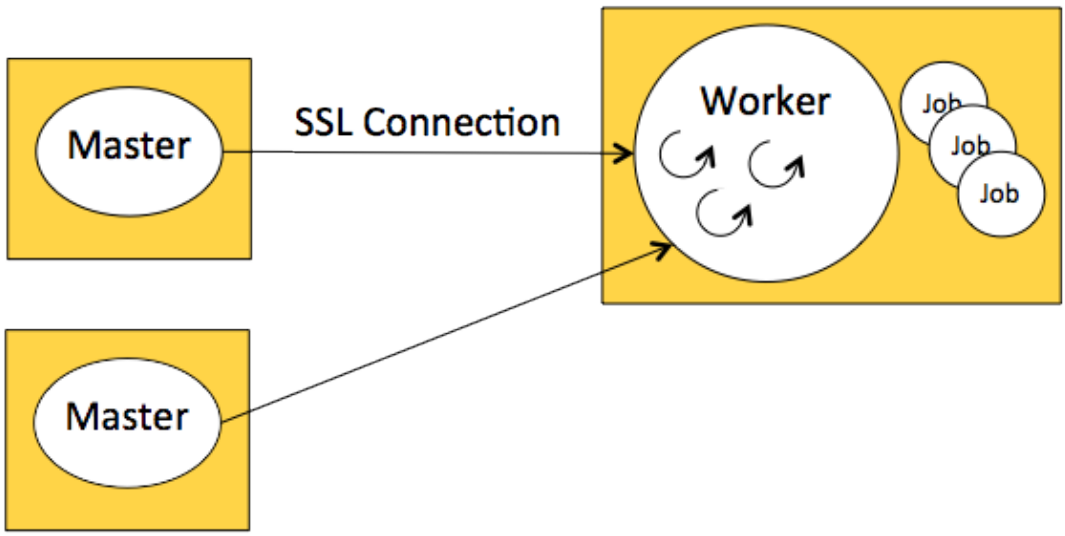
For the purpose of this assignment, you are required to use the Nectar cloud (https://www.nectar.org.au/research-cloud) for hosting the Workers. Alternatively, you can use another cloud provider (e.g. Amazon Web Services) at your expense. Each worker should be deployed in a separate Virtual Machine (VM) in Nectar or another cloud service.

A worker is a server process which can accept connections from multiple Master processes. It can accept their jobs, execute them, return back results and reports their statuses. Multiple masters' requests can be served at

the same time and thus Workers have to be multi-threaded.

Job Execution: A Worker executes a job by creating a new local process (not thread!) after its files (jar and input) are transferred from the master. You can use java's -Process- and -ProcessBuilder- standard library classes (https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html) to run and monitor processes.

Deployment: You can deploy each worker to its VM as a runnable jar file over SSH/SCP. You can start it though SSH.



Project

- This a team based project.
- Team size is 4 people (3 is also acceptable).
- The project must be demonstrated by all team members.
- Each team member will be asked about technical aspects of the project and will receive marks based on their answers and their familiarity with the code and parts of project they have done.

Requirements:

- Develop a job management system as described above.
- Worker must be able to accept SSL connections from more than one Master.
- Worker must be able to accept multiple jobs from different multiple Masters and run each job as a separate process.
- Each Master monitor the health status of the Workers (running, active, dead, etc.) and should be able to detect failed jobs.
- Each Worker monitors its jobs statuses and returns the output files to the Master after they are finished.
- You should handle failures of all types - e.g. faulty jobs, VM failures, network failures (e.g., connection timeout).

- Graphical user interface (GUI) is not required for this project. Command line functionality is enough. Extra points will be considered for projects with (GUI).

Extra Credit:

- Graphical User Interface (GUI) - 5%.
- An advanced job placement algorithm taking into account how overloaded each Worker is. A job is submitted to the worker node with the least number of active jobs or least CPU utilisation. - 5%.
- Allow the user to specify a deadline for each job. If the job does not finish within the deadline then terminate the process and mark the job as failed. - 2%.
- Allow the user to specify the maximum memory (RAM) each job can use. - 3%.

Technical aspects

- Use Java 1.7 or later.
- Use Nectar or alternative cloud services to host Workers.

Your Report

Use 10pt font, double column, 1 inch margin all around. Put your name, login name, and student number at the top of your report.

In up to 1500 words discuss:

- The overall architectural model of you application (use diagrams).
- How did you implement the communication between the master and worker nodes? What protocols and technologies did you use and why? Use UML Sequence diagrams to illustrate the main communication patterns.
- What threads and processes did you use in the assignment?
- How did you handle the different failure types - e.g. job and VM failure.
- Which functionality did each one of the team complete? You will be asked specific questions based on that.

Submission

You need to submit the following via LMS:

- Your report in PDF format only.
- Your source files in a .ZIP or .TAR archive only.
- Your executable master.jar and worker.jar files.

Submissions will be open and due on 11:59pm 31st of May.