**1. Introduction: What this project (focusing on Stage 2 is about, including the goal of the project and Stage 1. (1/2 page)**

This projects aim is to create a load balancing algorithm that meets a set of performance objectives that are derived from a set of predefined baseline algorithms. We will first define load-balancing, then outline the algorithms to compare against, and the performance metrics.

Load balancing is the process of assigning a series of jobs to a set of servers that con process them. Different algorithms can be used to assign the jobs to servers, and the characteristics of the jobs combined with the algorithm usually determines how effective different algorithms are, i.e. it is very rare that any single algorithm is ‘better’ than another in all cases.

We will be comparing the new algorithm to four existing algorithms: first-capable, first-fit, best-fit, and worst-fit. These algorithms are relatively simple, first-capable for example chooses the first server which is capable of running the current job regardless of whether it is already running a job. Understanding how these algorithms work can give us a starting point for building a new algorithm, and understanding what they don’t do can help us create better algorithms.

There are three performance metrics we can use to compare the effectiveness of our algorithms: average turnaround time, average resource utilisation, and total cost. Increasing the performance in one of these usually results in decreasing the performance in another, so will focus on decreasing the average turnaround time: the time between submission of a job, and the completion of that job.

**2. Problem definition (1/2 page): the description of scheduling problem and the definition of your objective function including the justification of your choice.**

If we have to sacrifice one performance metric to increase another, we have to make a decision on which we will focus on. We can rule out utilisation because of the way it is calculated; the baseline algorithms do not migrate jobs and servers are not simulating failures (at least I don’t think they are), which means the utilisation will always be 1, and we can’t have more than the maximum utilisation. That leaves turnaround time or cost.

Both average turnaround time and cost are valid focus’ when creating a new algorithm. Decreasing total cost would mean we have to know the individual cost of each server, then prioritise by cost. Unfortunately cost is only available through the xml configuration files that usually are specified when running ds-server, and without know the names of the files, becomes impossible. This leaves only turnaround time for our focus.

**3. Algorithm description (1 page; one sub-section per algorithm if you have more than one); NB: You need**

* **to provide a simple example scheduling scenario including a sample configuration, the schedule,**
* **the description and discussion; this is to visualise how your scheduling algorithm works.**

How can we decrease the average turnaround time? Examining the baseline algorithms, it becomes clear that they all have one characteristic in common: they never migrate scheduled jobs to use resources that have been made available post-schedule. An example of this is depicted in figure 1, where we can see server 2 is unused for the majority of the simulation because the algorithm doesn’t identify that a resource has become available. Clearly job 3 could have been executed on server 2 in parallel with job 1 as in figure 2, but the order of scheduling prevents this. Having servers that sit idly by while jobs are waiting is a huge waste of time that increases the average waiting time. To achieve this effect with the simulator, we can listen for a JCPL message which notifies us a job has completed and a server might be idle. We can then use the MIGJ command to move a waiting job to an idle server.

A picture containing graphical user interface

Description automatically generated

A picture containing chart

Description automatically generated

What happens to the cost when we use a different server? Cost is calculated by multiplying the runtime by the server cost/second. If server 1 and 2 are of the same type, then regardless of where we place job 3 (behind job 1 or 2), the cost of running in either scenario does not change. Only when server 1 and 2 have different running cost will the total cost change. Using this principle, we can decrease the average turnaround time.

**4. Implementation details including data structure(s) used (1/2 page).**

The implementation for the check algorithm is relatively simple. The pseudo-code is as follows:

1. Receive a job
2. Search for the first available server
3. If a server is available
   1. Assign the job to the server
4. Else
   1. Assign the job to the first capable server
5. If a server becomes idle through completing a job
   1. Find a waiting job that fits on the server
   2. Migrate the waiting job to the idle server

The class diagram for the algorithm implementation is in figure 3. The Client and Check classes are where the bulk of the functionality takes place; Client handles general ds-sim related messaging and program structure, Check contains the algorithm specific instructions like how to make scheduling decisions. The other classes are used as data structures to simplify ds-sim interaction, including Job, Server, JobState, and JobComplete. Generally, Client directly messages the simulation, receives messages in return, processes the information into data structures, and passes the data to a Scheduler.

Diagram

Description automatically generated

5. Evaluation (2 pages): simulation setup including test cases/configurations, results, comparisons (with FF, BF, WF and FC) and discussion including pros and cons of your algorithm. Use figures and tables effectively and be space conscious.