Project Report: Stage 2

COMP3100 Distributed Systems, S2, 2022

SID: 46438688, Name: Arion Dudley

Contents

1 Introduction

This document intends to map and summarise the development of a programming project involving a more complex ds-sim dispatcher/scheduler called MyClient, which is a continuation of the work done for the previous stage of this assignment, with the focus of this second stage being on improving turnaround time without sacrificing too much efficiency in regards to resource utilisation and rental cost. The development of this document occurred concurrently with the development of the programming project over the course of 21 days or roughly 3 weeks, beginning on May 9th 2023 and ending on May 30th 2023.

2 Problem Definition

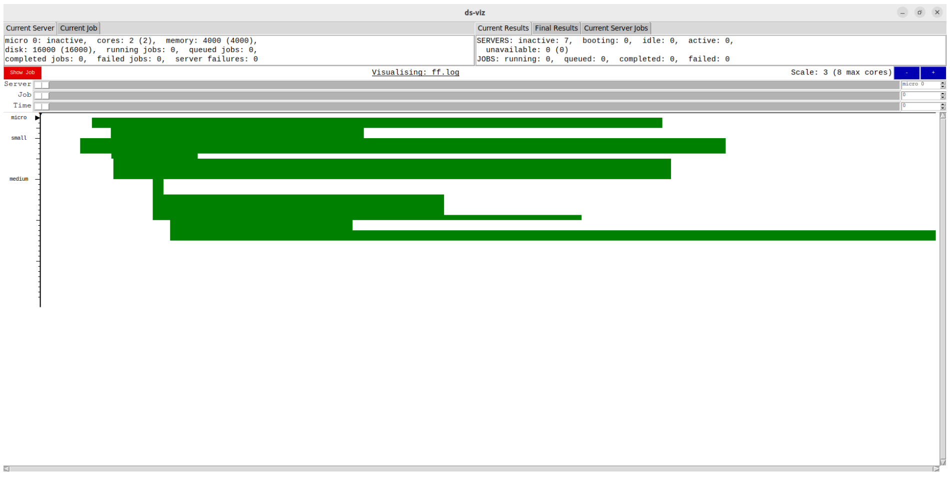
The main problem with the previous scheduling algorithm is that despite optimising resource utilisation and rental cost, it had very poor turnaround times for scheduled jobs. With the focus of this stage primarily being on improving turnaround times, it was decided to experiment with an iteration of best-fit, first-fit, and worst-fit models.

Experimentation with first-fit and best-fit showed the most promise, with worst-fit almost immediately discarded due to its similarity to the previous stage's scheduling logic. First-fit was attempted first, and had very good turnaround times, but still had poor resource utilisation and could not pass the first two stages of the implementation section of the assignment rubric. Efforts were then made to implement best-fit, which immediately showed drastically improved turnaround times but at the cost of resource utilisation and total rental cost scores.
The best-fit model thus was judged the best model for the scheduling algorithm and was implemented while attempting to cut away unnecessary functionalities such as unused statements and unnecessary calculations that existed in the previous stage's submission.

3 Algorithm Description

The main algorithm follows a best-fit scheduling preference, and follows a simple set of two parameters once the server information has been calculated and split into a readable String ArrayList called serverDataArray.

The first parameter is to check the number of needed CPUs against the number of available CPUs for a given server, and will requisition the named type and ID of the first server that clears the CPU requirements. The second parameter is used as a fallback if no best-fit server is identified, and simply calls the named type and ID of the first server in serverDataArray. One of these parameters is then stored in a String array called calculatedServers and returned to the main try-catch at the end of the algorithm.



```
micro 0: inactive,  cores: 2 (2),  memory: 4000 (4000),
disk: 16000 (16000),  running jobs: 0,  queued jobs: 0,
completed jobs: 0,  failed jobs: 0,  server failures: 0
```

```
SERVERS: inactive: 7,  booting: 0,  idle: 0,  active: 0,
  unavailable: 0 (0)
JOBS: running: 0,  queued: 0,  completed: 0,  failed: 0
```

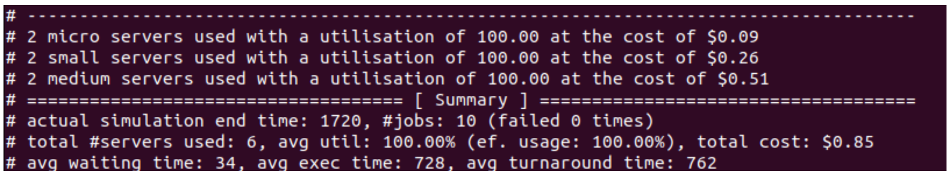
Visualising: ff.log

Scale: 3 (8 max cores)

Figure 1: Visualisation of MyClient running with ds-server using the config: sample-config01-2.xml found in configs/sample-configs. Generated with ds-viz.

A simple and demonstrative example scenario using sample-config01-2.xml begins with handshaking, then schedules the jobs in this order:

- SCHD 0 small 0 at 37 seconds
- SCHD 1 micro 0 at 60 seconds
- SCHD 2 micro 1 at 96 seconds
- SCHD 3 small 1 at 101 seconds
- SCHD 4 small 0 at 137 seconds
- SCHD 5 medium 0 at 156 seconds
- SCHD 6 medium 0 at 178 seconds
- SCHD 7 medium 1 at 190 seconds
- SCHD 8 medium 0 at 214 seconds
- SCHD 9 medium 1 at 229 seconds



```
# --------------------------------------------------------------------------------
# 2 micro servers used with a utilisation of 100.00 at the cost of $0.09
# 2 small servers used with a utilisation of 100.00 at the cost of $0.26
# 2 medium servers used with a utilisation of 100.00 at the cost of $0.51
# ================================== [ Summary ] ==================================
# actual simulation end time: 1720, #jobs: 10 (failed 0 times)
# total #servers used: 6, avg util: 100.00% (ef. usage: 100.00%), total cost: $0.85
# avg waiting time: 34, avg exec time: 728, avg turnaround time: 762
```

Figure 2: Final results from sample-config01-2.xml

From this simple scenario, resource utilisation stands at 100 percent with a correspondingly low rental cost. Turnaround time is also good, but this is a simple example, and as we will see later more complex configs have a much more challenging assessment.

4 Implementation

The implementation of the newer scheduling algorithm is based largely off the structure and design of the previously submitted scheduler for Stage 1 of this assignment project. Thus, code structure and statements are nearly identical, but with some important distinctions.

In this implementation, the program relies chiefly on 5 variable statements that come with descriptions for ease of readability and understanding. These 6 variables are;

- typeCapableServer, a string which is used to track a given server type's name, which constantly changes throughout the execution of the program.
- idCapableServer, an integer which is used to track a given server ID, which constantly changes throughout the execution of the program.
- serversCalculated, a boolean that is used to make sure the program only calculates server information when absolutely necessary. It begins FALSE and flips to TRUE in two circumstances. The first being if the method calculateServers is called, and the second being if the last message from the server was not JOBN, to make sure the program does not needlessly spend resources on a non-job query.
- messageTracker, this is a string that simply stores the most recently received message from the server.
- jobData, this is a string array which is used to split up important messages, usually containing information of a specific job to be scheduled, but which is also used to store other preliminary information that is not to be processed, such as JCPL and NONE ds-sim server messages.

These variables are then routinely used by a series of 6 modular functions which are named to directly describe their purpose. These 6 functions are;

- sendData, a function which utilises a socket OutputStream to send a message to the server.
- receiveData, a function which utilises a socket InputStream to receive messages from the server via a BufferedReader.
- receiveCompliantData, this is a function which is very similar to receiveData, but enforces a condition that the received message must exactly resemble an input string, if the received message does not meet this requirement, receiveCompliantData will instead return the string ”NONE” instead of the actually received message.
- calculateServers, the most complex function in this list, it begins by using sendData to transmit OK to the server then receives and splits an input string array into a string ArrayList, this is then used by the best-fit algorithm to dissect and analyse server information to decide which server type and ID to return as described in the Algorithm Description section.
- processJob, a function that checks if the input string is null or empty before proceeding to split the string array and determine if the input matches that of a JOBN message sent by the server, the function then returns the input regardless but this distinction allows for better debugging if debug is set to TRUE.
- scheduleJob, a simple but critical function that relies on an input string array, string, integer, OutputStream, and BufferedReader. It unceremoniously places these inputs into a jobMessage string which is then sent to the server via a sendData function.

5 Evaluation

When compared against baseline algorithms, the new scheduling algorithm meets an average performance level in average turnaround time. This will be demonstrated with a series of following images and tables. Specifically comparing the baseline ds-client best-fit compared to MyClient's best-fit approach using two configs from S2TestConfig, the first being config12-long-med and the second being config16-short-med.

The first comparison will show the final results from terminal of ds-client versus MyClient using config12-long-med.xml, then followed by a visualisation table for MyClient.

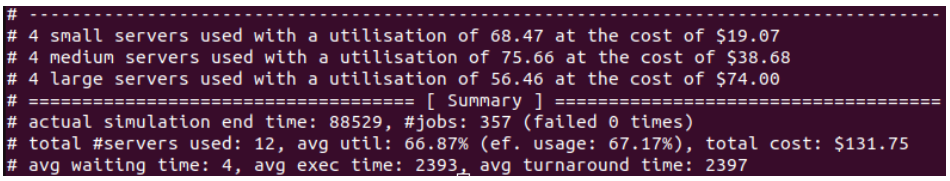
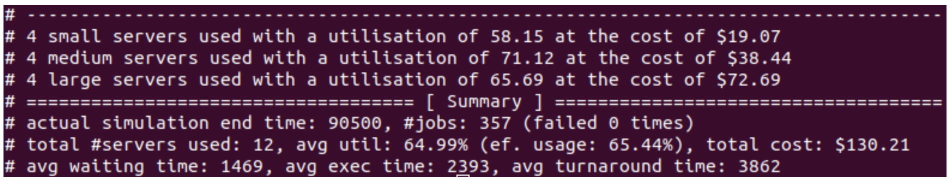

```
# --------------------------------------------------------------------------------
# 4 small servers used with a utilisation of 68.47 at the cost of $19.07
# 4 medium servers used with a utilisation of 75.66 at the cost of $38.68
# 4 large servers used with a utilisation of 56.46 at the cost of $74.00
# ==================================== [ Summary ] ===================================
# actual simulation end time: 88529, #jobs: 357 (failed 0 times)
# total #servers used: 12, avg util: 66.87% (ef. usage: 67.17%), total cost: $131.75
# avg waiting time: 4, avg exec time: 2393, avg turnaround time: 2397
```

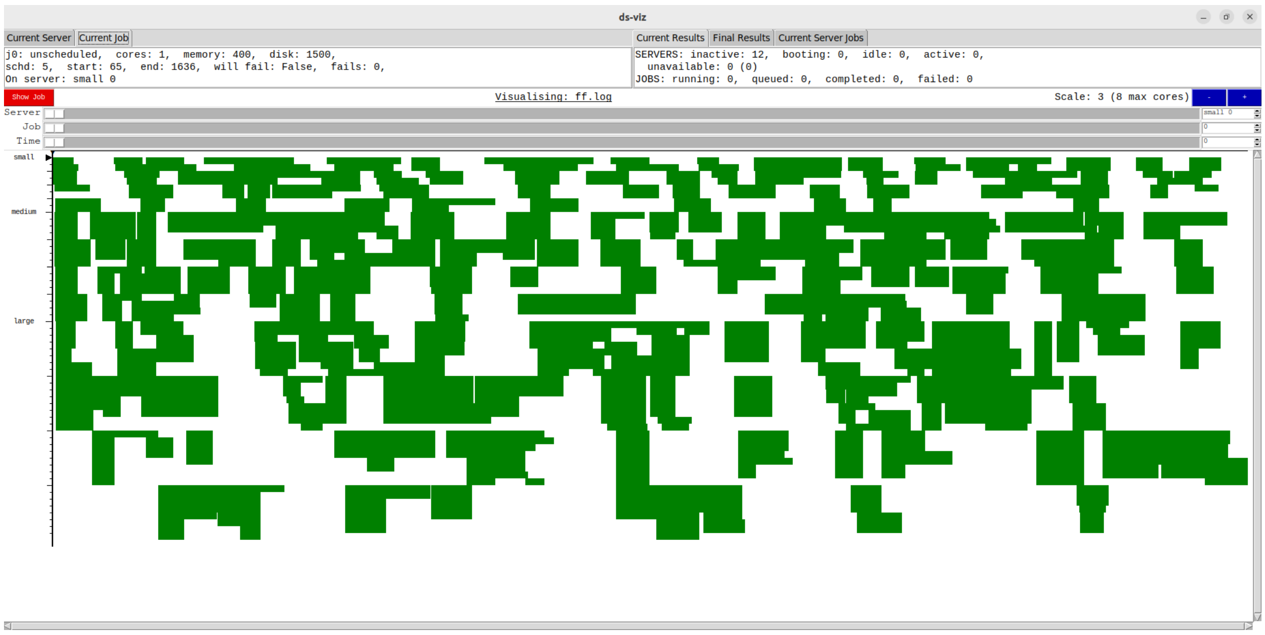
Figure 3: config12-long-med.xml running against ds-client's best-fit



```
# ------------------------------------------------------------------------------------
# 4 small servers used with a utilisation of 58.15 at the cost of $19.07
# 4 medium servers used with a utilisation of 71.12 at the cost of $38.44
# 4 large servers used with a utilisation of 65.69 at the cost of $72.69
# ==================================== [ Summary ] ====================================
# actual simulation end time: 90500, #jobs: 357 (failed 0 times)
# total #servers used: 12, avg util: 64.99% (ef. usage: 65.44%), total cost: $130.21
# avg waiting time: 1469, avg exec time: 2393, avg turnaround time: 3862
```

Figure 4: config12-long-med.xml running against MyClient's best-fit

As it can be seen, MyClient's scheduling logic distributes jobs to servers in a similar manner to ds-client. MyClient demonstrates an identical distribution between server types, a near-identical rental cost, and a roughly similar turnaround time. The optimisation of rental cost for MyClient is slightly better with this config, although the turnaround time for MyClient is slightly worse. An exact table of scheduling for MyClient is shown in Figure 5, showing the mapping of jobs being scheduled to servers, their scheduling time, waiting time, and completion time. The visualisation was compiled using ds-viz.



```
j0: unscheduled,  cores: 1,  memory: 400,  disk: 1500,
schd: 5,  start: 65,  end: 1636,  will fail: False,  fails: 0,
On server: small 0
```

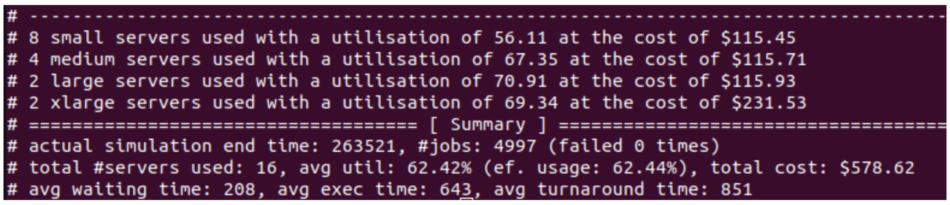
```
SERVERS: inactive: 12,  booting: 0,  idle: 0,  active: 0,
  unavailable: 0 (0)
JOBS: running: 0,  queued: 0,  completed: 0,  failed: 0
```

Visualising: ff.log

Scale: 3 (8 max cores)

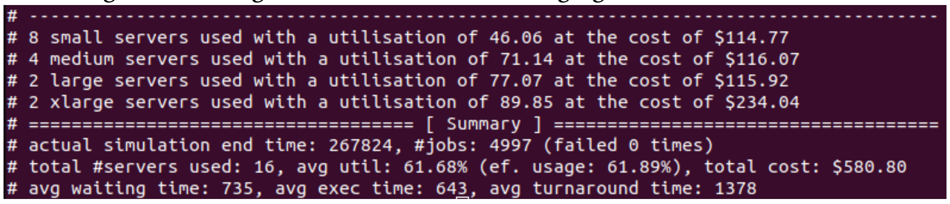
Figure 5: MyClient's best-fit for config12-long-med.xml visualised with ds-viz

Now, the second comparison will show the final results from terminal of ds-client versus MyClient using config16-short-med.xml, then followed by a visualisation table for MyClient.



```
# --------------------------------------------------------------------------------
# 8 small servers used with a utilisation of 56.11 at the cost of $115.45
# 4 medium servers used with a utilisation of 67.35 at the cost of $115.71
# 2 large servers used with a utilisation of 70.91 at the cost of $115.93
# 2 xlarge servers used with a utilisation of 69.34 at the cost of $231.53
# ==================================== [ Summary ] ====================================
# actual simulation end time: 263521, #jobs: 4997 (failed 0 times)
# total #servers used: 16, avg util: 62.42% (ef. usage: 62.44%), total cost: $578.62
# avg waiting time: 208, avg exec time: 643, avg turnaround time: 851
```

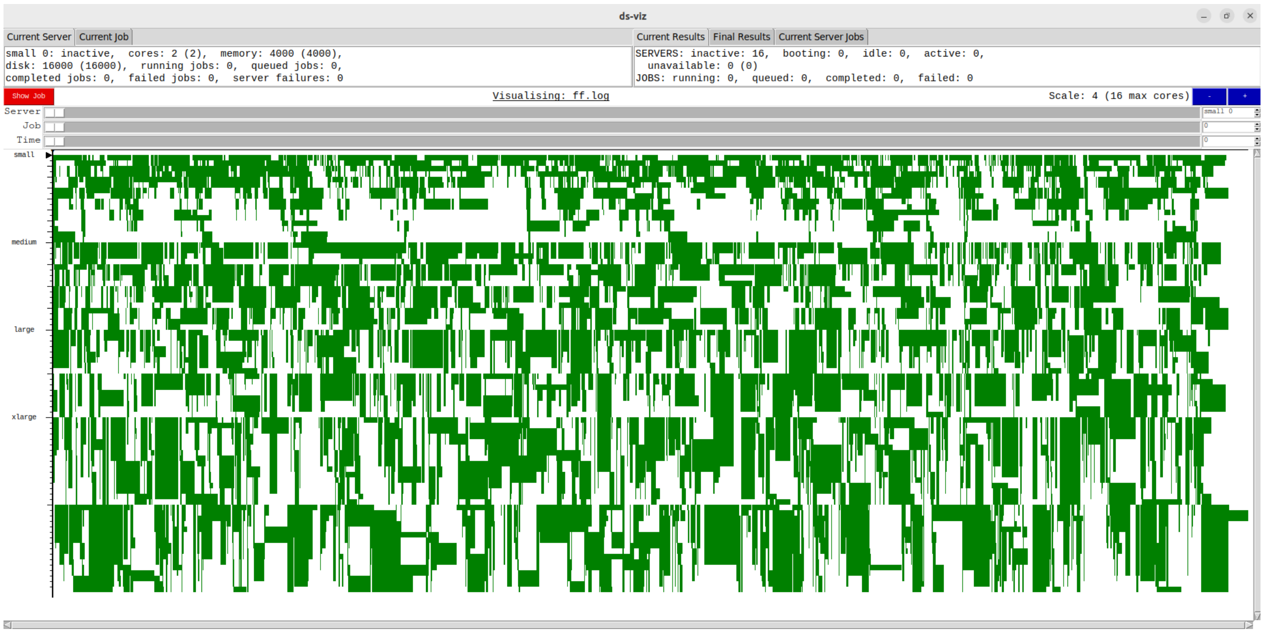
Figure 6: config16-short-med.xml running against ds-client's best-fit



```
# --------------------------------------------------------------------------------
# 8 small servers used with a utilisation of 46.06 at the cost of $114.77
# 4 medium servers used with a utilisation of 71.14 at the cost of $116.07
# 2 large servers used with a utilisation of 77.07 at the cost of $115.92
# 2 xlarge servers used with a utilisation of 89.85 at the cost of $234.04
# ==================================== [ Summary ] ====================================
# actual simulation end time: 267824, #jobs: 4997 (failed 0 times)
# total #servers used: 16, avg util: 61.68% (ef. usage: 61.89%), total cost: $580.80
# avg waiting time: 735, avg exec time: 643, avg turnaround time: 1378
```

Figure 7: config16-short-med.xml running against MyClient's best-fit

As with the previous test, MyClient's scheduling logic distributes jobs to servers in a similar manner to ds-client. MyClient demonstrates an identical distribution between server types, a near-identical rental cost, although with a slightly worse turnaround time compared to the previous config. Another contrast is evident compared to the previous config, as this time the optimisation of rental cost for MyClient is slightly worse, with the turnaround time for MyClient also being worse compared to ds-client. An exact table of scheduling for MyClient is shown in Figure 5, showing the mapping of jobs being scheduled to servers, their scheduling time, waiting time, and completion time. The visualisation was compiled using ds-viz.



```
small 0: inactive,  cores: 2 (2),  memory: 4000 (4000),
disk: 16000 (16000),  running jobs: 0,  queued jobs: 0,
completed jobs: 0,  failed jobs: 0,  server failures: 0
```

```
SERVERS: inactive: 16,  booting: 0,  idle: 0,  active: 0,
  unavailable: 0 (0)
JOBS: running: 0,  queued: 0,  completed: 0,  failed: 0
```

Visualising: ff.log

Scale: 4 (16 max cores)

Figure 8: MyClient's best-fit for config16-short-med.xml visualised with ds-viz

Overall, the benefits of using MyClient's algorithm are slight compared to ds-client's baseline best-fit model. Using the given S2TestConfigs from the Week 11 workshop show a roughly average level of performance, with MyClient's turnaround time and resource utilisation being slightly worse and MyClient's total rental cost being slightly better, with 1 config returning a superior score in regards to total rental cost. A use-case for MyClient is thus a matter of preference and not logic, as it does not significantly differ from ds-client's best-fit model as it demonstrates only marginal changes in performance, with only a small amount of configurations performing better for MyClient, and a smaller amount of configurations performing worse for MyClient.

6 Conclusion

This program successfully cleared the first two requirements of the implementation rubric when tested against the given S2TestConfigs supplied in the Week 11 workshop, yet failed to achieve success at creating a superior-than-average scheduling algorithm that could reliably clear more than one or two of the performance requirements in regards to resource utilisation and total rental cost.

While attempts were made at integrating a global queue into this implementation during the final days of development, all failed to achieve notable success and were scrapped before the final commit due to time constraints. In the future, attempting to holistically implement a global queue early on in development rather than attempting to integrate it after completing experimentation would be the main lesson learnt.

7 Citations and References

This client program relies entirely on two critical systems, ds-sim and Git.

ds-sim, which is an open-source, language-independent and configurable distributed systems simulator. It is designed to perform a quick and realistic simulation of job scheduling and execution in distributed systems, such as computer clusters and (cloud) data centres.

Git[1], as well as Github[2], was used extensively as the main project management tool for this program. Progress on the program maintained a steady pace throughout the weekly tutorials and through work at home. The first git push was a simple helloWorld program, and the following week's git push being a rudimentary ds-client that utilised UTF functions instead of a proper socket input/output stream and buffered reader. The project files can be found on this specific github[3].

References

[1] *Git: About Page*. URL: `https://git-scm.com/about`.

[2] *Github: Main Page*. URL: `https://github.com/`.

[3] *Github Project Page for Stage 2 by Arion Dudley*. URL: `https://github.com/FantasticRiceCrackers/comp3100Project`.