Project report: Stage 2
COMP3100 Distributed Systems, S1, 2022
SID: 46129448, Name: Mariel Anne Uykim

1 Introduction

DS Sim is a discrete event simulator that simulates a distributed system. The main goal of DS Sim is to schedule a list of jobs to a cluster of servers in a given system for them to execute. It consists of two main parts, the client side and the server side. This report will mainly focus on the client side aspect of DS Sim (DS Client) which is responsible for scheduling each job to an appropriate server. The scheduling of jobs is an important aspect of a distributed system. As this plays an important role in the overall performance of the system, it is important that the algorithm used to implement this is efficient. In the previous implementation of the DS Client program, the Largest Round Robin algorithm was implemented to schedule jobs. While this algorithm is able to schedule all given jobs, it was not efficient with assigning jobs to servers. The current program implements a more efficient algorithm which assigns jobs to a better fitted server (is further discussed in the next section). The GitHub repository for the final output can be viewed here (https://github.com/mariel-uykim/DSClient.git). The main DSClient file is titled 'DSClient.java'.

2 Problem Definition

The main objective of this stage is to implement a new algorithm that optimises the scheduling efficiency of the DS Sim. Efficiency is measured by multiple performance metrics including resource utilisation, total cost, and turnaround time. Resource utilisation refers to how well the system resources are used while total cost refers to the overall price of executing the tasks [1]. This report will specifically focus on improving the average turnaround time of a task. The turnaround time is defined as the gap between the time that the task was scheduled and the task was completed. A lower average turnaround time indicates that jobs are executed in a shorter time period [2]. The baseline algorithms used in this project are first capable, best fit, first fit, and worst fit algorithm. First capable algorithm selects the first server that is capable of executing the job. First fit algorithm finds the first server that satisfies the job requirements and assigns the job to it. Best fit searches for the smallest possible server that can handle the job. Worst fit finds the largest server that can handle the job and allocates the job to it [3].

The problem with the initially implemented LRR algorithm was that despite it having a good average resource utilisation compared to the other baseline algorithms, it had a high turnaround time compared to first fit, best fit, and worst fit. The goal of the new algorithm is to have the best turnaround time among the the baseline algorithms while still having better resource and utilisation and cost than one of the four baseline algorithms. Having a better turnaround time would generally lead to having a higher cost or a lower resource utilisation as a consequence.

Table 1: Average Performance Metrics on Algorithms with Stage 2 Test Configurations

Algorithm	Turnaround Time	Resource Utilisation	Rental Cost
LRR	18,228.94	96.93	509.04
First Capable	246,382.78	94.37	379.20
First Fit	1,803.50	63.30	638.18
Best Fit	1,866.72	60.72	636.89
Worst Fit	19,609.94	68.79	665.09

As shown in Table 1, first fit and best fit have the best turn around time but have lower resource utilisation compared to the other algorithms. They also operate at a higher cost compared to first capable and LRR which have the highest turnaround times.

3 New Scheduling Algorithm

The current algorithm aims to improve the turnaround time of the distributed jobs while minimising the negative effects it has on the other two performance metrics. It implements this by having multiple factors that determine which server to pick.

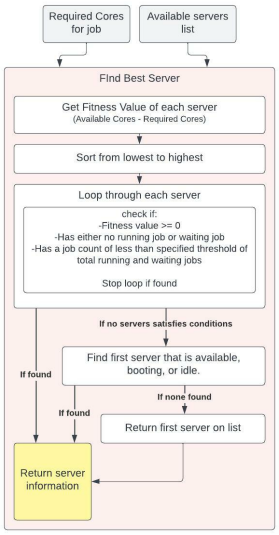


Figure 1: Process of deciding which server to use

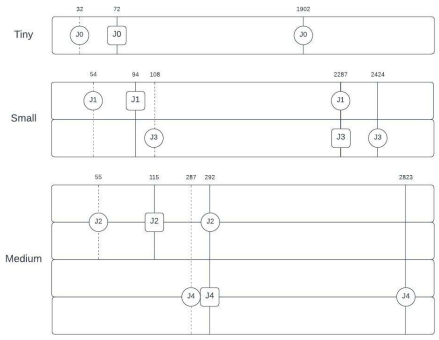
The figure above depicts how the algorithm decides which server to use. It requires two input parameters, the required cores for the current job and the available servers including details about them. The algorithm first gets all the fitness values of the servers which is calculated by the number of cores they have available subtracted by the number of cores required for the job. Second, the fitness values are sorted in ascending order. Third, a loop goes through each servers which are sorted by their fitness value. An if condition checks whether the current server satisfies the requirements of: having a fitness value greater than or equal to 0; has either no running or waiting jobs; and has a job count of less than or equal to the specified threshold of the total running and waiting jobs. The loop immediately breaks when a server fits these requirements.

The aforementioned conditions were chosen for several reasons. The fitness value makes sure that the smallest possible server is chosen or a running server that is already executing a job is used. This would reduce the total rental cost in most scenarios as lesser or smaller servers are chosen. This will also allow us to chose the best fitted server that is available as the smaller the fitness value, the less processor wastage there is. However, a negative fitness value connotes that the server does not have enough available capacity to execute the task. The second condition of needing to have either no running or waiting job is to ensure that the waiting time is not long. Similar to the previous condition, the third condition checks whether a server is overloaded depending on the specified percentage (in the code, a threshold of 20% was set) of the total jobs in queue and are being executed. This value can be modified, with a higher value indicating a higher maximum capacity for a server. This will ensure that a single server does not have most of the jobs and will minimise delays in execution time.

If no server satisfies these three condition, a loop goes through the each servers and break ones it finds the first server that is available, booting, or idle. If no servers satisfies these, a worst case scenario of selecting the first server on the list is performed. The scheduling process of the new algorithm is illustrated in Figure 2 which was executed with the week 9 configuration that has three servers and four jobs. As shown, Job 0 is assigned to server Tiny since it only requires one core. Job 1 is assigned to server Small since it requires two cores. The algorithm assigns the servers into the smallest servers that can execute them. However, Job 2 is assigned to server Medium despite only requiring 2 cores. Since server Small is still busy executing Job 1, the algorithm assigns it to the next available server. Job 3 is assigned to server Small since it only requires two cores and since there are no vacant servers, it has wait for Job 1 to finish executing. Job 4 is assigned to server Medium as this is only server that can accommodate 4 cores.

```
<!--  generated by: Y. C. Lee  -->
<config randomSeed="65535">
<servers>
<server type="tiny" limit="1" bootupTime="40" hourlyRate="0.4" cores="1" memory="4000"
disk="32000"/>
<server type="small" limit="1" bootupTime="40" hourlyRate="0.4" cores="2" memory="8000"
disk="64000"/>
<server type="medium" limit="1" bootupTime="60" hourlyRate="0.8" cores="4" memory="16000"
disk="128000"/>
</servers>
<jobs>
<job type="short" minRunTime="1" maxRunTime="60" populationRate="30"/>
<job type="medium" minRunTime="61" maxRunTime="600" populationRate="40"/>
<job type="long" minRunTime="601" maxRunTime="3600" populationRate="30"/>
</jobs>
<workload type="unknown" minLoad="20" maxLoad="60"/>
<termination>
<condition type="endtime" value="86400"/>
<condition type="jobcount" value="5"/>
</termination>
</config>
```

(a) Configuration File



(b) Scheduling

Figure 2: Schedule generated by new algorithm

4 Implementation

To create the DS-Sim Client-side simulator and to ensure it is able to communicate to the server properly to do its tasks, the following libraries, data structure, and functions were implemented. The process of establishing a connection, communicating with DS Server, and terminating the connection are similar to stage 1. The main difference lies in the scheduling algorithm used and the command used to retrieve server information.

4.1 Technologies and Libraries

DS Client utilises three in-built Java libraries. This includes the Java.net package to socket class which allows the application to connect to the server. The second package is the Java.io package which is used for reading inputs from the server and outputting data or queries to the server. The last package used is the Java.util package which is used to implement the ArrayList and HashMap data structure (discussed further in the next sub section).

4.2 Data Structure

DS Client uses arrays and array list to store information about server. The information handed out by DS Server is stored into an array and the array list methods such as sort and add are used to manipulate the list as needed. A hash map is used to store information on all the running and waiting jobs of each server and is a global variable that is accessed by other functions to obtain or change the data regarding server jobs.

4.3 Functions

DS Client has several functions that are used to implement certain features. The main code runs inside a while loop and continuously loops until no more job is found or an error has occurred. The initial code outside of the while loop, establishes a connection between the server and authenticates with it. The rest of the functions inside the loop, are responsible for getting server information and scheduling the jobs.

4.3.1 Server Connection

Using the Socket class from Java.net package, a function was implemented to connect to the server that is listening on the same port as the Client.

4.3.2 Authentication

The client authenticates itself to the DS Server by using the AUTH command followed by the name of the system. The server responds with an OK if authentication was successful.

4.3.3 Sending and Receiving Messages

The DS Client has separate functions for sending messages to the server and receiving messages from the server. Using Java.io's BufferReader and its readline method that reads from the InputStreamReader, the client is able to read the messages sent by the server. The function for sending messages uses the DataOutputStream

to write the output to be sent to the DS Server. These functions are used in order to communicate to the server by sending commands and receiving data.

4.3.4 Getting Server Information

To get information on the servers, the client has to use the GETS command which has a parameter of getting servers based on the capabilities, availability, or all the servers. The current DS Client application uses the GETS Avail command which responds with DATA nRec size. The nRec being the number of servers available. This command returns back all the available servers that are capable of executing the job. If no server is returned, the GETS Capable command is sent which returns back all capable servers.

4.3.5 Finding The Best Server

In order to find the best server type, the function goes through a list of array that is returned back from the GETS command and calculates the fitness value of each server. Using the sort method from the Java.util package, a duplicate array of the fitness values is sorted in ascending order. This is done in order to preserve the index of each value in the list. The .indexOf method is used to find the server that the fitness value belongs to or the first server with the fitness value. An if condition then checks if the server fits the conditions discussed in the previous section. If it fits, the loop stops and that server is returned. Otherwise, if no server fits the aforementioned criteria, the servers run through another loop that stops once it finds a server that is idle, booting, or active and returns this server back. In the eventuality that no server is still found, the first server in the list is returned back.

4.3.6 Scheduling Jobs

Once a server is returned from the getBestServer() function, the program creates a schedule message using the SCHED command, the server type and id, and the job id. It then sends it to DS Server which would then assign the job the selected server.

4.3.7 Closing Connection

After scheduling all jobs and successfully executing all of them, the Client sends a QUIT command to the server and closes the socket connection in the client side before terminating completely.

5 Evaluation

5.1 Setup

The new algorithm was tested on a total of 18 configuration files with varying types and length. Each configuration file has five server types (tiny, small, medium, large, x-large), five job types (instant, short, medium, long, x-long), and three workload pattern (low, medium, high). The five server types remain constant while job types proportion and workload pattern vary depending on the file. Job types refer to the length of a job and workload pattern refers to the percentage of system usage [4].

5.2 Results and Discussion

The results are measured using the three performance metrics: turnaround time, rental cost, and resource utilisation. The results show that the new algorithm has a significantly better turnaround time than the other baseline algorithms on configurations with high workload type while generally performs on par with the first fit and best fit on those with short and medium workload types. On the other hand, first capable and worst fit perform poorly in terms of turnaround time, however, due to space limitations is not displayed entirely.

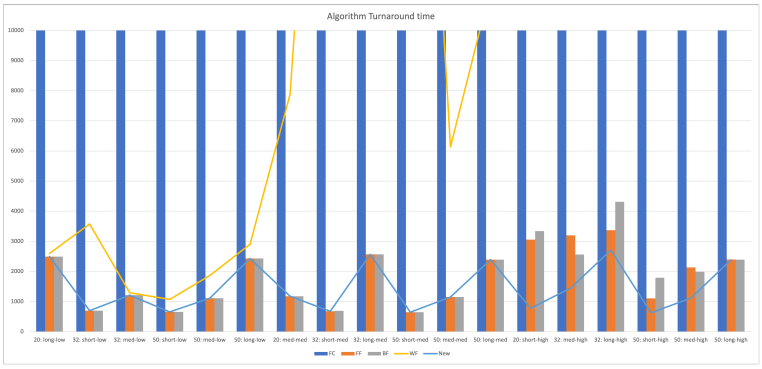


Figure 3: Turnaround time of each algorithm on the 18 configuration files

Algorithm	Turnaround Time	Resource Utilisation	Rental Cost
LRR	18,228.94	96.93	509.04
First Capable	246,382.78	94.37	379.20
First Fit	1,803.50	63.30	638.18
Best Fit	1,866.72	60.72	636.89
Worst Fit	19,609.94	68.79	665.09
New Algorithm	1,459.28	61.44	643.30

Table 2: Average Performance Metrics Comparison

The table above shows the average of each algorithm for the three performance metrics. As shown, first capable has the highest turn around time, however, also has the best resource utilisation and has the lowest cost. A similar pattern is also observed with the LRR algorithm implemented in Stage 1. The new algorithm has the best turnaround time among the other algorithms, however, has poor resource utilisation that is only better than best fit and has the second highest rental cost next to worst fit.

The advantages of using the new algorithm is that it generally has the lowest turnaround time which makes the overall time to execute the jobs shorter. However, as a consequence, the overall cost is higher and the resource utilisation is less efficient. This algorithm can be used if cost and resource utilisation is not as significant as the turnaround time for the user.

6 Conclusion

The new algorithm implemented had a better overall performance in terms of turnaround time however this could be further improved with the other types of workload and with the other performance metrics. As observed from the results, the algorithm works better with high workloads compared to the small and medium ones. It could also be improved upon in terms of resource utilisation and cost in future versions.

References

[1] I. N. Ivanisenko and T. A. Radivilova, “Survey of major load balancing algorithms in distributed system,” in *2015 Information Technologies in Innovation Business Conference (ITIB)*, pp. 89–92, Oct. 2015.

[2] V. Behal and A. Kumar, “Cloud computing: Performance analysis of load balancing algorithms in cloud heterogeneous environment,” in *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, (Noida, India), pp. 200–205, IEEE, Sept. 2014.

[3] “Cs online: Memory allocation - virginia tech.”

[4] Y. C. Lee, J. King, Y. K. Kim, and S.-H. Hong, “Robust Scheduling for Large-Scale Distributed Systems,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, (Guangzhou, China), pp. 38–45, IEEE, Dec. 2020.