**---------------------------------------------------------------------------**

**Scheduling Algorithm**

**Design and Implementation**

**---------------------------------------------------------------------------**

**Student Details**

|  |  |  |
| --- | --- | --- |
| Name | Student ID | GitHub Name |
| Edoardo Busano | 45757100 | edoardobusano |

**Introduction**

The project is centred on designing and developing a new scheduling algorithm for the client-side. The algorithm must not only work, but also needs to be optimised for at least one of the following objectives: minimisation of average turnaround time, maximisation of average resource utilisation, and minimisation of total server rental cost. The initial step to later achieve these goals is to first prioritize one of the objectives, then study how it is related to scheduling decisions, to finally develop the algorithm and translate it into code.

This report will include the design process, the development of the algorithm, an example of a scheduling scenario using the new algorithm, and the final evaluation, containing a comparison with the existing scheduling algorithms First Fit, Best Fit, Worst Fit, and All To Largest as well as the pros and cons of using it when taking in consideration the three objectives previously described.

**Problem Definition**

Between the turnaround time, the resource utilisation, and the rental cost, I decided to give less importance to the latter, and prioritise resource utilisation, while still taking in consideration the turnaround time. By analysing the four pre-existing algorithms, it is clear that both All To Largest and Worst Fit are the best fitted to maximise the average resource utilisation, however their average turnaround time is fairly high, as well as the total server rental cost for Worst Fit. By design, the resource utilisation of All To Largest is impossible to outdo, as the algorithm schedules every job into the first server with the largest server type, and the resource utilisation always ends up being maximised. However, Worst Fit has a high percentage for resource utilisation, but there is still a margin for improvement.

Hence, the idea of this new algorithm is to surpass Worst Fit in all three performance metrics, with a special focus on resource utilisation.

**Algorithm Description**

The algorithm, when given a job and a set of servers as the input, searches for the server with the highest core count to schedule the job; if two or more servers provide the same resources, the first one is chosen for scheduling. When there are no available servers that are able to run the job, it is scheduled to the server with the highest core count capable of running it.

The following scheduling scenario will provide an example of the algorithm’s functioning.

Configuration:

* Server 1: bootupTime=”40”, hourlyRate=”0.4”, coreCount=”1”, memory=”4000” disk=”32000”
* Server 2: bootupTime=”40”, hourlyRate=”0.4”, coreCount=”2”, memory=”8000” disk=”64000”
* Server 3: bootupTime=”60”, hourlyRate=”0.8”, coreCount=”4”, memory=”16000” disk=”128000”

Jobs to schedule:

* Job 0: coresRequied=”1” memoryRequired=”700” diskRequired= “600”
* Job 1: coresRequied=”1” memoryRequired=”400” diskRequired= “800”
* Job 2: coresRequied=”2” memoryRequired=”900” diskRequired= “1600”
* Job 3: coresRequied=”2” memoryRequired=”500” diskRequired= “3300”
* Job 4: coresRequied=”4” memoryRequired=”1600” diskRequired= “4600”

Actors:

S is the Server.

C is the Client.

Scheduling process:

The algorithm runs after the first REDY is acknowledged, and S sends the first job to schedule: Job 0. C reads the job information and asks for available servers for Job 0. S replies with a list of servers: Server 1, Server 2, and Server 3. C picks the server with the highest coreCount: Job 0 is scheduled to run on Server 3.

S sends Job 1 for scheduling: C reads the job information and asks for available servers for Job 1. S replies with a list of servers: Server 1 and Server 2. C picks the server with the highest coreCount: Job 1 is scheduled to run on Server 2.

S sends Job 2 for scheduling: C reads the job information and asks for available servers for Job 2. S replies with an empty list. C asks for capable servers for Job 2. S replies with a list of servers: Server 2 and Server 3. C picks the server with the highest coreCount: Job 2 is scheduled to run on Server 3.

S sends Job 3 for scheduling: C reads the job information and asks for available servers for Job 3. S replies with an empty list. C asks for capable servers for Job 3. S replies with a list of servers: Server 2 and Server 3. C picks the first server with the highest coreCount: Job 3 is scheduled to run on Server 2, as now they both have 1 core available.

S sends Job 4 for scheduling: C reads the job information and asks for available servers for Job 4. S replies with an empty list. C asks for capable servers for Job 4. S replies with Server 3. Job 4 is scheduled to run on Server 3.

The same schedule can be seen by running the ds-server using configs/sample-configs/ds-config01-wk9.xml.

This new algorithm, opposed to All To Largest, it has a more dynamic design, as the queue is processed in such a way that the resource utilisation is still kept high on average, but it is spread on a higher amount of servers, which also reduces the turnaround time as a result.

**Implementation**

To implement the algorithm, a few new methods were added to the client from Stage 1. Below is a list of the methods that are part of the algorithm.

theAlgorithm(): method that contains the communication between Client and Server for the scheduling part. It reads incoming jobs to be scheduled, and sends the job information to selectServer(String[] j), which decides which server should be picked for the specific job. Once selectServer(String[] j) has ended, it retrieves the server data to send the scheduling message to the Server. theAlgorithm() keeps running until there are no jobs left to schedule.

selectServer(String[] j): method that receives the job information in the form of an array of strings, then sends a request to the server for a list of servers able to run the job. If the list is not empty, it is sent to compareServers(String ds, int ln, String[] l), which picks the specific server that should run the job. If the list is empty, it sends a request to the server for a list of servers capable of running the job. The list is then is sent to compareServers(String ds, int ln, String[] l), which picks the specific server that should run the job.

compareServers(String ds, int ln, String[] l): method that receives a list of servers, and picks the server with the highest amount of cores. If there is more than one server with the highest number of cores, the first one is picked.

**Evaluation**

The new algorithm succeeds in optimising the average resource utilisation and surpassing in all the objectives Worst Fit. The algorithm was run using practice test script called “test\_results”.

The test simulation was run using the command:

* ./test\_results “java TheClient” -o ru -n -c ../../configs/other/

The following data is gathered from such test, and it represents the overall results of the pre-existing algorithms and the new one for all three objectives.

The order of results is All To Largest, First Fit, Best Fit, Worst Fit, New Algorithm.

Turnaround Time

Average |254086.33 |1473.33 |1462.83 |6240.72 |2638.61

Improvement: 13.74%

Resource Utilisation

Average |100.00 |66.79 |64.94 |72.85 |74.10

Improvement: 8.66%

Total Rental Cost

Average |256.05 |417.90 |414.42 |443.03 |434.96

Improvement: -2.32%

From these results, it is clear that the new algorithm outperforms Worst Fit in the three objectives. On the other hand, the Turnaround Time of the new algorithm is still much higher than First Fit and Best Fit. The same applies for the Total Rental Cost: All To Largest, First Fit and Best Fit are better, but the difference is not as wide as for the Turnaround Time.

What is surprising is that the Resource Utilisation for the new algorithm is actually the best out of all the scheduling algorithms, without taking in consideration All To Largest, as explained in the Problem Definition of this report.

**Conclusion**

The project was successful: the algorithm designed managed to outperform one of the existing scheduling algorithms, optimising Resource Utilisation as well.

However, it is clear from the test simulation that there is still margin for improvement, especially for Turnaround Time, but also for Total Rental Cost.

The overall outcome of the project is nonetheless satisfactory.

**References**

The GitHub repository URL that contains the algorithm:

https://github.com/edoardobusano/COMP3100\_Stage2