**Low-cost Scheduling Algorithm**

*# Student name: Le Thuy Hang Nguyen*

*# Student ID: 44499159*

*# Course: COMP3100 – Distributed System*

*# Github repository:* [*https://github.com/sotiey511/Low-cost-Scheduling-Algorithm*](https://github.com/sotiey511/Low-cost-Scheduling-Algorithm)

1. **Introduction**

In this stage, I have designed and implemented a new scheduling algorithm to optimise one of the most important performance objectives of distributed system - system’s execution cost. The algorithm’s goal is to minimize the number of servers used, utilization, and therefore, minimize the costs. However, the implementation of this algorithm will sacrifice the optimisation of waiting time and turnaround time.

1. **Simulation setup**

* Make sure you make downloaded the simulator source file from GitHub <https://github.com/sotiey511/Scheduling-Algorithm-with-cheap-total-cost>
* You need to check whether the following libraries are installed on your machine or not. You need the libraries before compiling and running the simulator code.

1. *libxml2*
2. *libxml2-dev*

To install the libraries, open a terminal and enter the following command (you will be asked to enter your password): sudo apt-get install libxml2 libxml2-dev

* After the installation became completed, go to the folder you have downloaded ds-sim. It is assumed that ‘Downloads’ is the directory for the file. cd Downloads
* Un-tar the file (you may need to check tar is installed on the system.) tar -xvf ds-sim
* Go to the folder ds-sim cd ds-sim
* To build the simulator you need to enter the following code. make
* You will have two files ds-server.c and FirstFit.java. To make them executable: chmod +x ds-server javac CostFit.java
* You have both ds-server and FirstFit files ready. To run the simulator, open two terminals
* Run the ds-server first: ./ds-server -c config\_simple5.xml -v all > log\_al.txt
* Then for the other terminal, run the FirstFit: java CostFit -a al

1. **Algorithm Description**

The algorithm attempts to exploit all servers have been booted up for utilization instead of using new ones. If a server has sufficient available resources to run a job and currently in Idle and Active state, it will be immediately chosen to run the job. In case, no severs are in Idle or Active state, or in Idle or Active state but do not have enough available resources to run the job, then choose servers with minimum available time. The implementation shows very positive results since the number of used servers and total execution cost are reduced significantly (please review section 3.2 for more details)

* 1. **Pseudo code**

For a given job ,

1. Set *minAvail* to very large number (e.g: INT\_MAX)
2. Obtain job state information
3. Send RESC Capable for information of all servers with sufficient initial resource to run job
4. Obtain server state information
5. For each server returned
6. If (*server state == 2 or server state == 3*)
7. And if *server has sufficient available resources* to run job then
8. Set scheduling message to server and return
9. Else if (*server state == 1 or server state == 4*) and (*serverAvail < minAvail*) then
10. Set scheduling message to server
11. Send scheduling message
    1. **Evaluation**

The table below shows comparison between my algorithm and other baseline algorithms.

*(All sample configurations are available at* [*https://github.com/sotiey511/Scheduling-Algorithm-with-cheap-total-cost/blob/master/s3sampleconfigs.tar*](https://github.com/sotiey511/Scheduling-Algorithm-with-cheap-total-cost/blob/master/s3sampleconfigs.tar)*)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Total cost** | **My algorithm** | **First fit** | **Best fit** | **Worst fit** |
| **Ds-config-1** | $9.39 | $13.86 | $13.86 | $15.99 |
| **Ds-config-2** | $2011.21 | $2241.44 | $2274.35 | $2728.55 |
| **Ds-config-3** | $26458.48 | $27643.44 | $27738.16 | $29933.08 |
| **Ds-config-4** | $138.24 | $572.01 | $568.64 | $598.86 |
| **Ds-config-5** | $94.92 | $6256.41 | $249.43 | $357.49 |
| **Ds-config-6** | $444.90 | $1013.62 | $1026.81 | $998.22 |
| **Ds-config-7** | $6256.41 | $16202.63 | $16082.36 | $20119.38 |

It is easy to see from the comparison table above that the new algorithm always has the lowest execution costs no matter what the configuration is.

However, despite the huge performance improvement in execution cost, as shown above, the turnaround time is getting huge. This means that it takes very long to process all job scheduling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Turnaround time** | **My algorithm** | **First fit** | **Best fit** | **Worst fit** |
| **Ds-config-1** | 14297 | 10245 | 10245 | 21180 |
| **Ds-config-2** | 3512181 | 1642392 | 1715644 | 2568687 |
| **Ds-config-3** | 8251270 | 8642647 | 8760321 | 11167544 |
| **Ds-config-4** | 110364 | 15 | 15 | 15 |
| **Ds-config-5** | 66689 | 5714 | 6162 | 6683 |
| **Ds-config-6** | 206944 | 10097 | 10328 | 16863 |
| **Ds-config-7** | 295835 | 8206 | 8512 | 38294 |

1. **Conclusion**

Even though the algorithm shows very positive results in attempt to develop a low-cost scheduling algorithm, I recommend further researches should be undertaken to balance between execution cost and turnaround execution time