**Cost Minimizing Algorithm**

45536155 Socheat Chhun

**Introduction**

This stage 2 assignment’s goal is to improve in one of the metrics compared to the three baseline algorithms which are Best Fit, First Fit, and Worst Fit. Our task in this stage 2 is to work on top of our stage 1 ds-client to improve on one or more metric which are minimizing of turnaround time, maximizing resource utilization, and minimizing the total cost.

**Problem Definition**

We three baseline algorithm are good for their own respective job. First fit, will find the first server that is capable and available for the job. Best Fit will go through all the servers that are capable, available, and best fit for the job. Finally, Worst Fit will do the same thing as Best Fit but instead of finding the best that fit the job it finds the worst. These baseline algorithms are aim for performance improvement. However, none of these baseline algorithms aim to reduce the rental cost of the server. The goal of my algorithm is to minimize the cost of the job scheduling. In short, it will find the smallest capable server that can do the job and will only schedule to that server. More description in Algorithm description.

**Algorithm Description**

From week 8 to week 12, during our workshop, we are giving the opportunity to understand the behavior of both ds-client and ds-server. I got to understand the difference between GETS Capable and GETS Avail. After understanding that there a huge different between the two options, I came up with an algorithm to use GETS Capable to request for the status of the server instead of GETS Capable.

Here are the steps:

* Client send a GETS Capable message to the server instead of GETS Avail.
* My client after getting the information of the capable machine to do the job, will schedule to the machine that is the smallest for doing the job. The reason for that is the cost of a small server compared to a large server is huge.
* When the next job is received, the client still then schedules to that same server if it is capable of doing the job. I did this reduce the idle time of the server. Hence reducing the cost.
* It will only schedule on to a new server unless the server that was initially used is not capable of doing the job, meaning the server does not have enough core.

**Example**

There are 100 jobs.

We have 15 servers in total:

* 5 tiny 1 core
* 5 small with 2 cores
* 5 mediums with 4 cores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MC | FF | WF | BF |
| Server Used | 3 | 13 | 7 | 13 |
| Cost | $2.54 | $5.61 | $5.54 | $5.61 |

I have attached a custom configuration file with the submission, and if you run that configuration file with my algorithm against the other three baselines. The result is as shown in the above table. As I am aiming to reduce the time, my algorithm only uses one of each of the type of server, resulting in only using 3 servers as compared to other baseline algorithms which uses 7 or more server.

**Implementation**

I started implementing the stage 2 of the assignment on top of the stage one source code. There are 4 main classes that are required to run this algorithm. They are Client, ServerObject, Job, and MinimizeCost. MinimizeCost is the new class that I added to handle the scheduling of jobs to the server. There are three main methods to this class.

 public void readPopulateServer(ArrayList<String> serverStatuses) {

        servers = new ArrayList<ServerObject>();

        for (String s : serverStatuses) {

            String[] sArray = s.split(" ");

            ServerObject sObj = new ServerObject(sArray);

            servers.add(sObj);

        }

    }

This method is to add serverObj from the server to the serverObject attribute of the class.

public ServerObject getSmallestServer() {

        ServerObject smallestServer = servers.get(servers.size() - 1);

        for (ServerObject s : servers) {

            // > mean smallest

            if (smallestServer.compareTo(s) > 0) {

                smallestServer = s;

            }

        }

        return smallestServer;

    }

This method is for finding the smallest server capable to do that job.

public String MinimizeCostAlgo(ArrayList<String> serverStatuses){

        readPopulateServer(serverStatuses);

        ServerObject  serverObject= getSmallestServer();

        return serverObject.type + " " + serverObject.id;

    }

This method is the method that return the server type and server id that is then passed back to the Client class.

The Client class will then call MinimizeCostAlgo on the object to get the string of server type and server id to send SCDH command on.

As you can see, I have designed my code to be modular. I decided to separate the server, job and algorithm in three separate classes and put everything together in the Client class. This will result in a more maintainable and readable code.

**Evaluation**

To evaluate my algorithm with the three-baseline algorithm and AllToLargest algorithm, I have drawn few visual graphics to aid the understanding. All these data are based on the test script provided to us by the unit convenor using the 18 configs file in week 11.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ATL | FF | BF | WF | MC |
| Turnaround | 254086ms | 1473ms | 1462ms | 6240ms | 193493ms |
| Utilization | 100% | 66.8% | 65% | 73% | 97.5% |
| Cost | $256 | $417 | $414 | $443 | $240 |

Figure1: The Table shows MC algorithms compared to the other 4 in three different metrices.

As you can see in figure 1 table, the minimize cost algorithm focus primary on the cost reduction metric. As a result, it does not take much of the turnaround metric into consideration. This led to a significant increase on the turnaround time of the jobs. This algorithm is best fit for when the user wants to reduce the cost of the servers regardless of how long it would take to complete all of the jobs.

Figure 2: The Chart shows MC algorithms compared to the other 4 in cost metric.

As you can see from the two graphs above, Cost Minimize Algorithm outperform all three baseline algorithms in the rental cost metric.

**Conclusion**

The three baseline algorithms are best for each of their own respective use case, while my minimize cost algorithm’s primary goal is the reduce the cost of the servers. What I have found about my algorithm is that it heavily focuses on one metric while increasing the other metric significantly. There are room for improvement. However, in the meantime, the algorithm does what it meant to do. What I could have done better is instead of only taking the current state of the servers, I could have taken into consideration the server information from the ds-system.xml file. This could slightly reduce the turnaround time.

In conclusion, my minimize cost algorithm focus primary on the reduce cost metric, and it does this by reducing the idle time of the server by only using one server of each capable job.

**References**

[1] https://github.com/chhunsocheat/comp3100-group47-assignment