Cloud Job Scheduler

Matthew Aliwarga (45422117)

Introduction

The focus of this project is to implement a scheduling algorithm that performs better than the baseline algorithms when scheduling jobs. The algorithm should fulfil one or more of the following objectives:

- Minimisation of average turnaround time
- Maximisation of average resource utilisation
- Minimisation of total server rental cost

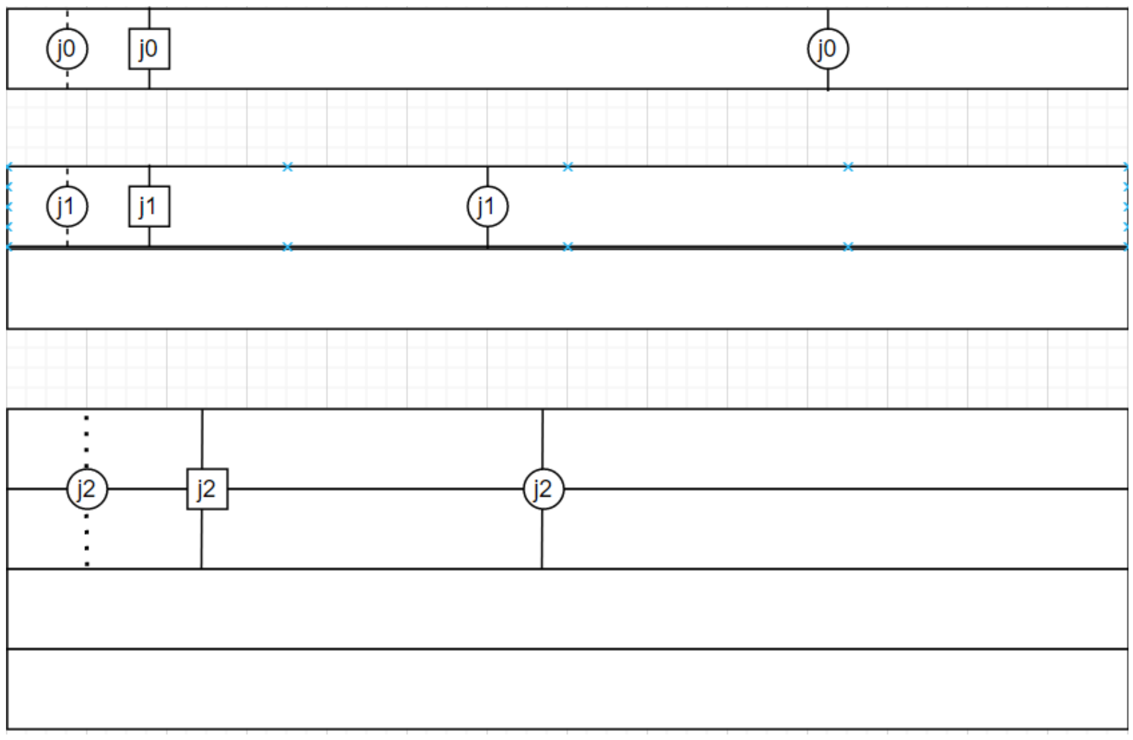
Problem Definition

The objectives stated above have conflicting performance objectives, and as such, this will require in the other objectives not being fulfilled. Minimizing the average turnaround time would reduce the amount of time between the job submission and the job completion as much as feasibly possible; however, if a server cannot take more requests, the job would be moved to another server, causing an increase in total server rental costs. Maximizing average resource utilization would ensure that each job would be allocated to each server based on what the server is currently capable of doing (whether it has enough memory to run the task, whether it has enough cores currently free at the moment, etc.); however, this may increase turnaround time by the amount of calculations an algorithm needs to do in order to accomplish the task of finding a compatible server. Minimizing total server rental costs would involve placing as many tasks on a server as possible, in order to avoid renting multiple servers and increasing costs; however, this would increase usage of resources on the servers that are running jobs, making it difficult to allocate jobs in the future by increasing average turnaround time, as jobs will be forced to be submitted on servers that cannot run them immediately.

The function chosen for this project prioritized maximizing the resource utilization and minimizing the total server rental cost.

Algorithm Description

The algorithm iterates through all of the given servers. For each server, it transmits an LSTJ command to the server to query the current running jobs on the server. It then calculates the maximum cores the server will be using at any given time for its running and remaining jobs by summing up all of the cores from all of the jobs returned from the query, but only if their runtimes overlap. An example is given below, with an example configuration. For the sake of simplicity, only cores are used as a metric to select the candidate in the example; however, cores, memory and disk are all considered by the algorithm.



Each bar above represents a different server core. Multiple bars attached together represent a server. A circle with a dotted line represents when the job is submitted. A square with a straight line represents the actual starting time of the job. A circle with a straight line through it represents the ending time for the job.

Example Scenario

Job j3 needs to be scheduled. It requires 2 cores. The algorithm will iterate through the servers to select its candidate and eliminate the top server in the diagram with 1 core due to the fact that it does not have enough cores to run the job. Since the middle server can support the job but requires that j1 be finished before it starts the job and the bottom server has 2 cores free to run the job before the middle server can, it will choose the bottom server.

Implementation Details

class Job

Used to store information about a job.

private String type; private int submitTime; private int jobID; private int estRunTime; private int core; private int memory; private int disk;

Represents the different fields of data a Job can have when received from the server.

public Job(String state);

Constructs a Job by parsing the given String state. This will parse the given string by splitting it by empty spaces. Each element is then assigned to its appropriate member variable – if it is a number, it is parsed first with Java’s Integer.parseInt(); function.

public int getJobID(); public String getType(); public int getCoreUsage(); public int getMemoryUsage(); public int getDiskUsage();

Getters for the private fields above.

class Server
Used to store information about a server.

public static final String STATE_ACTIVE = “active”; public static final String STATE_INACTIVE = “inactive”; public static final String STATE_IDLE = “idle”;
Constants representing the possible states a server can be in.

private String serverType; private int serverID; private String state; private int curStartTime; private int cores; private int memory; private disk; private int wJobs; private int rJobs;
Represents the different fields of data a Server can have when such information is queried.

public Server(String state);
Constructs a Server by parsing the given String state. This will parse the given string by splitting it by empty spaces. Each element is then assigned to its appropriate member variable – if it is a number, it is parsed first with Java’s Integer.parseInt(); function.

public String getServerType(); public int getServerID(); public int getState(); public int getNumberOfCores(); public int getMemoryUsage(); public int getDiskUsage();
Getters for the private fields above.

public void printServer();
Prints server data to the console (debug)

public static Tuple selectServer(Job current, ArrayList<Server> servers, DataInputStream din, DataInputStream dout);
Selects a suitable server as a candidate to be chosen to run the job. This function is now where the algorithm of this assessment is implemented.

Evaluation

Configurations
Configurations used can be found in ds-sim/configs/other/.

Results
Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	672786
config100-long-low.xml	316359	2458	2458	2613	316359
config100-long-med.xml	679829	2356	2362	10244	679829
config100-med-high.xml	331382	1184	1198	12882	331382
config100-med-low.xml	283701	1205	1205	1245	283701
config100-med-med.xml	342754	1153	1154	4387	342754
config100-short-high.xml	244404	693	670	10424	244404
config100-short-low.xml	224174	673	673	746	224174
config100-short-med.xml	256797	645	644	5197	256797
config20-long-high.xml	240984	2852	2820	10768	240984
config20-long-low.xml	55746	2493	2494	2523	55746
config20-long-med.xml	139467	2491	2485	2803	139467
config20-med-high.xml	247673	1393	1254	8743	247673
config20-med-low.xml	52096	1209	1209	1230	52096
config20-med-med.xml	139670	1205	1205	1829	139670

```
config20-short-high.xml     |145298    |768       |736       |5403      |145298
config20-short-low.xml      |49299     |665       |665       |704       |49299
config20-short-med.xml      |151135    |649       |649       |878       |151135
Average                 |254086.33 |1473.33   |1462.83   |6240.72   |254086.33
Normalised (ATL)         |1.0000    |0.0058    |0.0058    |0.0246    |1.0000
Normalised (FF)         |172.4568  |1.0000    |0.9929    |4.2358    |172.4568
Normalised (BF)         |173.6947  |1.0072    |1.0000    |4.2662    |173.6947
Normalised (WF)          |40.7143   |0.2361    |0.2344    |1.0000    |40.7143
Normalised (AVG [FF,BF,WF]) |83.0629   |0.4816    |0.4782    |2.0401    |83.0629

Resource utilisation
Config                        |ATL      |FF       |BF       |WF       |Yours
config100-long-high.xml     |100.0     |83.58     |79.03     |80.99     |100.0
config100-long-low.xml      |100.0     |50.47     |47.52     |76.88     |100.0
config100-long-med.xml      |100.0     |62.86     |60.25     |77.45     |100.0
config100-med-high.xml      |100.0     |83.88     |80.64     |89.53     |100.0
config100-med-low.xml       |100.0     |40.14     |38.35     |76.37     |100.0
config100-med-med.xml       |100.0     |65.69     |61.75     |81.74     |100.0
config100-short-high.xml    |100.0     |87.78     |85.7      |94.69     |100.0
config100-short-low.xml     |100.0     |35.46     |37.88     |75.65     |100.0
config100-short-med.xml     |100.0     |67.78     |66.72     |78.12     |100.0
config20-long-high.xml      |100.0     |91.0      |88.97     |66.89     |100.0
config20-long-low.xml       |100.0     |55.78     |56.72     |69.98     |100.0
config20-long-med.xml       |100.0     |75.4      |73.11     |78.18     |100.0
config20-med-high.xml       |100.0     |88.91     |86.63     |62.53     |100.0
config20-med-low.xml        |100.0     |46.99     |46.3      |57.27     |100.0
config20-med-med.xml        |100.0     |68.91     |66.64     |65.38     |100.0
config20-short-high.xml     |100.0     |89.53     |87.6      |61.97     |100.0
config20-short-low.xml      |100.0     |38.77     |38.57     |52.52     |100.0
config20-short-med.xml      |100.0     |69.26     |66.58     |65.21     |100.0
Average                 |100.00    |66.79     |64.94     |72.85     |100.00
Normalised (ATL)         |1.0000    |0.6679    |0.6494    |0.7285    |1.0000
Normalised (FF)         |1.4973    |1.0000    |0.9724    |1.0908    |1.4973
Normalised (BF)         |1.5398    |1.0284    |1.0000    |1.1218    |1.5398
Normalised (WF)          |1.3726    |0.9168    |0.8914    |1.0000    |1.3726
Normalised (AVG [FF,BF,WF]) |1.4664    |0.9794    |0.9523    |1.0683    |1.4664

Total rental cost
Config                  |ATL      |FF       |BF       |WF       |Yours
config100-long-high.xml     |620.01    |776.34    |784.3     |886.06    |620.01
config100-long-low.xml      |324.81    |724.66    |713.42    |882.02    |324.81
config100-long-med.xml      |625.5     |1095.22   |1099.21   |1097.78   |625.5
config100-med-high.xml      |319.7     |373.0     |371.74    |410.09    |319.7
config100-med-low.xml       |295.86    |810.53    |778.18    |815.88    |295.86
config100-med-med.xml       |308.7     |493.64    |510.13    |498.65    |308.7
config100-short-high.xml    |228.75    |213.1     |210.25    |245.96    |228.75
config100-short-low.xml     |225.85    |498.18    |474.11    |533.92    |225.85
config100-short-med.xml     |228.07    |275.9     |272.29    |310.88    |228.07
config20-long-high.xml      |254.81    |306.43    |307.37    |351.72    |254.81
config20-long-low.xml       |88.06     |208.94    |211.23    |203.32    |88.06
config20-long-med.xml       |167.04    |281.35    |283.34    |250.3     |167.04
```

config20-med-high.xml	255.58	299.93	297.11	342.98	255.58
config20-med-low.xml	86.62	232.07	232.08	210.08	86.62
config20-med-med.xml	164.01	295.13	276.4	267.84	164.01
config20-short-high.xml	163.69	168.7	168.0	203.66	163.69
config20-short-low.xml	85.52	214.16	212.71	231.67	85.52
config20-short-med.xml	166.24	254.85	257.62	231.69	166.24
Average	256.05	417.90	414.42	443.03	256.05
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.0000
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.6127
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.6178
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.5779
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.6023

Comparisons

The new algorithm performed poorly in minimizing turnaround time, with times often exceeding 40 times the turnaround time than any of the times given by running First Fit, Best Fit or Worst Fit.

Pros and Cons

The algorithm pays no mind to the turnaround time of a job; however, it maximizes Resource Efficiency, and, in most cases, minimizes the Total Rental Cost.

Conclusion

More steps could be taken to minimize the Total Rental Cost in order for this algorithm to more closely complete its objective.

References

Github: https://github.com/Shadoweagle7/COMP3100-Project-Stage2