COMP3100 - Stage 2 Report
Scheduling Algorithm Optimisation for Cloud-Based Job Scheduler

Christopher Stedman (45609888)

1. Introduction

Stage one of the distributed job scheduler project saw the development of a client which would connect with the supplied server (ds-server) and act as a job scheduler, receiving jobs from the server and allocating them to a specific server within the distributed system. The algorithm implemented as part of stage one was named ‘all-to-largest’ (ATL), and simply allocated all jobs to the largest server regardless of job requirements or scheduling metrics.

The goal of stage 2 is to develop a scheduling algorithm which improves upon the rudimentary ‘all to largest’ algorithm. An algorithm is only deemed successful if one of the metrics (rental cost, job turnaround time, resource utilisation) surpasses at least one of the provided baseline algorithms ‘first-fit’ (FF), ‘best-fit’ (BF) or ‘worst-fit’ (WF) as well as the ‘all-to-largest’ algorithm from stage one, without severely impacting the other metrics.

This report explains the design decisions faced when developing this algorithm and explores how it is an improvement over the previously mentioned algorithms.

2. Problem Definition

A job scheduling algorithm’s success can be measured by examining its performance in the three metrics, turnaround time, rental cost and resource utilisation. An optimal solution is an algorithm which is able to maximise resource utilisation while minimising rental costs and turnaround times. However, these metrics are at odds, with advancements in one metric resulting in the decline of another. A clear example of this can be seen by examining the ‘all-to-largest’ algorithm. This simple algorithm functions by scheduling all incoming jobs to the largest server in this distributed system. While the utilisation of only one server results in low rental costs and high utilisation, it creates a long queue of jobs on the server which leads to excessive turnaround time.

The algorithm developed for stage 2 of this project successfully minimises server rental costs and maximises resource utilisation compared to the three baseline algorithms, while limiting the negative impact on turnaround time. This offers an ideal compromise between the three metrics, allowing organisations who implement this algorithm to significantly reduce their server costs without severely impacting performance.

To achieve this, active and idle servers with available resources are targeted first. If none are available, the algorithm then roughly estimates the wait time of the cheapest severs and determines if new servers should be powered on or if jobs should be queued.

3. Cost Reduce Algorithm Description

The cost reduce algorithm was developed to maximise resource utilisation and reduce rental costs without significantly impacting turnaround time. To achieve this, the algorithm is composed of two parts:

1. Identify idling or active servers with available resources
2. Identify a cost-effective server with minimal waiting time. If not possible, boot a new server.

First, the client sends a GETS Avail request to the ds-server and uses the received server information to identify idling or active servers with no waiting jobs. This aims to minimise server idle time to maximise utilisation, while also minimising wait time. If a suitable server is identified, it is immediately scheduled.

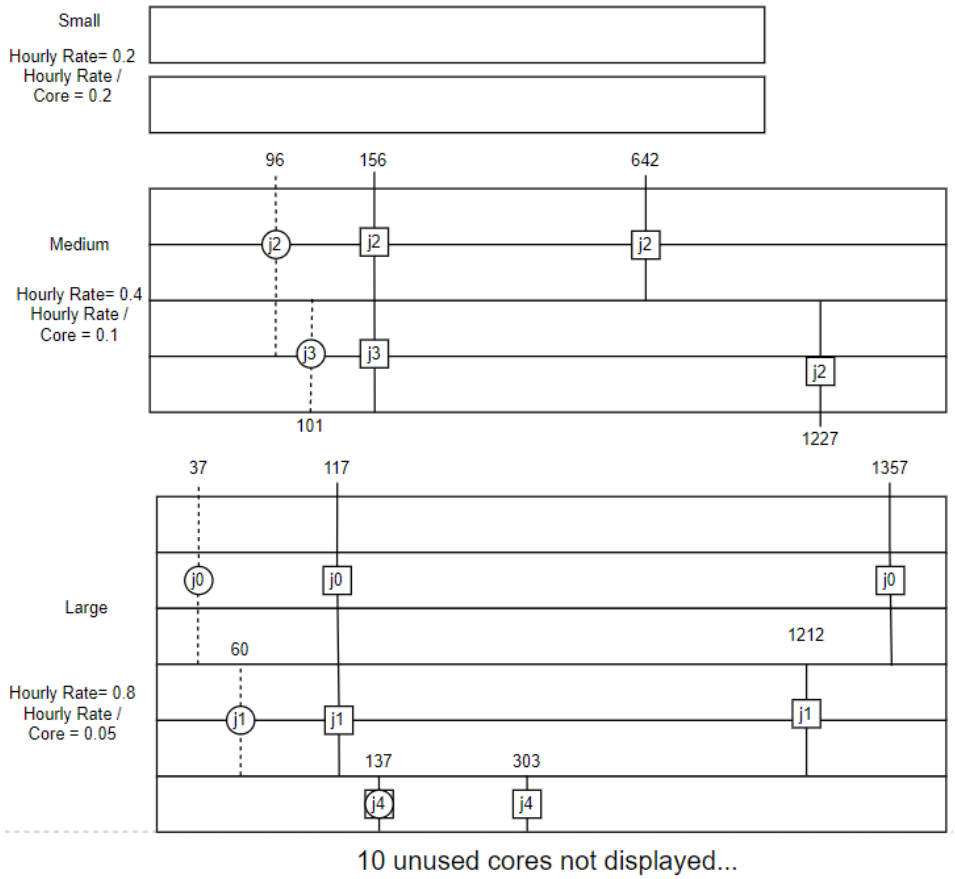
However, if a server with available resource cannot be found, the second part of the algorithm is initiated. A GETS Capable request is sent to the ds-server, containing all servers with hardware capable of handling the job. Each of these servers is then checked to see if they are in an active or booting state, as

this means they are powered on (if state is inactive, see below). As this algorithm aims to minimise cost, the cost per core of each server is checked against the current cheapest, ensuring that only the cheapest servers are used whenever possible. To identify the ideal server for a job, an EJWT request is sent to the ds-server to retrieve an estimated wait time, so that the server with the shortest wait time is identified.

In the event that a server state is inactive, the cost of running the server is compared to the current cheapest option to ensure that only the most cost-efficient servers are used.

Once every server has been examined, an algorithm determines if the job should be allocated to the server with the shortest wait time, or if a new server should be powered on, if one is available. This algorithm checks if the estimated wait time for the powered server is greater that the estimated run time of the job multiplied by a multiplier (currently set to 4.0 as this offered the best trade-off between turnaround time and cost). If the wait time is longer, then the most cost- effective new server is powered on for use, otherwise the job is added to the queue of the already powered server.

3.1. Scenario



Configuration – Jobs

ID	SubmitTime	Estimated Run Time	Cores	Memory	Disk
0	37	653	3	700	3800
1	60	2025	2	1500	2900
2	96	343	2	1500	2100
3	101	380	2	900	2500
4	137	111	1	100	2000

Configuration – Server types

Type	Limit	HourlyRate	BootupTime	CoreCount	Memory	Disk
Small	1	0.2	60	1600	4000	16000
Medium	2	0.4	60	4	16000	64000
Large	1	0.8	80	16	64000	512000

Scenario Explanation

In the above scenario, the cost reduce algorithm starts by allocating job 0 to the largest server, as this is the most cost-effective option available. Job 1 is also allocated to this server as the estimated wait time of the server is less than the jobs estimated running time multiplied by the defined multiplier (4). Job 2 and job 3 are then allocated to the medium server even though it is more expensive, as the wait time of the larger server exceeds their estimated running time multiplied by the multiplier. Job 4 is then allocated to the large server, as the server has available resources to handle it.

4. Implementation

Implementing this algorithm required some modifications to the stage 1 code, as well as the creation of a Scheduler class to modularise the scheduling functionality of the client.

ServerTypes class

- The variable 'hourlyRatePerCore' of type double was added to the server type data structure so the cost per core of each server could be quickly identified by the algorithm.

Client class

- getCapableServers() and getAvailableServers() functions were implemented so the required server information could be retrieved from ds-server. Server objects are created form the data and are stored in ArrayList<Server> objects in order, so the scheduling algorithm can iterate through each server from smallest to largest effectively.
- getServerList() function modified so server types parsed from ds-system.xml are stored in a HashMap<String, ArrayList>, with the name of the server acting as key. This allows the scheduling algorithm to quickly identify server information, such as hourly cost per core.

Scheduler class

When client runs, a new Scheduler object is created, with server type information and the client passed to the constructor (detailed below).

When a scheduling decision has to be made, the Client passes the job to the costReduceAlgorithm() function in the constructor. This algorithm first retrieves available server data from ds-server and tries to identify an 'active' or 'idle' server with no waiting jobs.

Failing this, the costReduceAlgorithm() retrieves capable server data from ds-server and identifies a cost-effective active server with minimal wait time and a cost-effective inactive server. It then determines if the job should be allocated to an inactive server or if it should be added to the queue of the already active server.

- HashMap<ServerTypes> passed to Scheduler constructor and defined. Allows scheduling algorithm to quickly identify server information
- Client passed to Scheduler constructor and defined. Allows the scheduling algorithm to access variables and functions within the Client.
- Multiplier variable was implemented as it allows a developer to tweak the performance of the scheduling algorithm. Setting the number higher will result in more jobs being added to a queue of an active server, reducing costs and increasing turnaround times, while a lower number will result in more servers being powered on, increasing costs and improving turnaround times.

For a more detailed explanation of the cost reduce algorithm, please see section 3. Cost Reduce Algorithm Description

5. Evaluation

Simulation setup

The simulation consists of 18 test cases which test different server and job configurations to evaluate different aspects of the scheduler's performance.

To run the simulation, execute *./test_results "java Client" -o co -n -c ../../configs/other/*

Results - Turnaround Time

Turnaround Time was not the primary focus of the cost reduce algorithm, so although it marginally fails to outperform the FF and BF algorithms in this area, it offers a significant improvement over the WF and ATL algorithms. This improvement is due to the initial strategy of identifying available servers, followed by the algorithm prioritising low wait servers.

Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	3218
config100-long-low.xml	316359	2458	2458	2613	2632
config100-long-med.xml	679829	2356	2362	10244	2624
config100-med-high.xml	331382	1184	1198	12882	1828
config100-med-low.xml	283701	1205	1205	1245	1286
config100-med-med.xml	342754	1153	1154	4387	1364
config100-short-high.xml	244404	693	670	10424	1447
config100-short-low.xml	224174	673	673	746	701
config100-short-med.xml	256797	645	644	5197	904
config20-long-high.xml	240984	2852	2820	10768	2991
config20-long-low.xml	55746	2493	2494	2523	2588
config20-long-med.xml	139467	2491	2485	2803	2517
config20-med-high.xml	247673	1393	1254	8743	1529
config20-med-low.xml	52096	1209	1209	1230	1263
config20-med-med.xml	139670	1205	1205	1829	1197
config20-short-high.xml	145298	768	736	5403	1531
config20-short-low.xml	49299	665	665	704	666
config20-short-med.xml	151135	649	649	878	689
Average	**254086.33**	**1473.33**	**1462.83**	**6240.72**	**1720.83**
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0068
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	1.1680
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	1.1764
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	0.2757
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	0.5626

Results - Resource Utilisation

Resource utilisation is closely linked to cost-effective job allocation, and these results clearly show that the cost reduce algorithm outperforms the three baseline algorithms in most situations. This is due to the initial strategy of selecting servers which are idling or have available resources, ensuring that servers spend little time running no tasks. Additionally, the algorithm focuses on allocating jobs to servers with lower waiting times, minimising the chance of servers idling.

Resource utilisation

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	100.0	83.58	79.03	80.99	99.01
config100-long-low.xml	100.0	50.47	47.52	76.88	79.13
config100-long-med.xml	100.0	62.86	60.25	77.45	92.29
config100-med-high.xml	100.0	83.88	80.64	89.53	99.24
config100-med-low.xml	100.0	40.14	38.35	76.37	75.83
config100-med-med.xml	100.0	65.69	61.75	81.74	96.73
config100-short-high.xml	100.0	87.78	85.7	94.69	98.15
config100-short-low.xml	100.0	35.46	37.88	75.65	62.24
config100-short-med.xml	100.0	67.78	66.72	78.12	96.4
config20-long-high.xml	100.0	91.0	88.97	66.89	98.49
config20-long-low.xml	100.0	55.78	56.72	69.98	90.51
config20-long-med.xml	100.0	75.4	73.11	78.18	91.81
config20-med-high.xml	100.0	88.91	86.63	62.53	91.02
config20-med-low.xml	100.0	46.99	46.3	57.27	89.1
config20-med-med.xml	100.0	68.91	66.64	65.38	82.66
config20-short-high.xml	100.0	89.53	87.6	61.97	96.1
config20-short-low.xml	100.0	38.77	38.57	52.52	61.53
config20-short-med.xml	100.0	69.26	66.58	65.21	83.83
Average	**100.00**	**66.79**	**64.94**	**72.85**	**88.00**
Normalised (ATL)	1.0000	0.6679	0.6494	0.7285	0.8800
Normalised (FF)	1.4973	1.0000	0.9724	1.0908	1.3177
Normalised (BF)	1.5398	1.0284	1.0000	1.1218	1.3551
Normalised (WF)	1.3726	0.9168	0.8914	1.0000	1.2080
Normalised (AVG [FF,BF,WF])	1.4664	0.9794	0.9523	1.0683	1.2905

Results – Rental Cost

The result below show that the cost reduce algorithm significantly outperforms the three baseline algorithms in almost all situations. This was achieved by maximising resource utilisation as described above to minimise idle time, and by comparing servers based upon the hourly cost per core. This method of comparing servers allowed for cost-effective decisions to be made for every job, resulting primarily in the widespread utilisation of large servers for both small and large jobs.

Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	661.82
config100-long-low.xml	324.81	724.66	713.42	882.02	484.43
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	751.08
config100-med-high.xml	319.7	373.0	371.74	410.09	343.84
config100-med-low.xml	295.86	810.53	778.18	815.88	476.27
config100-med-med.xml	308.7	493.64	510.13	498.65	351.29
config100-short-high.xml	228.75	213.1	210.25	245.96	219.98
config100-short-low.xml	225.85	498.18	474.11	533.92	372.31
config100-short-med.xml	228.07	275.9	272.29	310.88	235.98
config20-long-high.xml	254.81	306.43	307.37	351.72	274.66
config20-long-low.xml	88.06	208.94	211.23	203.32	120.03
config20-long-med.xml	167.04	281.35	283.34	250.3	214.38
config20-med-high.xml	255.58	299.93	297.11	342.98	291.99
config20-med-low.xml	86.62	232.07	232.08	210.08	117.01
config20-med-med.xml	164.01	295.13	276.4	267.84	231.28
config20-short-high.xml	163.69	168.7	168.0	203.66	168.67
config20-short-low.xml	85.52	214.16	212.71	231.67	156.37
config20-short-med.xml	166.24	254.85	257.62	231.69	207.93
Average	**256.05**	**417.90**	**414.42**	**443.03**	**315.52**
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.2323
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.7550
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.7614
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.7122
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.7422

Pros & Cons Comparison

Overall, the reduce cost algorithms manages to minimise costs by maximising resource utilisation and applying a cost-based comparison strategy to job allocation decisions. While it rarely manages to outperform the turnaround times of the baseline algorithms, the savings that can be achieved through the algorithm make it an excellent choice for job schedulers who are willing to suffer a small performance decrease .

6. Conclusion

Creating an algorithm which performs well in all three metrics is difficult as these metrics are at odds with one another. This means that developers of scheduling algorithms must determine the priorities of the scheduler based on the application and focus on optimising a particular metric and compromising others. The cost reduce algorithm developed as part of stage 2 of this project manages to lower rental costs and improve server utilisation, while only slightly increasing turnaround times compares to the three baseline algorithms. To provide developers additional control over the algorithm, the multiplier variable allows for performance to be adjusted based on requirements, which is a particularly useful feature.

To improve this algorithm further, more accurate estimations could be made about waiting times by including the remaining time of running jobs in the calculation. Additionally, the requirements of the waiting jobs queued on booting servers could be further detailed to allow server resources to be utilised more effectively.

7. References

GitHub https://github.com/ChrisStedman/COMP3100_Stage-2_ds-client