Project report: Stage 2
COMP3100 Distributed Systems, S1, 2023
SID: 46276688, Name: Le Minh Pham

1 Introduction

After stage 1 of the project, we learned that scheduling algorithms are an integral part of distributed systems due to their direct effect on such systems. However, there is no universal best-fit scheduling policy that can guarantee an optimal solution for every distributed system since the best scheduling is dependent on the distribution of jobs: submit time, run time, resource requirement, etc. Furthermore, scheduling has a number of conflicting performance metrics such as execution time, turnaround time, resource utilization, and rental cost. To optimize one is to sacrifice another. Therefore, defining the best scheduling algorithm is situational as it relies heavily on the system, objectives, and constraints.

In stage 2 of the project, I aim to design and implement one or more scheduling algorithms that can reduce the average turnaround time while not giving up too much resource utilization and server rental cost. Similar to stage 1, the schedulers are also implemented on the client side of ds-sim. The scheduling algorithms implemented are Least-Waiting-Time (LWT), Improved-First-Fit (IFF), Improved-Best-Fit (IBF), and Improved-Worst-Fit (IWF).

In this report, I discuss the problem definition, algorithm description, implementation, and evaluation in Section 2, Section 3, Section 4, and Section 5 respectively. The repository for this project can be accessed via the following link: https://github.com/LeMinhPham/COMP-3100-S1-2023-Assignment

2 Problem Definition

In this problem, the new schedulers need to outperform five baseline algorithms implemented in ds-client (FF, BF, FFQ, BFQ, and WFQ) in terms of turnaround time, i.e. achieve lower turnaround time, while not falling too far behind in resource utilization and rental cost. Since jobs' execution time is fixed regardless of available resources, the only way to reduce turnaround time is to minimize waiting time. Furthermore, the new algorithms work solely as a job scheduler, which means there will be no job migration or preemption, thus turnaround time has to be optimized at the time jobs are scheduled.

With all objectives and constraints considered, the objective functions are defined as follows:

- To ensure minimized average waiting time, the new scheduling algorithms have to make scheduling decisions in such a way that the jobs get the least amount of waiting time possible when it is scheduled to a server. (1)
- To ensure resource utilization and rental cost, the new scheduling algorithms need to prioritize scheduling jobs to servers in the following order according to their state: idle, active, booting, and inactive. Prioritizing booted-up servers will maximize resource utilization, and given the turnaround time is minimized, the rental cost will also be minimized. (2)

3 Algorithm Description

3.1 Least Wating Time

3.1.1 Description

This algorithm takes a naive approach and schedules jobs to the server with the least estimated amount of waiting time. Unlike the default **EJWT** command in ds-sim, LWT can estimate the remaining run time of running jobs and take into account the possibility of parallel execution of waiting jobs. To estimate waiting time, LWT employs a discrete-event simulation similar to ds-sim. Furthermore, LWT also tries to satisfy the objective function (2) by prioritizing idle and active servers to schedule jobs to and only scheduling jobs to inactive servers when it cannot find any server that can offer an amount waiting time less than the job's estimated run time.

The following is a pseudo-code of LWT:

```
Input:   a job j and a set of servers, S in ascending order of
         their resource capacity based on core count.
Output:  a server for j to be scheduled on.
Process: search for a server s*, from the first one to the last
         one in S, that satisfies the following conditions:
         1. The estimated waiting time of j when scheduled to S
            is smaller than its estimated run time.
         2. The estimated waiting time is the smallest

         If no s* found, select the first inactive server in S.
         If there is no inactive server, select the first server in S.
```

3.1.2 Sample scheduling scenario

Consider the following scheduling scenario with the following jobs:

jobID=0	submitTime=32	estRunTime=60	core=1	memory=900	disk=400
jobID=1	submitTime=55	estRunTime=20	core=2	memory=2100	disk=1200
jobID=2	submitTime=69	estRunTime=50	core=2	memory=2500	disk=1100

And the following servers:

type=small	core=2	memory=4000	disk=16000
type=medium	core=4	memory=16000	disk=64000
type=large	core=8	memory=32000	disk=256000

To simplify the calculating process, we will assume that the bootup time of the servers is 0. Firstly, job 0 will be scheduled to the small server since there is no active server.

small receives job 0 at time=32

When job 1 arrives at time=55, the remaining run time of job 0 is 37. If job 1 is scheduled to the small server, its waiting time will be 37, which is longer than its estimated run time of 20. Therefore, job 1 is scheduled to the first inactive server, which is the medium server.

medium receives job 1 at time=55

Lastly, job 2 arrives at time=69, when the remaining run time of jobs 0 and 1 is 23 and 6 respectively. If job 2 is scheduled to the small server, the waiting time will be 23. However, if it is scheduled to the medium server, the waiting time would be 0 as this server has enough resources to execute the job immediately. Therefore, job 2 is scheduled to the medium server.

medium receives job 2 at time=69

3.2 Improved First Fit, Improved Best Fit, and Improved Worst Fit

3.2.1 Description

As their names suggest, these algorithms are improved versions of their counterparts implemented by ds-client. After replicating the baseline algorithms, I realize that they have a major flaw that significantly increases the turnaround time: they are not aware that backfilling is not allowed in ds-sim. As a result, these baseline algorithms may get the illusion that a server is readily available to execute the job immediately while in fact, it may have some waiting jobs in the queue. When large and long jobs keep getting placed into the same waiting queue, the waiting time increases substantially. Therefore, the turnaround time of these algorithms can be optimized by making them ignore servers that have both running and waiting jobs.

It is clear that except for Improved Worst Fit, these algorithms satisfy both objective functions (1) and (2).

The pseudo-codes for these algorithms are identical to the pseudo-codes provided in Workshop 9.

3.2.2 Sample scheduling scenario

Consider the following scheduling scenario with the following jobs:

jobID=0	submitTime=32	estRunTime=60	core=1	memory=900	disk=400
jobID=1	submitTime=55	estRunTime=20	core=2	memory=2100	disk=1200
jobID=2	submitTime=69	estRunTime=100	core=2	memory=2050	disk=1500
jobID=3	submitTime=70	estRunTime=50	core=1	memory=500	disk=900

And the following servers:

type=small0	core=2	memory=4000	disk=16000
type=small1	core=2	memory=4000	disk=16000
type=medium	core=4	memory=16000	disk=64000

The scheduling results will be as follows:

IFF	IBF	IWF
small0 receives job 0 at time=32	small0 receives job 0 at time=32	medium receives job 0 at time=32
small1 receives job 1 at time=55	small1 receives job 1 at time=55	medium receives job 1 at time=55
medium receives job 2 at time=69	medium receives job 2 at time=69	small0 receives job 2 at time=69
small0 receives job 3 at time=70	small0 receives job 3 at time=70	small1 receives job 3 at time=70

4 Implementation

Different from stage 1, my client-side implementation in stage 2 has employed the Strategy Pattern to allow more flexibility in integrating more scheduling algorithms. Each scheduler is a separate class that implements the Scheduler interface. While the implementation details may differ, all algorithms employ the same pattern of using class variables to store necessary information about the next schedule, so that they do not need to be passed around helper methods. The class variables are as follows:

- Current job information: includes jobID, submit time (only LWT), estimated run time (only LWT), and required resources.
- List of capable servers: because the **LSTJ** command cannot be called while reading server outputs from the **GETS Capable** command, we need to store it for later check.
- List of server types information: each server type's initial resources are stored in this list for accurate calculation.
- Chosen server: the most suitable server to schedule the job to.

All algorithms follow the same structured and comprehensive job scheduling loop as shown in the following pseudo-code:

```
while having more jobs, do
    store job information
    read and store capable servers;
    if(not have initial resources of any server type)
        store initial resources;
    helper_method;
    schedule job to the server;

helper_method:
    read job information of capable servers;
    determine the most suitable server based on the algorithm;
```

5 Evaluation

To evaluate the performance of the algorithms, we will use the sample test suit provided on iLearn. The results are summarized in the following tables

Turnaround time									
Config	FF	BF	FFQ	BFQ	WFQ	IFF	IBF	IWF	LWT
config12-long-med	2400	2397	2802	2630	6880	2400	2397	3678	3145
config12-med-alt	388	373	813	698	3804	367	362	2030	2970
config12-med-med	654	653	893	831	4576	653	653	801	819
config12-short-med	61	60	106	100	677	60	60	74	73
config16-long-high	3123	4674	6536	6666	23972	2664	2639	21398	29883
config16-long-med	2548	2562	3997	3778	19538	2550	2566	6648	3360
config16-med-high	3149	5328	6123	5494	23740	1414	1467	22991	30026
config16-short-high	3215	8260	5517	4369	24814	995	893	26837	33328
config16-short-med	699	851	1952	1972	17139	665	660	7345	4460
config40-long-high	4506	4164	3642	3568	8461	3303	3287	15299	73688
config40-long-med	3553	3553	3553	3553	5928	3553	3553	5229	4457
config40-med-high	1505	896	940	933	3097	898	896	6344	33313
config40-med-med	941	941	941	941	1988	941	941	1983	2408
config40-short-high	485	1365	373	301	2817	228	228	7521	31841
config40-short-med	180	182	198	193	1945	180	180	3222	9294
Average	1867.1	2417.1	2559.1	2401.8	9958.4	1391.4	1385.5	8760.0	17537.7

Resource utilization									
Config	FF	BF	FFQ	BFQ	WFQ	IFF	IBF	IWF	LWT
config12-long-med	69.39	66.87	68.24	66.45	79.22	66.39	66.87	75.16	78.01
config12-med-alt	66.93	63.88	66.46	63.39	49.55	66.91	63.86	55.71	71.12
config12-med-med	66.22	63.06	65.92	62.68	71.52	66.22	63.08	64.96	74.18
config12-short-med	61.56	58.46	61.32	58.18	61.48	61.59	58.45	56.57	68.14
config16-long-high	79.97	74.70	77.70	74.06	66.17	79.80	76.51	67.68	80.56
config16-long-med	68.16	64.71	67.59	63.86	65.06	68.29	64.81	63.11	72.55
config16-med-high	77.45	73.06	76.10	73.14	48.18	78.61	75.81	56.82	79.33
config16-short-high	76.88	71.21	74.44	70.87	50.94	78.48	74.20	49.38	76.76
config16-short-med	66.22	62.42	65.54	62.08	61.40	66.18	62.80	54.15	70.75
config40-long-high	85.81	80.91	86.34	81.81	79.46	86.99	81.30	78.10	88.22
config40-long-med	72.16	67.37	72.16	67.37	76.71	72.16	67.37	83.82	75.93
config40-med-high	83.01	79.51	83.87	79.32	67.73	84.00	79.51	74.30	83.30
config40-med-med	72.77	65.83	72.77	65.83	79.25	72.77	65.83	83.82	71.45
config40-short-high	86.54	79.64	86.68	80.98	78.09	87.18	81.22	67.64	87.72
config40-short-med	77.23	71.66	77.18	71.68	87.92	77.23	71.74	76.20	83.38
Average	74.02	69.55	73.49	69.45	68.18	74.39	70.22	66.96	77.43

Total rental cost									
Config	FF	BF	FFQ	BFQ	WFQ	IFF	IBF	IWF	LWT
config12-long-med	131.4	131.8	136.0	133.2	132.0	131.4	131.8	127.5	132.0
config12-med-alt	113.3	113.2	115.8	115.2	118.8	113.3	113.2	111.6	119.8
config12-med-med	132.8	133.0	134.9	134.6	129.9	132.8	133.0	121.6	133.7
config12-short-med	21.5	21.5	21.7	21.7	21.0	21.5	21.5	20.3	21.5
config16-long-high	589.2	596.1	632.2	632.2	653.3	583.5	582.8	596.1	684.7
config16-long-med	581.9	581.5	600.4	599.3	580.8	581.9	579.8	542.7	592.6
config16-med-high	598.4	598.8	632.2	617.5	696.2	578.7	578.5	635.3	670.8
config16-short-high	594.1	612.4	635.5	626.1	674.3	580.5	579.1	667.0	685.4
config16-short-med	579.4	578.6	592.5	590.3	587.3	579.5	578.6	548.3	594.6
config40-long-high	1244.1	1236.9	1250.4	1239.6	1297.0	1228.5	1231.0	1338.6	1570.7
config40-long-med	1025.6	967.5	1025.6	967.5	1038.2	1025.6	967.5	997.5	1193.7
config40-med-high	614.4	602.1	605.5	605.9	657.5	599.5	602.1	649.7	755.8
config40-med-med	482.7	498.5	482.7	498.5	519.2	482.7	498.5	496.0	612.3
config40-short-high	600.4	612.8	601.8	598.0	655.3	592.6	591.9	657.3	775.1
config40-short-med	586.9	587.6	592.5	588.6	566.2	586.9	587.6	549.7	668.2
Average	526.4	524.8	537.3	531.2	555.1	521.3	518.5	537.3	614.0

Overall, in this setting, IFF and IBF are able to outperform the baseline algorithms in terms of turnaround time and total rental cost and outperform most of the baseline algorithms in terms of resource utilization, Although LWT has the best resource utilization, it induces significantly more waiting time, thus has higher

total rental cost. Furthermore, the results suggest that all improved versions of the baseline algorithms are able to optimize turnaround without sacrificing any other metrics.

6 Conclusion

In summary, out of the four new algorithms, only two of them prove to be superior to the baseline algorithms with respect to turnaround time: IFF and IBF. Furthermore, all baseline algorithms showed a more optimized turnaround after being aware that backfilling is not allowed in ds-sim.

Although IWF's poor performance is highly expected due to the known inefficient nature of the WF algorithm in ds-sim, it is surprising to see that LWT is also inefficient in terms of turnaround time despite the fact that it was intentionally designed to optimize this performance metric. This may suggest that "core scheduling" (scheduling jobs based on the number of cores) is the most effective way to optimize turnaround time when there is an absence of backfilling and job migration.