# Distributed Systems Job Scheduler – Stage 3

Connor Wilson – 44946058

## Introduction

In stage two of the Distributed System Job Scheduler, we implemented 3 algorithms Best fit, first fit and worst fit. Best fit trades time for accuracy which in effect gives average costs, worst fit is extremely fast however trades of for major costs and fast fit its extremely quick with minimal cost but long wait time. Stage 3 of our Distributed Systems job scheduler is about creating our own scheduling algorithms That increase at least one specific objective. These objectives can aim to *optimise* one or more things such as time, cost, server usage, number of servers used etc.

## Problem Definition

Since we have to improve upon Stage two we have multiple problems to deal with. Firstly, we have to fix our RESC ALL function as its broken in stage two, secondly, we have to fix the sorting of the Server Lists by core count. After we complete these tasks, we can begin working on the main problem optimisation. We must produce a sorting algorithm like Best fit, first fit and worst fit the optimises one or more objectives relative. In this report our focus will be set upon the cost vs server utilisation. To do this I will be testing three algorithms absolute best fit (abf), absolute(fast) first fit (fff) and Average First fit (aff).

## Algorithm Description

### Absolute Best Fit

The ABF algorithm aimed to reduce costs more then the Best fit algorithm by more accurately placing the servers correctly. This is done by checking the memory and disk space as well as core count. The Addition of a core check was added to allow the RESC ALL function to work, this compares the current servers’ cores to the original to ensure the server has enough cores to run the job.

Pesudo Code ouput

For a given job j

Set Min to smallest server

Loop: For each server in server list

Get the current server S and get the current job J

Loop: For each original server in original server list

Get the current og server OS

If OS == S

If J cores >= OS cores

If S cores <= min Cores

Set min = S

End if

End if

End if

End loop

End loop

Set the schedule Server to min

End

As the charts show this algorithm performs the same as Best fit, which given our basic config is expected. As such the Costs are also equal for this given config

### Absolute(Fast) first fit

The AFF algorithm aims to Increase allocation speeds and reduce costs, however the trade-off is Server wait time increases dramatically. It does this by immediately scheduling the current job to the smallest server with the minimum required cores. Of course by doing this we will end up stacking jobs onto one server hence the server wait time increases.

Pseudo Code output

For a given job j

Get current job J

Loop: for each og servers

Get og server OS

If (OS core >= J Core)

Set the schedule server to OS

End if

End loop

Due to all the Servers being piled onto one server you can see here that the average turnaround is far higher then the other algorithms, however the cost is virtually equal.

### Average Fast Fit

The AVFF is an improvement to the AFF algorithm above but sheds Server wait time for Cost. The idea is to split the jobs over the servers equally, while still being first fit. For example, if 3 jobs have 1 core and 2 have 4 cores, the 1 core servers will be split over tiny 0-2 and the 4 core split over medium 0-1 instead of them all being set to tiny 0. This implementation will use a *simpler* version due to time constraints. This implementation will only split the jobs over servers 0-n if the scheduling of jobs is concurrent. So for example 3 small servers will be split small 0-2 while small, large, small will be set 0 ,0, 0 respectively.

Pseudo Code Output

For a given job J

Mod count = 0 (global variable) MC

ServerType = " " ST

Server count =1 SC

Loop: for og servers

Get og servers OS

Loop: while ST == " "

If ( J core <= Os core)

ST = OS. name

End if

End loop

If (OS name == ST)

SC ++

End if

End loop

Loop: for og servers

If ( MC >= SC)

MC = 0;

End if

If(OS id == MC && OS name == ST)

MC ++

Set the schedule server with this ID

Return;

End if

End loop

return

end

As you can see the average turnaround is much lower than that of the AFF algorithm. The cost of the two algorithms is also equal, so this algorithm successfully reduces turnaround without effecting cost.

## Implementation details

I implemented a function and variable called *ogservers* which holds the original untouched serverlist. Ogservers() will get the list from the server and set this variable accordingly. I also use the Sort function that I created, this will be called on either the Allserver (resc all servers) or serverlist (resc avial servers). This will sort them by cores then memory then disk. I have also added some error arguments in the main to allow for my extra schedulers. Besides these added functions all variables have been taken from stage 2.

## Evaluation

#### Test 1

For a my first test I used the “config\_simple3.xml” This is more or less to test if the algorithms worked as intended.

Figure 2

Figure 1

Figure 3

Interestingly the cost for the AFF and AVFF algorithms are equivalent to fast fit ($4830.93). This is because in this test case the server costs are equivalent for each small server. Also take note of the Avg turnaround time difference AFF is around 85% higher then all the algorithms. This means that the AFF algorithm as expected has severe AVG turnaround time penalties, however has 4% higher avg Util then ff. AVFF algorithm reduces the wait time drastically (80%) however in turn reducing the server Util to only 86.7%. As a result I would debate that the AFF algorithm in this base case is just a slower FF , which depending on the service providers hidden costs, could be cheaper, as it will only use the minimum servers. while the AVFF algorithm reduces both wait time but also utilisation meaning, this would benefit the customer, but not to the extent as FF. ABF interestingly costs more then the base BF. ABF also has 4 more ticks in the avg turnaround time with 2% more server util. This result is extremely interesting as ABF is just BF but with memory and disk space checks and yet it works out to be worse than BF.

#### Test 2

The second test case what used with “config\_simple4.xml”. I used this test case because the servers are equivalent to “config\_simple3.xml” servers but the jobs are different. In this test case the jobs execution times will finish in 3647 ticks while in “config\_simple3.xml” the jobs would execute with 225062 seconds.

Figure 2

Figure 3

Figure 1

In test two the AFF algorithm has a average utilisation difference of 36% while in test one they have a difference of only 4%. As expected, both AFF and AVFF have equivalent costs to FF and ABF has more cost then BF. Although the avg Turnaround compared from test 1 is almost equivalent. This result tells us that AFF is actually more effective on servers that have multiple instances.

#### Test 3

This test is being used to see the results of “config\_simple4.xml” as it has 3 servers one with size 5, one with size 40 and the other size of 20. We will aim to see the AFF results against the fact there are less server amount and more server repetition.

Figure 2

Figure

Figure 3

Interestingly these results prove to us that AFF is more cost efficient when there are more multiples of servers. This is what we expected, however interestingly AVFF actually costs the most despite it effectively trying to mitigate the AFF algorithm. This suggests to us that mitigating turnaround time in a situation where there are more repetitions of a server the AVFF is actually far worse. Another fact to look at is the AFF has an average utilisation of 98%, 35% better then fast fit and a whopping 55% better than AVFF.

#### Test 4

This test aims to see the results is my thesis for my algorithms. I’m using “config\_simple8.xml” which contains one server with 100 repetitions

Figure

Figure 2

Figure 1

As expected, The results adhere to my thesis, the cost of the AFF algorithm is cheaper then all the other algorithms and it has 60% more server utilisation than any other algorithm. Also, to expected results AVFF costs equally as much as worst fit and has the worst average utilisation. AFF’s turnaround time is extremely bad however which we expected.

## Conclusion

In conclusion my AFF algorithm is extremely cost effective in situations that have less server count and more server repetition however in contrast my AVFF algorithm works far better when there are many servers and minimal repetition. My best fit ended up being far worse then the best fit given as it uses one extra server for an unknown reason. Ultimately I would class my AFF the champion however it truly depends on the situation for which algorithm you use.