Job Scheduler for Client Side Simulator

Name : Muntasir Adnan
ID : 44292252

The proposed scheduler will try to improve utiliasation of used servers, execution time and the cost simultaneously. Thus it relies on somewhere between best fit and worst fit. The major objective that it focuses on is utilisation and cost.

*****Each job comes with an estimated run time that provides us with an assumption of how long the job will run. An error margin of +20% has been added to the estimated runtime.***

Problem Definition

The existing best fit and worst fit algorithms can optimise the resource utilisation and cost separately but a customer who wants better resource utilisation and execution time whilst paying something close or sometimes even better than best fit can use the proposed algorithm. Medium data sensetive software like online video streaming, video chatting, online games or map reduce jobs can get benefitted from the algorithm.
In some cases, the waiting time can rise a lot which happens when required number of cores of the jobs is high and most of them are "long" type jobs. So, very data sensitive jobs like operating a driverless vehicle will not suit the algorithm unless the jobs complies with the specs mentioned in the list "**Proposed algorithm will work best when :**".

The scheduler will mostly help to the consumers who has a time limit but has a budget as well preventing them from renting all the large servers which will incur a hefty cost.

Resource utilisation is improved by using least amount of server possible. Mainly at most 1 server from each type will be used and everytime the customer buys a server, the algorithm makes sure the server utilisatin to be high.

Proposed algorithm will work best when :

- When most of the jobs are tiny or short and require less core. It's mainly because the algorithm uses first fit servers to minimise cost which means if there are jobs that requires high number of cores, execution time and waiting time will increase since jobs will not be able to run parrellely.
- If jobs can be processed to come in a sorted away.
- When there are less number of server types. This way, the cost will be close to the best fit cost since less number of servers mean all of the types of the servers will have to be used during the execution.

Algorithm overview :
The scheduler will check for a first fit server from system.xml to run a particular job.

When the first fit server is determined, the algorithm will check if the server type has been previously used to **minimise cost and improve waiting time and resource utilisation**.
If that particular server type has never been used, scheduler will check if ther's a used server of any type that can accommodate the job. If yes, scheduler will scheduler the job to the server. To get a better waiting time and cost, this algorithm will try to schedule the jobs to a server that is currenly active/busy regardless of whether the server is best or worst fit for the current job. **This way scheduler will avoid the 60 seconds boot up time of the server and won't cost the customer to rent a new server for another hour**.

If there is no server in the used server list that can accomadate the current job, scheduler will start the best fit server and schedule the job to that server. The reason why first fit server is the best choice to start is it comes with least risk of adding unnecessary cost and poor utilisation when **the proportion of core and cost is same for all the servers**. For example we can focus on this scenario:

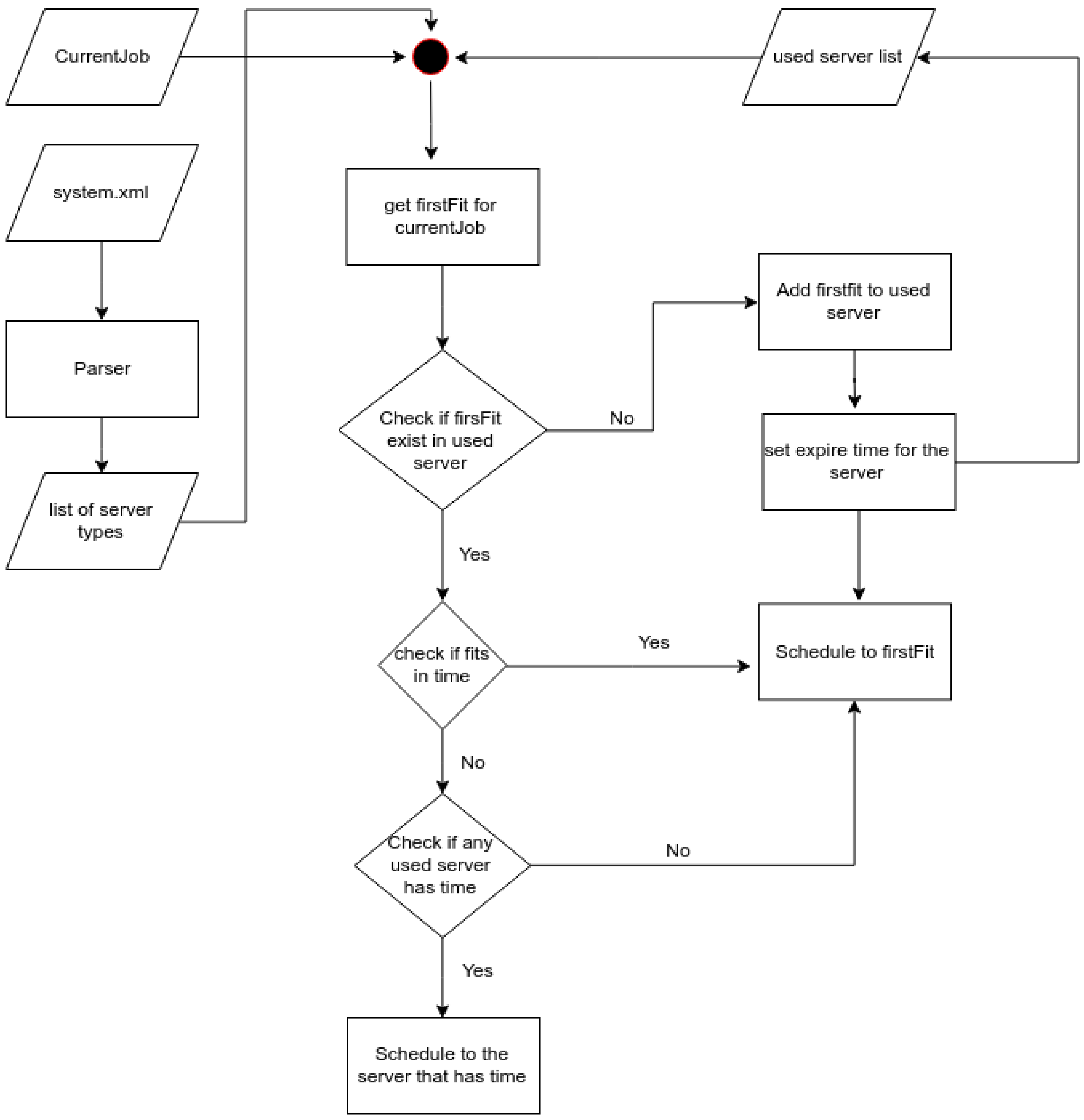
> *Job requirements : 1 core, 10 unit run time*
> *Servers available : Tiny with 1 core, hourly rate 0.1 unit and Large with 8 core, hourly rate 0.8 unit.*
> *In the worst case scenario where the current job is also the last job for the server, the cost to run this particular job in "tiny" for 10 units of time will be : (0.1 * 10)/3600 = 0.00027 whereas, The cost to run the same job in "large" for 10 units of time will be : (0.8 * 10)/ 3600 = 0.002 which is* ***7 times more than the cost to run the job in tiny and will have the lesat efficient utilisatin****.*
> *If we are runing the same job 8 times sequentially in tiny and parallelly in large, only then the cost to execute each job will be very close to same across tiny and large servers since 0.002/8 = 0.00025.*

If there's a server in the used list that can accommodate the job, scheduler will schedule the job to that server. **This way, scheduler will improve execution and waiting time.**

If the best fit server is present in the used list of the servers, the scheduler will have to check if the server can process the job in a time limit where it will not exceed the current one hour limit. To schedule the job, scheduler will take the submission time of the job, apply the error margin on the estimated run time and add it to the submissoin time to estime when the job will finish executing. On basis of that, ther scheduler will decide whether to schedule the job to the best fit server in the used list or to a different server from the server list. **To get a better waiting time and cost, this algorithm will try to schedule the job to a server that is currenly active/busy regardless of whether the server is best or worst fit for the current job**.

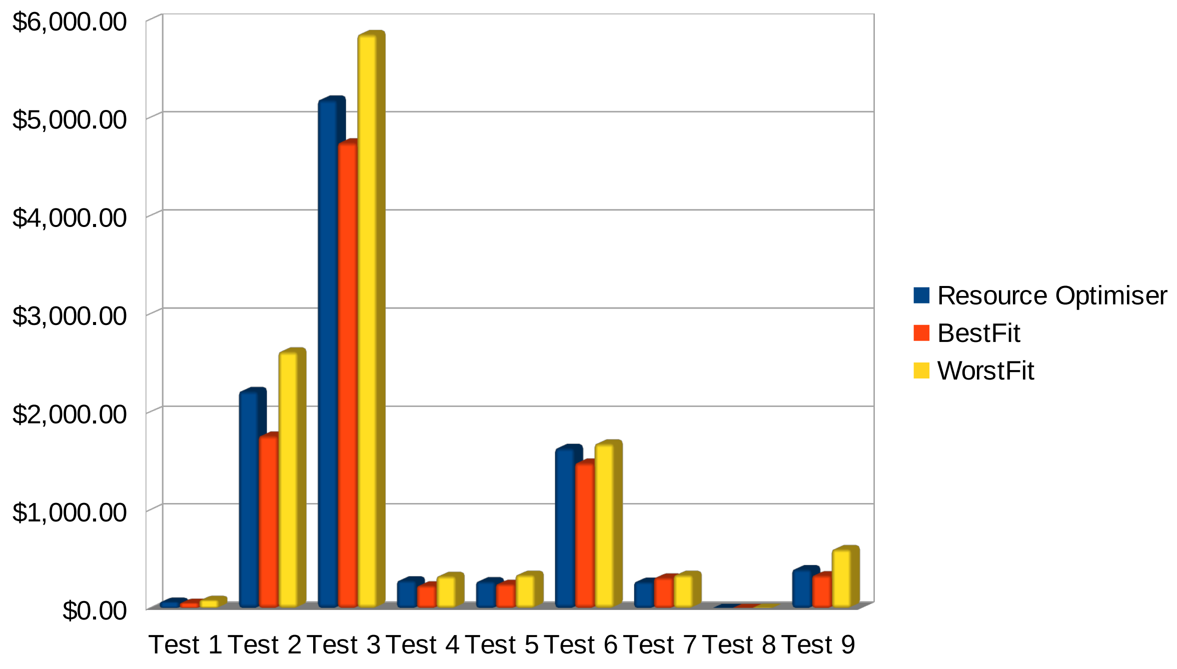
If none of the criterias meet, scheduler will start a best fit server and schedule the job to the server since it comes with least risk of poor utilisation and hefty cost.

Provided a UML diagram of the proposed scheduler:

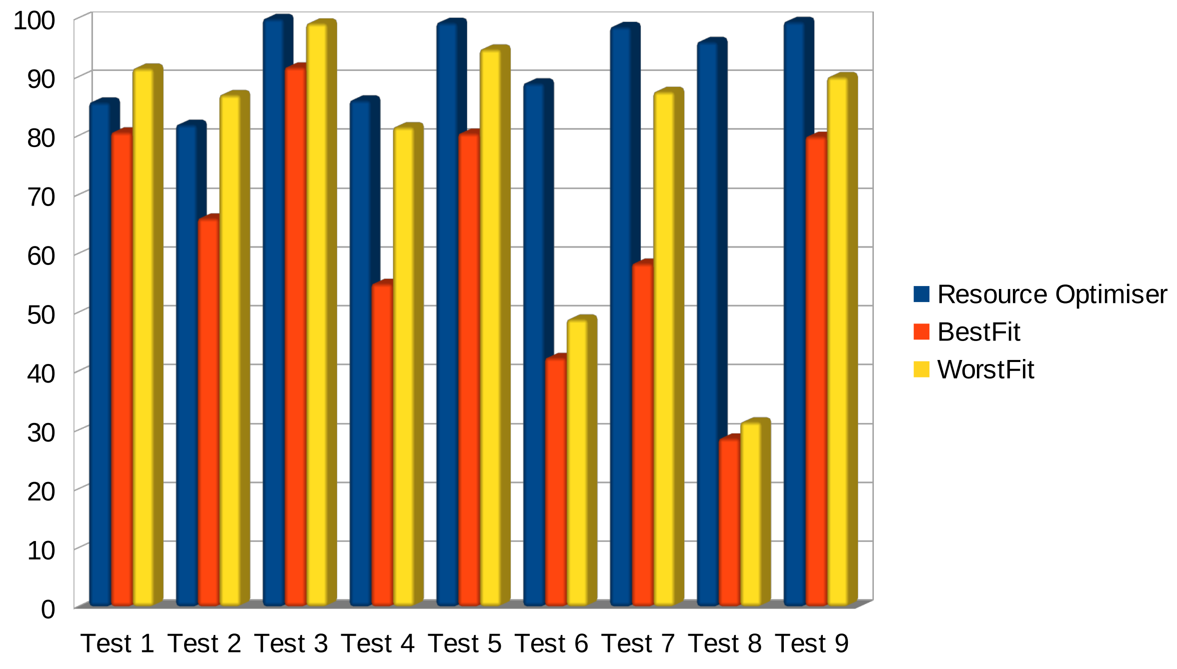

CurrentJob
system.xml
Parser
list of server types
used server list
get firstFit for currentJob
Add firstfit to used server
Check if firsFit exist in used server
No
set expire time for the server
Yes
check if fits in time
Yes
Schedule to firstFit
No
Check if any used server has time
No
Yes
Schedule to the server that has time

Evaluation :

Performance based on cost :



Performance based on resource utilisation :



Conclusion
The charts in evaluation highlights how the alogorithm works quite clearly and precisely. We can see that it optimises the resource utilisation comparing to worst fit in every test case except one ant beats the best fit in every case. On the other hand, its cost is always better than worstfit and very close to bestfit in fact it beats best fir when the jobs and servers has less types and most jobs are short or instant.
Clearly the optimisation of cost and resource utility does not go hand to hand but this algorithm priotarizes resource utilisation and at the same time brings the cost to an affordable margin.

Software Requirements
The software is written on a Java environment. To run it in a linux bash terminal java has to be installed. Please follow the mentioned procedures below to run the program :

- First open the terminal (Ctrl+Alt+T)
- Add the PPA (sudo aa-apt-repository ppa:webupd8team/java). Then input the password
- Update and install the installer script (sudo apt update; sudo apt install oracle-java9-installer).
- Check the java version (javac -version)
- Set java environment variables (sudo apt install oracle-java9-set-default)
- Extract the submission.
- Download the ds-sim tar in the same folder and extract it.
- Before running, compile all classes. For example, to compile use the command : javac client.java
- Best fit, worst fit and first fit algorithms can be run using bf/wf/ff : “java client -a ff/bf/wf”.
- If -a ff/bf/wf is not included, and only ‘java client’ line, then by default it will run allToLargest().
- To run the newly implemented algorithm use java_client_class -a `optimiseResource`
- Admittedly, before running the client class, ds-sim must be downloaded untarred from .tar and run the server with specific config_simple.xml or without it.