Stage 2

Implementation of a turn around time efficient job scheduler.

Francisco Butturini
45437408

COMP3100 Semester 1 2021

Introduction

Building on from the first stage of the project which was implementing a simple job scheduler which used an all to largest server algorithm (ATL) to dispatch its jobs. So I have inherited a simple job dispatcher which works with the DS-server protocols, going into the second stage I now need to implement a more intelligent job scheduler which has to optimise one or more metrics. Those metrics being minimisation of average turnaround time, maximisation of average resource utilisation or minimisation of total server rental cost. As well as optimising one or more metrics it obviously needs to be different to ATL and beat all baseline algorithms (BF,WF,FF) in the metric of our choice which is turn around time.

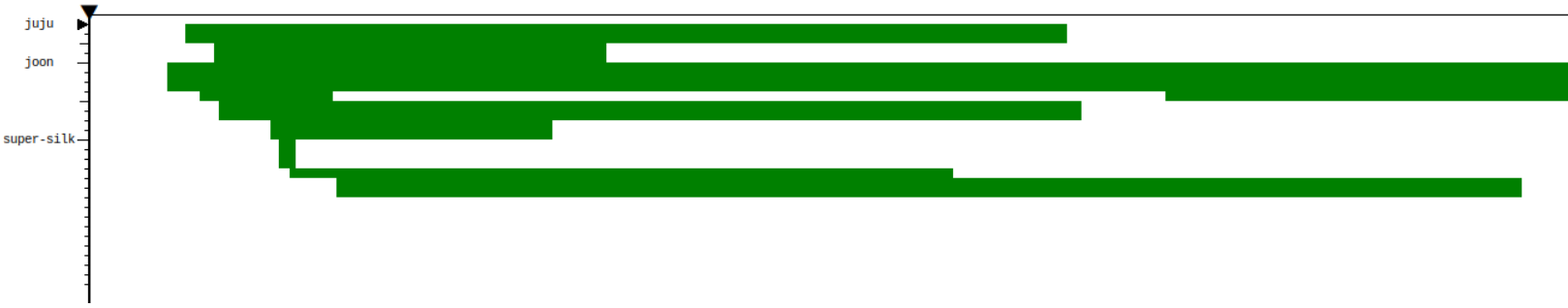
Problem Definition

When talking about job scheduling with multiple servers in a distributed system environment, we mainly talk about resource provisioning in terms of cores. Each server has a specific amount of cores and each job requires different amounts of cores depending on what the job needs. So the problem is allocating the correct jobs to the most appropriate servers so we can maximise how many jobs can be run concurrently across multiple servers. For my algorithm I have optimised the minimisation of average turnaround time, I have chosen this optimization as I see that if we can optimize how quick the jobs come back we can then try to see if we can optimise server utilization.

Implementation details

For my implementation, I have used several data structures including nested string arrays for my server keeping as well as obvious socket programming to make sure that my client and server can talk to each other. I have also used a XML parser so i can read details from ./ds-system.xml, this is an external library which needed to be imported. This allowed me to use and read the details of the server so I could create the correct scheduling decisions. The client is also built with the user needing the -n option on the ds-server to indicate its using newline characters.

Algorithm description



My algorithm is a turnaround time efficient job dispatcher which gets a job from the DS server, my algorithm initially uses the gets Avail command. The algorithm then parses the response from the server with all the servers which can currently run the job and is ready. Once it has collected all of the servers that can run the job it then filters through that list to see what server has the smallest core count. Once filtered it then sends the job to the server with the smallest core count and is available. However once there are no available servers to handle the current job it then hands over to another function which is where my algorithm comes into place. So once I receive a 0 from the ds server indicating there are no servers that are available that can handle my job it calls another function called handleGets, in this function the job is passed through and it is scheduled to the server which has the shortest wait time. This is done by using the command gets Capable, once the capable servers are fetched its then sorted by the server command EJWT which estimates the job waiting time. Which ever server has the smallest wait time the job is then scheduled to that server and waits its turn to be run. This is done until there are no jobs left and it disconnects.

Evaluation

Even though my algorithm is quite efficient at average turn around time it does not do well in other metrics such as total rental cost and resources utilisation. There is one glaring problem though when I look at my algorithm is that when jobs are scheduled it does not take into effect that when jobs finish other servers are capable and free to run the jobs that are scheduled and waiting, I would of like to implement a feature which checks jobs that are waiting to see if they can be migrated to another server which is available to run it instead of waiting to be run on that same server. My algorithm is better than WF/BF and ATL in all situations for turnaround time, and sometimes matches or beats the first fit option.

Conclusion

In conclusion my algorithm schedules the way it's supposed to do which is minimise turn around time in which it beats or comes close to all of the baseline algorithms (WF/BF/FF).

References

https://github.com/Rex781/Stage2

https://github.com/distsys-MQ/ds-sim