Cloud Job Scheduler

Nipun Shrestha (45549192)

Introduction
Distributed System, is when distributed computer systems and distributed related and unit-coordinated [1]. Due to the vast nature of these systems, they often have a variety of resources in terms of computation power and memory. The effective use of these resources is very important for high performance and scalability. To get the optimal assignment of the task to each resource, a scheduler is needed. Job Scheduler is to provide this optimal utilization of the resources by acting as a ‘manager’ and distributing jobs to servers.

This project is a continuation of the previous project and aims to simulate a Client-Server model job scheduler for a distributed system and provides an effective algorithm to optimize the utilization of the server resources. A server-side application has been provided which simulates a distributed system and all of its servers.

Stage 2 implements an algorithm to schedule the jobs to various servers available to find the optimal turnaround time in the distributed system. This algorithm is built to improve the performance of the First Fit algorithm by dispatching the job to the next capable server instead of the first one if there are no available servers and decreasing the waiting time of the first capable server.

Problem definition

The turnaround time of a process or task is the time of submission of a process to the time of the completion of the process. The turnaround time is important as it tells us if all the tasks are given the right priority to process them. This can help us identify if any tasks are not processed and dealing with starvation issues or not.

The scheduling algorithm focuses on reducing the turnaround time and optimizing the turnaround time of the first-fit algorithm. As the baseline first fit algorithm allocates jobs to the first available server and queues the job to the first capable server if there are no servers available. However, in my first fit algorithm, I have stored the last capable server used and instead of dispatching to the same capable server, I dispatch the job to the next capable server. This would significantly decrease the turnaround time of the jobs as there would not be large waiting queues and the jobs would be evenly distributed around all the capable servers.

Algorithm

In my algorithm of dispatching the job, I get all the available servers that can process the requirements of the current job that needs to be dispatched by using the "GETS Avail" message. If there are available servers I dispatch the jobs to the first server that can process the job. However, if there are no servers available currently to process the job, I get the list of all the servers that are capable of processing the job. I store all the server information in an array list as the ds-server sends the information of the capable servers one by one. Then I allocate the job to the first server that is capable of doing the job and store the server information. Thus now if a similar job of the same system requirements is seen, I would allocate the job to the next server that is capable of doing the job, therefore decreasing the turnaround time.

For example: In a scenario where we need to schedule a job of core count 2, memory 400 and disk 120 and there are no servers that are currently available to process the job immediately, the algorithm would schedule the job to the first capable server which can process the job in the future. The information of the server that the job was scheduled is also stored. Now if we need to schedule a job with similar system requirements and there were still no available servers, the algorithm would schedule the job to the next capable server instead of allocating it to the first server. This would therefore decrease the turnaround time of the scheduler.

Implementation

For the implementation of the scheduling algorithm, the connection and disconnection to the ds-server have been reused from Stage 1. The auxiliary function handshake() performs the connection.

The stage 2 implementation of the algorithm, included the use of standard Java libraries that stored and process information to schedule the job effectively.

Socket Programming has been used within this algorithm to allow communication to and from the server. The gathering of different information needed to process the scheduling decision has been done through sending and receiving data from the server.

The auxiliary method getServerAvail() performed the steps of finding the immediately available server to schedule the job. To this method, the message “GETS Avail” message was sent to the server and a variable was used to store the first server information sent by the server to the client. The job would then schedule the job to the server that the variable contained.

Furthermore, the implementation of the auxiliary function getServerCapable() performed the steps of finding the next capable server that the job can be scheduled to if there were no servers immediately available. This auxiliary function used data structures like ArrayList to store information of the capable servers that could process the job and HashMap to map the job requirements to the capable server information.

The use of an array list was needed to store all the information of the capable servers as the ds-server would send this information one after another. We also needed to know the next server that the job needs to be scheduled to thus the HashMap has been used.

Evaluation

Evaluation of my algorithm was done by using the test script and configuration files provided to me.

From the testing of different test scripts provided to use I can easily see the decrease of turnaround time in my algorithm as compared to the other baseline algorithm and the All to Largest algorithm.

To run the test script that was provided to me I have used different arguments to cater to the objective of my algorithm, which to optimize turnaround time. I used arguments such as “-o tt” which test the algorithm based on the turnaround time score. I also used the argument “-n” to specify the usage of a new line as my algorithm uses a new line character.

According to the first test script and the configuration file that was provided to me, my algorithm has been able to score a lower average turnaround time than all the other algorithms except the best-fit algorithm. My algorithm scored an average turnaround time of 1464.06 which is better than the first fit algorithm and we can see an improvement in the optimization of that has been the implementation of my algorithm.

Pros and Cons

Pros
The advantage of using my algorithm over the first fit algorithm and other baseline algorithms is that my algorithm tries to effectively allocate resources and mitigate the issue of starvation.

Moreover, this algorithm is also fast and efficient as it does not try to loop through every server and try to find the best or worst possible server. This effectively decreases the execution time and complexity of the algorithm.

Cons
The disadvantage of using this algorithm is that it does not provide the proper distribution of jobs. This algorithm simply finds the next capable server that can process the job and scheduling the job to that server without considering the waiting jobs queue in that server and the estimated completion of the active job in that server. This could result in improper distribution and cause one server to be heavily occupied than the other.

Moreover, this algorithm only stores the core count of the server as the key in the hashMap. This could result in conflicts if there were servers with the same core count but different disk and memory.

Conclusion
All in all, this algorithm is an optimization of the first fit algorithm which store the information of the servers and the job specifications and which schedules the next similar job to the next capable server which has not already been assigned, if there are no immediately available servers that can process the job.

As I mentioned above that this algorithm does not account for the state of the next capable server while scheduling the job. Usage of this information while scheduling the job would greatly improve the performance of the algorithm.

For proper job scheduling, I have found that it can be difficult to effectively schedule the job without knowing the information of the other jobs that are incoming. If we could somehow make a pool of jobs and then schedule the job based on the requirements, then that would have been much more effective. As the ds-server cannot provide us with information about the other job without scheduling the current job, making a pool of job is impossible thus we have to make choices based on the current job.

Reference

GitHub: https://github.com/nipunshrestha/COMP3100-Stage2

[1] Resource Allocation for Distributed Systems: A Review Kurdistan Was HamaAli & Sushi R.M. Zeebaree

[2] A. S. Tanenbaum, Distributed Systems: Principles and Paradigms, Harlow, Essex, England: Pearson Education Limited, 2014.