Client-Job Scheduler to minimize turnaround time.

**Name: 45942854 = Carlos Larino**

**Introduction:**

This project is to demonstrate the understanding of job dispatching and scheduling works in distributed systems like data centers. The overall aim was to design and implement a new scheduling algorithm that optimizes specific objectives. These objectives needed to dispatch jobs in a way that minimizes the average of turnaround time, Maximizes the average resource utilization, and minimize the total server cost. During this project, the focus will be the minimization of turnaround time and display how the new scheduling algorithm can be seen to be more efficient that the three-baseline algorithm. This report will explore the comparison of the First fit, best fit and worse fit with the new scheduling algorithm that focuses on the minimization of turnaround time. Hence, it will cover a qualitive analysis on the advantages and disadvantages of each algorithm and determine whether the design and implementation were successful.

**Problem definition:**

The design of the algorithm was to create a job scheduler that would minimize the turnaround time of job scheduling, doing so will make jobs complete earlier than something with a larger waiting time to completion. When designing the algorithm, other objectives were not possible to achieve as creating a job scheduler that minimizes turnaround time altered the total rental cost and resource utilization. However, this could be fixed by creating a more enhanced algorithm in which it will take all those objectives into consideration and trying to find the equal point between all those objectives. Although, this will not effectively for all three objectives. For example, making the turnaround time to its absolute minimum may mean that the total server rental cost is placed in maximum as the job scheduler will try to prioritize getting the job done earlier than reducing the cost of the server. Furthermore, this can also mean that minimizing turnaround time could affect the average resource utilization as some servers may not be used at all. If the algorithm is picking the earliest start time. Hence, almost no algorithm can absolutely make one objective more efficient without hindering another objective. Turnaround time was mainly picked due to the objective of making the jobs complete earlier and constantly having free servers if the need arises. For example, if the job scheduler could prioritize how early the jobs get done, it would mean that a downtime of a server could be reduce and more jobs could be placed within it in a real-world scenario.

**Algorithm: Turnaround time algorithm**

**Input:** A String Array *jobs*, A Server Arraylist *tempServers, shortestStartTime* server -> This will be the server with the shortest start time

**Output:** A scheduled job with the shortest start time server

**Process:** Grab jobs that need to complete, ask for all capable servers to run the current job in terms of core count, memory, and disk space. Once receive all the servers’ data, allocate all information to an array list. Then use that array list to sort and find the server with the shortest start time. Assign that shortest start time server to a variable and schedule the jobs in that server.

**Purpose:** The design of the new algorithm was to minimize the turnaround time of the jobs. This can be done by finding all capable servers that the job can run on and with those received details from the server, it can find the first server with the shortest start time. This will reduce the time taken for the server to start up and start completing the job. Hence, minimizing the turnaround time of the job. This algorithm is derived from the first fit base line algorithm with similarities such as finding the most capable servers in terms of whether the job can be completed within the server and assigning that job to that capable servers with the major difference is that it will focus on the server with the shortest wait time and assign the job to the shortest wait time compared to first fit where it will assign to the first capable server.

**Work Example:**

Considering the configuration file of “ds-config—wk9.xml”, there are job types that are categories with short, medium, long and requiring 1,2,3 cores, respectively. There are also server types that are categories as tiny, small, medium, and containing 1,2,4 cores, respectively. This information is needed to understand which capable servers that the algorithm needs to complete the job and how the new algorithm will use an extra attribute to determine where a job should fall into.

After the handshake and connection is established, the algorithm will tell the server that it is ready to schedule the jobs and dispatch them to the server. The client will enter a while loop that will keep running until all the jobs are schedule and completed. For this work example, we will imagine that there are 2 short jobs and one large job for simplicity. The while loop will enter its first job assignment for this case job 0 is considered as short job. The details are read using a list of strings to save required information such as jobID, coreCount, memory, diskspace. This information will then be used in handleGetsCapable function to determine which server is the best for the minimization of turnaround time. For example, job 0 will have a core count of 1, a memory size of 1000 and a required disk size of 10000. Once the function of the handleGetsCapable is called, it will pass the data within the list such as the coreCount, memory and diskspace. In the function, a “GETS Capable” message is sent to the server and the server should send back all its servers that can run the job. The client receives the list of servers and attaches all the servers’ attributes to a server array list so that data can be easily called and accessed. For this example, tiny, small, medium are all capable to run the job as the servers has sufficient coreCount, memory and diskspace. The algorithm will then pick search through all the capable servers start time and determine which one will have the shortest start time. This will make the job execute earlier. For this case, tiny (40) and small (40) have a quicker boot time than medium (60). The scheduler will then return the server ID and Server type of which server the job should be assigned to and then the client will schedule the job based on the server ID and server type. In this case, it will be tiny 0 as it is the first available resource. The algorithm will continue this process with the other jobs. For example, job 1 will get small (40) as it is the next earliest capable boot time, and the last medium job can only fall into medium as it is the last available job. Once all the jobs are complete, the client scheduler closes the connection.

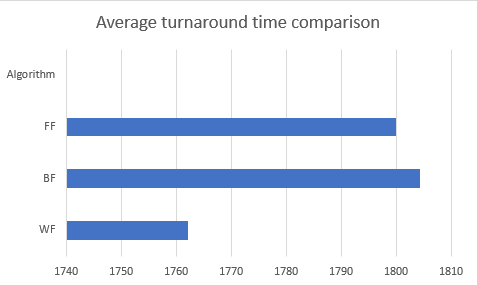
Take note, this example is very basic. In real world scenarios different servers could have different boot times. For example, small could have a longer boot time for some reason than medium. Hence, the algorithm in that case will prioritize medium (60) as it may have a smaller boot time than small (Eg. 100).

**Implementation:**

The implementation of the algorithm used data structures with two java classes defined. The first java class was the server class, this would be a foundation class to hold all the data received from the server and assign it within this class. This class holds setter and getter functions to easily call any server attributes need to complete the algorithm or to extend the algorithm. The main setter and getter function used in the server class was the current start time function. This was used to find the server with the shortest start time by comparing all the servers current start time with each other and find the shortest one. The client java class holds all the scheduling instructions to send to the server. The main method needed to establish a handshake with the server is used using DataOutputStreams to write to the server and establish a connection. A creation of an array list was used to save server information to use for the comparing of the servers to find the server with the shortest start time. An implementation of a parse information list was used to view incoming data of the jobs receive from the server to allow easy transference to required function calls. A function handleGetCapable is implemented to clean the code up but also to house the algorithm. Within this function, it will find all the capable servers to complete the job then using an array list of servers will find the shortest start time of the servers. This will ensure that smallest waiting time is achieved between the first jobs. The function returns the serverID and serverType which can be used as inputs into the schedule function. This schedule function will schedule the job in the most capable server. Finally, helper methods were used to allow ease of code reading. Functions such as msgSender and msgReciever both used to implement cleaner code. In msgSender basic DataOutputStream was used to send the message to the server and in the msgReciever a readline() function was used to read any incoming bytes received from the server which can be used to establish which server is the most capable.

**Evaluation:**

The three baseline algorithms gave a good standard when comparing the usability and efficiency of the new algorithm. The three baseline algorithms of first fit, best fit and worse fit gave logs which allow the determination of the best turnaround time. The configuration file of “ds-config—wk9.xml” was used to test all the algorithms and get acquire the specific timings of each algorithm. Since the goal of the algorithm was to reduce the turnaround time within job scheduling, this will be the focus of the evaluation.

To determine whether the algorithm was reducing the turnaround time for each job, we need to calculate each of the baseline and new algorithms turnaround time. This can be done by summing up all the jobs finishing time minus the submission time and dividing that by the number of jobs in that need to be scheduled. The main idea is to reduce the time it takes for the job to submit and complete. This can be effectively done by reducing the waiting time. For example, if the schedulers can pick the servers with the shortest wait time every time, then the jobs will be completed quicker. This can be seen clearly in the diagram below. Worst fit has the best turnaround time compared to all the other baseline algorithms this is because worse fit initializes only 2 servers to complete all its jobs. This is the small and medium server. Since the small and medium servers are already initialized the jobs do not need to wait for the server to start up till the next job is sent. Comparing that to the best fit and first fit where the tiny needs to be initialized, it adds extra time till it gets the job done. In a real-world scenario, if there are multiple different servers that are being initialize where only one can be used to finish a job could pad a lot of time. This can result in delay, latency, or downtime as all the servers need to be initialized. The new algorithm should have efficiently reduced the average turnaround time by eliminating the initialization of each server as it will always try to pick the shortest wait time to complete the job, however, it was not able to be tested.

**Pros and Cons:**

The benefits of the new algorithm should reduce the turnaround time of the jobs as the initialization process of server is reduced as it will always try to prioritize the short start time servers possible to complete the jobs. This algorithm should reduce the turnaround time whilst maintaining the resource utilization. For example, if there are only two jobs needed to be completed where 7 servers are always available then the algorithm should prioritize the shortest start time which allows the servers to be more readily available after the job is complete. Hence, the average turnaround time for each job should in theory be reduced.

The limitation of this new algorithm is that it will not be effective if all servers need to be ran to complete all the necessary job. If there are many jobs to be completed picking the capable server with the shortest start time may not be used as all servers will be initialized anyways. Although, it will reduce the average turnaround time a bit as it is trying to sort in a way that the jobs can finish early but if there are too many jobs, the algorithm may be useless. For example, if all servers are full of jobs the algorithm will not do anything else but assign the job to next available server as the initialization process of the server is finished.

Worse fit does provide theoretically better turnaround time than the new algorithm as it only needs to initialize the minimum number of servers to complete the job. In this case, it will be 2 (small and medium). However, worse fit also has the worse server utilization compared to best fit and first fit as it will try not to use a smaller server if it is not needed. The algorithm is designed conceptually like worse fit prioritizing the quicker start time servers, but it also knows to assign the jobs to another server if there are jobs already running within that server. Therefore, the algorithm may not be as efficient in reducing turnaround time compared to worse fit, but it is still able to be hybrid of efficiently balancing turnaround time and resource utilization.

**Conclusion:**

The design and implementation of the new algorithm was unsuccessful as the code did not properly work in scheduling the job. Although, the theory and logical of the design can be seen as viable and a better implementation may result into a more efficient and working algorithm which could resolve the requirements objectives of a minimization of turnaround time.

**References:**

Larino. Carlos (2021). COMP3100\_Stage2 [online]. Github. Available at: [https://github.com/Surga3/COMP3100\_Stage2]

Lee. Y. (2021). ds-sim[online]. GitHub. Available at: [https://github.com/distsys-MQ/ds-sim/]