Katrina David (45308748) Sakura Mukhopadhyay (45435073) Trideep Lal Das (45532125)
Group 8 (Thursday 4pm-6pm)

Intelligent Priority Scheduler for Distributed Systems

Introduction

Priority Scheduling is the task of scheduling jobs based on a scale of importance. This importance is based on the submission time in which a particular job is sent to the scheduler. Stage 2 of the project requires Group 8 to understand and implement three simple scheduling algorithms that are based on memory allocation policies. To our understanding, the main purpose of implementing algorithms to the job scheduler is to ensure resources are utilised efficiently and minimise the occurrence of resource starvation. The three scheduling algorithms implemented in Group 8's priority scheduler are First-Fit, Best-Fit and Worst-Fit algorithms schemes and . Stage 2 of this project also includes producing a simulation log generated by the server-side simulator and is based on simulation statistics that are presented to the team members at the end of simulation. Group 8's algorithm design document features the design considerations and preliminaries for data structures used in the Priority Scheduler as well as detailed algorithm descriptions for each of the three simple scheduling algorithms implemented; First-Fit, Best-Fit and Worst-Fit Algorithms.

Preliminaries

As per stage 2, three new files were added on top of the previous stage 1 implementation, namely: Server.java, Algorithm.java and Job.java all of which are required for the implementation of the algorithms. For the set-up, it remains relatively similar to Stage 1 requiring matching port numbers in this case being port '50000', creating a makefile and inputting the chmod commands for server and client, and ultimately running the ds-client file. The only difference is this time the client can be run using the three newly implemented algorithms using the argument '-a' in the command line followed by 'ff' for first-fit, 'bf' for best-fit and 'wf' for worst-fit or empty for largeServer();

RESC All Implementation

The implementation of RESC All is an important factor in Stage 2. Essentially the RESC command is one that is required for the client-server communication model to function as its main purpose is to send a request for resource (availability) information. This is important as this allows the client and the data structures implemented to obtain this information which in turn allows for the creation of algorithms. Conditions for the algorithms are based on the server state information that is obtained from RESC. This request of information is key to creating the algorithms as the conditions of the algorithms are dependent on the resource availability of the server in real-time.

A footnote on the implementation of the RESC would be mentioning that the command is sent by the client during the run() function. This means that the client is continuously requesting for server information and receiving information until the server has stopped sending information. As per this code, RESC had to be integrated into our current Client.java. The RESC All's purpose is to return information associated with each server to the client. This is evident in the Server.class where we parsed this in and appended this to an ArrayList to create the Algorithm.java class out of.

Katrina David (45308748) Sakura Mukhopadhyay (45435073) Trideep Lal Das (45532125)
Group 8 (Thursday 4pm-6pm)

Algorithm Design and Description

First-Fit Algorithm

The First-Fit Algorithm is an algorithm that scans through the computer memory from beginning to the end. It finds the first partition of memory large enough to accommodate the job given, then allocates the job to that memory partition. After this, there are two conditions we can take a look at; firstly being checking for leftover space which then becomes a separate free space and secondly, when the job size exceeds the biggest free space in terms of memory, it returns an error.

When looking at the FF algorithm, it is key to note that there are a few advantages and disadvantages to implementing this algorithm. The most apparent of the advantages is that compared to all other algorithms, FF is the fastest algorithm as it essentially just needs to find the nearest memory and then allocate the job to that partition. Although a point of contention for some, the biggest disadvantage to this algorithm is the wastefulness in terms of memory exhibited by this algorithm. The algorithm does not account for the size of the job compared to the size of the partition, leaving unused partitions of memory which is generally a rather small amount. As such, memory space is wasted, and jobs would frequently have to wait for another job to be completed to then be allocated to the memory.

Implementation

The first-fit algorithm was implemented into the job scheduler by Trideep, which requires the implementation of the 'RESC' action which essentially sends a for resource (availability) information. Touching briefly on RESC, which is essentially sent by the client to the server to get the server's resource information. The purpose of RESC for this algorithm is to essentially retrieve information on the cores, disk, memory and the state.

To then describe the actual workings of the first-fit algorithm, the RESC command is first required to be able to obtain the server information regarding the server ID/server type; this allows us to sort the server from smallest to largest which is the key first step to the algorithm(in this case, the function sortByID() is called to sort a given server based on its server ID).

Once the server has been sorted accordingly, we can then move to checking if the server has sufficient available resources to run the job. We use a loop in this scenario to loop through all the available servers one by one, checking to see if the server has the capacity (and is in the right state) to run the job. In this case the variables coreCount, disk and memory which have been appropriately named, reference the information of the job and the server, checking to see if the resources required by a job is less than that of the available resources on the server. If the job requires more resources than the resources currently available in a server, it will iterate through the loop and check the availability of the next server to be able to run the job.

One thing to note is that we continually check the server for the resources required to run the job. Taking this into consideration, you would need to continually iterate through the array from start to finish to find the next active server that can run the job(which appears as a second for-loop in the code, separate from the initial loop).

Best-Fit Algorithm

Design Considerations & Algorithm Description

The Best-Fit Algorithm is an algorithm that deals with allocating the smallest available free partition that meets the requirement of the requesting process. The algorithm when implemented in a job scheduler for distributed systems works by undertaking two key steps; a search for bR1 which is the local best fit in region R1 followed by a search of bR2 which is the best fit in region R2. After conducting a search through both regions, the best-fit algorithm checks the smallest and most appropriate block for the job and will return the overall best fit for the given job if it exists.

Thoroughly understanding the Best-Fit algorithm also involves analysing the advantages and disadvantages that affect the performance of the algorithm. The primary advantage of this algorithm in comparison to the first-fit and worst-fit algorithms is that it is fast and successful memory utilisation as it searches the smallest free partition first available. However, this algorithm also has a disadvantage, namely that it is slowest of them all and may have the tendency to fill up memory partitions with tiny purposeless holes.

Implementation

Katrina implemented the best-fit algorithm in Algorithm.java to the client side of the intelligent priority scheduler. Firstly, in order for all our Group 8 to implement their algorithms, RESC All had to be integrated into our current Client.java. For the best-fit algorithm, the purpose of the RESC is to return server information regarding the cores, disk space, available memory, the available time and the state of the server. Implementing the algorithm in a separate class (Algorithm.java) was a logical and uncomplicated solution as it became easier to create an instance of each class for important information that is received by the server as stated earlier.

To implement the Best-Fit algorithm, I have firstly declared the following variables so that they can be used in my algorithm: bestFit, minAvail, best and found. By using a for loop to iterate through the servers, we compare attributes of the server against the current job's coreCount, disk and memory. If the server passes these conditions, we declare the variable fitnessValue and initialize that to a deduction of the Server's coreCount against the current job's coreCount. After that, the algorithm will calculate the bestFit and minAvail values and compare the server's state which ensures that the bestFit has been found. In the event that bestFit is not found, the algorithm will then conduct calculations for the alternative server and will inevitably return the alternative server based on the initial resource capacity.

Worst-Fit Algorithm

Design Consideration & Description

The worst-fit algorithm follows the process of allocating job tasks to the largest memory space that is available and has enough storage space for the requested task. The algorithm selects the largest free memory space that is available and that has more memory space than the desired task's required memory space. It then stores the requested information directly into this vacant memory space. By implementing this algorithm, we observe the processor allocates any memory block that is vacant to the ongoing process. This leads to an increase in the probability of memory being wasted and hence is called the worst-fit algorithm. When the allocated tasks are of medium sized memory space tasks, the worst-fit algorithm is ideal and works efficiently. However, they break up the potential of using the largest free memory block for further tasks as these large memory spaces cannot be reallocated and end up being wasted and unused. It also causes external fragmentation and other algorithm schemes are usually higher choices in implementing into the distributed system.

Implementation

The worst-fit algorithm has been implemented by Sakura to the client side of the scheduler. As mentioned above, to implement the algorithms successfully, the RESC command has been integrated into the current Client.java file. The RESC command provides information of each server in the system and delivers it to the client. In this case, the RESC retrieves server information regarding the servers' cores, disks, memory, available time, boot-up time and the current state of the server.

In the program, I have defined coreCount as the size of the server based on the type of job task to be processed. The algorithm decides which available server the task is to be allocated to depending on its memory, disk and core size required for the requested task to be processed. The algorithm compares these values to the memory, disk and core values. It forms a calculation producing a value known as the fitness value. The fitness value in the worst-case algorithm is the variable that chooses which server will be the 'worst-fit' for the requested job to be allocated to. In the worst-fit algorithm I have implemented, the fitness value calculation is the difference between the coreCount value defined for the requested job compared to the coreCount value of the available servers. The algorithm checks the server array which stores information of the availability of the servers. The server availability is continuously updated in the client side through the RESC command. After checking each server, the highest fitness value is selected and we can say it has the most suitable worstFit value. The requested job is then scheduled to the server completing the requirements for a worstFit algorithm scheme. As the highest fitness value is selected in the worstFit algorithm, all the servers that the jobs are scheduled to are large servers can be seen in the output when the worst-fit is implemented.

References

Castillo, C., Rouskas, G. and Harfoush, K., 2007. *Efficient Implementation Of Best-Fit Scheduling For Advance Reservations And Qos In Grids*. [online] Rouskas.csc.ncsu.edu. Available at: <https://rouskas.csc.ncsu.edu/Publications/Conferences/EVGM-Castillo-2007.pdf> [Accessed 1 May 2020].

David, K., Mukhopadhyay, S. and Lal Das, T., 2020. *COMP3100*. [GitHub Repository] https://github.com/Katrina1999/COMP3100.git.

GeeksforGeeks. 2020. *Program For Best Fit Algorithm In Memory Management - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/program-best-fit-algorithm-memory-management/> [Accessed 2 May 2020].

Tutorialspoint.com. 2020. *OS Memory Allocation Q & A #2 - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/operating_system/os_memory_allocation_qa2.htm> [Accessed 1 May 2020].

T. Singhal, "First-Fit Allocation in Operating Systems - GeeksforGeeks", GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/first-fit-allocation-in-operating-systems/> [Accessed: 05- May- 2020]

S. Amjad, "First fit , Best fit, Worst fit", Slideshare.net, 2020. [Online]. Available: <https://www.slideshare.net/ShahzebAmjad/first-fit-best-fit-worst-fit> [Accessed: 05- May- 2020]