**Improving the efficiency of distributed system through the scheduling algorithm**

Seonghun Park (44401213)

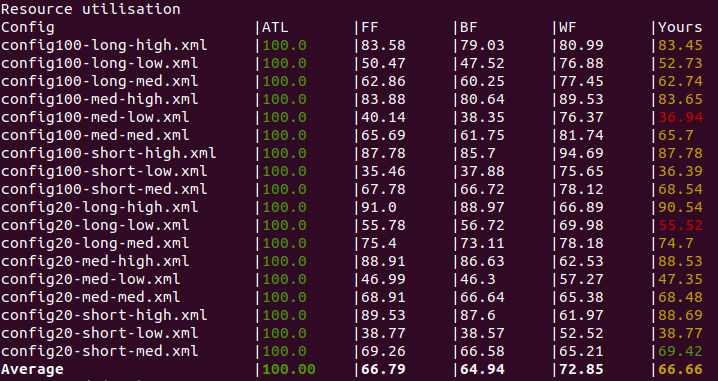
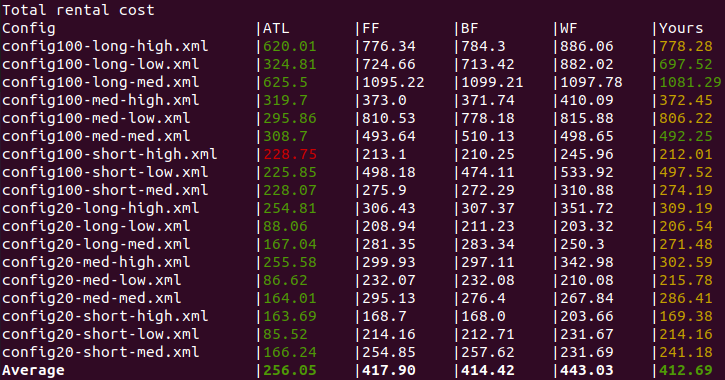
**Introduction:**

The distributed system implemented through the previous project is designed to perform the handshake process with the server through the vanilla version of the client, read data from the server, and allocate all jobs to the server with the largest core in the server list. Unfortunately, this AllToLargest algorithm has very low efficiency because it allocates all jobs to one server. In order to solve this problem, a client that introduced FF (First Fit), BF (Best Fit), and WF (Worst Fit) algorithms were used to calculate and compare the turnaround time, rental cost, and resource utilization of each algorithm. It. However, a series of efficiency problems were also found in the above algorithm. This project aims to increase the efficiency of the server by introducing a new algorithm suitable for distributed systems in order to improve this. In particular, we focus on minimizing the average turnaround time, maximizing average resource utilization, or minimizing the total server rental cost.

**Problem Definition:**

So far, BF, FF and WF have been applied to this project. First of all, in FF, a job is allocated to the first memory space found. When the process is allocated in the case of BF, the job is allocated where the least space is left. Conversely, WF allocates to the largest space. It's built on the exact opposite of Best Fit, where you have to leave the largest amount of space left, and there is a high probability that other processes will be allocated there. First Fit and Best Fit were found to be better than Worst Fit in terms of speed and memory efficiency. There is not much difference between First Fit and Best Fit in terms of memory efficiency, but of course, First Fit is faster than Best Fit. This is because the first Fit allocates as soon as the first free space occurs, and Best Fit calculates all the free space to find out where the least free space is left when the process is allocated. When comparing in terms of speed and memory efficiency, in conclusion, the first Fit> Best Fit> Worst Fit is the most efficient. However, whether it is First Fit or Best Fit, external fragmentation occurs in the end. That is, memory is wasted. This project aims to improve the shortcomings of Best Fit among these three algorithms, resulting in higher efficiency.

**Algorithm description:**

While the existing BF algorithm determines only the fit function of the server and assigns jobs to the server, the improved BF, Optimized Fit, determines the number of jobs currently waiting in the server and gives jobs to the server. The Fit function is the number of cores in the server minus the number of cores that need a job. In other words, the server has a smaller fit value as it is closer to the number of jobs it needs. If Fit is 0, the server has precisely as many cores as it needs the job. The smaller the fit value, the higher the server's suitability. The algorithm first checks the server list and the job to find the fit value of each server and then assigns the job to the server with the minor waiting list among the servers with the smallest fit value. In other words, the server with the highest fit and shortest queue will be assigned a new job. Such an algorithm increases the efficiency of server utilization and reduces rental cost by giving jobs to various servers.  

This can be seen in the chart below comparing resource utilization and total rental cost.

**Implementation details including data structure(s) used:**

JobParser records the following information of the job: JobType, JobId, JobCoreCount, JobMemory, JobDisk.

Server records the following information from the server: ServerName, ServerId, ServerCoreCount, ServerJobsWaiting.

**Conclusion:**

The concept of fit used by the BF algorithm was added to assigning jobs to the server with the lowest weight, but inefficient memory wastage is still occurring. In order to solve this problem, it is necessary to introduce an algorithm such as Compaction, which increases efficiency by gathering memory holes between processes. After all, the efficiency of a distributed system is how to manage memory holes. This is because memory is wasted when a specific process is larger than the hole and cannot enter the hole. One of the algorithms to solve this problem is paging. Paging uses a fixed partitioning method that cuts each process in units of physical addresses in memory and arranges them so that no space is left behind. That is, not only simply assigning the job to the server but also improving the job level by mobilizing internal fragmentation of the process to develop a distributed system with higher efficiency.

Reference:

<https://github.com/44401213SeonghunPark/COMP3100_Stage2/tree/main/ds-sim-master/src/pre-compiled>