Server Farm Simulation with Queues

Abstract

In this project we designed a job and server system to evaluate the performance of different job dispatcher types in a server farm simulation. This project more than anything served to test our queue data structure. The key purpose of this assignment was to apply use our queues to analyze the efficiency of various dispatching strategies, with a focus on their impact on average waiting times. The simulations were conducted with different dispatcher types and various job sequences of various lengths. The base simulation uses simulated time to process jobs. However, we also implemented a simulated server method that uses threads to run each job processor. This method utilizes a timer data structure that sends alerts to a job dispatcher at specified times. Once these alerts are received, the dispatcher sends out the corresponding job to its assigned server. Due to the real-time processing and real-time dispatching some amount of lag is introduced; however, this is generally nanosecond to microsecond lag.

Results

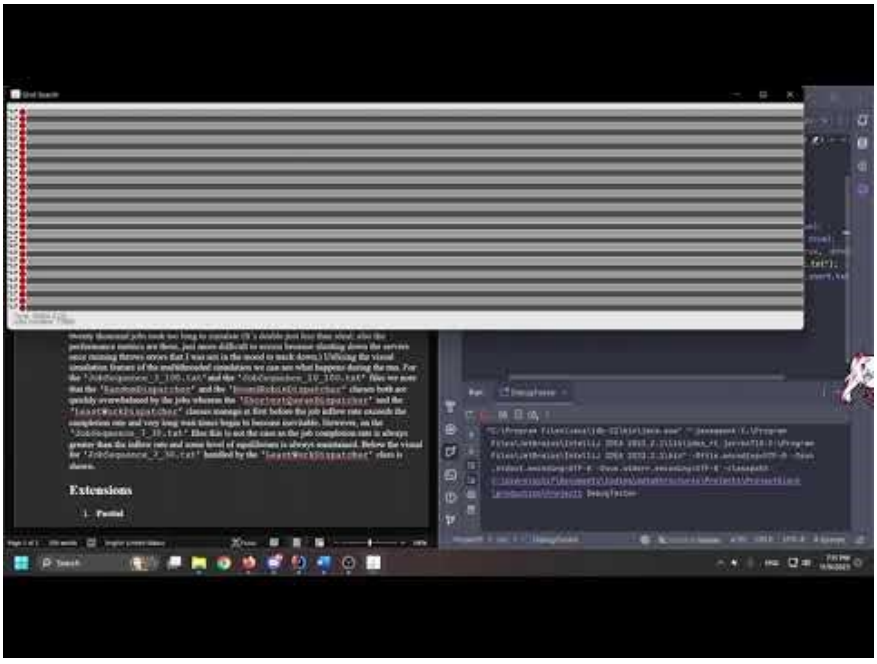
In the following table, we present the average waiting times (in milliseconds) obtained for each dispatcher type when run with 30 servers and the three provided job sequence files. The waiting times were computed by simulating the server farm and observing how each dispatcher type managed incoming jobs. In this table file names are shortened for brevity.

Dispatcher	3_100_short	10_100_short	7_30_short
'Random'	19035.571573984376	152.68130125	35.3873746875
'RoundRobin'	18561.496547890623	107.52086828125	30.10963421875
'ShortestQueue'	18065.642185625	99.850945859375	30.047984296875
'LeastWork'	18052.716426875	99.850945859375	30.047984296875

The following table represents the same tests done with the long versions of these files; their names too have been shortened for brevities sake.

Dispatcher	3_100	10_100	7_30
'Random'	6663818.628070798	12171060.821952388	5.04740422578125
'RoundRobin'	6601034.51239276	12121103.41596075	4.30333335234375
'ShortestQueue'	6595763.382918036	12121081.742753785	4.2947563546875
'LeastWork'	6534710.454442539	11998122.356208075	4.2947563546875

Using both the shorter and the longer files we come to the same conclusions. The `'RandomDispatcher'` class consistently came in as the slowest dispatcher out of the bunch, in some cases being as much as 50% slower than its competition. The next slowest was generally the `'RoundRobinDispatcher'` class; however, this class performed considerably better than the random dispatcher. Finally, the `'ShortestQueueDispatcher'` and the `'LeastWorkDispatcher'` classes came in ranked pretty similarly; however, the `'LeastWorkDispatcher'` class edged out over the competition when average job sizes were longer. This pattern was maintained in the multithreaded implementation; however, the fact that the multithreaded simulation runs in real time makes it such that any job queue exceeding twenty thousand jobs took too long to simulate (It's doable just less than ideal; also the performance metrics are there, just more difficult to access because shutting down the servers once running throws errors that I was not in the mood to track down.) Utilizing the visual simulation feature of the multithreaded simulation we can see what happens during the run. For the `'JobSequence_3_100.txt'` and the `'JobSequence_10_100.txt'` files we note that the `'RandomDispatcher'` and the `'RoundRobinDispatcher'` classes both are quickly overwhelmed by the jobs whereas the `'ShortestQueueDispatcher'` and the `'LeastWorkDispatcher'` classes manage at first before the job inflow rate exceeds the completion rate and very long wait times begin to become inevitable. However, on the `'JobSequence_7_30.txt'` files this is not the case as the job completion rate is always greater than the inflow rate and some level of equilibrium is always maintained. Below the visual for `'JobSequence_7_30.txt'` handled by the `'LeastWorkDispatcher'` class is shown.



All in all, these results are pretty much as expected. The least work dispatcher distributed the work more evenly than the other methods and as such is faster. When the jobs are shorter, they can be handled quicker and less of a queue is allowed to build up. As such the average waitime goes down.

Extensions

1. **Real-time Thread-based Simulation:**
    - I created alternative versions of some of the methods that allow this whole project to use simulated servers instead of the traditional implementation intended in this project.
    - Each of these servers runs on its own thread and receives and processes jobs in real time.
    - In addition to this there is a utility thread running at all times which serves as a timer. This timer tracks start times for each server relative to the job dispatcher start time (actually technically it's the timer start) this is all in nanoseconds. The timer both handles sending alerts to the dispatcher to initiate the dispatch of the next job, as well as providing a central time reference which all of the servers can call to set their own local time. On top of all this the timer also updates the server local time for each server "subscribed" to its updates.
    - In addition, I implemented a slight tweak to visualization that allows these simulated servers to be better simulated with the real-time simulations. Basically, a time-based updater that updates the visualization every microsecond (I think).
    - One note, I disabled the simulation on the `'ServerFarmSimulation'` class because you would either run out of threads or have to close threads both of which posed problems.
    - Also, I have a 64 thread (semi-server grade) processor, this code will not run on lower end setups, much less with all 30 servers, so tweak values to adjust for this.
2. **Large File Handling**:
    - This wasn't super intentional, as my code just happened to be pretty efficient on my first attempt. However, I made an alternate "work remaining" function that was more efficient to allow the `'leastWorkDispatcher'` class to dispatch more efficiently. All the large versions of the files work even with the simulated versions, although, the real-time nature of the servers would make that take forever as one time unit equals one millisecond. This conversion was a necessity based on the fact that most operating systems and Java don't go beyond nanosecond precision and the jobs had enough sig figs to force me to use milliseconds.

Reflection

This project was a comprehensive exploration into the performance of different job dispatcher types in a simulated server farm environment. The primary focus of the project was to evaluate the efficiency of various dispatching strategies and their impact on average waiting times, utilizing queue data structures.

The implementation involved simulating the server farm under different dispatcher types and job sequences of varying lengths. Two key methods were employed: a base simulation using simulated time to process jobs and a simulated server method using threads to run each job

processor in real-time. The latter method incorporated a timer data structure for alerts and introduced some nanosecond to microsecond lag due to real-time processing and dispatching.

The results, presented in tables for both short and long job sequence files, consistently showed the `'RandomDispatcher'` class as the slowest, often up to 50% slower than other dispatchers. The `'RoundRobinDispatcher'` class performed better than the random dispatcher, while the `'ShortestQueueDispatcher'` and `'LeastWorkDispatcher'` classes ranked similarly, with the `'LeastWorkDispatcher'` having an edge in longer average job sizes.

The visual simulations provided insights into the dispatcher behavior under different job sequence scenarios. Notably, the `'LeastWorkDispatcher'` demonstrated better work distribution, resulting in faster processing times. The reflection correctly interprets these results, highlighting the effectiveness of the least work dispatcher in evenly distributing work, especially for shorter jobs that can be handled quickly.

The extension of the project included a real-time thread-based simulation, allowing servers to run on separate threads and process jobs in real time. The implementation involved a timer thread for alerts, time reference, and server synchronization. A time-based updater for visualization was also incorporated. However, limitations were acknowledged, such as potential thread issues and system requirements for optimal performance.

Another extension focused on large file handling, with an efficient "work remaining" function enhancing the `'LeastWorkDispatcher'` class's dispatch efficiency. The project successfully handled large files, even in simulated versions, although the real-time nature of the servers could make processing time-consuming.