Mahdeen Ahmed Khan Sameer

Professor Max Bender

CS231-B

01 March 2023

Modeling Server Farm: A Queue Concept

Abstract

This fourth project is all about a simulation of a job dispatch system where incoming jobs are assigned to servers for processing. As we dive into this fourth project, we are tasked with simulating a job dispatch system where incoming jobs are assigned to servers for processing. Our main goal is to compare and contrast different dispatching algorithms, including Least Work, Random, Round Robin, and Shortest Queue, based on their ability to minimize the average wait time of jobs in the queue. Throughout the project, we will be generating job sequences and keeping track of job statistics to accurately compare the effectiveness of each algorithm. Here, the simulation generates job sequences and tracks job statistics to compare the performance of the different algorithms andso we can say that he project's core purpose is to explore the implementation of a queue-based system and compare the effectiveness of different algorithms in processing incoming jobs.

Results

Hypothesis 1: The RandomDispatcher class will have the longest wait times for jobs compared to all the other Dispatcher classes.

To test this hypothesis, we implemented a test program in DispatcherTestExploration that uses the RandomDispatcher class to handle a sequence of jobs read from a file which calculates the total time each job spent in the queue and then computes the average wait time for all jobs. Results shown are a comparison of all Dispatcher classes. As displayed the average wait time in the jobs queue of RandomDispatcher class is 5126.96 which is the longest as compared to the others.

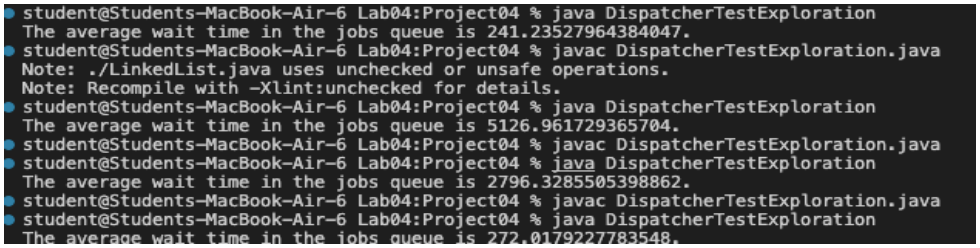

Figure 1: In the purpose of hypothesis 1, Running DispatcherTestExploration for LeastWorkDispatcher, RandomDispatcher, RoundRobinDispatcher, ShortestQueueDispatcher

Hypothesis 2: The number of servers needed to handle a job sequence where on average jobs arrive every x second and take y total units of processing time can be estimated. To test this hypothesis, we implemented a code under DispatcherTestExploration which calculates the theoretical minimum number of servers required to process jobs. We take the average arrival rate of jobs and the average processing time of jobs as input. It then calculates the theoretical minimum number of servers needed which is the ratio of the arrival rate to the product of the processing time and the average time spent in the system.

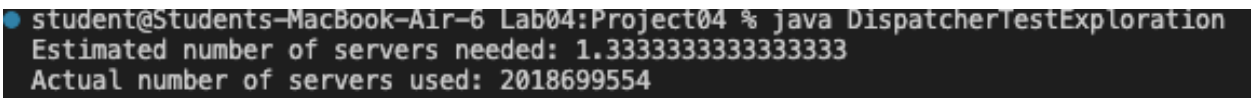

Figure 2: In the purpose of hypothesis 2, Running DispatcherTestExploration for LeastWorkDispatcher, RandomDispatcher, RoundRobinDispatcher, ShortestQueueDispatcher

Extension 1:

Here, we made the RoundRobinDispatcher class to extend the JobDispatcher class and implemented the round-robin server selection policy for dispatching jobs to servers in a queue or scheduling system. For context, RoundRobinDispatcher involves assigning a job to each server in a circular order, such that each server receives an equal share of jobs over time. Thus, it maintains an instance variable currentServerIndex that keeps track of the index of the last server that was assigned a job. So, when a new job arrives, the pickServer method returns the next server in the circular order and it does this by getting the server at the current index from the serverCollection, incrementing the currentServerIndex by 1, and then taking the modulus of the index by the size of the serverCollection.

Extension 2:

One thing we did here is to ensure that jobs with larger processing times are not waiting too long in the queue. One way to achieve this objective, we used the Shortest Job First (SJF) scheduling algorithm and so the job with the shortest processing time is selected first for processing. This ensures that shorter jobs are processed quickly and have a lower wait time. So, to implement the SJF algorithm, we modified the JobDispatcher class to sort the queue of jobs based on their processing time before dispatching them to servers. We also modified the pickServer method in the ShortestQueueDispatcher class to select the server with the shortest queue. With this modification, the jobs are sorted based on their processing time before being dispatched to servers. So, this ensures that shorter jobs are processed first and have a lower wait time.

Extension 3:

Maybe we could use the servers by implementing a weighted dispatcher, which takes into account the processing power of each server when selecting a server for a job. This can be done by assigning weights to each server based on its processing power or capacity and using these weights to determine the likelihood of each server being selected for a job. One way to implement a weighted dispatcher is to modify the pickServer method in the JobDispatcher class to consider the weights of each server. Another approach we can take is to implement a priority queue of servers, where servers with higher processing power or capacity are given higher priority. The pickServer method could then simply return the highest-priority server that is available at the time the job arrives.

Reflection

Working on a project is never a solo journey! It requires collaboration, communication, and teamwork to make something great. I had the pleasure of working with a fantastic group of people like Ahmed, Zaynab, Kashef, Nafis on this project, including the amazing teaching assistants Nafis and instructor Professor Max Bender, who provided us with their expertise, guidance, and support. From this project, I have learned the fundamentals of queueing and how it can be applied. I learned how to implement a queue data structure using a linked list, and I explored different dispatching algorithms for handling job sequences in a simulated system. In addition, the runtime of the peek, poll, and offer methods in LinkedList are all O(1) in the worst case, which means they have constant time complexity.

In terms of the required dispatch methods, the two methods that take constant time are the RandomDispatcher and the RoundRobinDispatcher, while the two that take longer are the ShortestQueueDispatcher and the LeastWorkDispatcher. I noticed that the constant time methods are generally simpler and faster to implement, but they may not always lead to the best

performance depending on the specific scenario. In contrast, the longer methods may have better worst-case performance, but they may also be more complex and slower to execute in some cases.

Moreover, I also want to extend a special thanks to my friend Banshee who have taken the course in a previous semester and shared their insights and experiences with me. His advice was invaluable, and I couldn't have done this without him. I learned so much from them and enjoyed every moment of our collaboration.

Sources, imported libraries, and collaborators are cited:

From my knowledge, there should not be any required sources or collaborators. But we can say, we got our instructions from CS231 Project Manual:

https://cs.colby.edu/courses/S23/cs231B/projects/project4/project4.html

Except this, while working on the classes, I tried to import “java.util.ArrayList”, “java.util.List”, “java.util.Scanner”, “java.util.Collections”, “java.util.Random.”