Load Balancing With Random Choices

CSC 446 Final Project

David J - V00855972
Chenghao L - V00841275
Ayush M - V00835885

Table Of Content

Table Of Figure

Table Of Table

1.0 Abstract

The goal of the project is to simulate a system with several parallel servers that use a load balancer to process each event. The desired outcome is to compare different load-balancing algorithms and determine the optimal algorithm that minimizes the processing time for each event. This simulation is written in Python to model the system in question and to try and collect enough statistics to determine the optimal algorithm. The insight we came out with after analyzing is that: RandMin is better overall, than RoundRobin and PureRand.

2.0 Brief problem (Project Description)

The problem in our project deals with simulating a system with a set number of customers and numbers of parallel servers with an intermediary load balancer. This load balancer will utilize 3 methods to distribute the incoming customers: PureRand, RoundRobin, RandMin. Each method consists of a different process for distributing the customers to a server. The final objective is to run the simulation on the 3 methods and analyze the output to determine whether one method is better than the other or just more efficient in one category. Additionally, testing will be conducted with manipulating the parameters including trying out different arrival rate, service rate, number of queues.

3.0 The simulation model

The system in this project can be modeled by multiple M/M/1 queues. The model consists of several nodes: A customer and multiple servers. Each customer is served by one server. Each server is modeled as a FIFO queue system. When a customer enters the system, they must first go through the load balancer. The choice of the server is determined by the load balancer.

The customer arrivals follow a Poisson process with a mean arrival rate of lambda. The service time for each server follows an exponential distribution with rate parameter mu. For the purposes of this study, the processing time of the load balancer is ignored. The load balancer uses different methods to dispatch each customer to a server. In this project, we consider the following randomized strategy as follows:

RandMin (RM):

1. The load balancer randomly selects d servers ($d < m$)
2. The load balancer checks the queue length of the selected d servers
3. The load balancer dispatches the incoming customer to the server that has the shortest queue length.

The goal of this project is to compare the performance of the method above with the following two methods and determine the most optimal algorithm for processing customers.

- **Purely Random (PR)**: The load balancer randomly selects one server from the m servers and dispatches the incoming customer to the selected server.
- **Round-robin (RR)**: The load balancer uses the round-robin method to dispatch incoming customers.

4.0 Simulation parameters

Individual Parameters

The individual parameters are those that change with each run, they are to rigorously test the method. All parameters are instantiated at the beginning of the program and are static for that run. (make bullet point)

- Arrival Rate
    - The process for getting interarrival time is implemented to mimic the Poisson distribution process using the numpy library. The values are set to many different variables to test out the 3 methods: whether the distribution of customers to queues makes an impactful change.
- Service Rate
    - The exponential distribution process determines how long the service takes to complete for each customer in the queue. Depending on the run, the values can range from 5 to 25.
- Number of Queues
    - The total number of parallel servers, depending on the run, is set to 4, 7, 10, 15, and 25 parallel servers.
- Load Balancer Method
    - Each run will be purely based on either PureRand, RoundRobin, or RandMin.
- Number of customers
    - For testing purposes, the range can be from 10 to 1000.
    - However, for final results, the value will be set at 400.

Queue Characteristics

For each queue, the capacity will be finite, although they were monitored to see if the value needed to be larger to accommodate a queue logjam. Therefore, we assumed balking might occur, where a customer decides not to join a queue if it is too long, so if the load balancer puts too many customers in an already busy queue, customers may be skipped and not recorded. Furthermore, jockeying, where customers leave their current queue for another queue if they have waited too long for a service, and reneging, where a customer leaves the queue if they have

waited too long for a service. Thus, jockeying and reneging are not implemented for our simulation.

Analysis Parameters (Goals)

These parameters we need for analysis in the section analysis later in the paper.

1. Interarrival times (**λ**) & Exponential times (**μ**): This is to ensure that these two times are following the distribution.
2. Max Queue Length: Every time an event is enqueued, check the size of the queue against the current maximum queue length, and if it is longer, then update the value.
3. Average Queue Length: Essentially, the average queue length among the **m**-number of servers. This will be calculated by taking the sum of queues and dividing it by the total number of servers available in the run.
4. Average System Time: Simply, we add all the time spent in the system by each customer, and divide by the total number of customers in the simulation run.

5.0 Methodology

The simulation was built using a discrete event system simulation modeled by the Java program provided in the course. The program provided simulates a simple M/G/1 queue and was translated to Python and modified to simulate several M/M/1 queues and to implement the load balancer.

In order to generate and simulate customer arrivals and departures in real-time, an implementation of a splay tree was used along with a node class in order to keep track of each event. The Python queue module was used to simulate each FIFO queue belonging to a server. The simulation consists of four primary methods:

1. The Load Balancer
2. Processing Arrival Events
3. Scheduling Departure Events
4. Processing Departure Events

The Load Balancer

The load balancer is a simple implementation; essentially, it calls PureRand, RoundRobin, and RandMin, and they return a random, and hopefully idle and unoccupied, server. Finally, the load balancer enqueues the customers' ID into the server that was returned by one of the methods.

```
def load_balancer(self, method):

    if method == "PureRand":
        queue = 'Q' + str(self.PureRand())
    elif method == "RoundRobin":
        queue = 'Q' + str(self.RoundRobin())
    else:
        # returns the server 'Q1','Q2',..
        queue = self.RandMin()

    return queue
```

Fig 1. Code implementation of load_balancer

Processing Arrivals

This function is responsible for processing the arrival of a new customer in the simulation. It takes an event object as an argument, which contains information about the customer's arrival, such as the time of arrival and the queue to which the customer belongs. The function first adds the customer to the selected queue and increments the queue length. If there are no customers currently being served, it schedules the departure of the customer using the ScheduleDeparture() function. The function also updates the total busy time of the servers and schedules the next arrival of a customer using the Poisson distribution

Scheduling Departures

This function schedules the departure of a customer after being served. It generates a random service time using the exponential distribution and calculates the departure time as the sum of the customer's arrival time and the service time. The function then inserts the departure event into the futureEventList, a Splay Tree data structure, with the departure time as the key, the type of event (departure), and the server queue that the customer is assigned to. The function also updates the number of customers being served and the queue length.

Processing Departures

The ProcessDeparture function handles the departure of a customer from the system. It first retrieves the queue associated with the event from the DICT_OF_SERVERS dictionary. Then, it removes the finished customer from the queue using the get() method. Next, it checks whether there are still customers in the queue. If there are, it schedules the next departure using the ScheduleDeparture function. If there are no customers left in the queue, it sets the "busy" status of the queue to False. The function also calculates the response time for the finished customer by subtracting their arrival time (stored in finished.key) from the current simulation time

(self.clock). It then updates the sum of response times, the total busy time, and the number of departures accordingly. Finally, it sets the lastEventTime to the current simulation time.

Load balance, Process Arrival, Scheduling Departures and Processing Departures work together to simulate a multi-server queuing system where customers arrive, get served and depart according to the specified arrival and service rate(different combinations) based on what we have defined from the main function.

6.0 Simulation Setup

Each experiment was specified by the method selected by the load balancer and parameter values used. The three sets of parameter values are listed in Table #. Every experiment consisted of five runs and the same five random seeds were used in every experiment. The total number of experiments is six. In an ideal scenario, each experiment would be run multiple times over each load balancing method to remove any resulting artifacts in the experiment.

While PureRand and RoundRobin are relatively simple and intuitive implementation, we wanted to prove our model for RandMin works correctly, and as we can see the additions to the queues are random and follow the simple objective of ensuring a server which is already busy does not receive more customers, while other servers are relatively idle.

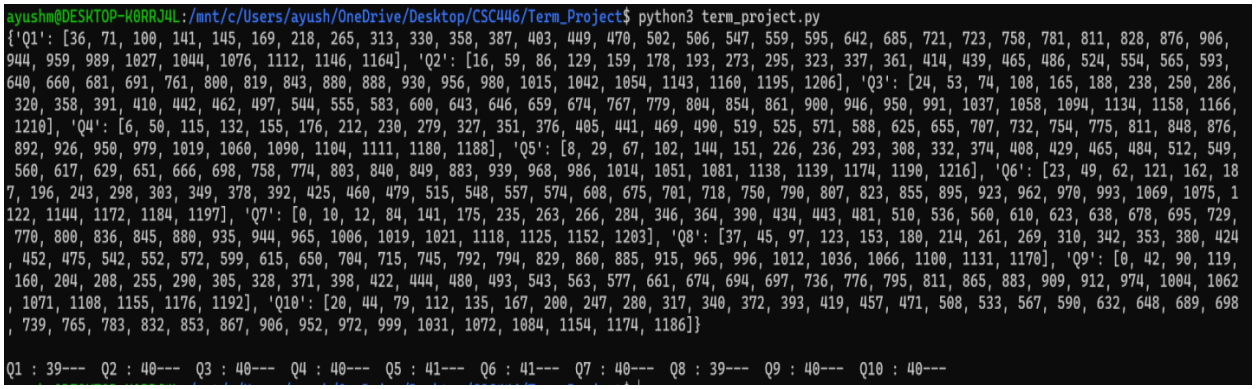

Fig 2. Ensuring RandMin works as intended

7.0 Data Collection and Statistics

For each simulation, run output variables (within-replication analysis) were calculated and recorded. The sample mean, sample variance and 95% confidence intervals are computed for each variable of interest (between-replication analysis). Each load balancer method is compared under each parameter set using the variance reduction technique of estimating the mean difference between models (model comparison). The output variables include the maximum queue length across all the queues, the average queue length across all queues, and the average amount of time each customer spent in the system. The latter two variables were calculated by summing the variable of interest and dividing by the total number of data points collected.

8.0 Analysis

Each configuration was run for five different seeds and each simulation run consisted of the following combination of parameters.

#	ARRIVAL_LAMBDA_RATE	SERVICE_RATE	MAX_SERVERS
1	5	5	4
2	8	15	7
3	20	22	10
4	15	10	15
5	25	22	20

Table 1: Parameter Combinations

The average results across each run are recorded in the ables below.

Run #	Algorithm	Max Queue Length	Avg Queue Length	Avg System Time
1	PureRand	4.6	0.9703618706917645	2.5011076202017373
2	PureRand	4.75	1.1780028363546675	2.585418431559955
3	PureRand	4.0	0.8855607506980973	2.538781283566849
4	PureRand	3.5	0.8928003249637404	2.6353840772202553
5	PureRand	5.0	1.0727391292072483	2.667554448751006
1	RoundRobin	2.6	0.6999339040011432	2.6329067668554127
2	RoundRobin	2.5	0.6991361300014289	2.631176116854397
3	RoundRobin	2.5	0.6991361300014289	2.6298526786183256
4	RoundRobin	2.5	0.6991361300014289	2.6288078589582695
5	RoundRobin	2.5	0.6991361300014289	2.627962052566796
1	RandMin	1.4	0.5556993979645353	2.616396164525963
2	RandMin	1.75	0.5992926613616268	2.6116086974820996
3	RandMin	1.25	0.5463881826948365	2.643628883167975
4	RandMin	1.75	0.5653240777054409	2.646161074593736
5	RandMin	1.5	0.5490444275245608	2.637280928579973

Table 2: Average measurement results across parameter combinations

Method	Max Queue Length	Average Queue Length	Average Time in System
PureRand	0.372	0.015610616	0.0046314903
RoundRobin	0.002	0.00000012728867	0.0000038322848
RandMin	0.04825	0.00046161394	0.0002545226

Table 3: Average variance results across parameter combinations

Method	Max Queue Length	Average Queue Length	Average Time in System
PureRand	0.0000000, 2.217836055	4.130115098, 4.629884902	2.443889893, 2.716110107
RoundRobin	0.6105572809, 0.7894427191	2.519286449, 2.520713551	2.632084756, 2.639915244
RandMin	0.1266823473, 1.005317653	1.487029594, 1.572970406	2.632084756, 2.659907529

Table 4: 95% confidence interval results across parameter combinations

Based on the results above, the sample means, variances, and confidence intervals are consistent with the expected results.

Each method went under 5 runs with a single seed, and with 5 runs of different parameters with that particular seed. The parameters that were changed are the **λ**, **μ**, and **m.** The insight we came out with after analyzing is that: RandMin is better overall, than RoundRobin and PureRand.

For PureRand the average queue length was obviously the highest, and average system time was very close to each other, since it is purely random the averages converge to the averages.For RoundRobin, predictively it is better than PureRand, since it cycles through all the queues, meaning every queue is ensured to be busy. For RandMin, it had the shortest queue length, as it smartly picks the shortest queue length therefore, it is more effective in keeping queue lengths small.

9.0 Conclusion

Summary - Quantitatively

From the mean and variances, we can conclude the workload of servers distributed by PureRandom is inefficient, RoundRobin improves upon it, but still, we could see a long line up in a single server, if a single customer has a long service time, thereby holding up the entire queue. Finally, we can see that RandMin is the most efficient method to pass workload in a fair manner for both servers and customers, as they are not stuck in a long queue.

Conclusion based on simulation study

In this paper, we went over 3 methods that are available for a load balancer and compared their efficiencies, including talking about each method's unique strengths and weaknesses. As we kept adding more servers, more inadequacies presented themselves for PureRand and RoundRobin.

Suggestions for Improvements

Additional testing on PR, RR, and RM is required to find that average system time is much more efficient on RM. Because PR had the overall lowest average system time in our simulations, we would need to conduct additional tests to prove that in a real-world scenario RM would be the best implementation.