Comparison of Load Balancing Methods in a Parallel Server Queuing System

Cam Rohwer, Owen Thurston and Andrei Mazilescu

Department of Computer Science, University of Victoria

CSC 446

Professor Kui Wu

April 11, 2023

Introduction

Load Balancing is a method of distributing traffic across a number of servers or other resources. It is often used to increase capacity and reliability of queuing systems. Simple forms of load balancing such as Round-Robin(RR) or Purely Random(PR) are easily implemented, but not always optimal. A Random Minimum(RM) approach may be used, assigning server loads based on randomly selecting servers and assigning to the shortest queue of those servers.

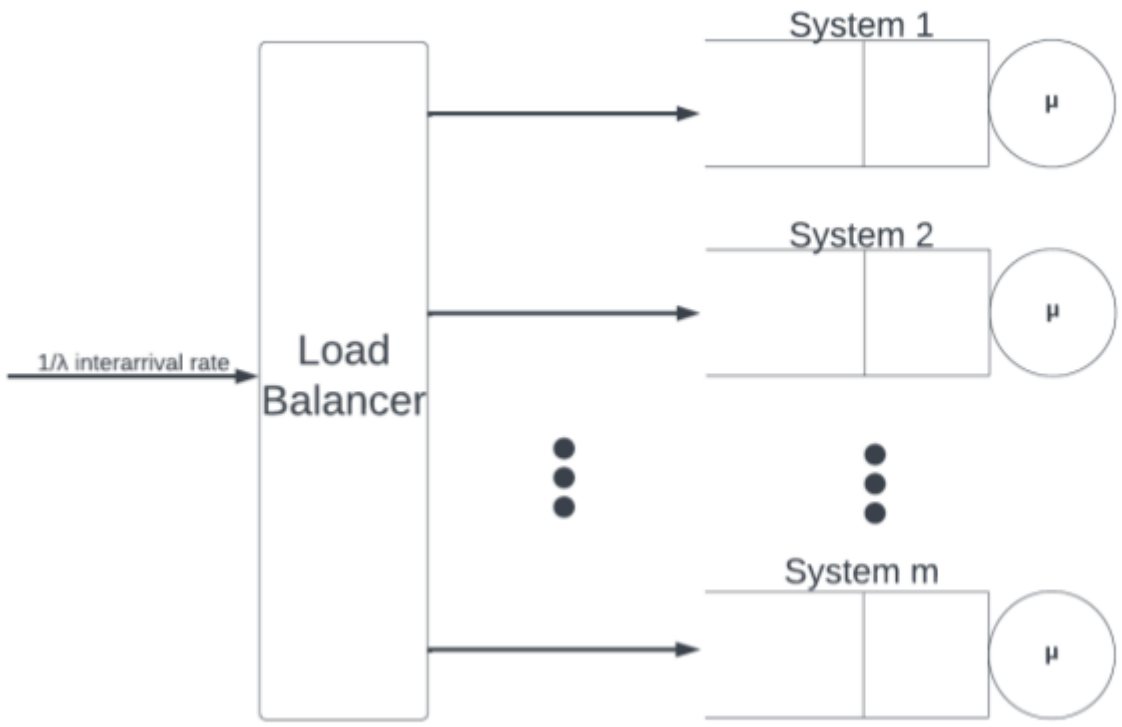

Figure 1: Diagram of a load balancer distributing traffic to M | M | m parallel servers

The goal of this project was to implement the theory and tools learned in CSC446 and to analyze the performance of an RM load balancing approach. Comparison of RM, RR and PR load balancing was done under similar parameter configurations to compare the efficiency of these forms of load balancing.

Simulation Model

To model load balancing used in systems such as DNS, we can use parallel servers working to serve customers. This simulation uses an M|M|m model: *'m'* number of FIFO (First-in, First-out) Queueing/server systems with exponential inter-arrivals and exponential service times. We will use the three assignment methods discussed previously for the load balancer in order to dispatch a customer to a server.

Simulation Parameters		
Number of Servers	Total Servers	m
RandMin Servers	Number of Servers selected by RandMin	d
Mu	Mean Service Time	μ
Lambda	Arrival Rate	λ

RandMin randomly selects *'d'* servers from the total *'m'* servers, checks queue length of each respective server and assigns the customer to the server with the shortest length. PureRand randomly selects one server and dispatches a customer to that server regardless of queue length. Round-Robin dispatches customers by the order of servers. For the sake of simplicity, the Load Balancer's response and service time are instantaneous, negating the need for an initial arrival queue.

Methodology

All the coding was done in Python 3.10. Much of the code was originally sourced from Java code that had been translated during class assignments. As well, a Python Splay Tree implementation was adapted from user: anoopj's code which was found on Github. The single server queueing system code that was introduced during the semester was modified to allow multiple servers to run concurrently, with additional system queues being maintained in the main simulation program.

After baselines are run on PR and RR with 5 CRN (common random numbers) seeds, the RM simulations can be run. Three separate runs of different parameter value sets were run with the same CRN seeds.

Set	d	μ	λ
A	1	1	6
B	3	1	6
C	5	1	6

The parameters were set to allow for both testing of the efficiency of RM, as well as to compare to the baseline runs. For all simulations, 6 servers were run in parallel for 10,000 customers. Mean service times of 1 and mean arrival rate of 6 were chosen to

create a theoretically stable system with 100% server utilization assuming optimal load balancing. Full server utilization permits better comparison of the three load balancers.

$$\rho = \frac{\lambda}{m\mu} = \frac{6}{6 \cdot 1} = 100\%$$

5, 3 and 1 were selected for the d values. These values were selected to be able to compare the change in efficiency as the number of servers available to the load balancer increases. 5 was selected to allow the most possible options to select the shortest queue, while still being less than the total number of servers to maintain the random aspect of the selection. 1 was selected to compare to the PR load balancing method. 3 was selected as an intermediate option.

Results and Analysis

Our simulation outputs results for all sets of runs needed for comparison at once. These results are exported as CSV files for statistical analysis. Maximum workload, average workload and average system time are recorded for each run. Our results for each set of parameters and each random number seed are aggregated in a spreadsheet. The Excel spreadsheet of this data is included alongside this report.

Seed	Load Balan	Avg Q Len	Max Q Len	Avg TCSS
0	RandMin	94	68	29.30517
0	PureRand	202	105.1667	49.70033
0	RR	78	53	21.19434
1	RandMin	88	57.16667	23.67421
1	PureRand	106	71.16667	29.67766
1	RR	101	65.33333	29.75616
2	RandMin	89	56.5	22.10934
2	PureRand	146	90.5	44.78702
2	RR	83	59.83333	26.69382
3	RandMin	133	96.5	43.83544
3	PureRand	115	71.33333	29.2138
3	RR	62	47	17.33406
4	RandMin	95	57.66667	25.27557
4	PureRand	95	67.16667	25.44311
4	RR	98	60.16667	27.54129

Figure 2: Example of Performance Metrics output of Set A (d = 1)

For each simulation run, output variables were returned as a table of Customer Numbers, Arrival and Departure Times, Service times and Server Numbers. In addition the performance metrics for each run are outputted for analysis.

Runs with 5 CRNs were performed for each of the load balancing methods for 10,000 customers. This was done for 3 sets of parameters, with the *'d'* value being used for RM being changed in each set. The following tables present the cross-replication results including the maximum queue length, the average queue length, and the average time customers spent in the system (TCSS).

Mean Values	Load Balancer	Max Q Length	Avg Q Length	Avg. TCSS
Set A	RandMin	99.8	67.1666667	173.0396811
d = 1	PureRand	92.4	60.03333333	154.9459182
	RR	68.8	46.46666667	121.5904007

Mean Values	Load Balancer	Max Q Length	Avg Q Length	Avg. TCSS
Set B	RandMin	37.4	36.9	102.8471266
d = 3	PureRand	77.6	53.23333333	136.1607124
	RR	60	39.5	96.74187312

Mean Values	Load Balancer	Max Q Length	Avg Q Length	Avg. TCSS
Set C	RandMin	25	24.66666667	68.01674767
d = 5	PureRand	82.6	55.1	153.6869526
	RR	77.2	45.86666667	119.7996226

The cross-replication performance summary metrics show that as *'d'*, the number of servers available for each RM load balance increases, the maximum queue length, average queue length, and average time customers spend in the system all decrease.

Discussion and Conclusion

As Expected, the performance of the RM load balancing improved with each increase in *'d'* server quantity. When only 1 server was available to RM during assignment, It performed very similarly to PR load balancing and was outperformed by RR balancing. But as the available servers to the RM balancer was increased to 3, both baseline methods were outperformed. With the increase of *'d'* =5 servers, RM was 330% more efficient than PR and 308% more efficient than RR.

There are many more potential parameter combinations to be tested in future simulations. Testing larger numbers of parallel servers in order to determine when the efficiency begins to diminish relative to the workload would be beneficial. Adding an additional baseline of always selecting the shortest queue would eliminate the random aspect of the shortest server queue selection in RM. Testing a wider range of parameters, such as different combinations of μ and λ, to test different levels of server utilization. Another potential change in the simulation would be to introduce dynamic Arrival Rates, testing how servers can handle large spikes in arrivals.