

Load-balancing with Random Choices

CSC 446 Project



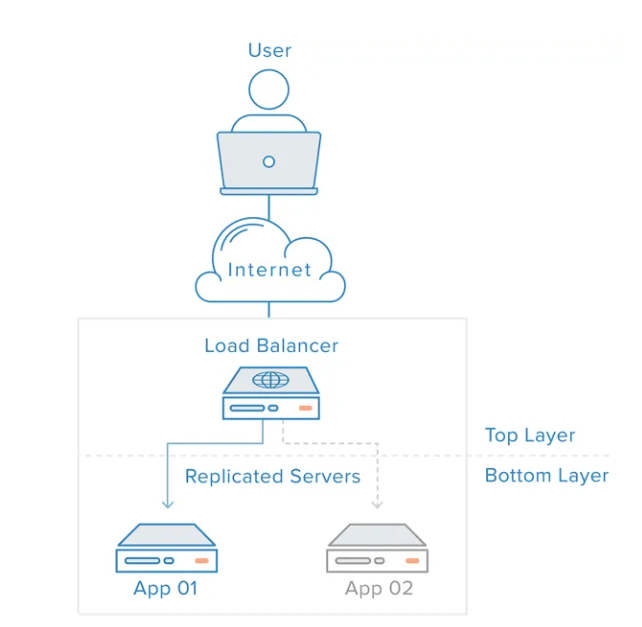
Team members: Junyi Ma, Patrick Meng

**Table of Contents**

1. [Introduction](#Introduction)
2. [The simulation model (Load Balancing Algorithm)](#Thesimulationmodel)
3. [Simulation objectives and parameters](#Simulationparameters)
4. [Methodology](#Methodology)
5. [Results & Analysis](#Results)
6. [Conclusion](#Conclusion)

**1.Introduction**

Load balancing is a method of distributing network traffic evenly across the pool of resources supporting an application. Modern applications must handle millions of users simultaneously and return the correct text, video, images and other data to each user in a fast and reliable manner. To handle such a high volume of traffic, most applications have many resource servers that contain a lot of duplicate data between them. A load balancer is a device that sits between a user and a server group, acting as an unseen coordinator and ensuring equal use of all resource servers.



Project Outline

In this project, we used a load balancing algorithm to distribute client requests to multiple servers for the purpose of balancing the workload of the servers. Specifically, we used two common load balancing algorithms, the random algorithm and the polling algorithm.

We try to simulate the event of n clients entering m servers, that is, we simulate the scenario of multiple client requests entering multiple servers at the same time. Through this simulation, we draw some conclusions, including the maximum workload, average workload, and average system time.

The maximum workload is the maximum time to complete processing requests on all servers. The average workload is the average of the time to complete processing a request on all servers. Average system time is the average time for a client request to enter the system until it gets a response.

By analyzing these metrics, we can evaluate the effectiveness of the load balancing algorithm we are using and the performance of the servers. If an algorithm causes some servers to be overloaded, then the maximum workload and average system time may be very high, while the average workload may be very low. Conversely, if the algorithm distributes requests in a more balanced manner, then the values of these metrics should be more even.

**2.****The simulation model (Load Balancing Algorithm)**

This project involves simulating a queuing system with a load balancer and multiple servers. The system consists of m parallel servers, each modeled as a FIFO queueing system with service time following an exponential distribution of rate parameter mu.

When a new customer arrives, they must first go through a load balancer that selects a server to handle the customer. The load balancer uses a scheduling method called RandMin, which involves the following steps:

The load balancer randomly selects d servers (d < m).

The load balancer checks the queue length of the selected d servers.

The load balancer dispatches the incoming customer to the server that has the shortest queue length among the selected d servers.

To compare the performance of RandMin with two baseline methods, the following two methods are also considered:

Purely random (PureRand): 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, i.e., the order of servers for dispatching customers is 1 to m.

The customer arrivals follow the Poisson arrival process with mean arrival rate lambda. We also assume that the processing time of the load balancer is negligible.

1. RandMin: This load balancing algorithm is also relatively simple, which means that whichever server has fewer connections is directly assigned to whichever server, which is very sensible and reasonable.
2. Purely random (PureRand): Unlike the previous algorithm, the random algorithm is also used in more scenarios, that is, a random number is generated, and the server corresponding to the number is the server to be assigned, this random algorithm has a greater chance and uncertainty, in fact, there is no difference between the use of time and the polling method, except that the order of the random method is not as strict.
3. Round-robin (RR): This algorithm is the most common load balancing algorithm, that is, regardless of the situation, all servers are allocated to each server in order of crude oil in turn. This algorithm treats all server requests equally, so it is more suitable for those cases where the server hardware conditions are similar.

**3.****Simulation parameters**

We assume that the service times of all servers follow an exponential distribution with rate parameter mu

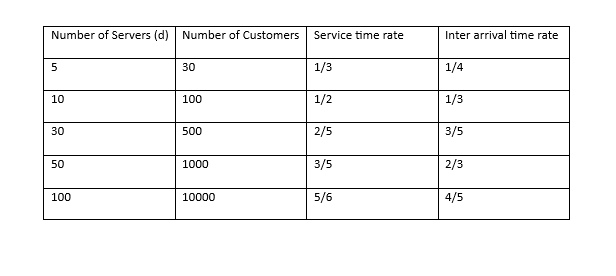
We assume that client arrivals follow a Poisson arrival process with an average arrival rate of lambda .

We ignore the processing time of the load balancer.

To achieve the simulation goal of comparing the performance of the RandMin scheduling method with the two baseline methods (PureRand and RR), we can define the following simulation parameters:

1. Number of servers: m - The number of servers in the system can be set to different values to test the performance of the system under different workloads. We can choose values for m such as 2, 4, and 8 to simulate low, medium, and high workloads, respectively.
2. Arrival rate: λ - The arrival rate of clients can be set to different values to simulate different traffic scenarios. We can choose values for λ such as 0.1, 0.5, and 0.9 to simulate low, medium, and high traffic, respectively.
3. Service rate: μ - The service rate of each server can be set to different values to simulate different server capabilities. We can choose values for μ such as 0.2, 0.5, and 0.8 to simulate slow, medium, and fast servers, respectively.
4. Dispatched servers: d - The number of servers selected by the load balancer can be set to different values to test the performance of the RandMin scheduling method. We can choose values for d such as 1, 2, and 3 to simulate different dispatching strategies.

By varying these simulation parameters, we can simulate different scenarios and measure the system performance in terms of maximum workload, average workload, and average system time. This will help us compare the performance of the RandMin scheduling method with the two baseline methods (PureRand and RR) and draw conclusions based on our simulation results.



Simulation result:

Maximum workload: defined as the longest queue length in m servers.

Average workload: defined as the average queue length in the m-servers.

Average system time: defined as the average time that all clients stay in the system.

**4.****Methodology**

Based on the tutorials we studied online, we got a deep understanding of the 3 load balancing algorithms. Our group wrote 3 algorithms using l had jupyter notebook software. And presented the data in the form of a table.

List the files below (with all algorithms):

RandMin.ipynb

PureRandom.ipynb

RoundRobin.ipynb

RandMin: We first select $d$ servers randomly from $m$ servers where $d$ < $m$. And then, we allocate each customer from the simulation into the server with the minimum queue length. If there are multiple server that have the shortest queue length, simply pick the first shortest queue length server we see and store the customer into the server while updating its queue length by 1. This algorithm will ensure that no server is overloaded with too many customers as it distributes the customer evenly into different servers equally.

Purely random: This algorithm will randomly select $d$ servers from $m$ servers where $d$ < $m$. After that, the algorithm will store each customer from the simulation by generating a corresponding server id randomly from the selected $d$ servers. By processing each customer from the simulation, the customer will be added to the random server and updating its length by 1. This algorithm has a great uncertainty because for every random choice of $d$ servers, we don’t know which exact server will be chosen and we might need to consider the worst case where the algorithm only picks one server for all customers. This means that we will need to pre-emptively allocate customer size as the queue length for every $d$ servers which makes it take up a lot of memory space.

Round-robin: We assign n clients to m servers sequentially. Assigning clients to servers in a sequential rotation, it treats each server on the back end in a balanced manner, without caring about the actual number of connections to the server or the current system load.

**5.****Results & Analysis**

We have used 3 different algorithms where we compare its efficiency of each algorithm.

RandMin Algorithm analysis:

1. This algorithm selects $d$ servers randomly from $m$ server where $d$ < $m$.

2. Checks the queue length of the selected $d$ servers and dispatches incoming customers to the server with the shortest queue length.

3. Since it always picks the server with the shortest queue length, it implies that each server will have an equal/even load without having the servers being overloaded with excessive customers.

4. The maximum workload of RandMin algorithm is approximately equal to the average workload of RandMin algorithm as the size of the customer grows.

5. We can say that RandMin algorithm efficiently handles customers into each server without having the servers to be overloaded.

Pure Random Algorithm analysis:

1. This algorithm randomly selects one server from $m$ servers while dispatching the incoming customers into the selected queue.

2. Since this algorithm is randomized, we can’t accurately predict which server to get chosen for the next iteration.

3. We can, however, assume that there is the worst-case scenario where only one of the chosen servers from $m$ servers always gets incoming customers dispatched for every iteration.

4. When this algorithm randomly chooses the same server and dispatches into the same queue for each iteration (which is possible), we need to preemptively allocate each server queue length from the server list to the size of all customers which requires a lot of computer memory.

5. Due to the nature of pure random algorithms, there is a significant uncertainty in this algorithm. For example, the maximum workload of Pure Random algorithm can be a lot greater than the average workload of Pure Random algorithm as we increase the size of the customers.

Round Robin Algorithm analysis:

1. This algorithm dispatches customers into each server in the order of server $1, 2, …, m$ where there are $m$ servers. If the size of customer is greater than $m$ servers then we repeat the order of server $1, 2, …, m$ until all customers are dispatched into each server queue.

2. As we can see, this algorithm will utilize all $m$ servers but the load of each server will be evenly distributed since this algorithm picks the server sequentially.

3. The Round Robin algorithm ensures that all servers will have an even load without preemptively allocating too much memory space for each server.

4. As we grow the size of the customers, the maximum workload of Round Robin algorithm is approximately equal to the average workload of Round Robin algorithm. So, this algorithm is relatively efficient.

Result from RandMin table:

|  |  |  |
| --- | --- | --- |
| 100 Customers, 10 servers | RandMin Algorithm | Mu = 1/3, lambda = 1/4 |
| Seed | Average time customer in the system | Maximum workload |
| 10 | 24.389218 | 10 |
| 50 | 6.521469 | 10 |
| 100 | 6.253589 | 10 |
| 150 | 8.380837 | 10 |
| 200 | 11.376272 | 10 |
| Mean | 11.384277 | 10 |
| Variance | 57.03001 | 0 |
| Confidence +/- | 11.384277  ±9.3768256 | 0 |

|  |  |  |
| --- | --- | --- |
| 1000 Customers, 50 servers | RandMin Algorithm | Mu = 1/3, lambda = 1/4 |
| Seed | Average time customer in the system | Maximum workload |
| 10 | 10.600646 | 20 |
| 50 | 11.455988 | 20 |
| 100 | 14.723724 | 20 |
| 150 | 10.445708 | 20 |
| 200 | 15.478245 | 20 |
| Mean | 12.540862200000001 | 20 |
| Variance | 5.6810413514952005 | 0 |
| Confidence +/- | 12.54086220  ±2.959498 | 0 |

Result from Pure Random table:

|  |  |  |
| --- | --- | --- |
| 100 Customers, 10 servers | Pure Random Algorithm | Mu = 1/2, lambda = 1/3 |
| Seed | Average time customer in the system | Maximum workload |
| 10 | 10.472224 | 18 |
| 50 | 3.868261 | 16 |
| 100 | 3.786326 | 14 |
| 150 | 5.070622 | 16 |
| 200 | 6.020316 | 18 |
| Mean | 5.8435498 | 16.4 |
| Variance | 7.546806032478202 | 2.8000000000000003 |
| Confidence +/- | 5.8435498  ±3.411031 | 16.4  ±2.07770 |

|  |  |  |
| --- | --- | --- |
| 1000 Customers, 50 servers | Pure Random Algorithm | Mu = 1/2, lambda = 1/3 |
| Seed | Average time customer in the system | Maximum workload |
| 10 | 5.544348 | 33 |
| 50 | 5.905982 | 31 |
| 100 | 7.613973 | 30 |
| 150 | 5.511890 | 33 |
| 200 | 7.064864 | 27 |
| Mean | 6.328211400000001 | 30.8 |
| Variance | 0.9137350173257996 | 6.2 |
| Confidence +/- | 6.3282114  ±1.1869 | 30.8  ±3.09172 |

Result from Round Robin table:

100 Customers, 10 servers, Mu = 1/2, lambda = 1/3

|  |  |  |  |
| --- | --- | --- | --- |
| Seed | Maximum workload | Average workload | Average time customer in the system |
| 10 | 10 | 10.0 | 2.234594723931633 |
| 50 | 10 | 10.0 | 1.8320887458838226 |
| 100 | 10 | 10.0 | 1.841785222237539 |
| 150 | 10 | 10.0 | 1.8092635398165582 |
| 200 | 10 | 10.0 | 2.1304315045597493 |
| Mean |  |  | 1.9696327 |
| Variance |  |  | 0.039260679 |
| Confidence +/- |  |  | 1.9696327 ± 0.0388 |

1000 Customers, 50 servers RandMin Algorithm Mu = 1/3, lambda = 1/4

|  |  |  |  |
| --- | --- | --- | --- |
| Seed | Maximum workload | Average workload | Average time customer in the system |
| 10 | 20 | 20.0 | 3.013659575294919 |
| 50 | 20 | 20.0 | 3.0534817044340827 |
| 100 | 20 | 20.0 | 3.1925294017317967 |
| 150 | 20 | 20.0 | 3.0377690499163945 |
| 200 | 20 | 20.0 | 3.070354647274634 |
| Mean |  |  | 3.0735589 |
| Variance |  |  | 0.0048590459 |
| Confidence +/- |  |  | 3.0735589 ± 0.00432 |

Rand-Min variance of average time customer spent in the system:

Rand-Min variance of max workload:

Pure-Rand variance of average time customer spent in the system:

Pure-Rand variance of max workload:

**6.****Conclusion**

We have used 3 different algorithms to manage load-balancing while comparing the efficiency and performance of each algorithm.

For Rand-Min algorithm, the variance value of the average time of a customer spent in the system decreases as we increase the number of customers in the simulation. The variance value of maximum workload of Rand-Min algorithm is 0 for both cases since the maximum workload of Rand-Min is the same for each of the $d$ selected servers.

For Pure Random algorithm, the variance value of maximum workload increases as we increase the number of customers in the simulation. However, the variance value of average time a customer spent in the system decreases as we increase the customer size in the simulation. One possible explanation for the maximum workload variance value is that there is a great uncertainty of the maximum workload. The uncertainty is caused by the Pure Random algorithm since it randomly picks the server to store arriving customers without considering the load of the server. As we increase the size of the customers, the servers are more prone to being overloaded which can be shown with a large variance value.

For Round Robin algorithm, the variance of the maximum workload is the same for 100 customers and 1000 customers. One explanation why this is happening is because the algorithm stores each customer to the server queue in a sequential fashion from 1 to m while doing this process repeatedly. This process incrementally increases the server queue length evenly without having any servers overloaded with excessive customers. In addition, the average time customer spent in the system is relatively consistent with different customer size as the variance value of the average time customer spent in the system is relatively small with respect to the average time customer spent with different seed input.