

Adwait Purao- 2021300101 TE Comps B - Batch B

DC LAB

**DISTRIBUTED COMPUTING EXPERIMENT 10**

**Aim:**

Implement a load balancing algorithm.

**Theory:**

Load balancing is the method of distributing network traffic equally across a pool of

resources that support an application. Modern applications must process millions of users

simultaneously and return the correct text, videos, images, and other data to each user quickly and reliably.

**What are load balancing algorithms?**

A] Static load balancing: Static load balancing algorithms follow fixed rules and are

independent of the current server state. The following are examples of static load balancing.

**1) Round-robin method:**

Servers have IP addresses that tell the client where to send requests. The IP address is a

long number that is difficult to remember. To make it easy, a Domain Name System maps

website names to servers. When you enter aws.amazon.com into your browser, the request

first goes to our name server, which returns our IP address to your browser. In the round-

robin method, an authoritative name server does the load balancing instead of specialized

hardware or software. The name server returns the IP addresses of different servers in the

server farm turn by turn or in a round-robin fashion.

**2) Weighted round-robin method:**

In weighted round-robin load balancing, you can assign different weights to each server

based on their priority or capacity. Servers with higher weights will receive more incoming

application traffic from the name server.

**3) IP hash method:**

In the IP hash method, the load balancer performs a mathematical computation, called

hashing, on the client IP address. It converts the client IP address to a number, which is

then mapped to individual servers.

**B] Dynamic load balancing:**

Dynamic load balancing algorithms examine the current state of the servers before distributing traffic. The following are some examples of dynamic load balancing algorithms.

**1) Least connection method:**

A connection is an open communication channel between a client and a server. When the

client sends the first request to the server, they authenticate and establish an active

connection between each other. In the least connection method, the load balancer checks

which servers have the fewest active connections and sends traffic to those servers. This

method assumes that all connections require equal processing power for all servers.

**2) Weighted least connection method:**

Weighted least connection algorithms assume that some servers can handle more active

connections than others. Therefore, you can assign different weights or capacities to each

server, and the load balancer sends the new client requests to the server with the least

connections by capacity.

**3) Least response time method:**

The response time is the total time that the server takes to process the incoming requests

and send a response. The least response time method combines the server response time

and the active connections to determine the best server. Load balancers use this algorithm

to ensure faster service for all users.

**4) Resource-based method:**

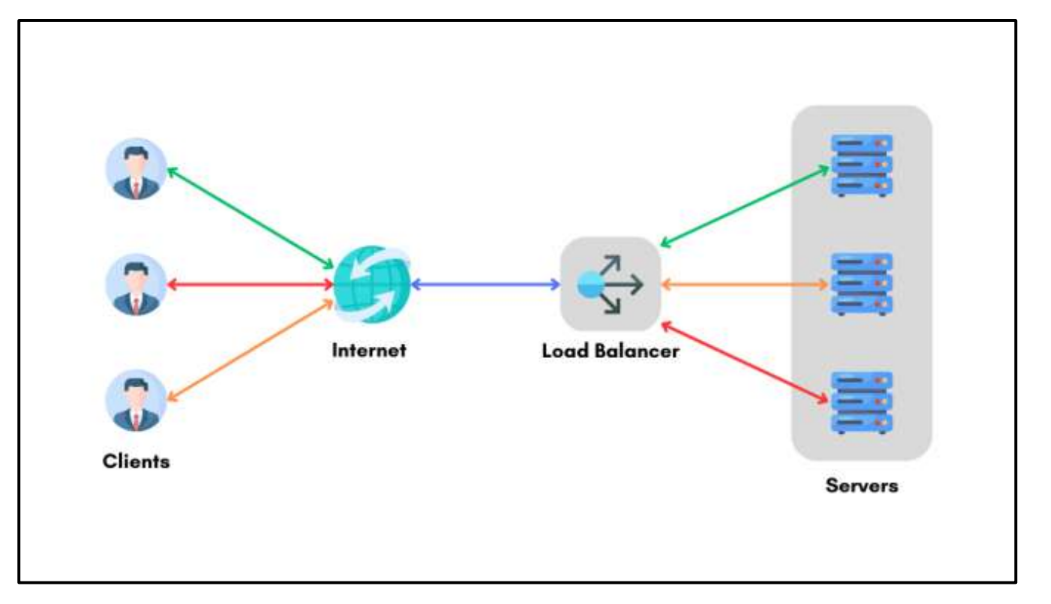
In the resource-based method, load balancers distribute traffic by analyzing the current

server load. Specialized software called an agent runs on each server and calculates usage

of server resources, such as its computing capacity and memory. Then, the load balancer

checks the agent for sufficient free resources before distributing traffic to that server.

**Diagram:**

****

**Code:**

class InventoryServer:

inventory = {}

def \_\_init\_\_(self, server\_name):

self.server\_name = server\_name

self.inventory = InventoryServer.inventory

def process\_request(self, action, item, quantity):

if action == 1:

self.add\_product(item, quantity)

elif action == 2:

self.subtract\_quantity(item, quantity)

elif action == 3:

self.add\_quantity(item, quantity)

elif action == 4:

self.view\_inventory()

else:

print("Invalid action: {}".format(action))

def add\_product(self, item, quantity):

if item in self.inventory:

self.inventory[item] += quantity

else:

self.inventory[item] = quantity

print("\n{}: Added {} units of {}.\nInventory: {}".format(

self.server\_name, quantity, item, self.inventory))

def subtract\_quantity(self, item, quantity):

if item in self.inventory:

if self.inventory[item] >= quantity:

self.inventory[item] -= quantity

print("\n{}: Subtracted {} units of {}.\nInventory: {}".format(

self.server\_name, quantity, item, self.inventory))

else:

print("\n{}: Not enough units of {} to subtract.".format(

self.server\_name, item))

else:

print("\n{}: Item {} not found in inventory.".format(

self.server\_name, item))

def add\_quantity(self, item, quantity):

if item in self.inventory:

self.inventory[item] += quantity

print("\n{}: Updated {} to {} units.\nInventory: {}".format(self.server\_name, item, quantity, self.inventory))

else:

print("\nProduct not found")

def view\_inventory(self):

print("\n{}: Inventory :".format(self.server\_name))

print("-------------------------")

if self.inventory.\_\_len\_\_() > 0:

for item in self.inventory:

print(f"Name : {item}")

print(f"Quantity : {self.inventory[item]}")

print("-------------------------")

else:

print("\nInventory in Empty.")

class RoundRobinLoadBalancer:

def \_\_init\_\_(self, servers):

self.servers = servers

self.current\_server\_index = 0

def get\_next\_server(self):

next\_server = self.servers[self.current\_server\_index]

self.current\_server\_index = (self.current\_server\_index + 1) % len(self.servers)

return next\_server

class InventoryManagementSystem:

def \_\_init\_\_(self, servers):

self.load\_balancer = RoundRobinLoadBalancer(servers)

def process\_inventory\_request(self, action, item, quantity):

server = self.load\_balancer.get\_next\_server()

server.process\_request(action, item, quantity)

# Example usage

server\_names = ["Server1", "Server2", "Server3"]

inventory\_servers = [InventoryServer(name) for name in server\_names]

inventory\_system = InventoryManagementSystem(inventory\_servers)

while True:

print("\nActions:")

print("1. ADD Product")

print("2. SUBTRACT item quantity")

print("3. ADD item quantity")

print("4. VIEW Inventory")

print("5. EXIT")

choice = int(input("Enter your choice (1-5): "))

if choice == 4:

inventory\_system.process\_inventory\_request(choice, 0, 0)

elif choice >= 1 and choice < 5:

item = input("Enter item name: ")

quantity = int(input("Enter item quantity: "))

inventory\_system.process\_inventory\_request(choice, item, quantity)

elif choice == 5:

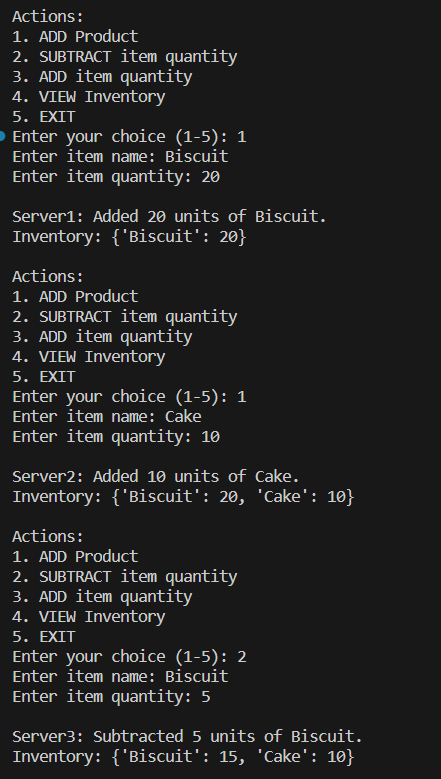
print("\nExited.")

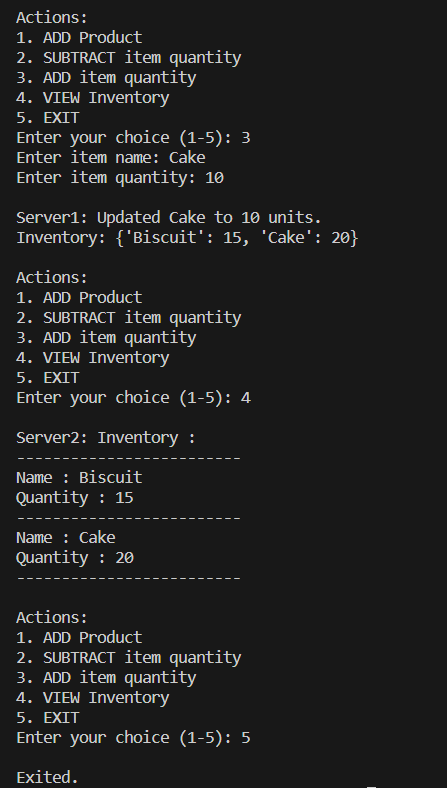
break

else:

print("Invalid input. Try again!")

**Output:**

****

****

**Conclusion:**

From this experiment, we learned about load balancing algorithms. We saw how load balancing works and also how to implement it in python. We saw how it is important and also how it plays a pivotal role in distributed systems. We saw how it increases the efficiency by accurately sending the requests to the appropriate server. Thus, load balancing proves to be an essential part of distributed systems and builds the backbone of any efficient system and improves all the user’s experience across the website.