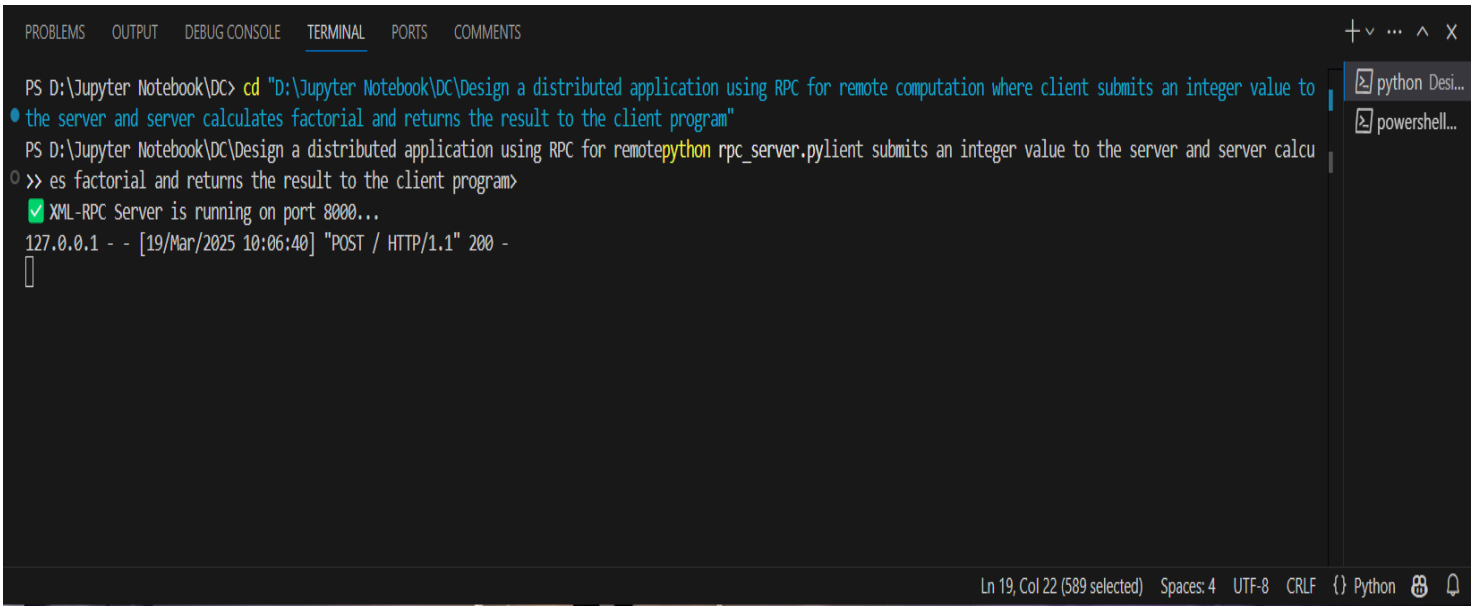
Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

rpc_server.py

```python
from xmlrpc.server import SimpleXMLRPCServer
import math

# Function to calculate factorial
def compute_factorial(n):
    if n < 0:
        return "Factorial not defined for negative numbers"
    return str(math.factorial(n))  # Convert to string to avoid integer limits

# Create an XML-RPC server
server = SimpleXMLRPCServer(("localhost", 8000), allow_none=True)
print("✅ XML-RPC Server is running on port 8000...")

# Register the function
server.register_function(compute_factorial, "factorial")

# Keep the server running
server.serve_forever()
#python rpc_client.py
```

Output:-



rpc_client.py

```python
import xmlrpc.client

# Connect to the XML-RPC server
server = xmlrpc.client.ServerProxy("http://localhost:8000/")

# Get user input
num = int(input("Enter a number to compute factorial: "))

# Call the remote function
result = server.factorial(num)

# Display the result
print(f"✅ Factorial of {num} is: {result}")
#python rpc_server.py
```

Output:-

Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

#install pip install Pyro5

rmi_server.py

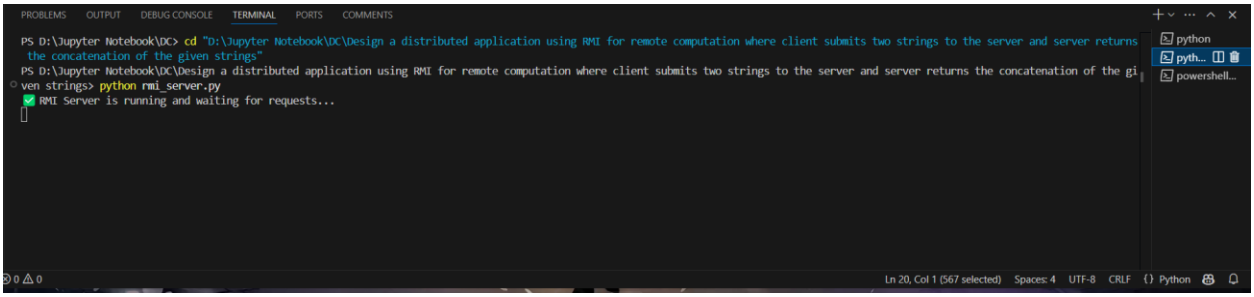
```
import Pyro5.api

# Define the remote class
@Pyro5.api.expose
class StringConcatenation:
    def concatenate(self, str1, str2):
        return str1 + str2

# Start the Pyro5 server
def main():
    daemon = Pyro5.api.Daemon()  # Create a server daemon
    ns = Pyro5.api.locate_ns()  # Locate the Pyro5 nameserver
    uri = daemon.register(StringConcatenation)  # Register the remote object
    ns.register("string.concat", uri)  # Register with a unique name

    print("✅ RMI Server is running and waiting for requests...")
    daemon.requestLoop()  # Keep server running

if __name__ == "__main__":
    main()
```

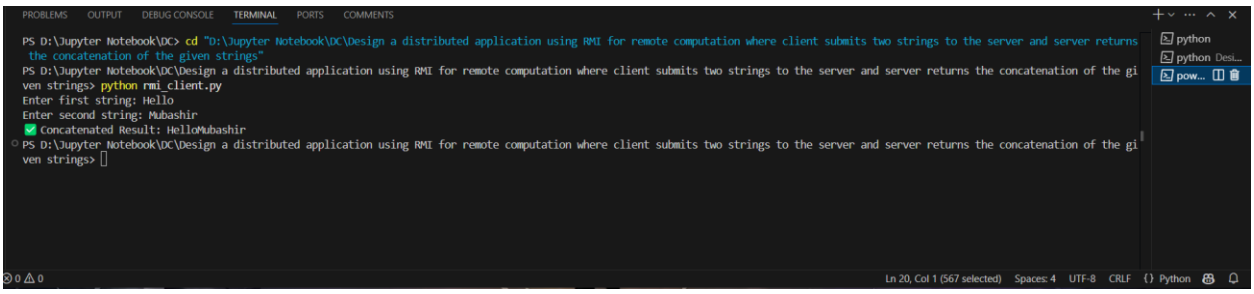
rmi_client.py

```
import Pyro5.api

# Connect to the remote object
def main():
    ns = Pyro5.api.locate_ns()  # Locate the Pyro5 nameserver
    uri = ns.lookup("string.concat")  # Look up the registered object
    remote_object = Pyro5.api.Proxy(uri)  # Get a proxy for the remote object

    # Get input from user
    str1 = input("Enter first string: ")
    str2 = input("Enter second string: ")

    # Call the remote method
    result = remote_object.concatenate(str1, str2)
```

```
    print(f"✅ Concatenated Result: {result}")

if __name__ == "__main__":
    main()
```

Outputs:-





Design a distributed application using MapReduce under Hadoop for: a) Character counting in a given text file. b) Counting no. of occurrences of every word in a given text file.

input.txt

Hello Hadoop World

Hadoop is great

Hello World

char_mapper.py

```
#!/usr/bin/env python3
import sys

# Read input line by line
for line in sys.stdin:
    line = line.strip()  # Remove whitespace
    for char in line:
        if char:  # Ensure it's not an empty character
            print(f"{char}\t1")  # Emit (char, 1)
```

char_reducer.py

```
#!/usr/bin/env python3
import sys
from collections import defaultdict

char_count = defaultdict(int)

# Read input from standard input
for line in sys.stdin:
    line = line.strip()
    if not line or "\t" not in line:  # Ignore empty or invalid lines
        continue

    try:
        char, count = line.split("\t")
        char_count[char] += int(count)
    except ValueError:
        continue  # Ignore lines that don't match expected format

# Print the final counts
for char, count in char_count.items():
    print(f"{char}\t{count}")
```

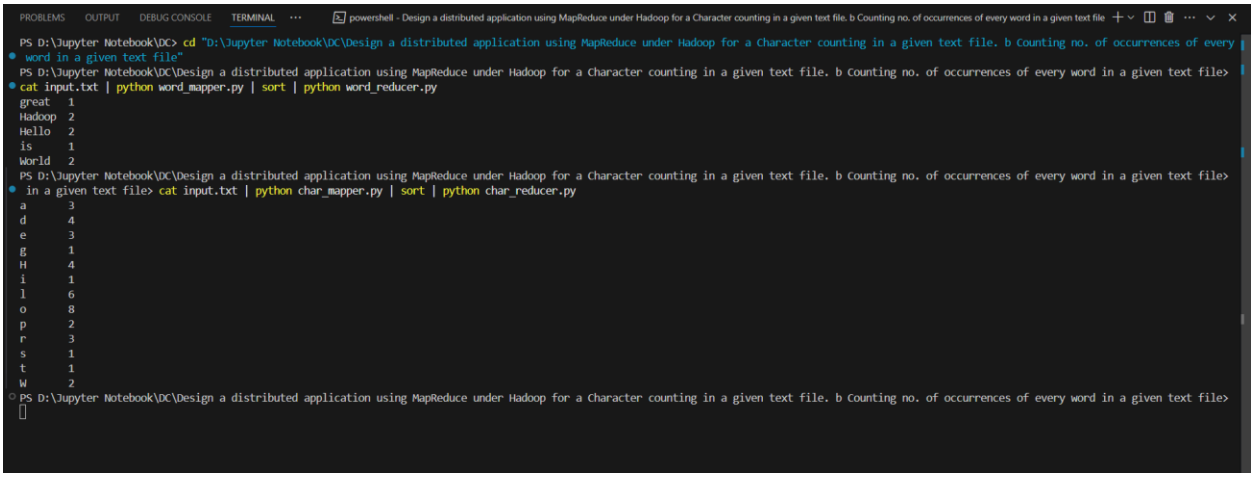
word_mapper.py

```
#!/usr/bin/env python3
import sys

# Read input line by line
for line in sys.stdin:
    line = line.strip()  # Remove whitespace
    words = line.split()  # Split into words
    for word in words:
        print(f"{word}\t1")  # Emit (word, 1)
```

word_reducer.py

```
#!/usr/bin/env python3
import sys
from collections import defaultdict

word_count = defaultdict(int)

# Read input from standard input
for line in sys.stdin:
    word, count = line.strip().split("\t")
    word_count[word] += int(count)

# Print the final counts
for word, count in word_count.items():
    print(f"{word}\t{count}")
```

Output:-



Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations.

fuzzy_sets.py

```
#Fuzzy Set Operations
#!/usr/bin/env python3

def fuzzy_union(A, B):
    """ Union of two fuzzy sets (max operation) """
    return {x: max(A.get(x, 0), B.get(x, 0)) for x in set(A) | set(B)}

def fuzzy_intersection(A, B):
    """ Intersection of two fuzzy sets (min operation) """
    return {x: min(A.get(x, 0), B.get(x, 0)) for x in set(A) & set(B)}

def fuzzy_complement(A):
    """ Complement of a fuzzy set (1 - membership value) """
    return {x: round(1 - A[x], 2) for x in A}

def fuzzy_difference(A, B):
    """ Difference of two fuzzy sets (A - B = min(A, 1 - B)) """
    return {x: min(A.get(x, 0), 1 - B.get(x, 0)) for x in A}

if __name__ == "__main__":
    A = {'a': 0.5, 'b': 0.7, 'c': 0.9}
    B = {'a': 0.2, 'b': 0.8, 'd': 0.6}

    print("Union:", fuzzy_union(A, B))
    print("Intersection:", fuzzy_intersection(A, B))
    print("Complement of A:", fuzzy_complement(A))
    print("Difference A - B:", fuzzy_difference(A, B))
```

fuzzy_relations.py

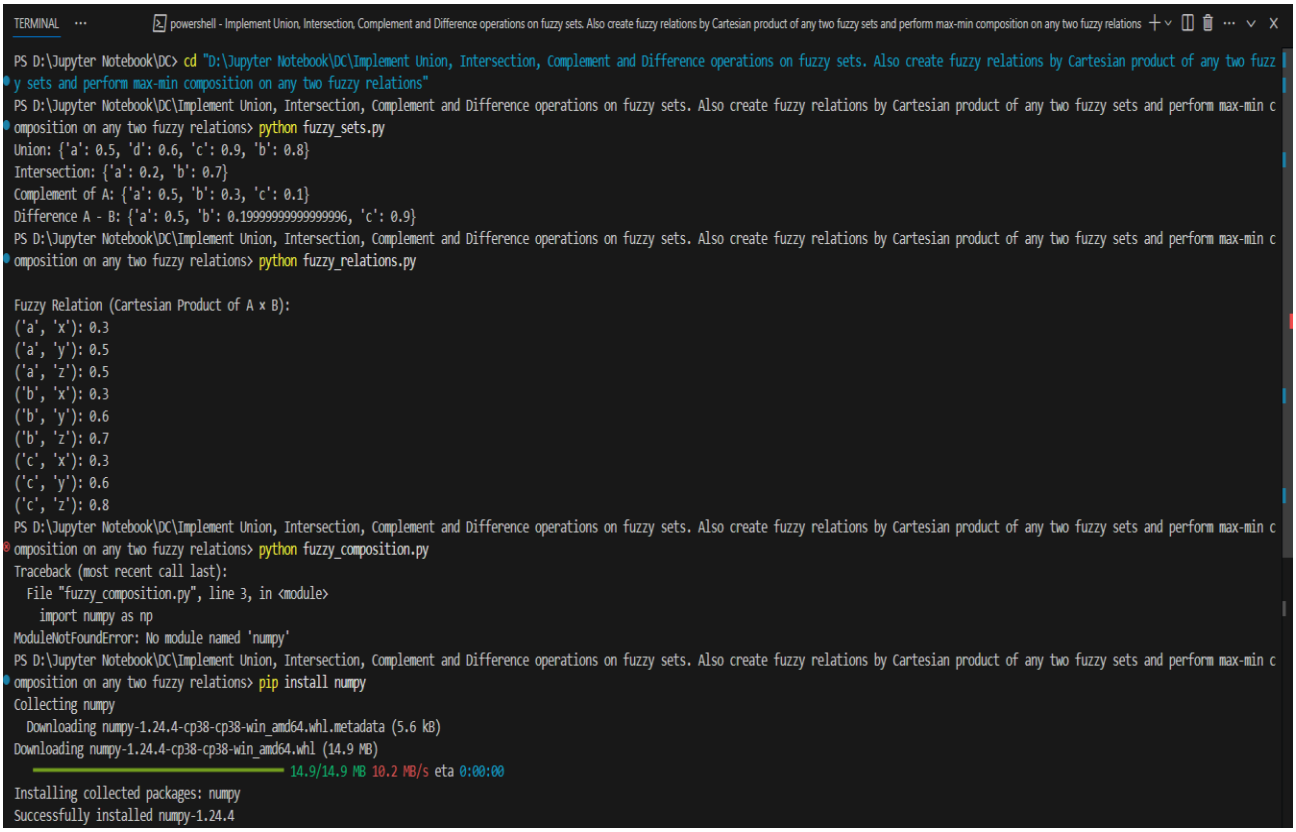
```
#Cartesian Product Relations
#!/usr/bin/env python3

def cartesian_product(A, B):
    """ Cartesian product of two fuzzy sets to form a fuzzy relation """
    return {(x, y): round(min(A[x], B[y]), 2) for x in A for y in B}

if __name__ == "__main__":
    A = {'a': 0.5, 'b': 0.7, 'c': 0.9}
    B = {'x': 0.3, 'y': 0.6, 'z': 0.8}
```

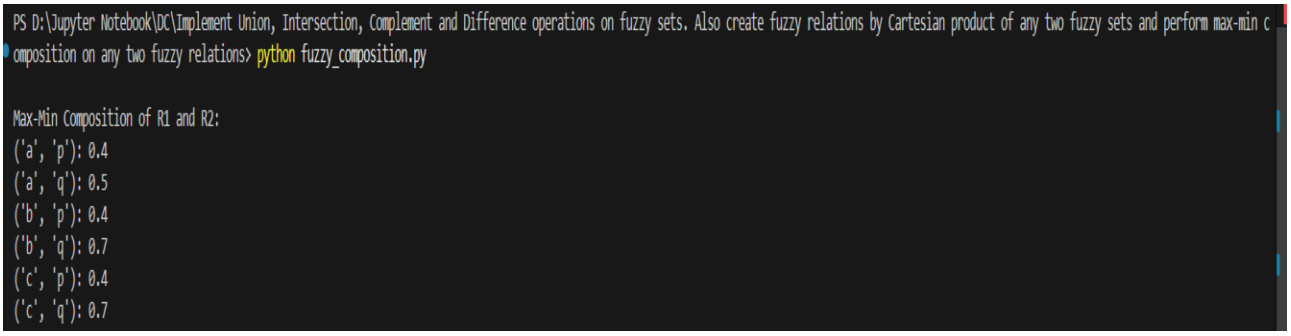
```
    R = cartesian_product(A, B)

    print("\nFuzzy Relation (Cartesian Product of A × B):")
    for pair, value in R.items():
        print(f"{pair}: {value}")
```

fuzzy_composition.py

```
#Max-Min Composition
#!/usr/bin/env python3
import numpy as np

def max_min_composition(R1, R2):
    """ Max-Min Composition of two fuzzy relations """
    X = sorted(set(x for x, _ in R1))
    Y = sorted(set(y for _, y in R1))
    Z = sorted(set(z for _, z in R2))

    matrix_R1 = np.array([[R1.get((x, y), 0) for y in Y] for x in X])
    matrix_R2 = np.array([[R2.get((y, z), 0) for z in Z] for y in Y])

    composed_matrix = np.zeros((len(X), len(Z)))

    for i in range(len(X)):
        for j in range(len(Z)):
            composed_matrix[i][j] = max(min(matrix_R1[i, k], matrix_R2[k, j]) for
k in range(len(Y)))

    return {(X[i], Z[j]): composed_matrix[i, j] for i in range(len(X)) for j in
range(len(Z))}

if __name__ == "__main__":
    A = {'a': 0.5, 'b': 0.7, 'c': 0.9}
    B = {'x': 0.3, 'y': 0.6, 'z': 0.8}
    C = {'p': 0.4, 'q': 0.7}

    R1 = {(a, b): min(A[a], B[b]) for a in A for b in B}
    R2 = {(b, c): min(B[b], C[c]) for b in B for c in C}

    print("\nMax-Min Composition of R1 and R2:")
    composition = max_min_composition(R1, R2)
    for pair, value in composition.items():
        print(f"{pair}: {value}")
```

Output:-


TERMINAL
powershell - Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations
PS D:\Jupyter Notebook\DC> cd "D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations"
PS D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations> python fuzzy_sets.py
Union: {'a': 0.5, 'd': 0.6, 'c': 0.9, 'b': 0.8}
Intersection: {'a': 0.2, 'b': 0.7}
Complement of A: {'a': 0.5, 'b': 0.3, 'c': 0.1}
Difference A - B: {'a': 0.5, 'b': 0.19999999999999996, 'c': 0.9}
PS D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations> python fuzzy_relations.py
Fuzzy Relation (Cartesian Product of A x B):
('a', 'x'): 0.3
('a', 'y'): 0.5
('a', 'z'): 0.5
('b', 'x'): 0.3
('b', 'y'): 0.6
('b', 'z'): 0.7
('c', 'x'): 0.3
('c', 'y'): 0.6
('c', 'z'): 0.8
PS D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations> python fuzzy_composition.py
Traceback (most recent call last):
File "fuzzy_composition.py", line 3, in <module>
import numpy as np
ModuleNotFoundError: No module named 'numpy'
PS D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations> pip install numpy
Collecting numpy
Downloading numpy-1.24.4-cp38-cp38-win_amd64.whl.metadata (5.6 kB)
Downloading numpy-1.24.4-cp38-cp38-win_amd64.whl (14.9 MB)
14.9/14.9 MB 10.2 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.24.4


PS D:\Jupyter Notebook\DC\Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations> python fuzzy_composition.py
Max-Min Composition of R1 and R2:
('a', 'p'): 0.4
('a', 'q'): 0.5
('b', 'p'): 0.4
('b', 'q'): 0.7
('c', 'p'): 0.4
('c', 'q'): 0.7

Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.

load_balancer.py

```
import random
import time
import threading

class Server:
    """ Represents a Server handling requests """
    def __init__(self, server_id):
        self.server_id = server_id
        self.active_connections = 0

    def process_request(self, request_id):
        """ Simulate processing a request """
        self.active_connections += 1
        print(f"🟤 Server {self.server_id} processing Request {request_id}
(Active: {self.active_connections})")
        time.sleep(random.uniform(1, 3))  # Simulate processing time
        self.active_connections -= 1
        print(f"□ Server {self.server_id} finished Request {request_id} (Active:
{self.active_connections})")

class LoadBalancer:
    """ Distributes client requests among available servers """
    def __init__(self, servers, algorithm="round_robin"):
        self.servers = servers
        self.algorithm = algorithm
        self.request_count = 0
        self.lock = threading.Lock()

    def distribute_request(self, request_id):
        """ Distribute requests based on the chosen load balancing algorithm """
        with self.lock:
            if self.algorithm == "round_robin":
                selected_server = self.servers[self.request_count %
len(self.servers)]
                self.request_count += 1
            elif self.algorithm == "least_connections":
                selected_server = min(self.servers, key=lambda s:
s.active_connections)
            elif self.algorithm == "random":
                selected_server = random.choice(self.servers)
            else:
```

```
                raise ValueError("Invalid load balancing algorithm!")

        # Process request on the selected server
        threading.Thread(target=selected_server.process_request,
args=(request_id,)).start()

def simulate_requests(load_balancer, num_requests=10, delay=0.5):
    """ Simulate multiple client requests """
    for i in range(1, num_requests + 1):
        print(f"\n⚡ Client Request {i} Sent")
        load_balancer.distribute_request(i)
        time.sleep(delay)

if __name__ == "__main__":
    # Create 3 server instances
    servers = [Server(i) for i in range(1, 4)]

    # Choose load balancing algorithm: "round_robin", "least_connections",
"random"
    algorithm = "round_robin"  # Change this to test different algorithms

    # Create Load Balancer
    load_balancer = LoadBalancer(servers, algorithm=algorithm)

    # Simulate client requests
    simulate_requests(load_balancer, num_requests=10, delay=1)
```

Output:-