ASSIGNMENT 1

PROBLEM STATEMENT: -

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.

OBJECTIVE:

1. To learn and understand how to communicate between processes on different workstations.
2. To Learn and design a distributed application using Remote Procedure Call.

THEORY:

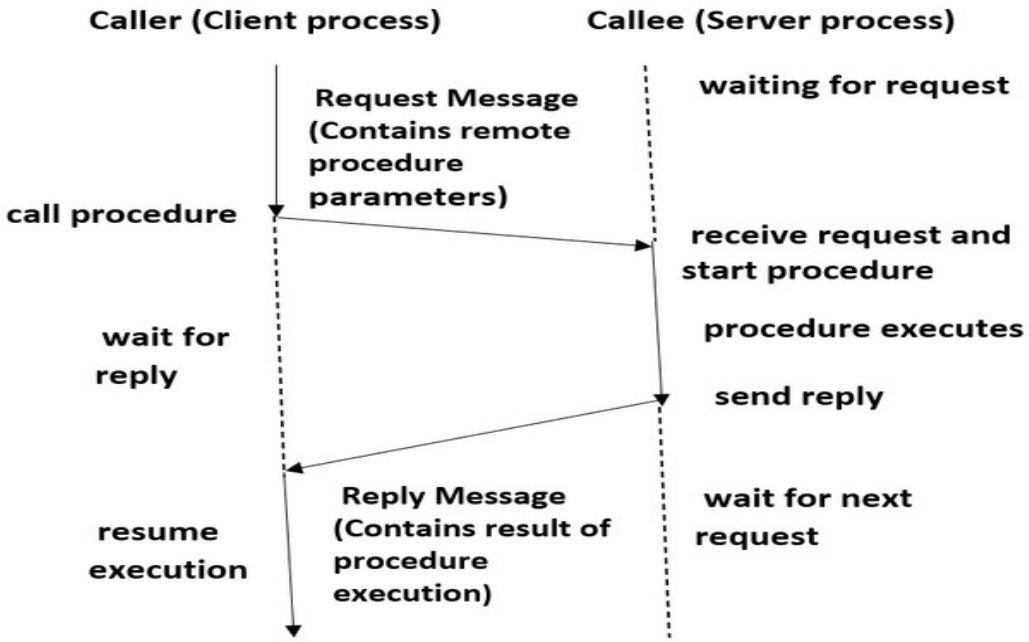
Remote Procedure Call (RPC) is a communication technology that is used by one program to make a request to another program for utilizing its service on a network without even knowing the network's details. A function call or a subroutine call are other terms for a procedure call.

It is based on the client-server concept. The client is the program that makes the request, and the server is the program that gives the service. An RPC, like a local procedure call, is based on the synchronous operation that requires the requesting application to be stopped until the remote process returns its results. Multiple RPCs can be executed concurrently by utilizing lightweight processes or threads that share the same address space. Remote Procedure Call program as often as possible utilizes the Interface Definition Language (IDL), a determination language for describing a computer program component's Application Programming Interface (API). In this circumstance, IDL acts as an interface between machines at either end of the connection, which may be running different operating systems and programming languages.

Working Procedure for RPC Model

- The process arguments are placed in a precise location by the caller when the procedure needs to be called.
- Control at that point passed to the body of the method, which is having a series of instructions.
- The procedure body is run in a recently created execution environment that has duplicates of the calling instruction's arguments.
- At the end, after the completion of the operation, the calling point gets back the control, which returns a result.

- The call to a procedure is possible only for those procedures that are not within the caller's address space because both processes (caller and callee) have distinct address space and the access is restricted to the caller's environment's data and variables from the remote procedure.
- The caller and callee processes in the RPC communicate to exchange information via the message-passing scheme.
- The first task from the server-side is to extract the procedure's parameters when a request message arrives, then the result, send a reply message, and finally wait for the next call message.
- Only one process is enabled at a certain point in time.
- The caller is not always required to be blocked.
- The asynchronous mechanism could be employed in the RPC that permits the client to work even if the server has not responded yet.
- In order to handle incoming requests, the server might create a thread that frees the server for handling consequent requests.



Implementation Steps:

1st file:Factserver.py

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler
```

```
class FactorialServer:
    def calculate_factorial(self, n):
        if n < 0:
            raise ValueError("Input must be a non-negative integer.")
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)
# Create server
with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) as server:
    server.register_introspection_functions()
    # Register the FactorialServer class
    server.register_instance(FactorialServer())
    print("FactorialServer is ready to accept requests.")
    # Run the server's main loop
    server.serve_forever()
```

2nd file: Factclient.py

```
import xmlrpc.client
# Create an XML-RPC client
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as proxy:
    try:
        # Replace 5 with the desired integer value
       input_value = 5
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Execution Steps

1. Open Command Prompt:

On Windows: Press Win + R, type cmd, and press Enter.

On Linux/macOS: Open a terminal.

2. Navigate to the Script's Directory:

Use the cd command to change to the directory where your Python script is located.

For example:

```bash
cd path\to\your\script\directory
```

3. Run the Python Script:

Use the python command followed by the name of your Python script to run it.

For example:

- On Windows:

```bash
python script.py
```

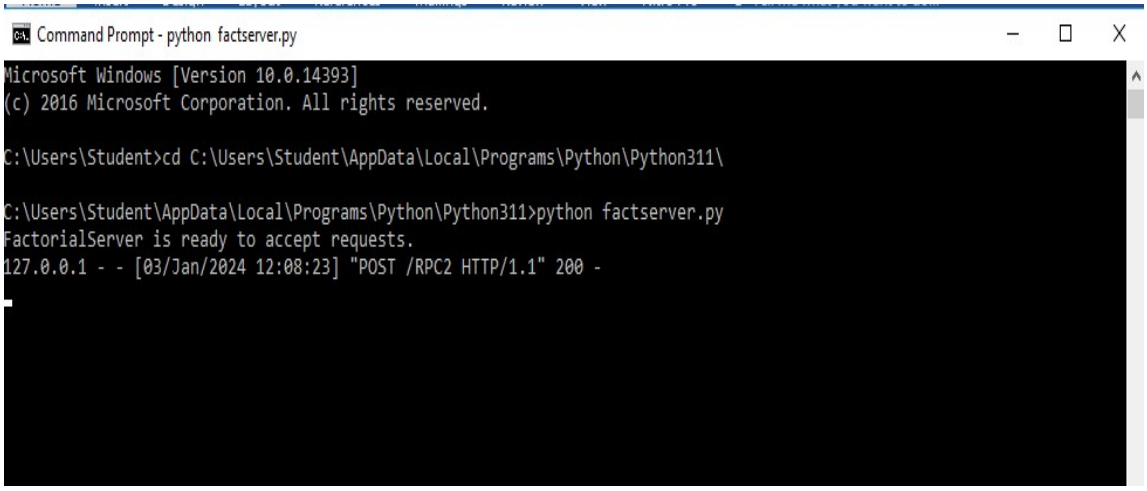
If you're using Python 3, you might need to use `python3` instead:

```bash
python3 script.py
```

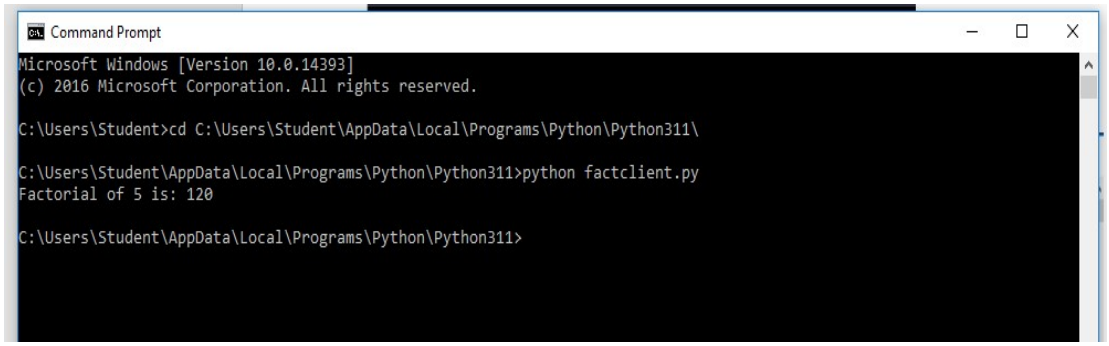
- On Linux/macOS:

```bash
python3 script.py
```

First execute factserver.py on command prompt



Then open another Command Prompt window and execute factclient.py



CONCLUSION:

In this way we have 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.

ASSIGNMENT QUESTION

1. Define Remote Procedure Call (RPC) and explain how it enables communication between a client and a server in a distributed system?
2. Discuss the advantages and challenges of using RPC in distributed systems?
3. Describe the steps involved in implementing the RPC mechanism for the factorial computation?
4. Discuss the key components and their responsibilities in the client and server RPC implementations?
5. Explain how the client and server can exchange error messages in the RPC system?

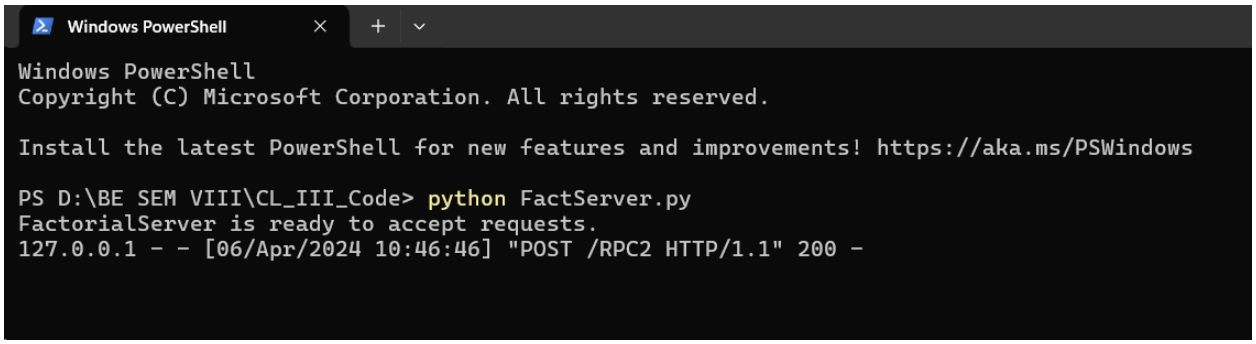
FactServer.py

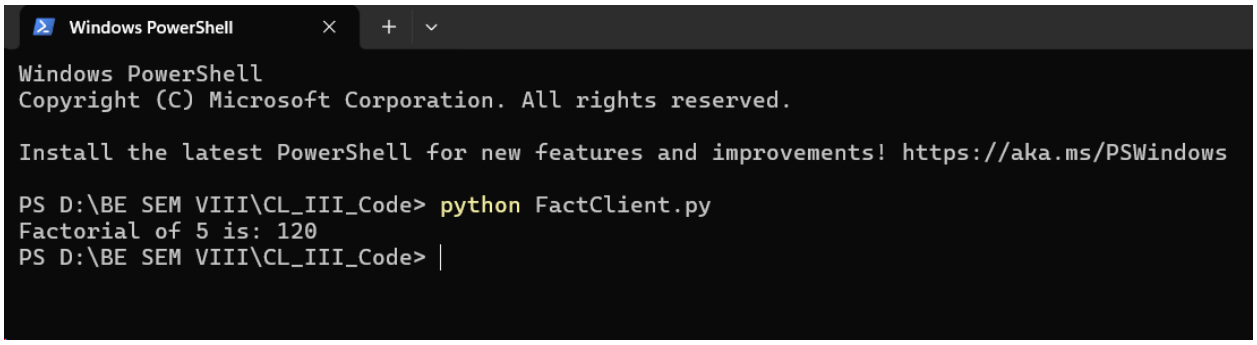
```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

class FactorialServer:
    def calculate_factorial(self, n):
        if n < 0:
            raise ValueError("Input must be a non-negative integer.")
        result = 1

        for i in range(1, n + 1):
            result *= i
        return result

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
requestHandler=RequestHandler) as server:
    server.register_introspection_functions()
    # Register the FactorialServer class
    server.register_instance(FactorialServer())
    print("FactorialServer is ready to accept requests.")
    # Run the server's main loop
    server.serve_forever()
```

FactClient.py

```
import xmlrpc.client

# Create an XML-RPC client
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as
proxy:
    try:
        # Replace 5 with the desired integer value
        input_value = 5
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Output:


Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS D:\BE SEM VIII\CL_III_Code> python FactServer.py
FactorialServer is ready to accept requests.
127.0.0.1 - - [06/Apr/2024 10:46:46] "POST /RPC2 HTTP/1.1" 200 -


Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS D:\BE SEM VIII\CL_III_Code> python FactClient.py
Factorial of 5 is: 120
PS D:\BE SEM VIII\CL_III_Code>

ASSIGNMENT 2

PROBLEM STATEMENT: -

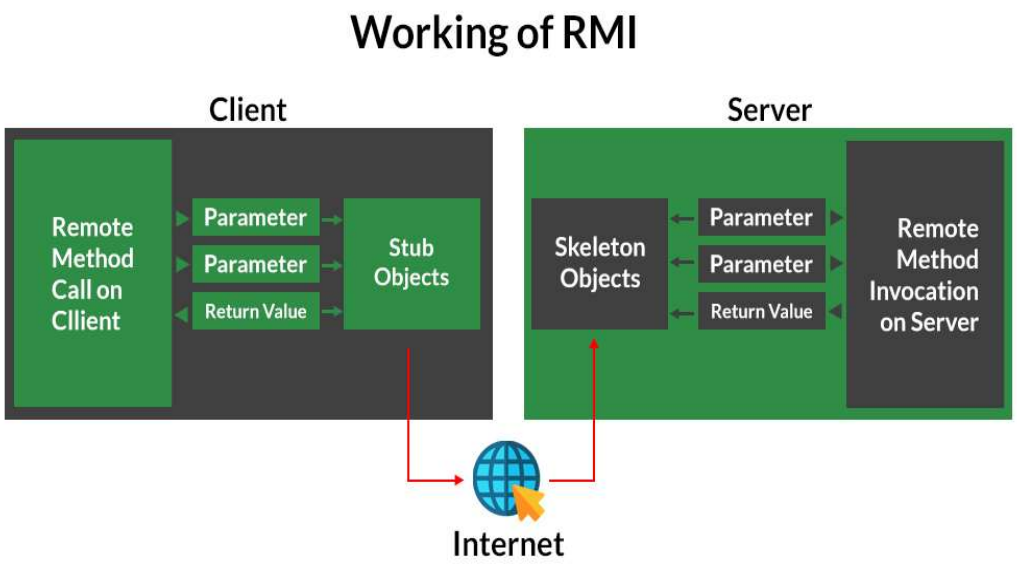
Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

OBJECTIVE:

1. Students should be able to Implement the server class that enables the remote interface.
2. Students should be able to start the RMI registry on the server side. This will allow clients to look up the server object by name.

THEORY:

In this practical students need to design and develop a distributed Hotel booking application using Java RMI. A distributed hotel booking system consists of the hotel server and the client machines. The server manages hotel rooms booking information. A customer can invoke the following operations at his machine i) Book the room for the specific guest ii) Cancel the booking of a guest.



Remote Method Invocation (RMI) is a Java technology that allows a Java object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. It enables distributed computing by providing a way for objects to interact across different JVMs, making it suitable for building distributed applications.

Following steps are involved in design of distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

In Python, you can use the Pyro4 library to implement a distributed application using RMI (Remote Method Invocation) for remote computation. Pyro4 is a Python Remote Objects library that simplifies the process of building distributed applications.

Before we start, we need to install the Pyro4 library. we can install it using:

```bash
pip install Pyro4
```

Server Implementation:

 # server.py

```
import Pyro4

@Pyro4.expose
class StringConcatenationServer:
    def concatenate_strings(self, str1, str2):
        result = str1 + str2
        return result


def main():
    daemon = Pyro4.Daemon()  # Create a Pyro daemon
    ns = Pyro4.locateNS()  # Locate the Pyro nameserver

    # Create an instance of the server class
    server = StringConcatenationServer()

    # Register the server object with the Pyro nameserver
    uri = daemon.register(server)
    ns.register("string.concatenation", uri)

    print("Server URI:", uri)

    with open("server_uri.txt", "w") as f:
        f.write(str(uri))
```

```
    daemon.requestLoop()

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

Client Implementation:

 # client.py

```
import Pyro4

def main():
    with open("server_uri.txt", "r") as f:
        uri = f.read()

    server = Pyro4.Proxy(uri)  # Connect to the remote server

    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")

    result = server.concatenate_strings(str1, str2)

    print("Concatenated Result:", result)

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

Steps to Run:

Install Pyro4 library

Then Use command Pyro4-ns

And use following steps

1)Save the server code in a file, e.g., server.py

2)Save the client code in a file, e.g., client.py

3)Open a terminal and run the server: python server.py

4)you will get server uri paste it inserver_uri.txt file.keep it in same folder where you have stored python files.

5)Open another terminal and run the client: python client.py

6)enter the values for concatenation.

CONCLUSION:

In this way we have 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.

ORAL QUESTION

1. What is RMI (Remote Method Invocation) and how does it facilitate communication between distributed Java applications?

2. Explain the client-server architecture in the context of RMI-based distributed applications.

3. How would you define the role of the client and the server in an RMI-based system for remote computation?

4. Can you outline the steps involved in implementing an RMI-based solution for remote computation in Java?

5. Describe the process of registering a remote object with the RMI registry.

Server.py

```
import Pyro4

@Pyro4.expose
class StringConcatenationServer:
    def concatenate_strings(self, str1, str2):
        result = str1 + str2
        return result

def main():
    daemon = Pyro4.Daemon()  # Create a Pyro daemon
    ns = Pyro4.locateNS()  # Locate the Pyro nameserver

    # Create an instance of the server class
    server = StringConcatenationServer()

    # Register the server object with the Pyro nameserver
    uri = daemon.register(server)
    ns.register("string.concatenation", uri)

    print("Server URI:", uri)

    with open("server_uri.txt", "w") as f:
        f.write(str(uri))

    daemon.requestLoop()

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

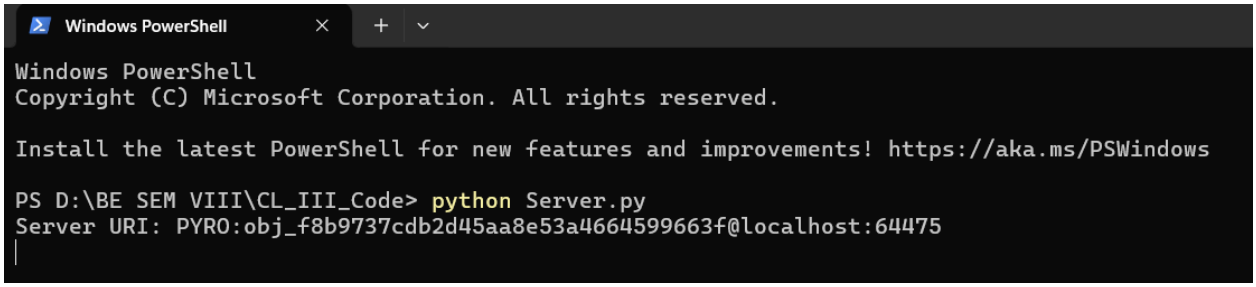
Client.py

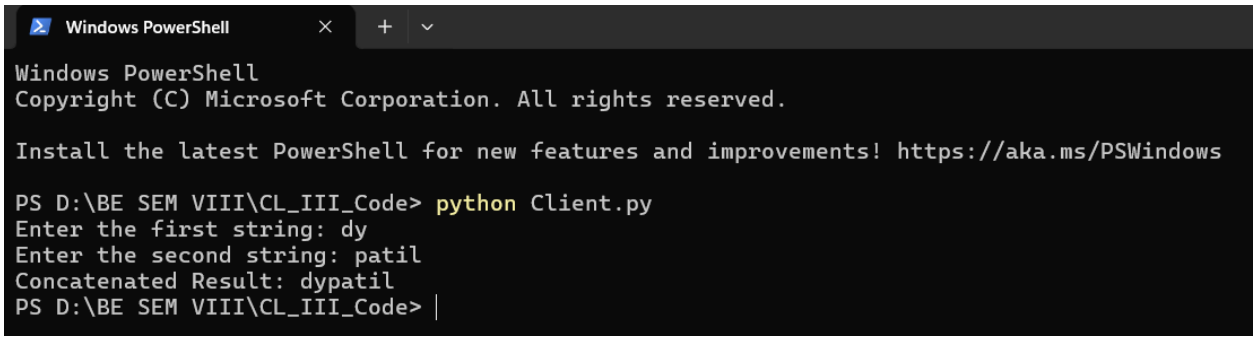
import Pyro4

```
def main():
    with open("server_uri.txt", "r") as f:
        uri = f.read()

    server = Pyro4.Proxy(uri)  # Connect to the remote server
    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")
    result = server.concatenate_strings(str1, str2)
    print("Concatenated Result:", result)
```

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

Output:





ASSIGNMENT 3

PROBLEM STATEMENT: -

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.

OBJECTIVE:

3. To learn and understand the different operations on fuzzy sets.
4. To learn and understand the min-max composition on fuzzy set.

THEORY:

A fuzzy set is an extension of a classical set in which each element has a degree of membership between 0 and 1. Unlike classical sets where an element either belongs (membership degree 1) or does not belong (membership degree 0), fuzzy sets allow for gradual membership degrees, representing the degree to which an element belongs to the set. The membership degree is often described by a membership function.

Operations on Fuzzy Sets

Having two fuzzy sets $\tilde{A}$ and $\tilde{B}$, the universe of information U and an element y of the universe, the following relations express the union, intersection and complement operation on fuzzy sets.

Union of fuzzy sets:

- The union of two fuzzy sets A and B with membership functions μA and μB, respectively, is a fuzzy set C, denoted C = A ∪ B, with the membership function μC.
- There are two definitions for the union operation: the max membership function and the product rule, as defined in following equations:

$$\mu C(x) = \max [\mu A(x), \mu B(x)] \quad (2.12)$$

$$\mu C(x) = \mu A(x) + \mu B(x) - \mu A(x)\mu B(x)$$

where x is an element in the universe of discourse X.

- Example:

  Let A and B be two fuzzy sets in the universe of discourse X and (x1, x2, x3, x4) ∈ X defined

as follows:

A = {0/x1 + 1/x2 + 0.7/x3 + 0.4/x4 + 0.2/x5 + 0/x6}

B = {0/x1 + 0.4/x2 + 0.7/x3 + 0.8/x4 + 1/x5 + 0/x6}

The union of fuzzy sets A and B using the max membership function is:

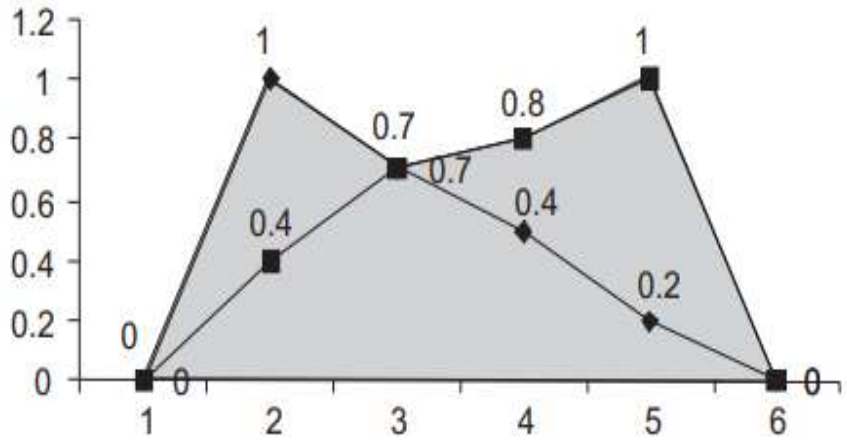
Cmax = A ∪ B = {0/x1 + 1/x2 + 0.7/x3 + 0.8/x4 + 1/x5 + 0/x6}

where μC(xi) is calculated from max [μA(xi), μB(xi)] for i = 1, 2, 3,..., 6.

Alternatively, using the product rule it is:

Cprod = A ∪ B = {0/x1 + 1/x2 + 0.91/x3 + 0.88/x4 + 1/x5 + 0/x6}

where μC(xi) is calculated using [μA(xi) + μB(xi) − μA(xi) ∗ μB(xi)] for i = 1, 2, 3,..., 6.



Union of fuzzy sets *A* and *B* using max operation

Intersection of fuzzy sets:

- The intersection of two fuzzy sets A and B with membership functions μA and μB, respectively, is a fuzzy set C, denoted C = A ∩ B, with membership function μC defined using the min membership function or the product rule as follows:

$$\mu C(x) = \min [\mu A(x), \mu B(x)]$$

$$\mu C(x) = \mu A(x) * \mu B(x)$$

- Example:

  Let A and B be two fuzzy sets in the universe of discourse X and (x1, x2, x3, x4) □ X defined as in the previous example. The intersection of fuzzy sets A and B using the min membership function is

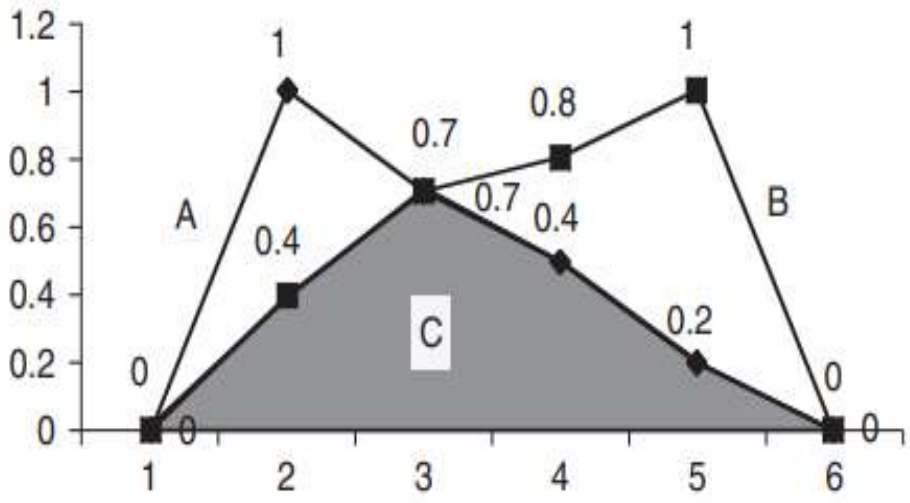
  Cmin = A ∩ B = {0/x1 + 0.4/x2 + 0.7/x3 + 0.4/x4 + 0.2/x5 + 0/x6}

  where μC(xi) is calculated from μC(x) = min [μA(x), μB(x)] for i = 1, 2, 3,..., 6.

  Alternatively using the product rule it is

  Cprod = A ∩ B = {0/x1 + 0.4/x2 + 0.49/x3 + 0.32/x4 + 0.2/x5 + 0/x6}

  where μC(xi) is calculated from μC(x) = μA(x) □ μB(x) for i = 1, 2, 3,..., 6.



Intersection of fuzzy sets *A* and *B* using the min operation

Complement of fuzzy set:

- The complement of a fuzzy set A with membership function μA is a fuzzy set, denoted ~A, with membership function $\mu_{\sim A}$ defined as

$$\mu_{\sim A}(x) = 1 - \mu_A(x)$$

- Example:

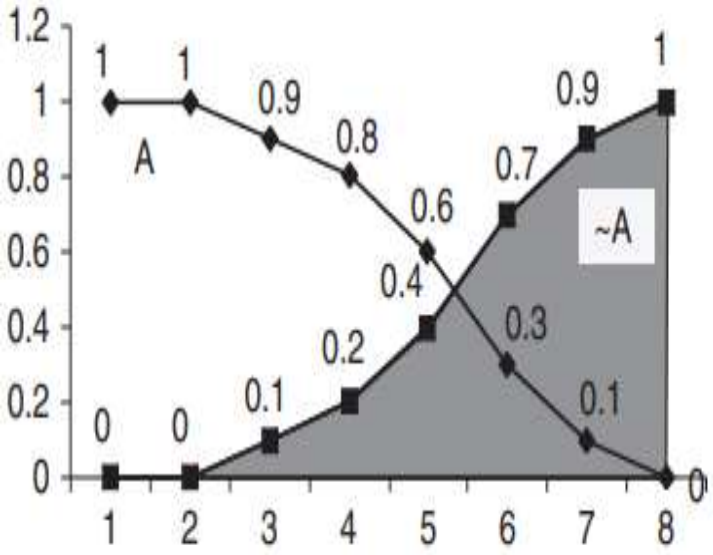
  Let A be a fuzzy set in the universe of discourse X and (x1, x2, x3, x4, x5, x6, x7, x8) ∈ X defined as follows

  A = {1/x1 + 1/x2 + 0.9/x3 + 0.8/x4 + 0.7/x5 + 0.3/x6 + 0.1/x7 + 0/x8}

  The complement of fuzzy set A is ~A :

  ~ A = {0/x1 + 0/x2 + 0.1/x3 + 0.2/x4 + 0.3/x5 + 0.7/x6 + 0.9/x7 + 1/x8}

  where μ~A(x) is calculated from [1 − μA(x)] for i = 1, 2, 3,..., 8.



Complement of fuzzy set *A*

Fuzzy Relations

- The concept of a relation has a natural extension to fuzzy sets and plays an important role in the theory of such sets and their applications.
- A fuzzy relation *R* from the fuzzy set *A* in *X* to the fuzzy set *B* in *Y* is a fuzzy set defined by the Cartesian product $A \times B$ in the Cartesian product space $X \times Y$.
- R is characterized by the membership function expressing various degrees of strength of

relations:

$$R = A \times B = \sum \mu_R(x, y)/(x, y) = \sum \min(\mu_A(x), \mu_B(y))$$

$$R = A \times B = \sum \mu_R(x, y)/(x, y) = \sum \mu_A(x)^* \mu_B(y)$$

- In Equations the sum does not mean a mathematical summation operation, it means all possible combinations of all elements.
- R is also called the relational matrix. The Cartesian product is implemented in the same fashion, as is the cross product of two vectors. For example, fuzzy set A with 4 elements (a column vector of dimension 4×1) and fuzzy set B with 5 elements (a row vector of dimension 1×5) will provide the resulting fuzzy relation R which is represented by a matrix of dimension 4×5.
- Example:

  Let A and B be two fuzzy sets defined by

  A = {1/1 + 0.8/2 + 0.6/3 + 0.5/4}

  B = {0.5/1 + 1/2 + 0.3/3 + 0/4}

  The fuzzy relation (i.e., the Cartesian product of A and B using the min operation) will be

$$\mathbf{R} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} \{1,.5\} & \{1,1\} & \{1,.3\} & \{1,0\} \\ \{.8,.5\} & \{.8,1\} & \{.8,.3\} & \{.8,0\} \\ \{.6,.5\} & \{.6,1\} & \{.6,.3\} & \{.6,0\} \\ \{.5,.5\} & \{.5,1\} & \{.5,.3\} & \{.5,0\} \end{bmatrix} = \begin{bmatrix} 0.5 & 1 & 0.3 & 0 \\ 0.5 & 0.8 & 0.3 & 0 \\ 0.5 & 0.6 & 0.3 & 0 \\ 0.5 & 0.5 & 0.3 & 0 \end{bmatrix}$$

  The fuzzy relation using the product operation will be

$$\mathbf{R} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} \{1,.5\} & \{1,1\} & \{1,.3\} & \{1,0\} \\ \{.8,.5\} & \{.8,1\} & \{.8,.3\} & \{.8,0\} \\ \{.6,.5\} & \{.6,1\} & \{.6,.3\} & \{.6,0\} \\ \{.5,.5\} & \{.5,1\} & \{.5,.3\} & \{.5,0\} \end{bmatrix} = \begin{bmatrix} 0.5 & 1 & 0.3 & 0 \\ 0.4 & 0.8 & 0.24 & 0 \\ 0.3 & 0.6 & 0.18 & 0 \\ 0.25 & 0.5 & 0.15 & 0 \end{bmatrix}$$

Max-min composition

- If R is a fuzzy relation in X × Y and A is a fuzzy set in X then the fuzzy set B in Y is given by

$$B = A \circ R$$

B is inferred from A using the relation matrix R which defines the mapping between X and Y and the operation '∘' is defined as the max/min operation.

- Example:

Let A be a fuzzy set defined by

A = {0.9/1 + 0.4/2 + 0/3}

With the fuzzy relation R given by the following relational matrix:

$$\mathbf{R} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1 & 0.8 & 0.1 \\ 0.8 & 0.6 & 0.3 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}$$

Then the fuzzy output B in Y using the max/min operation will be

$$\mathbf{B} = \mathbf{A} \circ \mathbf{R} = \begin{bmatrix} \frac{0.9}{1} & \frac{0.4}{2} & \frac{0}{3} \end{bmatrix} \circ \begin{bmatrix} 1 & 0.8 & 0.1 \\ 0.8 & 0.6 & 0.3 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \{0.9, 1\} & \{0.9, 0.8\} & \{0.9, 0.1\} \\ \{0.4, 0.8\} & \{0.4, 0.6\} & \{0.4, 0.3\} \\ \{0, 0.6\} & \{0, 0.3\} & \{0, 0.1\} \end{bmatrix}$$

Taking the minimum values row-wise, we obtain

$$\mathbf{B} = \begin{bmatrix} 0.9 & 0.8 & 0.1 \\ 0.4 & 0.4 & 0.3 \\ 0 & 0 & 0 \end{bmatrix}$$

Taking the maximum values column-wise, we obtain the fuzzy set B from the compositional relation:

$$\mathbf{B} = \begin{bmatrix} 0.9 & 0.8 & 0.3 \end{bmatrix}$$

CONCLUSION:

In this way we have explored the operations on fuzzy sets, fuzzy relations by Cartesian product and max-min composition on fuzzy relations.

ORAL QUESTION

1. Explain the concept of a fuzzy set and its representation.
2. Define a fuzzy relation and its purpose in fuzzy logic.
3. How is the Cartesian product of two fuzzy sets computed?
4. Discuss the steps involved in performing max-min composition on fuzzy relations.

Code:

```
import numpy as np

# Function to perform Union operation on fuzzy sets
def fuzzy_union(A, B):
    return np.maximum(A, B)

# Function to perform Intersection operation on fuzzy sets
def fuzzy_intersection(A, B):
    return np.minimum(A, B)

# Function to perform Complement operation on a fuzzy set
def fuzzy_complement(A):
    return 1 - A

# Function to perform Difference operation on fuzzy sets
def fuzzy_difference(A, B):
    return np.maximum(A, 1 - B)

# Function to create fuzzy relation by Cartesian product of two fuzzy
sets
def cartesian_product(A, B):
    return np.outer(A, B)

# Function to perform Max-Min composition on two fuzzy relations
def max_min_composition(R, S):
    return np.max(np.minimum.outer(R, S), axis=1)

# Example usage
A = np.array([0.2, 0.4, 0.6, 0.8]) # Fuzzy set A
B = np.array([0.3, 0.5, 0.7, 0.9]) # Fuzzy set B

# Operations on fuzzy sets
union_result = fuzzy_union(A, B)
intersection_result = fuzzy_intersection(A, B)
complement_A = fuzzy_complement(A)
difference_result = fuzzy_difference(A, B)

print("Union:", union_result)
print("Intersection:", intersection_result)
print("Complement of A:", complement_A)
print("Difference:", difference_result)

# Fuzzy relations
```

```
R = np.array([0.2, 0.5, 0.4]) # Fuzzy relation R
S = np.array([0.6, 0.3, 0.7]) # Fuzzy relation S

# Cartesian product of fuzzy relations
cartesian_result = cartesian_product(R, S)

# Max-Min composition of fuzzy relations
composition_result = max_min_composition(R, S)

print("Cartesian product of R and S:")
print(cartesian_result)

print("Max-Min composition of R and S:")
print(composition_result)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Fuzzy.py"
Union: [0.3 0.5 0.7 0.9]
Intersection: [0.2 0.4 0.6 0.8]
Complement of A: [0.8 0.6 0.4 0.2]
Difference: [0.7 0.5 0.6 0.8]
Cartesian product of R and S:
[[0.12 0.06 0.14]
 [0.3  0.15 0.35]
 [0.24 0.12 0.28]]
Max-Min composition of R and S:
[0.2 0.5 0.4]
PS D:\BE SEM VIII>
```

ASSIGNMENT 4

PROBLEM STATEMENT: -

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

OBJECTIVE:

1. Students should be able to Implement various load balancing algorithms to distribute requests among servers efficiently.
2. Students should be able to Simulate the processing of requests by servers.

THEORY:

What are Load Balancers?

In case multiple servers are present the incoming request coming to the system needs to be directed to one of the multiple servers. We should ensure that every server gets an equal number of requests. The requests must be distributed in a uniform manner across all the servers. The component which is responsible for distributing these incoming requests uniformly across the servers is known as Load Balancer. A Load Balancer acts as a layer between the incoming requests coming from the user and multiple servers present in the system.

We should avoid the scenarios where a single server is getting most of the requests while the rest of them are sitting idle. There are various Load Balancing Algorithms that ensure even distribution of requests across the servers.

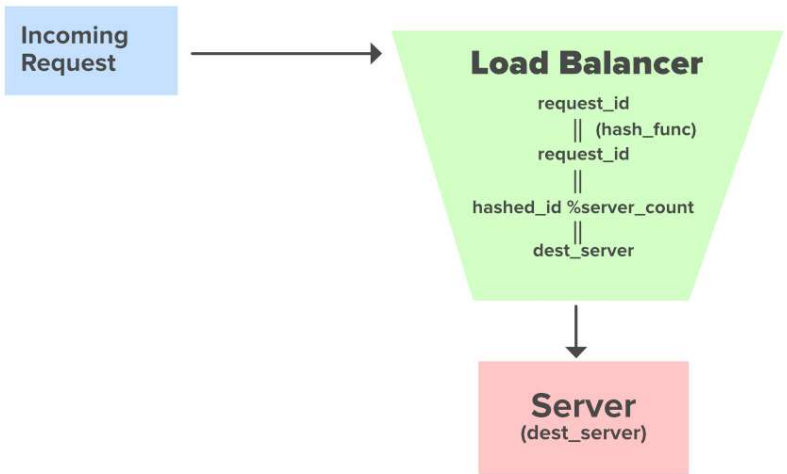


Fig. Hashing Approach to direct requests from the Load Balancer

We will be discussing the Hashing Approach to direct the requests to multiple servers uniformly. Suppose we have server_count as the Total number of servers present in the System and a load_balancer to distribute the requests among those servers. A request with an id request_id enters the system. Before reaching the destination server it is directed to the load_balancer from where it is further directed to its destination server. When the request reaches the load balancer the hashing approach will provide us with the destination server where the request is to be directed.

- **request_id** : Request ID coming to get served
- **hash_func** : Evenly distributed Hash Function
- **hashed_id** : Hashed Request ID
- **server_count** : Number of Servers

Java Code:

```
class GFG {
  public static int hash_func(int request_id)
  {
    // Computing the hash request id
    int hashed_id = 112;
    return hashed_id;
  }

  public static void route_request_to_server(int dest_server)
  {
    System.out.println("Routing request to the Server ID : " + dest_server);
  }

  public static int request_id = 23; // Incoming Request ID
  public static int server_count = 10; // Total Number of Servers

  public static void main(String args[])
  {
    int hashed_id = hash_func(request_id); // Hashing the incoming request id
    int dest_server = hashed_id % server_count; // Computing the destination server id

    route_request_to_server(dest_server);
  }
}
```

Python:

The Load Balancer class has two methods: round robin for round-robin load balancing and random selection for random load balancing. The simulate_client_requests function simulates client requests and prints the server selected by each algorithm for each request.

```
import random


class LoadBalancer:
  def __init__(self, servers):
    self.servers = servers
```

```
        self.server_index_rr = 0

    def round_robin(self):
        server = self.servers[self.server_index_rr]
        self.server_index_rr = (self.server_index_rr + 1) % len(self.servers)
        return server

    def random_selection(self):
        return random.choice(self.servers)

def simulate_client_requests(load_balancer, num_requests):
    for i in range(num_requests):
        # Simulating client request
        print(f"Request {i+1}: ", end="")

        # Using Round Robin algorithm for load balancing
        server_rr = load_balancer.round_robin()
        print(f"Round Robin - Server {server_rr}")

        # Using Random algorithm for load balancing
        server_random = load_balancer.random_selection()
        print(f"Random - Server {server_random}")

        print()

if __name__ == "__main__":
    # List of servers
    servers = ["Server A", "Server B", "Server C"]

    # Create a LoadBalancer instance
    load_balancer = LoadBalancer(servers)

    # Simulate 10 client requests
    simulate_client_requests(load_balancer, 10)
```

CONCLUSION:

In this way we have design a code that simulates client requests and distributes them among servers using the Round Robin load balancing algorithm. This demonstrates a basic implementation of load balancing in a simulated environment.

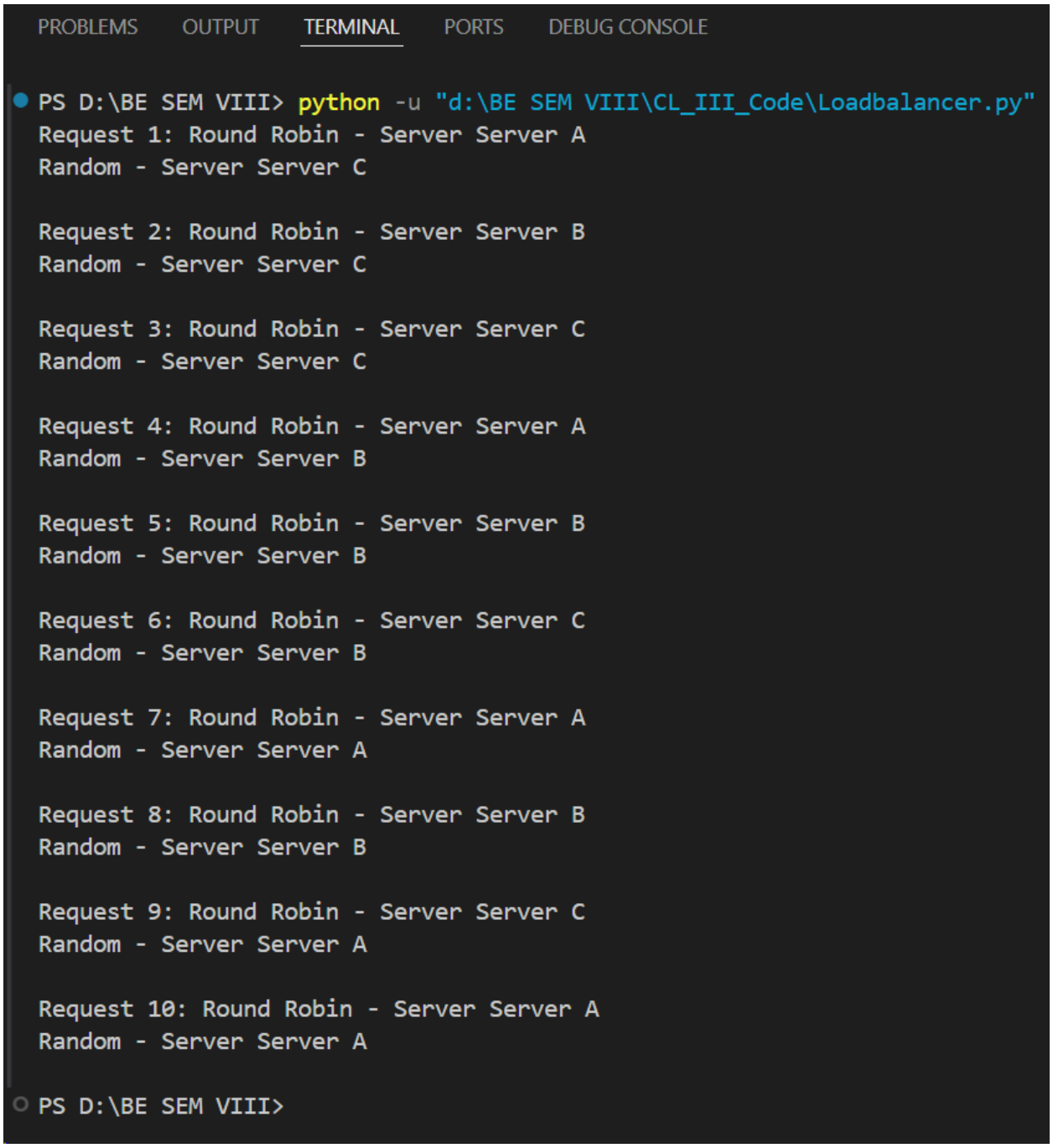
ORAL QUESTION:

1. What are the common load balancing algorithms?
2. How does the round-robin algorithm distribute requests among servers?
3. How does load balancing contribute to the scalability and efficiency of a distributed system?

Code:

```
import random

class LoadBalancer:
    def __init__(self, servers):
        self.servers = servers
        self.server_index_rr = 0

    def round_robin(self):
        server = self.servers[self.server_index_rr]
        self.server_index_rr = (self.server_index_rr + 1) %
len(self.servers)
        return server

    def random_selection(self):
        return random.choice(self.servers)

def simulate_client_requests(load_balancer, num_requests):
    for i in range(num_requests):
        # Simulating client request
        print(f"Request {i+1}: ", end="")

        # Using Round Robin algorithm for load balancing
        server_rr = load_balancer.round_robin()
        print(f"Round Robin - Server {server_rr}")

        # Using Random algorithm for load balancing
        server_random = load_balancer.random_selection()
        print(f"Random - Server {server_random}")
        print()

if __name__ == "__main__":
    # List of servers
    servers = ["Server A", "Server B", "Server C"]

    # Create a LoadBalancer instance
    load_balancer = LoadBalancer(servers)

    # Simulate 10 client requests
    simulate_client_requests(load_balancer, 10)
```

Output:


PROBLEMS
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Loadbalancer.py"
Request 1: Round Robin - Server Server A
Random - Server Server C
Request 2: Round Robin - Server Server B
Random - Server Server C
Request 3: Round Robin - Server Server C
Random - Server Server C
Request 4: Round Robin - Server Server A
Random - Server Server B
Request 5: Round Robin - Server Server B
Random - Server Server B
Request 6: Round Robin - Server Server C
Random - Server Server B
Request 7: Round Robin - Server Server A
Random - Server Server A
Request 8: Round Robin - Server Server B
Random - Server Server B
Request 9: Round Robin - Server Server C
Random - Server Server A
Request 10: Round Robin - Server Server A
Random - Server Server A
PS D:\BE SEM VIII>

ASSIGNMENT 5

PROBLEM STATEMENT: -

Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.

OBJECTIVE:

1. To learn and understand the genetic algorithm and its parameters.
2. Optimize genetic algorithm parameters to create a more robust and accurate hybrid model that can effectively simulate and predict outcomes in the spray drying of coconut milk.

THEORY:

Genetic Algorithms (GAs):

- GAs are inspired by the principles of natural selection and genetics.
- They operate on a population of candidate solutions, where each solution represents a set of parameters.
- They operate on a population of candidate solutions, where each solution represents a set of parameters.
- GAs use selection, crossover, and mutation operators to iteratively evolve the population towards better solutions.
- The fitness function evaluates how well a solution performs on the given problem.
- GAs are often used for optimization problems where the search space is complex and non-linear.

Neural Networks (NNs):

- NNs are computational models inspired by the structure and function of biological neural networks in the brain.
- They consist of interconnected layers of artificial neurons (nodes), each performing simple mathematical operations.
- NNs are trained using algorithms like backpropagation, where the model adjusts its parameters to minimize the difference between predicted and actual outputs.
- NNs excel at capturing complex patterns and relationships in data, making them suitable for various prediction tasks, including regression and classification.

Hybrid Genetic Algorithm-Neural Network Modeling:

- The hybrid approach combines the strengths of GAs and NNs to create a more powerful optimization and prediction framework.
- GAs are used to optimize the parameters of the neural network model, such as the architecture (number of layers, neurons per layer), activation functions, learning rates, etc.
- The genetic algorithm evolves a population of neural network configurations, searching for the combination of parameters that yields the best performance on a given objective function or fitness metric.
- By integrating GAs with NNs, the hybrid approach can effectively explore the high-dimensional parameter space, overcoming local optima and finding near-optimal solutions.

Application to Spray Drying of Coconut Milk:

- In the context of spray drying of coconut milk, the hybrid genetic algorithm-neural network modeling approach aims to optimize the process parameters (e.g., inlet temperature, feed flow rate, air flow rate, nozzle pressure, etc.) for achieving desired product characteristics (e.g., moisture content, particle size distribution, color, etc.).
- The hybrid model can simulate the spray drying process and predict the quality attributes of the dried coconut milk products.
- By optimizing genetic algorithm parameters, such as population size, crossover rate, and mutation rate, the hybrid model can efficiently search the parameter space to find optimal or near-optimal solutions for improved spray drying process performance and product quality.

Following are the steps to implement the optimization of genetic algorithm parameters in a hybrid genetic algorithm-neural network modeling framework for the application of spray drying of coconut milk:

1. Gather data related to the spray drying process of coconut milk. This may include various parameters such as temperature, humidity, airflow rate, feed composition, etc. Preprocess the data as needed, including normalization, encoding categorical variables, and splitting into training and testing sets.
2. Define the structure of the neural network that will be used in the hybrid model. This includes specifying the number of layers, neurons per layer, activation functions, and other architectural parameters.
3. Determine the parameters of the genetic algorithm that will be optimized. This may include population size, mutation rate, crossover rate, selection mechanisms, and termination criteria.
4. Encode the neural network architecture parameters and other relevant parameters into a form that can be optimized by the genetic algorithm. This typically involves defining a chromosome representation for the parameters.

5. Define a fitness function that evaluates the performance of the hybrid model based on the given parameters. This function typically involves training the neural network using the training data and evaluating its performance on the testing data.
6. Implement the genetic algorithm to search for the optimal set of parameters. This involves initializing a population of parameter sets, evaluating their fitness using the fitness function, and applying genetic operators such as selection, crossover, and mutation to evolve the population over multiple generations.
7. Define termination criteria for the genetic algorithm optimization process. This could be a maximum number of generations, reaching a certain fitness threshold, or convergence criteria.
8. After the optimization process completes, analyze the results to identify the optimal set of parameters and evaluate the performance of the hybrid model using these parameters.

CONCLUSION:

In this way we have optimized genetic algorithm parameters in a hybrid genetic algorithm-neural network model for coconut milk spray drying holds promise for enhancing predictive accuracy and process optimization

ORAL QUESTION

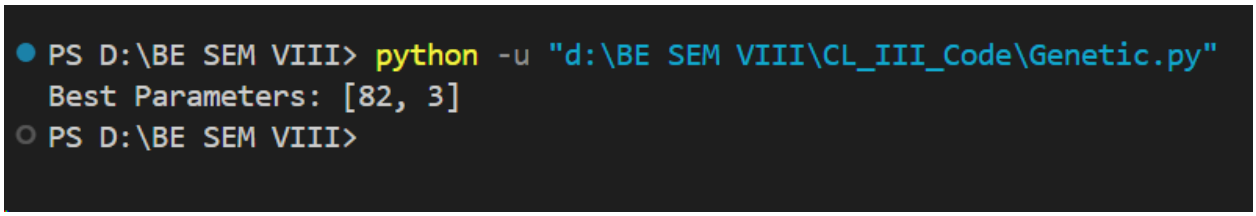
1. What are the key parameters of the genetic algorithm that can be optimized for improved performance?
2. Explain Genetic Algorithms.
3. How do genetic algorithms and neural networks complement each other in this hybrid model?

Code:

```
import random
from deap import base, creator, tools, algorithms

# Define evaluation function (this is a mock function, replace this
with your actual evaluation function)
def evaluate(individual):
    # Here 'individual' represents the parameters for the neural
network
    # You'll need to replace this with your actual evaluation
function that trains the neural network
    # and evaluates its performance
    # Return a fitness value (here, a random number is used as an
example)
    return random.random(),

# Define genetic algorithm parameters
POPULATION_SIZE = 10
GENERATIONS = 5

# Create types for fitness and individuals in the genetic algorithm
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# Initialize toolbox
toolbox = base.Toolbox()

# Define attributes and individuals
toolbox.register("attr_neurons", random.randint, 1, 100)  # Example:
number of neurons
toolbox.register("attr_layers", random.randint, 1, 5)    # Example:
number of layers
toolbox.register("individual", tools.initCycle, creator.Individual,
(toolbox.attr_neurons, toolbox.attr_layers), n=1)
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

# Genetic operators
toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=1, up=100,
indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

```
# Create initial population
population = toolbox.population(n=POPULATION_SIZE)

# Run the genetic algorithm
for gen in range(GENERATIONS):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5,
mutpb=0.1)

    fitnesses = toolbox.map(toolbox.evaluate, offspring)
    for ind, fit in zip(offspring, fitnesses):
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))

# Get the best individual from the final population
best_individual = tools.selBest(population, k=1)[0]
best_params = best_individual

# Print the best parameters found
print("Best Parameters:", best_params)
```

Output:



ASSIGNMENT 6

PROBLEM STATEMENT:

Implementation of Clonal selection algorithm using Python.

OBJECTIVE:

To efficiently solve optimization problems by emulating the adaptive immune system's principles.

THEORY:

The Clonal Selection Algorithm (CSA) is a bio-inspired optimization algorithm based on the principles of clonal selection and affinity maturation observed in the immune system. The theory behind implementing CSA using Python involves understanding its key components and mechanisms:

- **Clonal Selection:**

  In the immune system, B cells undergo clonal selection, where those with receptors (antibodies) that bind strongly to antigens are replicated (cloned) in response to an antigenic stimulus. This principle is mimicked in CSA, where candidate solutions (antibodies) with higher affinity to the problem's objective function (antigens) are replicated.

- **Affinity Maturation:**

  After clonal selection, B cells undergo affinity maturation, a process where their receptors undergo mutations to improve binding affinity to antigens. Similarly, in CSA, cloned candidate solutions undergo mutation or hypermutation to explore the search space and potentially improve their fitness.

- **Population Initialization:**

  The CSA starts with an initial population of candidate solutions (antibodies) representing potential solutions to the optimization problem.

- **Cloning Operation:**

  Candidate solutions are selected based on their fitness (affinity) to the problem's objective function, and a proportion of the fittest solutions are replicated or cloned to form an expanded population.

- **Mutation Operation:**

  Cloned candidate solutions undergo mutation or hyper mutation to introduce diversity into the population and explore new regions of the search space.

- **Selection Operation:**

  The mutated candidate solutions are evaluated, and the fittest individuals are selected to form the next generation population.

- **Termination Criteria:**

  The algorithm iterates through these operations until a termination criterion is met, such as reaching a maximum number of iterations or achieving a satisfactory solution quality.

Clonal Selection Algorithm (CSA):

1. Start by generating an initial population of candidate solutions (antibodies) randomly or using a heuristic method.
2. Evaluate the fitness of each candidate solution using the objective function, determining how well it performs in solving the optimization problem.
3. Select a proportion of the fittest candidate solutions based on their fitness scores to undergo cloning. The number of clones generated for each selected solution is proportional to its fitness.
4. Introduce diversity into the population by applying mutation or hypermutation to the cloned candidate solutions. This involves randomly modifying the cloned solutions to explore new regions of the search space.
5. Evaluate the fitness of the mutated candidate solutions.
6. Select candidate solutions for the next generation population based on their fitness, considering both the original candidate solutions and their mutated clones.
7. Repeat steps 3-6 for a predefined number of generations or until a termination criterion is met (e.g., a satisfactory solution is found, or a maximum number of iterations is reached).
8. Return the best candidate solution found during the optimization process as the solution to the optimization problem.

By following these steps, the Clonal Selection Algorithm iteratively evolves a population of candidate solutions, favoring those with higher fitness, to efficiently search for an optimal solution to the given optimization problem.

CONCLUSION

We've implemented a Clonal Selection Algorithm using Python. This implementation offers a practical and versatile solution for solving optimization problems.

ORAL QUESTION

1. How does the Clonal Selection Algorithm differ from traditional optimization algorithms?
2. What are the key components of your implementation of the Clonal Selection Algorithm?

Code:

```
import numpy as np

# Generate dummy data for demonstration purposes (replace this with
your actual data)
def generate_dummy_data(samples=100, features=10):
    data = np.random.rand(samples, features)
    labels = np.random.randint(0, 2, size=samples)
    return data, labels

# Define the AIRS algorithm
class AIRS:
    def __init__(self, num_detectors=10, hypermutation_rate=0.1):
        self.num_detectors = num_detectors
        self.hypermutation_rate = hypermutation_rate

    def train(self, X, y):
        self.detectors = X[np.random.choice(len(X),
self.num_detectors, replace=False)]

    def predict(self, X):
        predictions = []
        for sample in X:
            distances = np.linalg.norm(self.detectors - sample,
axis=1)
            prediction = int(np.argmin(distances))
            predictions.append(prediction)
        return predictions

# Generate dummy data
data, labels = generate_dummy_data()

# Split data into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * len(data))
train_data, test_data = data[:split_index], data[split_index:]
train_labels, test_labels = labels[:split_index],
labels[split_index:]

# Initialize and train AIRS
airs = AIRS(num_detectors=10, hypermutation_rate=0.1)
airs.train(train_data, train_labels)

# Test AIRS on the test set
predictions = airs.predict(test_data)
```

```
# Evaluate accuracy
accuracy = np.mean(predictions == test_labels)
print(f"Accuracy: {accuracy}")
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Clonal.py"
Accuracy: 0.05
PS D:\BE SEM VIII>
```

ASSIGNMENT 7

PROBLEM STATEMENT:

Implement DEAP (Distributed Evolutionary Algorithms) using Python

OBJECTIVE:

Students should be able to implement DEAP (Distributed Evolutionary Algorithms) using Python.

THEORY:

DEAP, which stands for Distributed Evolutionary Algorithms in Python, is a Python library specifically designed to implement and run Distributed Evolutionary Algorithms (DEAs). DEAP provides a flexible framework for developing and running evolutionary algorithms in distributed computing environments. It offers a wide range of tools and functionalities to facilitate the design, implementation, and execution of distributed evolutionary algorithms.

Key features of DEAP include:

- **Evolutionary Algorithm Components**: DEAP provides implementations of common evolutionary algorithm components such as individuals, populations, selection operators, mutation operators, and crossover operators. These components can be easily customized and combined to create various types of evolutionary algorithms tailored to specific optimization problems.
- **Parallel Evaluation**: DEAP supports parallel evaluation of individuals in a population, allowing for concurrent fitness evaluation across multiple processors or computing nodes. This feature enables efficient utilization of computational resources and accelerates the evolutionary search process.
- **Distributed Computing Support**: DEAP is designed to work in distributed computing environments, allowing evolutionary algorithms to be deployed across multiple machines or computing clusters. It provides support for communication, synchronization, and load balancing among distributed computing nodes, enabling scalable and efficient execution of evolutionary algorithms.
- **Flexibility and Extensibility**: DEAP offers a high degree of flexibility and extensibility, allowing users to easily customize and extend the library to suit their specific requirements. It provides a modular architecture that enables the integration of additional components and algorithms, as well as the implementation of novel evolutionary operators and strategies.
- **Integration with Optimization Problems**: DEAP can be seamlessly integrated with various optimization problems and domains, including combinatorial optimization, numerical optimization, and machine learning. It provides tools for encoding problem-specific representations, defining fitness evaluation functions, and specifying optimization objectives and constraints.

- **Community and Documentation**: DEAP has an active community of users and developers who contribute to its development and provide support to fellow users. The library is well-documented, with comprehensive tutorials, examples, and API documentation available to help users get started and effectively use the library for their projects.

Overall, DEAP is a powerful and versatile tool for implementing distributed evolutionary algorithms in Python, making it suitable for a wide range of optimization problems across different domains and applications.

High-level algorithm for the distributed evolutionary algorithm:

1. Generate an initial population of individuals using random values within the defined search space.
2. Evaluate the fitness of each individual in the population using the objective function.
3. Repeat for a fixed number of generations:
    a. Select individuals from the population for reproduction based on their fitness.
    b. Generate offspring by applying crossover and mutation operators on selected individuals.
    c. Evaluate the fitness of the generated offspring.
    d. Determine the individuals that will survive to the next generation. This may involve replacing the current population entirely or merging it with the offspring population.
    e. Check termination conditions. If satisfied, stop the algorithm; otherwise, continue to the next generation.
4. Output the best individual found during the evolutionary process.

CONCLUSION

In this way we have implement DEAP (Distributed Evolutionary Algorithms) using Python.

ORAL QUESTION

1. What is DEAP, and what is its primary purpose?
2. Explain the key components of DEAP?
3. What are the steps involved in using DEAP to solve optimization problem

Code:

```
import random
from deap import base, creator, tools, algorithms

# Define the evaluation function (minimize a simple mathematical
function)
def eval_func(individual):
    # Example evaluation function (minimize a quadratic function)
    return sum(x ** 2 for x in individual),

# DEAP setup
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# Define attributes and individuals
toolbox.register("attr_float", random.uniform, -5.0, 5.0)  # Example:
Float values between -5 and 5
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, n=3)  # Example: 3-dimensional individual
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

# Evaluation function and genetic operators
toolbox.register("evaluate", eval_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1,
indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# Create population
population = toolbox.population(n=50)

# Genetic Algorithm parameters
generations = 20

# Run the algorithm
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5,
mutpb=0.1)

    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
```

```
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))

# Get the best individual after generations
best_ind = tools.selBest(population, k=1)[0]
best_fitness = best_ind.fitness.values[0]

print("Best individual:", best_ind)
print("Best fitness:", best_fitness)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\DEAP.py"
Best individual: [-0.011174776506688588, -0.0063488374813361935, -0.033035424484573764]
Best fitness: 0.0012565226382148342
PS D:\BE SEM VIII>
```

ASSIGNMENT 8

PROBLEM STATEMENT: -

Design and develop a distributed Hotel booking application using Java RMI. A distributed hotel booking system consists of the hotel server and the client machines. The server manages hotel rooms booking information. A customer can invoke the following operations at his machine i) Book the room for the specific guest ii) Cancel the booking of a guest.

OBJECTIVE:

1. Students should be able to Implement the server-side logic that manages hotel room booking information. This includes storing information about booked rooms and handling booking and cancellation requests from clients.
2. Students should be able to Implement remote objects that implement the remote interfaces defined earlier. These objects will expose the booking and cancellation functionality to clients via RMI.
3. Students should be able to Start an RMI registry on the server machine. This registry will allow clients to look up remote objects by their names and obtain references to them.

Software Required: BlueJ IDE

THEORY:

Designing and developing a distributed Hotel Booking application using Java RMI involves creating a system where the server manages hotel room booking information, and clients interact with the server to book or cancel rooms for guests.

Components of the System:

1. Server: The server is responsible for managing hotel room booking information. It exposes remote methods that clients can invoke to book or cancel rooms.
2. Client: The client is the interface through which users interact with the system. It communicates with the server to perform booking and cancellation operations.

Remote Method Invocation (RMI):

Java RMI (Remote Method Invocation) allows Java objects to invoke methods on remote Java objects running on different JVMs (Java Virtual Machines). RMI provides a mechanism for distributed communication, enabling remote method calls between different Java applications.

Key Concepts in Java RMI:

1. Remote Interface: Define an interface that extends java.rmi. Remote. This interface declares the methods that can be invoked remotely by clients.
2. Remote Object: Implement the remote interface to create a remote object. This object is instantiated on the server and provides the implementation of the methods declared in the remote interface.
3. Server Implementation: Create a server class that instantiates the remote object, binds it to a registry, and listens for incoming client requests.
4. Client Implementation: Create a client class that looks up the remote object from the registry and invokes its methods.

Operations in the Distributed Hotel Booking System:

1. Book Room Operation:
    - The client invokes a remote method on the server to book a room for a specific guest.
    - The server updates its booking information and returns a confirmation to the client.
2. Cancel Booking Operation:
    - The client invokes a remote method on the server to cancel the booking of a guest.
    - The server updates its booking information and returns a confirmation to the client.

Workflow:

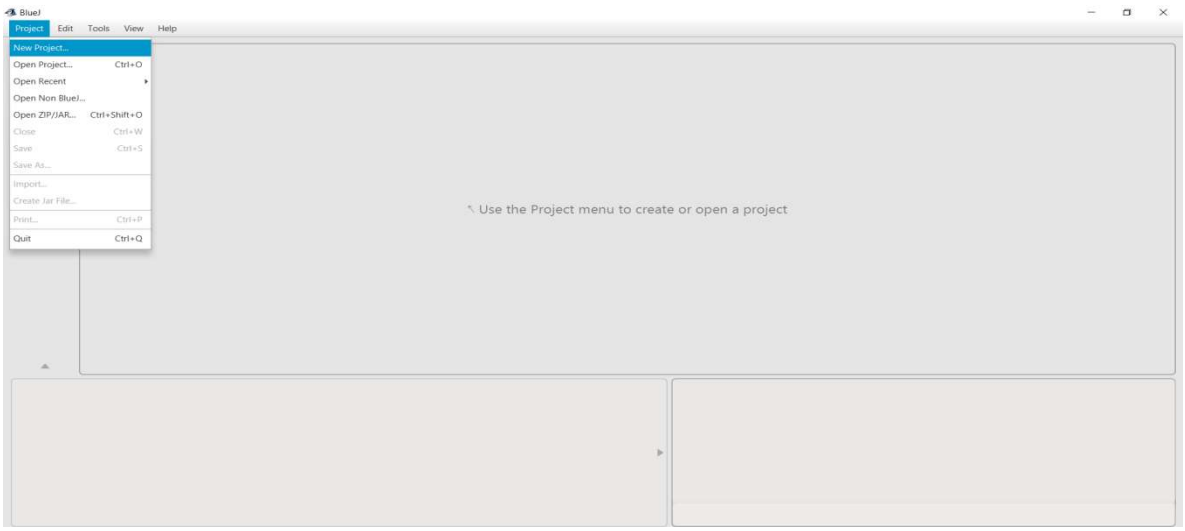
1. Server Setup:
    - The server creates an instance of the remote object implementing the booking system functionality.
    - The server binds this object to a specific name in the RMI registry, making it accessible to clients.
2. Client Interaction:
    - Clients retrieve the remote object reference from the RMI registry using its registered name.
    - Clients invoke methods on the remote object to perform booking or cancellation operations.
3. Server Response:

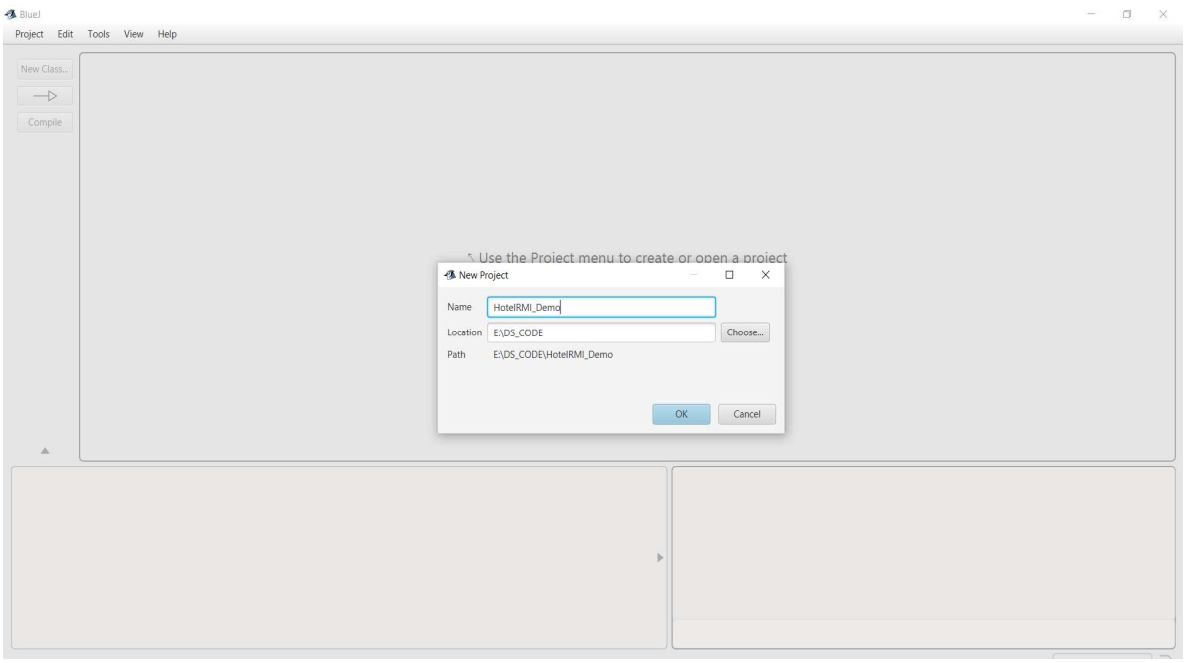
- The server processes client requests, updates booking information, and returns appropriate responses to clients.

Advantages of Java RMI in this Context:

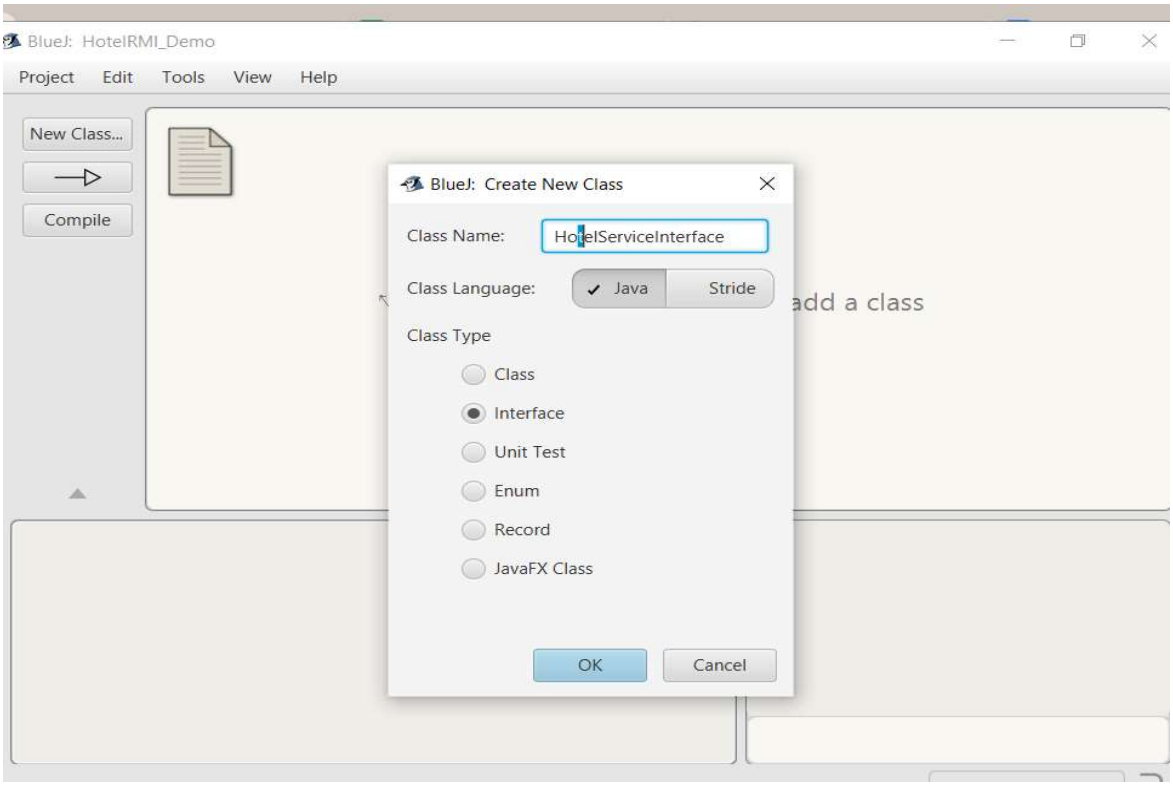
1. Ease of Development: Java RMI simplifies the development of distributed systems by abstracting network communication details.
2. Language Compatibility: Clients and servers can be written in Java, allowing seamless integration with existing Java applications.
3. Security: Java RMI provides built-in support for security features such as authentication and encryption, ensuring secure communication between clients and servers.
4. Performance: RMI is optimized for performance, making it suitable for real-time applications like hotel booking systems.

Step 1: Create Project and provide suitable name

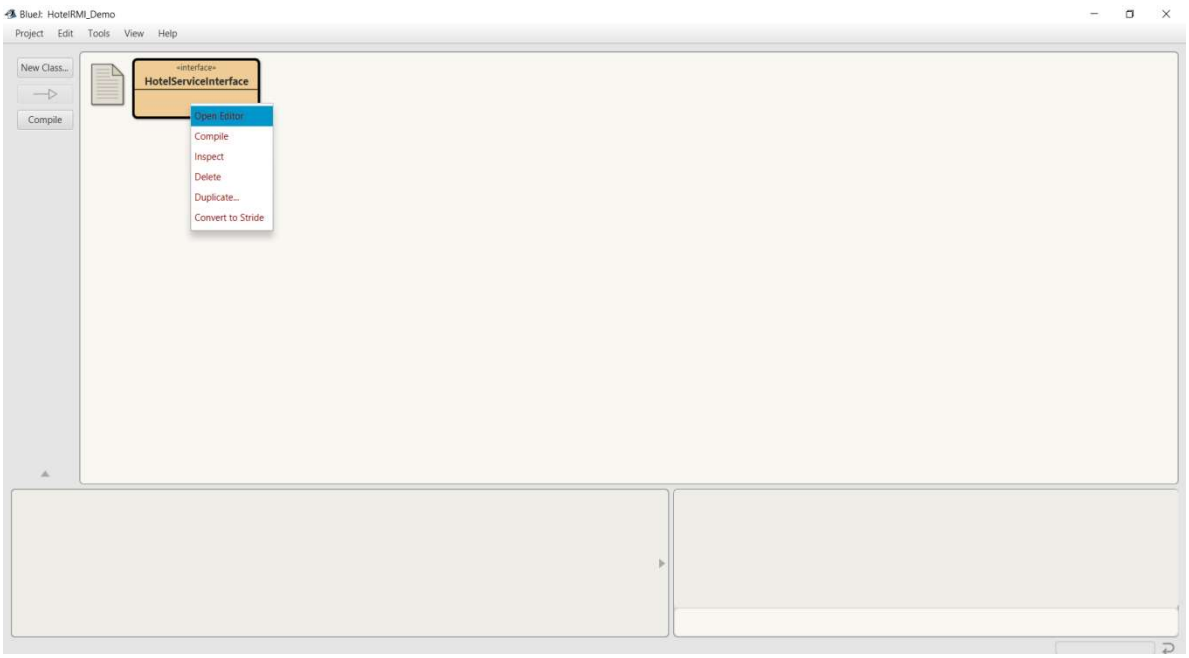




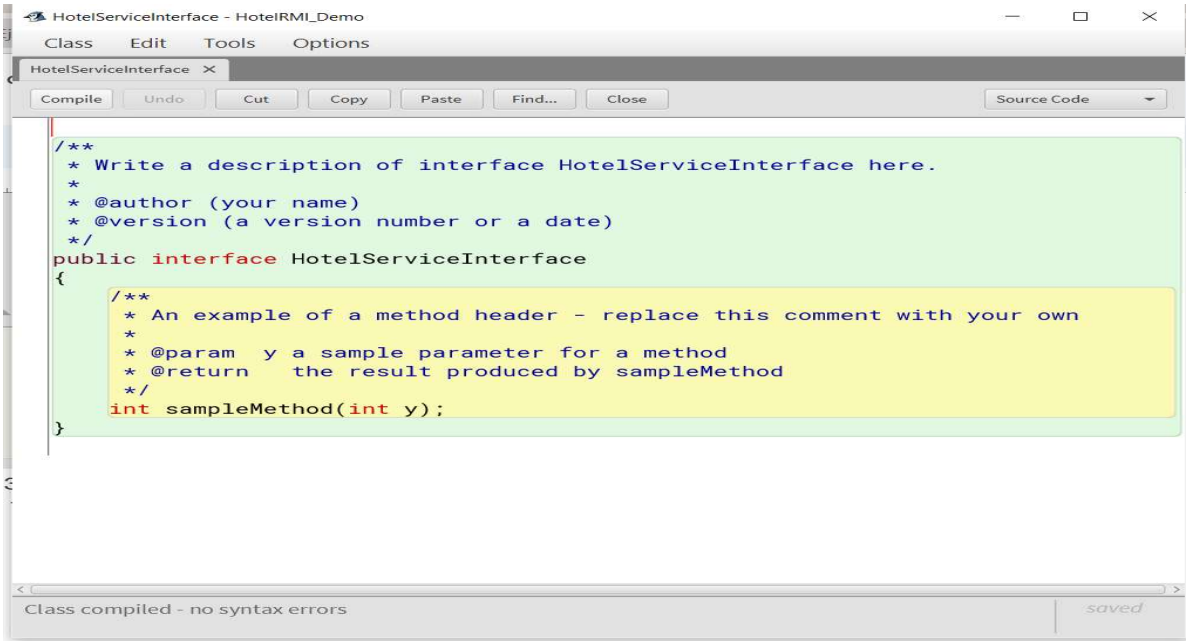
Step 2: Create Interface



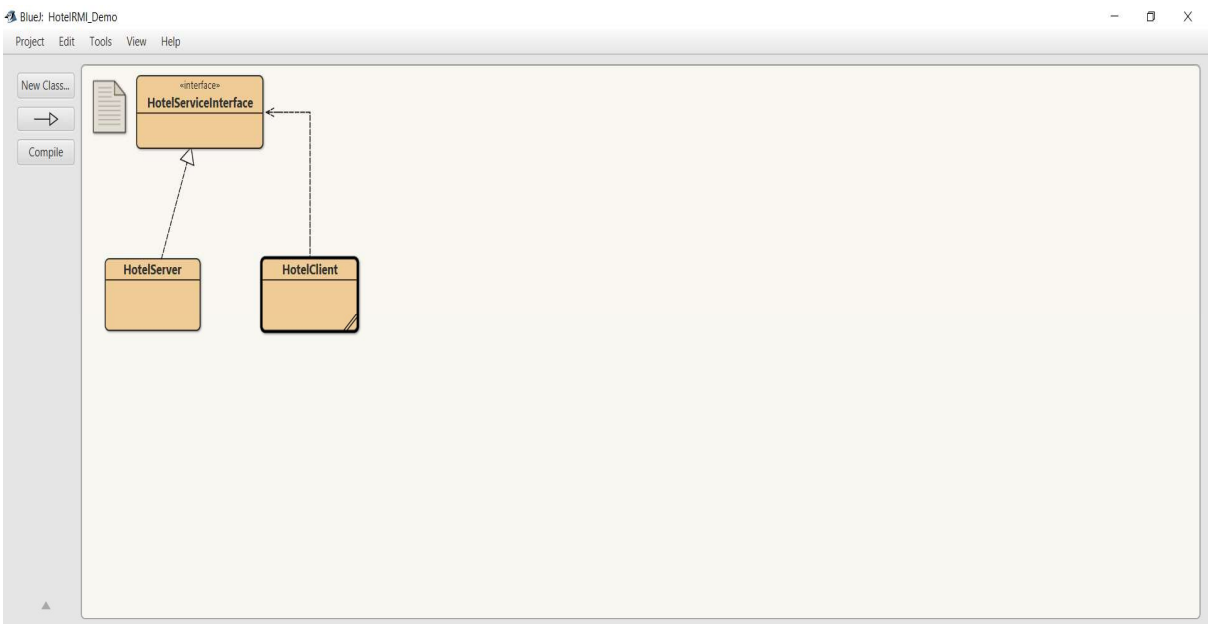
Step 3: Interface has been Created . Right click on Block of Interface.



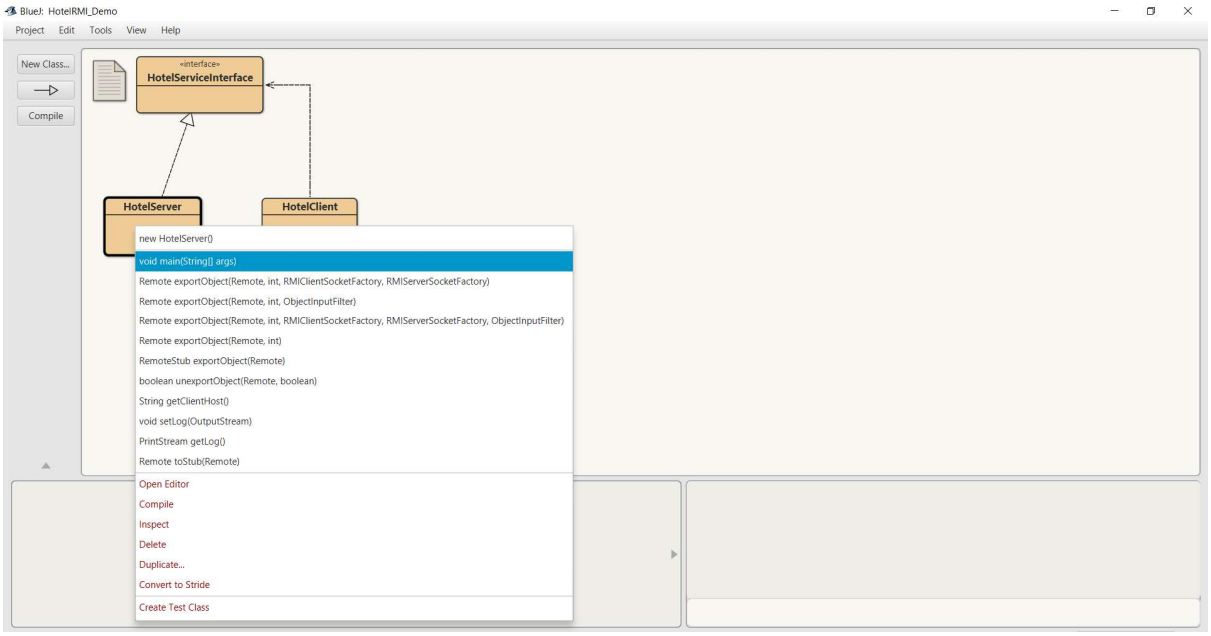
Step 4 : Click on Open Editor. Write the code of Interface here and Compile it.

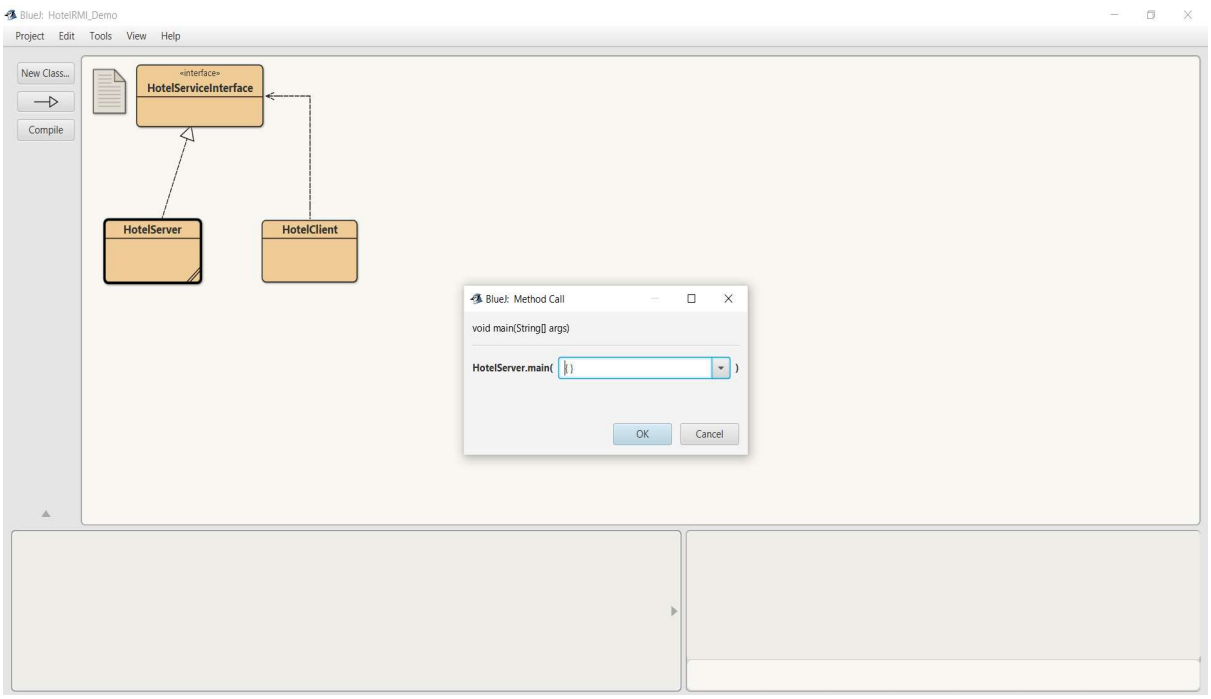


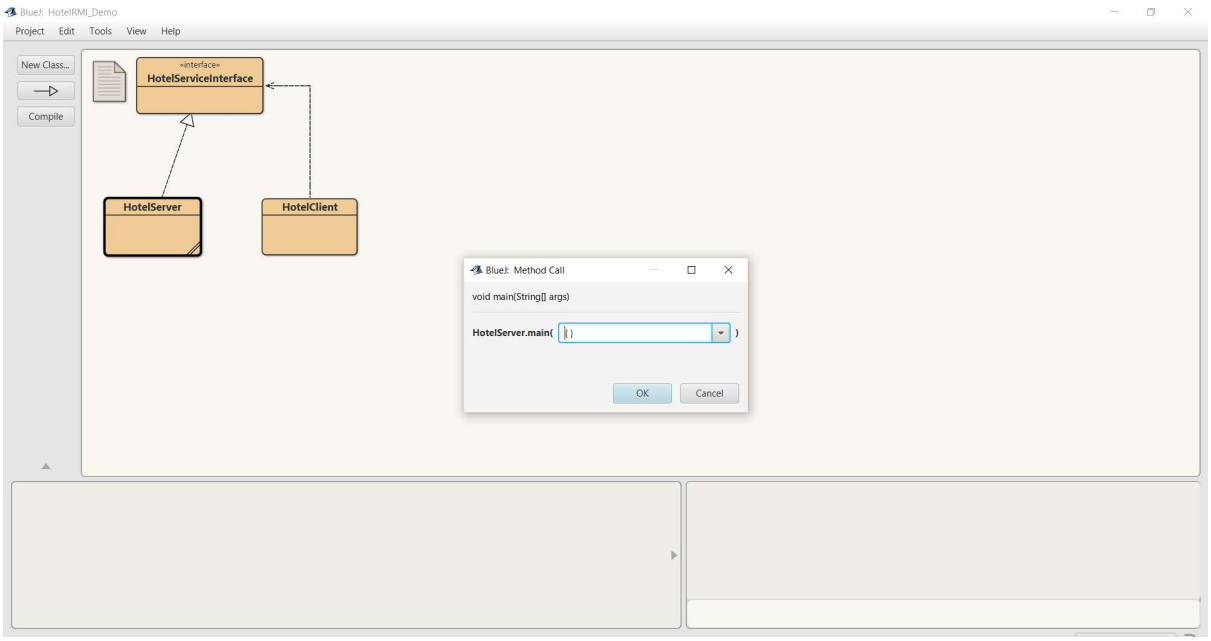
Step 5: In the similar way create two more classes HotelServer, and HotelClient. Open the editor and write the code and compile the both classes. On successfully completed the above procedure the diagram will be.



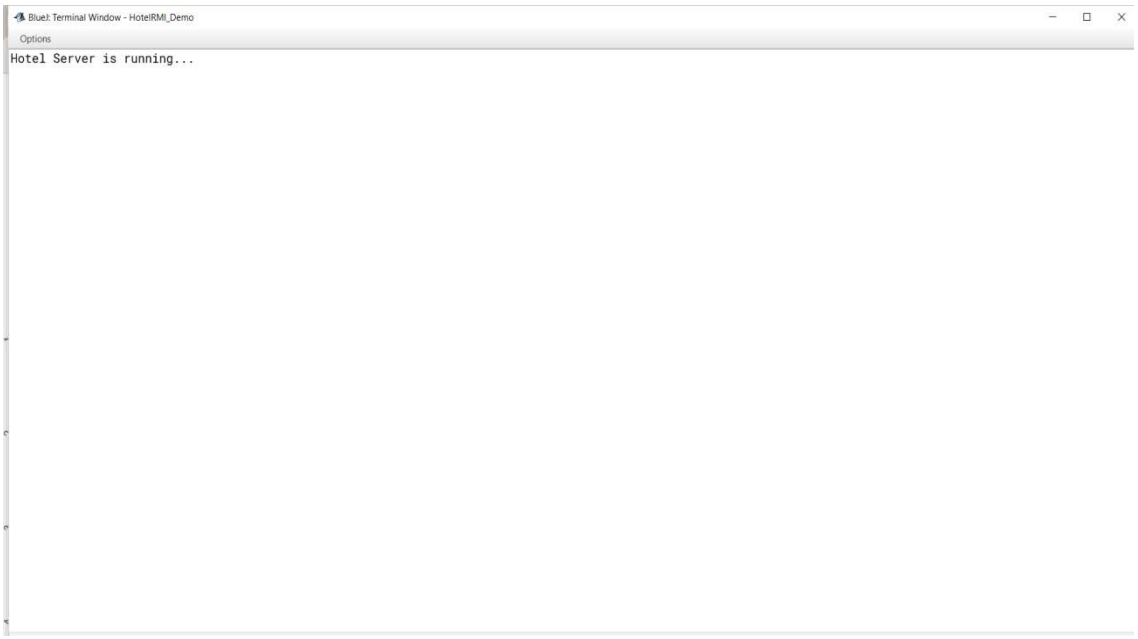
Step 6: Write Click on the block of HotelServer Class and select the main method call.



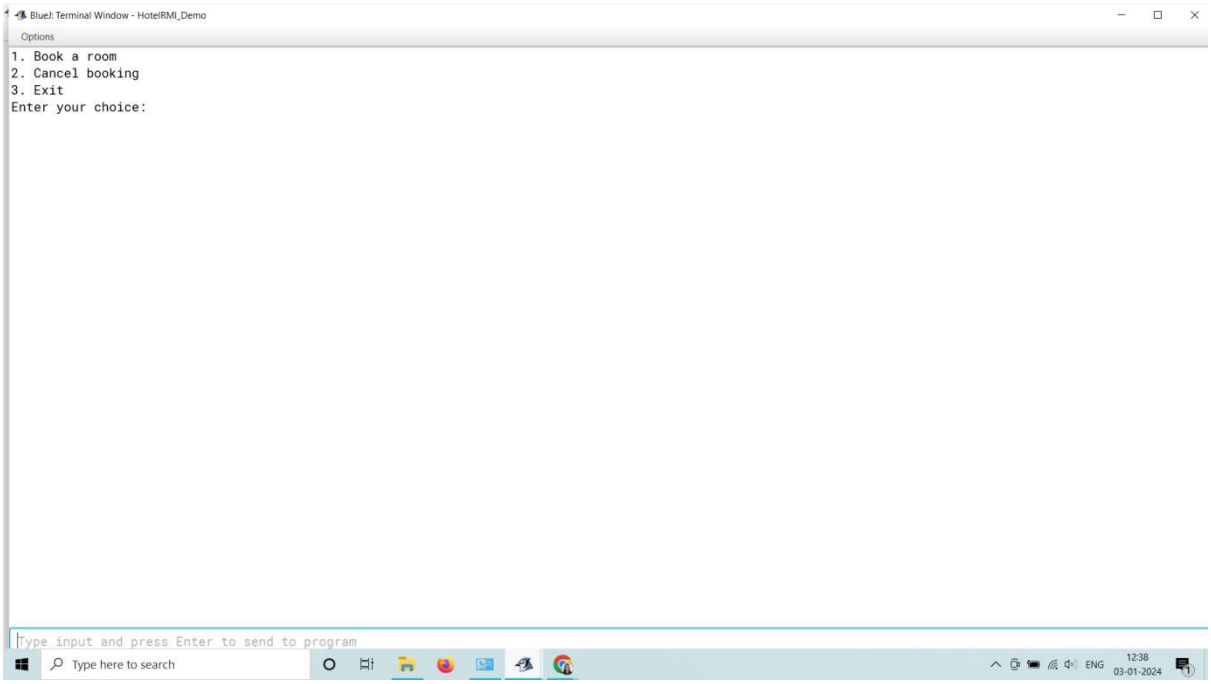




Step 7: Observe the Hotel Server is running on Terminal Window.



Step 8: In the similar way Right click on the block of Hotel Client Class and select the main method call. Observe the Hotel Client is running on Terminal Window.



Step 9

Give the appropriate choice in the Terminal window and Run the code successfully.

Implementation

Hotel Client

```
import java.rmi.Naming;
import java.util.Scanner;

public class HotelClient {
   public static void main(String[] args) {
      try {
         // Look up the RMI server object from the registry
         HotelServiceInterface hotelService = (HotelServiceInterface)
Naming.lookup("rmi://localhost/HotelService");

         Scanner scanner = new Scanner(System.in);

         while (true) {
            System.out.println("1. Book a room");
            System.out.println("2. Cancel booking");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // consume the newline character

            switch (choice) {
               case 1:
                  System.out.print("Enter guest name: ");
                  String guestName = scanner.nextLine();

                  System.out.print("Enter room number: ");
                  int roomNumber = scanner.nextInt();

                  boolean booked = hotelService.bookRoom(guestName, roomNumber);

                  if (booked) {
                     System.out.println("Room booked successfully!");
                  } else {
                     System.out.println("Room booking failed.");
                  }
                  break;

               case 2:
                  System.out.print("Enter guest name for cancellation: ");
                  String cancelGuestName = scanner.nextLine();

                  boolean canceled = hotelService.cancelBooking(cancelGuestName);

                  if (canceled) {
```

```
                    System.out.println("Booking canceled successfully!");
                } else {
                    System.out.println("Booking cancellation failed.");
                }
                break;

            case 3:
                System.out.println("Exiting the client application.");
                System.exit(0);
                break;

            default:
                System.out.println("Invalid choice. Please enter a valid option.");
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Hotel server

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import java.util.Map;

public class HotelServer extends UnicastRemoteObject implements
HotelServiceInterface {
  private Map<Integer, String> bookedRooms;

  public HotelServer() throws RemoteException {
    bookedRooms = new HashMap<>();
  }

  @Override
  public synchronized boolean bookRoom(String guestName, int roomNumber)
throws RemoteException {
    if (!bookedRooms.containsKey(roomNumber)) {
      bookedRooms.put(roomNumber, guestName);
      System.out.println("Room " + roomNumber + " booked for guest: " +
guestName);
      return true;
```

```
        } else {
            System.out.println("Room " + roomNumber + " is already booked.");
            return false;
        }
    }

    @Override
    public synchronized boolean cancelBooking(String guestName) throws
RemoteException {
        for (Map.Entry<Integer, String> entry : bookedRooms.entrySet()) {
            if (entry.getValue().equals(guestName)) {
                bookedRooms.remove(entry.getKey());
                System.out.println("Booking for guest " + guestName + " canceled.");
                return true;
            }
        }
        System.out.println("No booking found for guest " + guestName);
        return false;
    }

    public static void main(String[] args) {
        try {
            HotelServer server = new HotelServer();

            // Create and export the RMI registry on port 1099
            java.rmi.registry.LocateRegistry.createRegistry(1099);

            // Bind the server object to the registry
            Naming.rebind("HotelService", server);

            System.out.println("Hotel Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Hotel service Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HotelServiceInterface extends Remote {
    boolean bookRoom(String guestName, int roomNumber) throws RemoteException;
    boolean cancelBooking(String guestName) throws RemoteException;
```

}

CONCLUSION:

In this way we have by implementing the distributed Hotel Booking application using Java RMI, we can create a robust and scalable system for managing hotel room bookings efficiently.

ORAL QUESTION

1. Can you explain the benefits of using a distributed system for a hotel booking application?

2. What are the main components of the system, and how do they interact with each other?

3. What is Java RMI (Remote Method Invocation), and how does it facilitate communication between distributed components?

4. What are the necessary methods that need to be defined in the remote interface to support these operations?

Code:

HotelClient.java

```
import java.rmi.Naming;
import java.util.Scanner;
public class HotelClient {
 public static void main(String[] args) {
 try {
 // Look up the RMI server object from the registry
 HotelServiceInterface hotelService = (HotelServiceInterface)
Naming.lookup("rmi://localhost/HotelService");
 Scanner scanner = new Scanner(System.in);
 while (true) {
 System.out.println("1. Book a room");
 System.out.println("2. Cancel booking");
 System.out.println("3. Exit");
 System.out.print("Enter your choice: ");
 int choice = scanner.nextInt();
 scanner.nextLine(); // consume the newline character
 switch (choice) {
 case 1:
 System.out.print("Enter guest name: ");
 String guestName = scanner.nextLine();
 System.out.print("Enter room number: ");
 int roomNumber = scanner.nextInt();
 boolean booked = hotelService.bookRoom(guestName, roomNumber);
 if (booked) {
 System.out.println("Room booked successfully!");
 } else {
 System.out.println("Room booking failed.");
 }
 break;
 case 2:
 System.out.print("Enter guest name for cancellation: ");
 String cancelGuestName = scanner.nextLine();
 boolean canceled = hotelService.cancelBooking(cancelGuestName);
 if (canceled) {
 System.out.println("Booking canceled successfully!");
 } else {
 System.out.println("Booking cancellation failed.");
 }
 break;
 case 3:
 System.out.println("Exiting the client application.");
```

```
 System.exit(0);
 break;
 default:
 System.out.println("Invalid choice. Please enter a valid option.");
 }
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

HotelServer.java

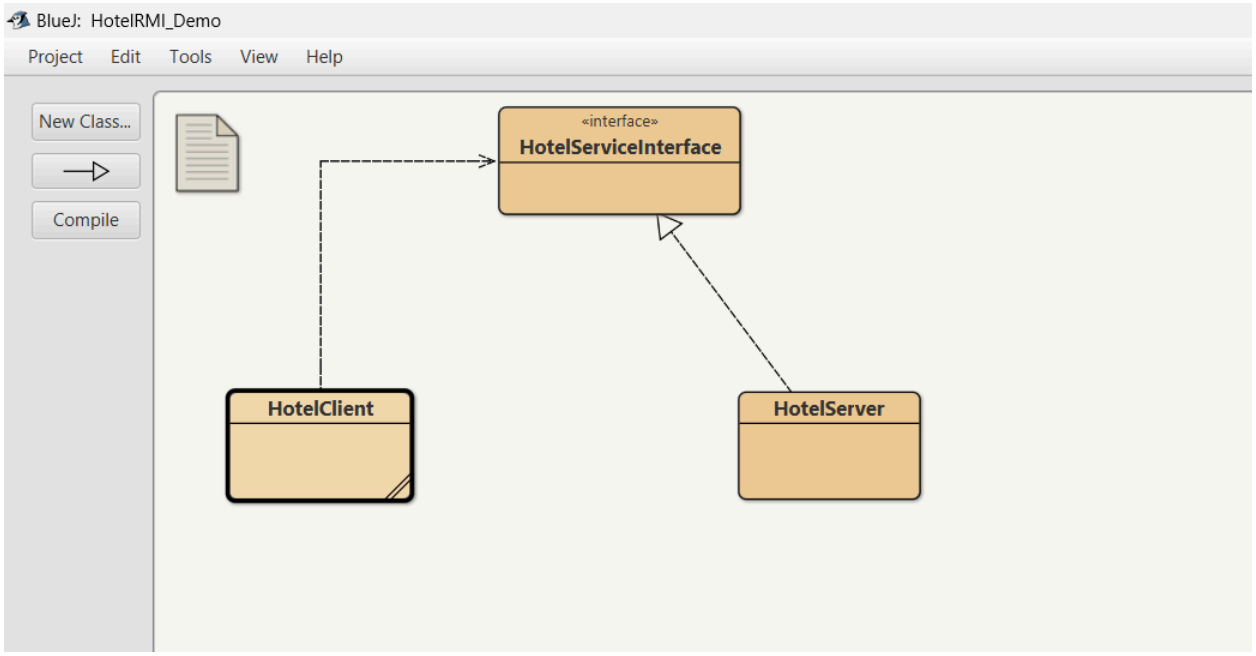
```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import java.util.Map;
public class HotelServer extends UnicastRemoteObject implements
HotelServiceInterface {
 private Map<Integer, String> bookedRooms;
 public HotelServer() throws RemoteException {
 bookedRooms = new HashMap<>();
 }
 @Override
 public synchronized boolean bookRoom(String guestName, int
roomNumber)
throws RemoteException {
 if (!bookedRooms.containsKey(roomNumber)) {
 bookedRooms.put(roomNumber, guestName);
 System.out.println("Room " + roomNumber + " booked for guest: " +
guestName);
 return true;
 } else {
 System.out.println("Room " + roomNumber + " is already booked.");
 return false;
 }
 }
 @Override
 public synchronized boolean cancelBooking(String guestName) throws
RemoteException {
 for (Map.Entry<Integer, String> entry : bookedRooms.entrySet()) {
 if (entry.getValue().equals(guestName)) {
 bookedRooms.remove(entry.getKey());
```

```
 System.out.println("Booking for guest " + guestName + " canceled.");
 return true;
 }
 }
 System.out.println("No booking found for guest " + guestName);
 return false;
 }
 public static void main(String[] args) {
 try {
 HotelServer server = new HotelServer();
 // Create and export the RMI registry on port 1099
 java.rmi.registry.LocateRegistry.createRegistry(1099);
 // Bind the server object to the registry
 Naming.rebind("HotelService", server);
 System.out.println("Hotel Server is running...");
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

HotelServiceInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HotelServiceInterface extends Remote {
 boolean bookRoom(String guestName, int roomNumber) throws
RemoteException;
 boolean cancelBooking(String guestName) throws RemoteException;
}
```

Output:



BlueJ: Terminal Window - HotelRMI_Demo

Options

```
Hotel Server is running...
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 1
Enter guest name: abc
Enter room number: 101
Room 101 booked for guest: abc
Room booked successfully!
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 2
Enter guest name for cancellation: abc
Booking for guest abc canceled.
Booking canceled successfully!
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 3
Exiting the client application.
```

ASSIGNMENT 9

PROBLEM STATEMENT: -
Design and develop a distributed application to find the coolest/hottest year from the available weather data. Use weather data from the Internet and process it using Map Reduce.

OBJECTIVE:
1.Students should be able to design the Map function to extract year and temperature data from each record.

2.Students should be able to Implement the Reduce function to aggregate temperature readings for each year and calculate the average temperature.

THEORY:

To design and develop a distributed application to find the coolest/hottest year from available weather data using Map Reduce, we need to break down the process into several steps. Let's outline the theory behind this process:

1. **Data Collection:**
   - Obtain weather data from reliable sources available on the internet. This data can be historical weather data collected over several years.
2. **Data Preprocessing:**
   - Clean the raw weather data to remove any inconsistencies or errors.
   - Organize the data into a suitable format for processing, such as CSV or JSON.
3. **Map Reduce Programming Model:**
   - Map Reduce is a programming model for processing and generating large datasets in a distributed environment.
   - In Map Reduce, computations are divided into two phases: the map phase and the reduce phase.
   - The map phase involves processing individual data elements and emitting intermediate key-value pairs.
   - The reduce phase aggregates and processes intermediate key-value pairs to produce the final output.
4. **Map Function:**
   - In our case, the map function will read each weather data record and extract the year as the key and the temperature as the value.
   - It will emit key-value pairs where the key is the year and the value is the temperature.
5. **Partitioning:**
   - The Map Reduce framework partitions the intermediate key-value pairs based on the keys.
   - Each partition is processed independently by a reduce task.
6. **Reduce Function:**
   - The reduce function receives all intermediate key-value pairs for a particular key (year) from different map tasks.
   - It computes the average temperature for each year.

- Finally, it identifies the year with the highest or lowest average temperature based on the requirement.

7. **Combining Multiple Jobs:**
   - In some cases, multiple Map Reduce jobs might be required to achieve the desired result.
   - For example, one job could compute the average temperature for each year, and another job could find the hottest or coolest year based on the output of the first job.
8. **Output:**
   - The output of the Map Reduce job will contain the year with the hottest or coolest temperature, along with the corresponding temperature value.
9. **Scalability and Fault Tolerance:**
   - Map Reduce is designed to scale horizontally, allowing the processing of large datasets across multiple nodes in a distributed cluster.
   - The framework provides fault tolerance by automatically restarting failed tasks on other nodes.
10. **Implementation:**
    - Implement the Map Reduce job using a suitable programming framework, such as Apache Hadoop or Apache Spark.
    - Configure the cluster environment to distribute the computation across multiple nodes.
    - Monitor the job execution and optimize performance as needed.

By following this approach, we can design and develop a distributed application to find the coolest/hottest year from available weather data using the Map Reduce programming model. This approach enables efficient processing of large datasets in a distributed environment, making it suitable for big data analytics tasks like weather data analysis.

Hadoop Installation

1. Java Installation

- sudo apt update
- Install JDK

https://download.oracle.com/otn/java/jdk/9.0.1+11/jdk-9.0.1_linux-x64_bin.tar.gz?AuthParam=1683886421_1603b66108615d82845d5ab9e22ec42e
OR
https://kenfavors.com/code/how-to-manually-install-oracle-java-9-on-ubuntu-16-04/

Step to Retrieve Java Path

- dirname $(dirname $(readlink -f $(which java)))

/usr/lib/jvm/java-11-openjdk-amd64

- **Change java**

$ update-alternatives --config java

2. SSH Installation:

- ssh-keygen -t rsa

- cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
- chmod 640 ~/.ssh/authorized_keys
- sudo apt-get install openssh-server
- ssh localhost

3. Hadoop Configuration:

- wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
- tar xzvf hadoop-3.3.4.tar.gz
- Rename hadoop3.3.4 to Hadoop
- gedit ~/.bashrc

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop path of hadoop folder
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

- source ~/.bashrc
- Switch to Hadoop Directory
  /hadoop/etc/hadoop
- Edit Core-site.xml: gedit core-site.xml

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

- Edit mapred-site xml :gedit mapred-site xml

```
<configuration>
<property>
  <name>mapreduce.job.tracker</name>
  <value>localhost:9870</value>
</property>
</configuration>
```

- Edit mapred-site xml :gedit hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

- Edit mapred-site xml :gedit Hdfs-site.xml

Hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
```

</configuration>

4. Switch to root

hdfs namenode -format

5. Cd hadoop

Cd hadoop/sbin

./start-all.sh

```
2023-05-24 10:17:09,336 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2023-05-24 10:17:09,336 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2023-05-24 10:17:09,339 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2023-05-24 10:17:09,339 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry ca
che entry expiry time is 600000 millis
2023-05-24 10:17:09,340 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2023-05-24 10:17:09,340 INFO util.GSet: VM type       = 64-bit
2023-05-24 10:17:09,340 INFO util.GSet: 0.029999999329447746% max memory 1.9 GB = 602.1 KB
2023-05-24 10:17:09,340 INFO util.GSet: capacity      = 2^16 = 65536 entries
2023-05-24 10:17:09,355 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1290038774-127.0.1.1-168490
3629350
2023-05-24 10:17:09,383 INFO common.Storage: Storage directory /tmp/hadoop-gurukul/dfs/name has been suc
cessfully formatted.
2023-05-24 10:17:09,403 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-gurukul/dfs/n
ame/current/fsimage.ckpt_0000000000000000000 using no compression
2023-05-24 10:17:09,475 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-gurukul/dfs/name/cur
rent/fsimage.ckpt_0000000000000000000 of size 402 bytes saved in 0 seconds .
2023-05-24 10:17:09,485 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-24 10:17:09,519 INFO namenode.FSNamesystem: Stopping services started for active state
2023-05-24 10:17:09,520 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-05-24 10:17:09,523 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-05-24 10:17:09,523 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at gurukul-ThinkCentre-M800/127.0.1.1
************************************************************/
gurukul@gurukul-ThinkCentre-M800:~$
```

jps

./stop-all.sh

Following Steps need to perform incase of any system Error:

Error localhost: rcmd: socket: Permission denied

https://tecadmin.net/how-to-install-apache-hadoop-on-ubuntu-22-04/

I also encountered the same thing, I did so I found that my pdsh default rcmd is rsh, not ssh, rsh and ssh remote login authentication is not the same, when installing hadoop I configured ssh localhost password-free login, but rsh is not possible.

so, try :

1.check your pdsh default rcmd rsh

pdsh -q -w localhost

See what your pdsh default rcmd is.

2.Modify pdsh's default rcmd to ssh

export PDSH_RCMD_TYPE=ssh

you can be added to ~/.bashrc, and source ~/.bashrc

3.sbin / start-dfs.sh

Steps to Check Hadoop is Properly Installed or Not

- Switch to hadoop bin
- hadoop fs -mkdir -p /user/gurukul/input
- http://localhost:9870

Steps to Compiler and Run the application

- Command to run Weathercode Mapreduce

  javac -d . WeatherDriver.java WeatherMapper.java WeatherReducer.java -cp "$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.3.4.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-3.3.4.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-3.3.4.jar:~/WeatherMapReduce/*:$HADOOP_HOME/lib/*"

  hadoop fs -put /home/gurukul/WeatherMapReduce/sample_weather.txt /user/gurukul

  hadoop jar /home/gurukul/WeatherMapReduce/weather.jar WeatherDriver input out

Mapper class

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MapperClass extends Mapper<LongWritable, Text, Text, LongWritable>{
        @Override
        protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
String w[] =value.toString().split(" ");
for (String word:w)
{
context.write(new Text(word), new LongWritable(1));
```

```
}
}
}
```

Reducer Class

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class ReducerClass extends Reducer<Text, LongWritable, Text, IntWritable> {
@Override
protected void reduce(Text key, Iterable<LongWritable> value,Context context)
throws IOException, InterruptedException {
int cnt=0;
for(LongWritable i:value)
{
cnt=cnt+1;
}
context.write(key, new IntWritable(cnt));
}
}
```

Driver Class

```
import org.apache.hadoop.conf.Configured;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class DriverClass extends Configured implements Tool {
@Override
public int run(String[] arg0) throws Exception {
Job job= new Job(getConf(),"KRN");
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setMapperClass(MapperClass.class);
job.setReducerClass(ReducerClass.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(LongWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path("input"));
```

```
FileOutputFormat.setOutputPath(job, new Path("out"));
job.setJarByClass(DriverClass.class);// to Run on hadoop
job.waitForCompletion(true);//Logs Display
return 0;
}
public static void main(String[] args) throws Exception {
ToolRunner.run(new DriverClass(), args);
}
}
```

CONCLUSION:

In this way we have by design a distributed application to find the coolest/hottest year from the available weather data.

ORAL QUESTION:

1. Can you briefly explain what MapReduce is and how it works?

2. What are the advantages of using MapReduce for processing large-scale data?

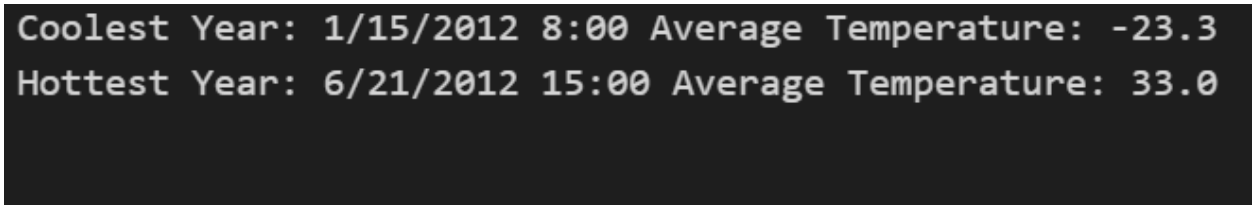
3. How does data shuffling occur in MapReduce, and why is it important?

Code:

```
import csv
from functools import reduce
from collections import defaultdict

# Define mapper function to emit (year, temperature) pairs
def mapper(row):
    year = row["Date/Time"].split("-")[0]  # Extract year from
"Date/Time" column
    temperature = float(row["Temp_C"])  # Convert temperature to
float
    return (year, temperature)

# Define reducer function to calculate sum and count of temperatures
for each year
def reducer(accumulated, current):
    accumulated[current[0]][0] += current[1]
    accumulated[current[0]][1] += 1
    return accumulated

# Read the weather dataset
weather_data = []
with open("weather_data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        weather_data.append(row)

# Map phase
mapped_data = map(mapper, weather_data)

# Reduce phase
reduced_data = reduce(reducer, mapped_data, defaultdict(lambda: [0,
0]))

# Calculate average temperature for each year
avg_temp_per_year = {year: total_temp / count for year, (total_temp,
count) in reduced_data.items()}

# Find coolest and hottest year
coolest_year = min(avg_temp_per_year.items(), key=lambda x: x[1])
hottest_year = max(avg_temp_per_year.items(), key=lambda x: x[1])

print("Coolest Year:", coolest_year[0], "Average Temperature:",
coolest_year[1])
```

```
print("Hottest Year:", hottest_year[0], "Average Temperature:",
hottest_year[1])
```

Output:



ASSIGNMENT 10

PROBLEM STATEMENT

Implement Ant colony optimization by solving the Traveling salesman problem using python Problem statement- A salesman needs to visit a set of cities exactly once and return to the original city. The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.

OBJECTIVE

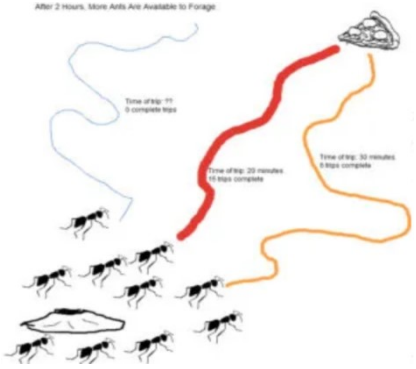
1. To learn and understand the Ant colony optimization algorithm.
2. Implement Ant colony optimization by solving the Traveling salesman problem.

THEORY

Ant colony optimization (ACO)

- Ant colony optimization (ACO) is a method inspired by the behavior of ants when they search for food.
- It is a metaheuristic optimization algorithm inspired by the behavior of ants. It is commonly used to solve combinatorial optimization problems like the Traveling Salesman Problem (TSP), the job shop scheduling problem, the vehicle routing problem, and the Knapsack problem.

- When ants look for food, they move randomly, leaving a pheromone trail behind them. Other ants follow these pheromone trails to find the food. The more ants that follow a trail, the stronger the pheromone trail becomes.



- In this example (figure), first, some Ant goes through the gray line, and it doesn't find the food. Since only some Ant's went through this way, the "pheromone" left by the ant will be minimal. These ants communicate with other ants not to follow this path.
- Second another set of some Ants goes through the orange line, and it finds food. These ants communicates with other ants that the food is found. So the other ant knows how to reach food following the "pheromone" train (orange path).
- Once the location of food is found, the other set of ants diverge to find the shortest path to the food from their ant colony. If the shortest path is found, then the shortest path is communicated to other ants. Now more ant's travel through the shortest path (red line) thus creating very strong "pheromone" trail. So finally ant's use the (red line) to go from their colony to food source.
- Similarly, in ACO, we have a group of artificial ants that are trying to find the best solution to a problem.
- For example, if we want to find the shortest path between two points, the artificial ants move randomly from one point to another, leaving a pheromone trail behind them.
- The pheromone trail indicates the quality of the path that the ant took. The ants prefer to follow the pheromone trail that is stronger, which corresponds to a better path.

Steps involved in Ant Colony Optimization:

1. Initialization: We start by placing the ants on the starting point.
2. Movement: Each ant selects a point to move to based on a probabilistic function that takes into account the pheromone level and the heuristic information. The heuristic information can be thought of as a measure of how good a particular point is.
3. Updating pheromone trails: The pheromone trail on each edge is updated based on the quality of the solution found by the ant that traversed that edge.
4. Termination: We stop the algorithm after a certain number of iterations or when a satisfactory solution is found.

By repeating these steps, the ants will gradually converge on the best solution to the problem. ACO can be used to solve a wide range of optimization problems, such as the traveling salesman problem, where the goal is to find the shortest path that visits a set of cities.

Pros & Cons of Ant colony optimization:

Advantages:

- Ant colony optimization is easy to implement and does not require complex mathematical knowledge.
- It is a very flexible algorithm and can be used in various problem domains.
- It can handle multiple objectives and constraints.
- It is a metaheuristic algorithm that can quickly find high-quality solutions in a large solution space.

- It has the ability to find near-optimal solutions, even in cases where the search space is very large or poorly understood.

Disadvantages:

- The algorithm may converge to a suboptimal solution if the parameter settings are not carefully selected.
- The algorithm may become computationally expensive if there are a large number of ants and/or iterations required to find a solution.
- The quality of the solution may be dependent on the pheromone initialization, which can be difficult to optimize.
- The algorithm may require a large number of iterations to converge to a solution, which may be a disadvantage if fast convergence is required.

Traveling Salesman Problem

- The Traveling Salesman Problem (TSP) is a famous mathematical problem in computer science and operations research.
- In this problem, a salesman needs to visit a set of cities exactly once and return to the original city.
- The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.
- The TSP is an optimization problem and can be applied to various real-world scenarios.
- For example, a delivery person may want to visit a set of cities to deliver packages and return to the starting point while minimizing the distance traveled. Another example is a circuit board drilling machine that needs to drill holes at various locations on the board.
- Finding the exact solution for TSP becomes computationally expensive as the number of cities increases.
- The number of possible paths between cities grows exponentially with the number of cities. Hence, it is necessary to use heuristic algorithms like ant colony optimization or genetic algorithms to find an approximate solution to the TSP.

CONCLUSION

We have implemented Ant colony optimization by solving the Traveling salesman problem using Python.

ORAL QUESTION

1. What is the Traveling Salesman Problem (TSP)?
2. How does Ant Colony Optimization (ACO) work, and what are its key components?
3. What role do pheromone trails play in ACO?
4. What data structures would you use in Python to represent the graph of cities and distances between them?

.

Code:

```
import numpy as np
import random

# Define the distance matrix (distances between cities)
# Replace this with your distance matrix or generate one based on
your problem
# Example distance matrix (replace this with your actual data)
distance_matrix = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])

# Parameters for Ant Colony Optimization
num_ants = 10
num_iterations = 50
evaporation_rate = 0.5
pheromone_constant = 1.0
heuristic_constant = 1.0

# Initialize pheromone matrix and visibility matrix
num_cities = len(distance_matrix)
pheromone = np.ones((num_cities, num_cities))  # Pheromone matrix
visibility = 1 / distance_matrix  # Visibility matrix (inverse of
distance)

# ACO algorithm
for iteration in range(num_iterations):
    ant_routes = []
    for ant in range(num_ants):
        current_city = random.randint(0, num_cities - 1)
        visited_cities = [current_city]
        route = [current_city]

        while len(visited_cities) < num_cities:
            probabilities = []
            for city in range(num_cities):
                if city not in visited_cities:
                    pheromone_value = pheromone[current_city][city]
                    visibility_value = visibility[current_city][city]
                    probability = (pheromone_value **
pheromone_constant) * (visibility_value ** heuristic_constant)
                    probabilities.append((city, probability))
```

```
            probabilities = sorted(probabilities, key=lambda x: x[1],
reverse=True)
            selected_city = probabilities[0][0]
            route.append(selected_city)
            visited_cities.append(selected_city)
            current_city = selected_city

        ant_routes.append(route)

    # Update pheromone levels
    delta_pheromone = np.zeros((num_cities, num_cities))

    for ant, route in enumerate(ant_routes):
        for i in range(len(route) - 1):
            city_a = route[i]
            city_b = route[i + 1]
            delta_pheromone[city_a][city_b] += 1 /
distance_matrix[city_a][city_b]
            delta_pheromone[city_b][city_a] += 1 /
distance_matrix[city_a][city_b]

    pheromone = (1 - evaporation_rate) * pheromone + delta_pheromone

# Find the best route
best_route_index =
np.argmax([sum(distance_matrix[cities[i]][cities[(i + 1) %
num_cities]] for i in range(num_cities)) for cities in ant_routes])
best_route = ant_routes[best_route_index]
shortest_distance = sum(distance_matrix[best_route[i]][best_route[(i
+ 1) % num_cities]] for i in range(num_cities))

print("Best route:", best_route)
print("Shortest distance:", shortest_distance)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\TSP.py"
d:\BE SEM VIII\CL_III_Code\TSP.py:24: RuntimeWarning: divide by zero encountered in divide
  visibility = 1 / distance_matrix  # Visibility matrix (inverse of distance)
Best route: [0, 1, 3, 2]
Shortest distance: 80
PS D:\BE SEM VIII>
```