

ASSIGNMENT No: 01

Title:-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

Objective:

1. Implement a distributed application using Remote Procedure Call (RPC) to allow a client to submit an integer value to the server.
2. Enable the server to calculate the factorial of the received integer and return the result to the client program.
3. Demonstrate the functionality and efficiency of RPC in remote computation tasks.

Outcome-

- Successfully implement an RPC-based client-server architecture for remote factorial

Unit 1

calculation.

- Demonstrate the ease of implementing distributed applications using RPC.

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirement

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of RPC

What is RPC?

Remote Procedure Call (RPC) is an interprocess communication technique. The Full form of RPC is Remote Procedure Call. It is used for client-server applications. RPC mechanisms are used when a computer program causes a procedure or subroutine to execute in a different address space, which is coded as a normal procedure call without the programmer specifically coding the details for the remote interaction.

This procedure call also manages low-level transport protocol, such as User Datagram Protocol, Transmission Control Protocol/Internet Protocol etc. It is used for carrying the message data between programs.

Types of RPC

Three types of RPC are:

- Callback RPC
- Broadcast RPC
- Batch-mode RPC

Callback RPC

This type of RPC enables a P2P paradigm between participating processes. It helps a process to be both client and server services.

Functions of Callback RPC:

- Remotely processed interactive application problems
- Offers server with clients handle
- Callback makes the client process wait
- Manage callback deadlocks
- It facilitates a peer-to-Peer paradigm among participating processes.

Broadcast RPC

Broadcast RPC is a client's request, that is broadcast on the network, processed by all servers which have the method for processing that request.

Functions of Broadcast RPC:

- Allows you to specify that the client's request message has to be broadcasted.
- You can declare broadcast ports.
- It helps to reduce the load on the physical network

Batch-mode RPC

Batch-mode RPC helps to queue, separate RPC requests, in a transmission buffer, on the client-side, and then send them on a network in one batch to the server.

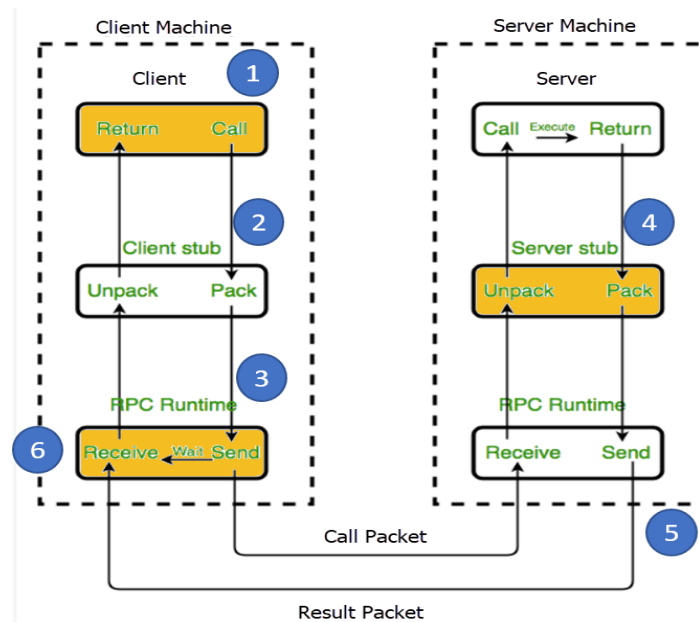
Functions of Batch-mode RPC:

- It minimizes overhead involved in sending a request as it sends them over the network in one batch to the server.
- This type of RPC protocol is only efficient for the application that needs lower call rates.
- It needs a reliable transmission protocol.

RPC Architecture

RPC architecture has mainly five components of the program:

1. **Client**
2. **Client Stub**
3. **RPC Runtime**
4. **Server Stub**
5. **Server**



How RPC Works?

Following steps take place during the RPC process:

Step 1) The client, the client stub, and one instance of RPC run time execute on the client machine.

Step 2) A client starts a client stub process by passing parameters in the usual way. The client stub stores within the client's own address space. It also asks the local RPC Runtime to send back to the server stub.

Step 3) In this stage, RPC accessed by the user by making regular Local Procedural Cal. RPC Runtime manages the transmission of messages between the network across client and server. It also performs the job of retransmission, acknowledgment, routing, and encryption.

Step 4) After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.

Step 5) In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.

Step 6) In this stage, the client stub demarshalls (unpack) the return parameters, in the resulting packet, and the execution process returns to the caller.

Characteristics of RPC

Here are the essential characteristics of RPC:

- The called procedure is in another process, which is likely to reside in another machine.
- The processes do not share address space.
- Parameters are passed only by values.
- RPC executes within the environment of the server process.
- It doesn't offer access to the calling procedure's environment.

Features of RPC

Here are the important features of RPC:

- Simple call syntax
- Offers known semantics
- Provide a well-defined interface
- It can communicate between processes on the same or different machines

Advantages of RPC

Here are Pros/benefits of RPC:

- RPC method helps clients to communicate with servers by the conventional use of procedure calls in high-level languages.
- RPC method is modeled on the local procedure call, but the called procedure is most likely to be executed in a different process and usually a different computer.
- RPC supports process and thread-oriented models.
- RPC makes the internal message passing mechanism hidden from the user.
- The effort needs to re-write and re-develop the code is minimum.
- Remote procedure calls can be used for the purpose of distributed and the local environment.
- It commits many of the protocol layers to improve performance.
- RPC provides abstraction. For example, the message-passing nature of network communication remains hidden from the user.

- RPC allows the usage of the applications in a distributed environment that is not only in the local environment.
- With RPC code, re-writing and re-developing effort is minimized.
- Process-oriented and thread-oriented models support by RPC.

Disadvantages of RPC

Here are the cons/drawbacks of using RPC:

- Remote Procedure Call Passes Parameters by values only and pointer values, which is not allowed.
- Remote procedure calling (and return) time (i.e., overheads) can be significantly lower than that for a local procedure.
- This mechanism is highly vulnerable to failure as it involves a communication system, another machine, and another process.
- RPC concept can be implemented in different ways, which is can't standard.
- Not offers any flexibility in RPC for hardware architecture as It is mostly interaction-based.
- The cost of the process is increased because of a remote procedure call.

RPC ArchitectureTo implement this in Python, we can use the **xmlrpc** library, which provides support for writing RPC servers and clients. The server program will create an XML-RPC server using **SimpleXMLRPCServer**, register a function to compute the factorial, and then start the server to listen for incoming requests. The client program will create an XML-RPC proxy object to communicate with the server, then call the remote procedure with the integer value as an argument to request the factorial calculation.

```
# Client-side code
import requests
```

```
def get_factorial(server_address, n):
    url = f"http://{server_address}/factorial"
    response = requests.post(url, json={"n": n})
```

```
factorial = response.json()["factorial"]
return factorial

if __name__ == "__main__":
    server_address = "localhost:50051" # Replace with your server's address and port
    n = int(input("Enter a non-negative integer: "))
    factorial = get_factorial(server_address, n)
    print(f"Factorial of {n} is {factorial}")

# Server-side code (using gRPC)
import grpc
import factorial_pb2
import factorial_pb2_grpc

class FactorialServicer(factorial_pb2_grpc.FactorialServiceServicer):
    def Calculate(self, request, context):
        n = request.n
        if n < 0:
            context.abort(grpc.status_code.INVALID_ARGUMENT, "n must be non-negative")
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        return factorial_pb2.FactorialResponse(factorial=factorial)

def serve():
    server = grpc.server(concurrent=4)
    factorial_pb2_grpc.add_FactorialServiceServicer_to_server(FactorialServicer(), server)
    server.add_insecure_port("[::]:50051") # Replace with your desired port
    server.start()
    server.wait_for_termination()
```

Conclusion:

Thus Implemented 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

Lab Assignment No.	02
Title	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
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III:Computational Intelligence
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 02

Title:- Distributed String Concatenation Application using RMI

Objective:

1. Develop a client-server application using Remote Method Invocation (RMI) for remote computation.
2. Enable the client to submit two strings to the server for concatenation.
3. Ensure that the server concatenates the given strings and returns the result to the client program.

Software Requirements -

- Jupyter Notebook, any Java IDE

Hardware Requirements -

- A machine with at least 8GB of RAM is recommended for model training.
- A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) is highly recommended.

Prerequisites -

- Basic understanding of Java programming

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

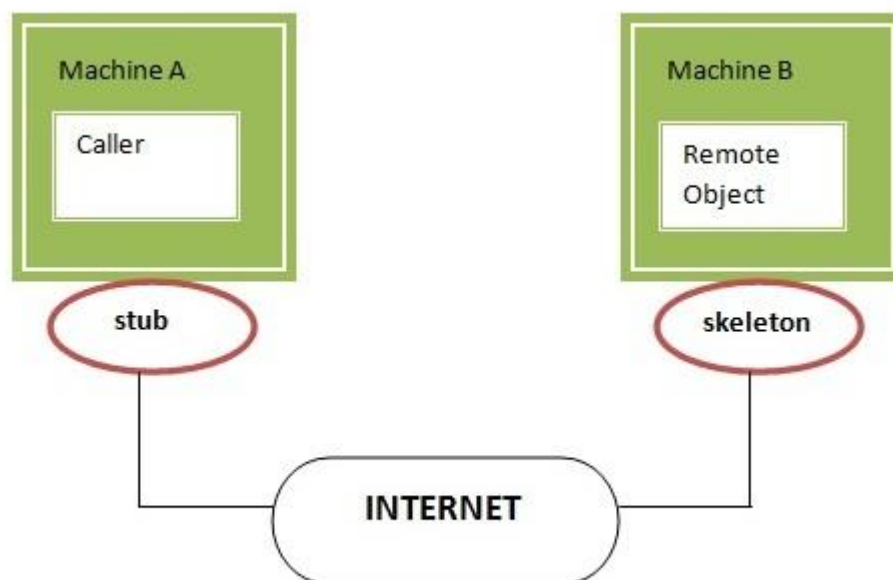
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),

3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller. In the Java 2 SDK, an stub protocol was introduced that eliminates the need for



skeletons.

Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Java RMI Example

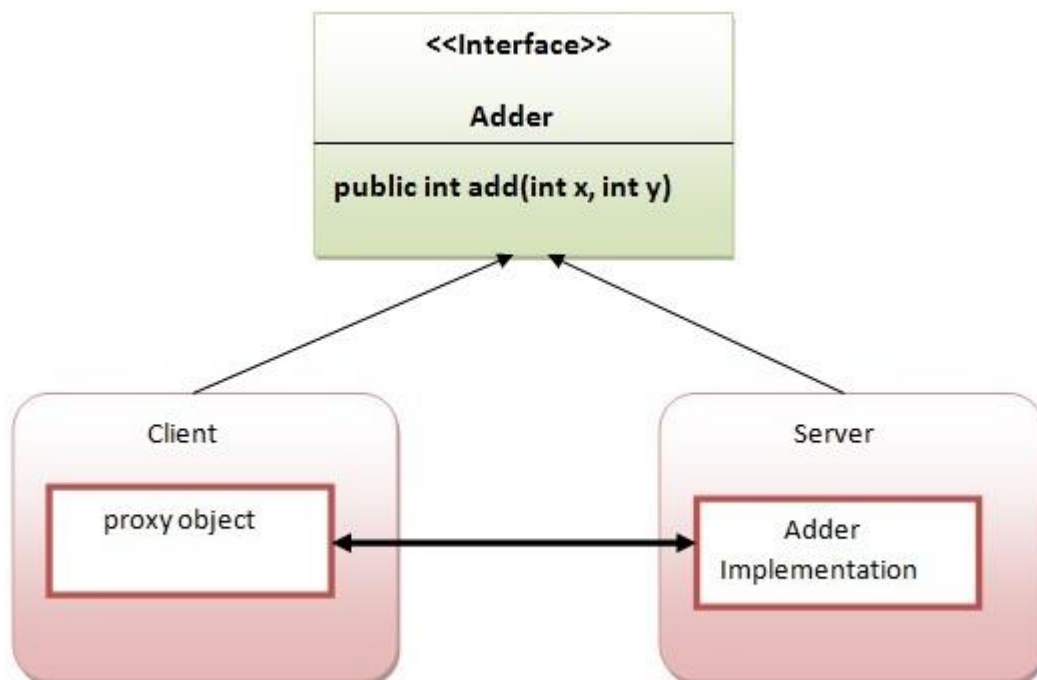
The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application

6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

1. **import** java.rmi.*;
2. **public interface** Adder **extends** Remote{
3. **public int** add(int x,int y)**throws** RemoteException;
4. }

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares

RemoteException.

```
1. import java.rmi.*;
2. import java.rmi.server.*;
3. public class AdderRemote extends UnicastRemoteObject implements Adder{
4.   AdderRemote()throws RemoteException{
5.     super();
6.   }
7.   public int add(int x,int y){return x+y;}
8. }
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
1. rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
1. rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object.

Conclusion: Thus This application demonstrates the use of RMI to create a distributed application for string concatenation, where the server receives two strings from the client, concatenates them, and returns the result to the client.

Lab Assignment No.	03
Title	Design a distributed application using MapReduce under Hadoop for: a) Character counting in a given text file. b) Counting no. of occurrences of every word in a given text file.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III-Computational Intelligence
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 03

Title:- Distributed Application using MapReduce for Text Analysis :

- a) Character counting in a given text file.
- b) Counting no. of occurrences of every word in a given text file.

Problem Statement:

Objective:

1. Implement a MapReduce program under Hadoop for character counting in a given text file.
2. Develop a MapReduce program under Hadoop for counting the number of occurrences of every word in a given text file.

Outcome-

1. Successfully count the number of characters in the given text file using MapReduce.
2. Count the occurrences of every word in the given text file using MapReduce.

Software Requirement

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirement

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites-

Counting the number of words in any language is a piece of cake like in C, C++, Python, Java, etc. MapReduce also uses Java but it is very easy if you know the syntax on how to write it.

It is the basic of MapReduce. You will first learn how to execute this code similar to “Hello World” program in other languages. So here are the steps which show how to write a MapReduce code for Word Count.

What is MapReduce?

MapReduce is a programming model and framework within the Hadoop ecosystem that enables efficient

processing of big data by automatically distributing and parallelizing the computation. It consists of two fundamental tasks: Map and Reduce.

In the Map phase, the input data is divided into smaller chunks and processed independently in parallel across multiple nodes in a distributed computing environment. Each chunk is transformed or “mapped” into key-value pairs by applying a user-defined function. The output of the Map phase is a set of intermediate key-value pairs. The Reduce phase follows the Map phase. It gathers the intermediate key-value pairs generated by the Map tasks, performs data shuffling to group together pairs with the same key, and then applies a user-defined reduction function to aggregate and process the data. The output of the Reduce phase is the final result of the computation.

Map Reduce example allows for efficient processing of large-scale datasets by leveraging parallelism and distributing the workload across a cluster of machines. It simplifies the development of distributed data processing applications by abstracting away the complexities of parallelization, data distribution, and fault tolerance, making it an essential tool for big data processing in the Hadoop ecosystem.

Advantages of using MapReduce

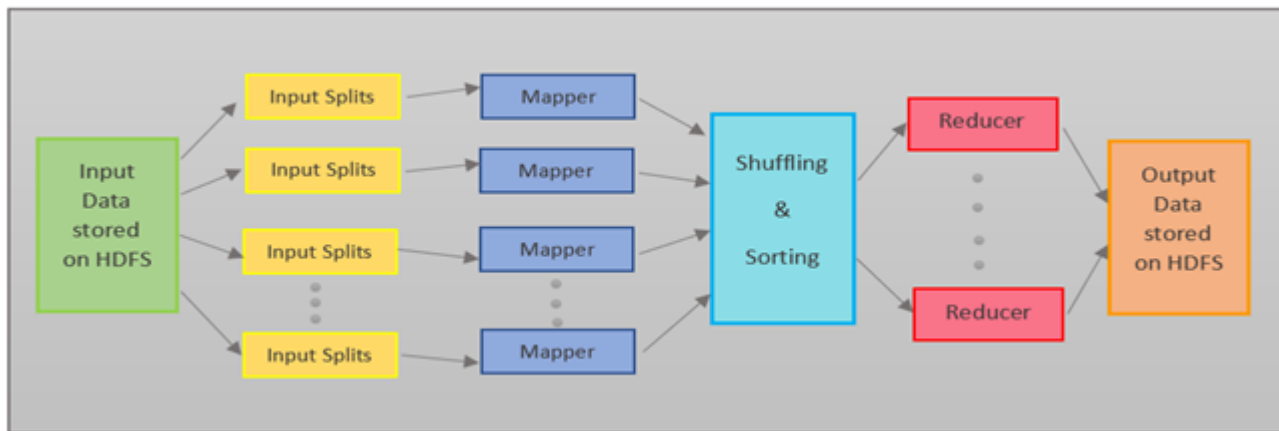
The advantages of using MapReduce are as follows:

- MapReduce can define mapper and reducer in several different languages using Hadoop streaming.
- MapReduce facilitates automatic parallelization and distribution, reducing the time required to run programs.
- MapReduce provides fault tolerance by re-executing, writing map output to a distributed file system, and restarting failed map or reducer tasks.
- Processing of data using MapReduce is a cost-effective solution.
- MapReduce processes large volumes of unstructured data very quickly.
- Using HDFS and HBase security, Map Reduce ensures data security by allowing only approved users to access data stored in the system.
- MapReduce programming utilizes a simple programming model to handle tasks more efficiently and quickly and is easy to learn.
- MapReduce is flexible and works with several Hadoop languages to handle and store data.

MapReduce Architecture

Map Reduce example process has the following phases:

1. Input Splits
2. Mapping
3. Shuffling
4. Sorting
5. Reducing



Input Splits

MapReduce splits the input into smaller chunks called input splits, representing a block of work with a single mapper task.

Mapping

The input data is processed and divided into smaller segments in the mapper phase, where the number of mappers is equal to the number of input splits. RecordReader produces a key-value pair of the input splits using TextFormat, which Reducer later uses as input. The mapper then processes these key-value pairs using coding logic to produce an output of the same form.

Shuffling

In the shuffling phase, the output of the mapper phase is passed to the reducer phase by removing duplicate values and grouping the values. The output remains in the form of keys and values in the mapper phase. Since shuffling can begin even before the mapper phase is complete, it saves time.

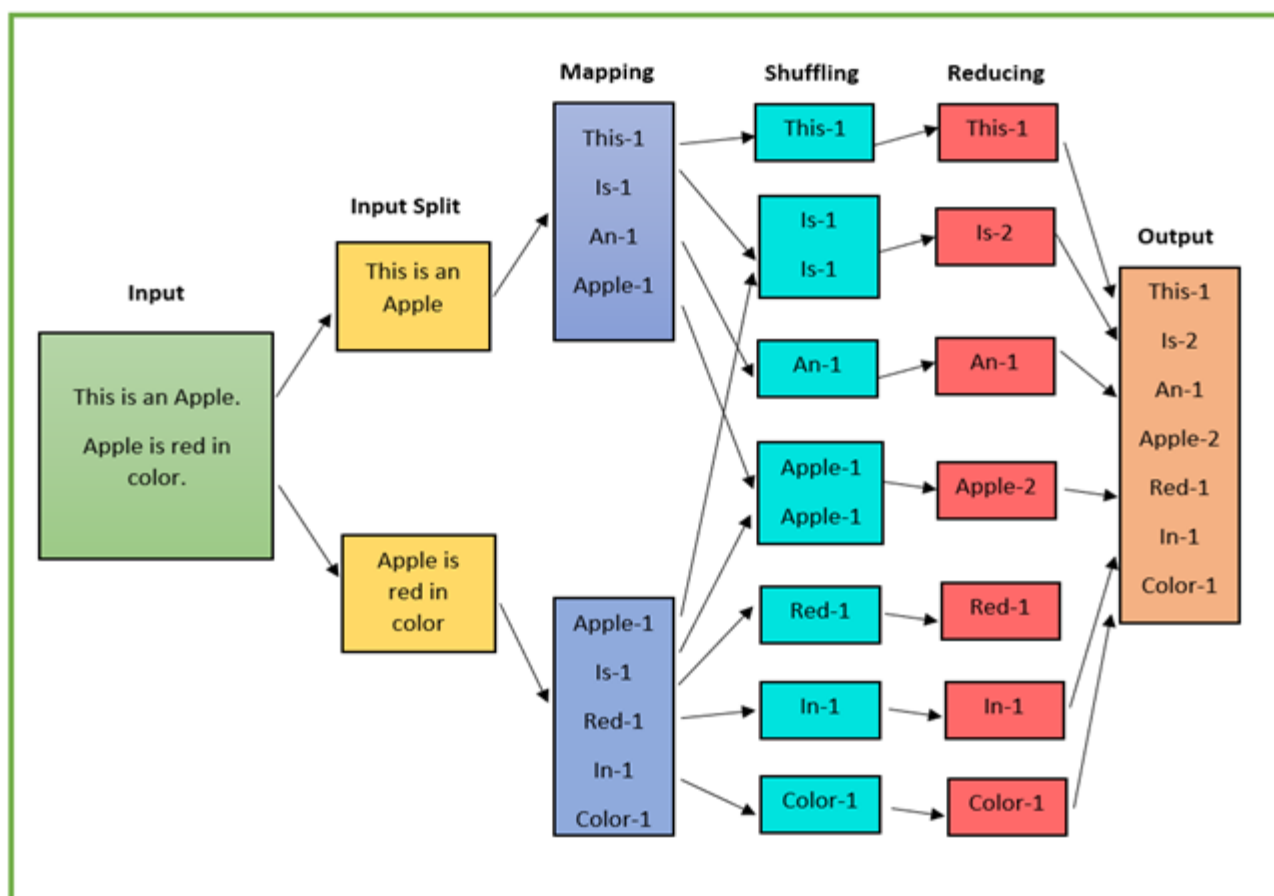
Sorting

Sorting is performed simultaneously with shuffling. The Sorting phase involves merging and sorting the output generated by the mapper. The intermediate key-value pairs are sorted by key before starting the reducer phase, and the values can take any order. Sorting by value is done by secondary sorting.

Reducing

In the reducer phase, the intermediate values from the shuffling phase are reduced to produce a single output value that summarizes the entire dataset. HDFS is then used to store the final output.

Here's an Map Reduce example to count the frequency of each word in an input text. The text is, "This is an apple. Apple is red in color."



- The input data is divided into multiple segments, then processed in parallel to reduce processing time. In this case, the input data will be divided into two input splits so that work can be distributed over all the map nodes.
- The Mapper counts the number of times each word occurs from input splits in the form of key-value pairs where the key is the word, and the value is the frequency.
- For the first input split, it generates 4 key-value pairs: This, 1; is, 1; an, 1; apple, 1; and for the second, it generates 5 key-value pairs: Apple, 1; is, 1; red, 1; in, 1; color.
- It is followed by the shuffle phase, in which the values are grouped by keys in the form of key-value pairs. Here we get a total of 6 groups of key-value pairs.
- The same reducer is used for all key-value pairs with the same key.
- All the words present in the data are combined into a single output in the reducer phase. The output shows the frequency of each word.
- Here in the example, we get the final output of key-value pairs as This, 1; is, 2; an, 1; apple, 2; red, 1; in, 1; color, 1.
- The record writer writes the output key-value pairs from the reducer into the output files, and the final output data is by default stored on HDFS.

Limitations of MapReduce

Map Reduce example also faces some limitations, and they are as follows:

- MapReduce is a low-level programming model which involves a lot of writing code.
- The batch-based processing nature of MapReduce makes it unsuitable for real-time processing.
- It does not support data pipelining or overlapping of Map and Reduce phases.
- Task initialization, coordination, monitoring, and scheduling take up a large chunk of MapReduce's execution time and reduce its performance.
- MapReduce cannot cache the intermediate data in memory, thereby diminishing Hadoop's performance.

Conclusion: Thus designed Distributed Application using MapReduce for Text Analysis.

<https://www.geeksforgeeks.org/how-to-execute-character-count-program-in-mapreduce-hadoop/>

Lab Assignment No.	04
Title	Design and implement a CNN for Image Classification a) Select a suitable image classification dataset (medical imaging, agricultural, etc.). b) Optimized with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-IV: Deep Learning
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 04

Title:-

Design and implement 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.

Problem Statement: Design and implement Union, Intersection, Complement, and Difference operations on fuzzy sets. Create fuzzy relations by the Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations.

Objective:

1. To implement Union, Intersection, Complement, and Difference operations on fuzzy sets.
2. To create fuzzy relations using the Cartesian product of fuzzy sets and perform max-min composition on fuzzy relations.
3. To demonstrate the application of these operations in fuzzy logic.

Outcome-

1. Successfully implement Union, Intersection, Complement, and Difference operations on fuzzy sets.
2. Create fuzzy relations using the Cartesian product of fuzzy sets and perform max-min composition on fuzzy relations.

Software Requirement

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirement

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites-

What is Fuzzy Set ?

Fuzzy refers to something that is unclear or vague . Hence, Fuzzy Set is a Set where every key is associated with value, which is between 0 to 1 based on the certainty .This value is often called as degree of membership. Fuzzy Set is denoted with a Tilde Sign on top of the normal Set notation.

Operations on Fuzzy Set with Code :

1. Union :

Operations on Fuzzy Set with Code :

1. Union : Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Union of them, then for every member of A and B, Y will be:

$$\text{degree_of_membership}(Y) = \max(\text{degree_of_membership}(A), \text{degree_of_membership}(B))$$

EXAMPLE :

Example to Demonstrate the

Union of Two Fuzzy Sets

A = dict()

B = dict()

Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print("The First Fuzzy Set is :", A)

print("The Second Fuzzy Set is :", B)

for A_key, B_key in zip(A, B):

 A_value = A[A_key]

 B_value = B[B_key]

 if A_value > B_value:

 Y[A_key] = A_value

 else:

 Y[B_key] = B_value

print("Fuzzy Set Union is :", Y)

Output

The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}

Fuzzy Set Union is : {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}

2. Intersection :

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Intersection of them, then for every member of A and B, Y will be:

$\text{degree_of_membership}(Y) = \min(\text{degree_of_membership}(A), \text{degree_of_membership}(B))$

EXAMPLE :

Example to Demonstrate

Intersection of Two Fuzzy Sets

A = dict()

B = dict()

Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):

 A_value = A[A_key]

 B_value = B[B_key]

 if A_value < B_value:

 Y[A_key] = A_value

 else:

 Y[B_key] = B_value

```
print('Fuzzy Set Intersection is :', Y)
```

Output

The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}

Fuzzy Set Intersection is : {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}

3. Complement :

Consider a Fuzzy Sets denoted by A , then let's consider Y be the Complement of it, then for every member of A , Y will be:

$$\text{degree_of_membership}(Y) = 1 - \text{degree_of_membership}(A)$$
EXAMPLE :

```
# Example to Demonstrate the
```

```
# Difference Between Two Fuzzy Sets
```

```
A = dict()
```

```
Y = dict()
```

```
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
```

```
print('The Fuzzy Set is :', A)
```

```
for A_key in A:
```

```
Y[A_key]= 1-A[A_key]
```

```
print('Fuzzy Set Complement is :', Y)
```

Output

The Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

Fuzzy Set Complement is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}

4.Difference

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Intersection of them, then for every member of A and B, Y will be:

$\text{degree_of_membership}(Y) = \min(\text{degree_of_membership}(A), 1 - \text{degree_of_membership}(B))$

EXAMPLE :

Example to Demonstrate the

Difference Between Two Fuzzy Sets

A = dict()

B = dict()

Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):

 A_value = A[A_key]

 B_value = B[B_key]

 B_value = 1 - B_value

 if A_value < B_value:

 Y[A_key] = A_value

 else:

 Y[B_key] = B_value

print('Fuzzy Set Difference is :', Y)

Conclusion: Fuzzy logic provides a powerful framework for dealing with uncertainty and imprecision in data. By implementing operations on fuzzy sets and relations, we can model complex relationships and make more flexible decisions in various applications such as control systems, artificial intelligence, and pattern recognition. The Union, Intersection, Complement, and Difference operations allow us to manipulate fuzzy sets, while fuzzy relations and max-min composition enable us to represent and combine fuzzy relationships between sets. These operations and techniques expand the capabilities of traditional logic and set theory, offering a more nuanced approach to handling vague and ambiguous information.

Lab Assignment No.	05
Title	Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III: Computational Intelligence
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 05

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

Problem Statement: Design and implement the spray drying process of coconut milk involves complex interactions of multiple parameters that affect the quality of the dried product.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

1. To optimize the parameters of the genetic algorithm in a hybrid genetic algorithm-neural network model for the spray drying process of coconut milk.
2. To improve the accuracy and efficiency of the modeling process by tuning the genetic algorithm parameters.
3. To demonstrate the effectiveness of the optimized hybrid model in predicting the quality of the dried coconut milk.

Outcomes:

1. Identification of the optimal parameters for the genetic algorithm in the hybrid model.
2. Improved prediction accuracy and efficiency of the hybrid model compared to traditional modeling approaches.

Theory: Modeling complex processes like spray drying coconut milk presents a multi-faceted challenge. Nonlinear relationships and numerous influential factors demand sophisticated methods beyond traditional linear models. While Artificial Neural Networks (ANNs) excel in capturing such nonlinearities, they can be prone to initialization issues and require substantial data for optimal performance. Genetic Algorithms (GAs), on the other hand, adeptly navigate global optimization landscapes but can be computationally expensive. This research explores the intriguing avenue of optimizing GA parameters in a hybrid GA-NN approach to model spray drying, aiming to synergistically leverage the strengths of both techniques while mitigating their limitations.

Our endeavor is driven by three key objectives:

1. **Fine-tuning GA Parameters:** We seek to identify a set of GA parameters that efficiently navigate the solution space, ultimately leading to optimized network weights for modeling spray drying. Population size, crossover rate, and mutation rate are prominent parameters we intend to carefully calibrate.
2. **Elevating Model Accuracy:** Our ambition is to develop a model that surpasses the accuracy of both standalone ANNs and GA-NN models using unoptimized parameters. This enhanced model should faithfully represent the intricate dynamics of the spray drying process, offering superior predictive capabilities.
3. **Striking a Balance:** Achieving high accuracy is commendable, but it shouldn't come at the expense of exorbitant computational costs. We delve into the interplay between accuracy and computation time, striving to find GA parameter settings that strike a favorable balance between thorough exploration and focused exploitation, minimizing execution time without compromising model efficacy.

Through this endeavor, we aspire to achieve two significant outcomes:

1. **A Champion Model:** The culmination of our efforts will be a meticulously optimized GA-NN model capable of precisely predicting spray drying performance based on diverse input parameters. This model should stand out from its peers, demonstrating superior accuracy and robustness.
2. **Performance Benchmarking:** We will meticulously quantify the model's improved accuracy and the reduction in computational cost compared to alternative approaches. This comparative analysis will illuminate the effectiveness of our optimization strategy, providing valuable insights and paving the way for further advancements.

The optimization process typically involves the following steps:

Parameter Initialization: Initialize the genetic algorithm parameters, including population size, crossover rate, mutation rate, and selection strategy.

Fitness Evaluation: Evaluate the fitness of each individual in the population using the NN model. The fitness function is usually based on the error between predicted and actual drying parameters.

Selection: Select individuals from the population based on their fitness scores to form the next generation.

Crossover and Mutation: Apply crossover and mutation operators to create offspring for the next generation.

Replacement: Replace the current population with the new generation of individuals.

Termination: Repeat the process until a termination condition is met, such as reaching a maximum number of generations or achieving a desired level of convergence.

By optimizing the genetic algorithm parameters in the hybrid GA-NN model, we expect to improve the accuracy of predicting the quality of dried coconut milk powder. This optimized model can provide valuable insights for optimizing the spray drying process and enhancing the quality of the final product.

Here's an example of how you can optimize the genetic algorithm parameters in a hybrid GA-NN model for predicting the quality of dried coconut milk powder using Python. This example uses the **genetic_algorithm** library for the genetic algorithm part and **scikit-learn** for the neural network part. Please note that this is a simplified example for demonstration purposes and may require further customization for your specific application.

```
import numpy as np
from genetic_algorithm import GeneticAlgorithm
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Sample data (features and target)
X = np.random.rand(100, 5) # Example features
y = np.random.rand(100)    # Example target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the neural network model
nn_model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', random_state=42)

# Define the fitness function for the genetic algorithm
def fitness_function(params):
    # Unpack genetic algorithm parameters
    population_size, crossover_rate, mutation_rate = params

    # Create the genetic algorithm object
    ga = GeneticAlgorithm(population_size=population_size, crossover_rate=crossover_rate,
                           mutation_rate=mutation_rate)

    # Train the neural network model
    nn_model.fit(X_train, y_train)

    # Evaluate the model
    y_pred = nn_model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    return mse

# Define the parameter ranges for the genetic algorithm
parameter_ranges = {
```

```
'population_size': (50, 100),  
'crossover_rate': (0.6, 0.9),  
'mutation_rate': (0.01, 0.1)  
}  
  
# Create the genetic algorithm object  
ga = GeneticAlgorithm(fitness_function, parameter_ranges)  
  
# Optimize the genetic algorithm parameters  
best_params = ga.optimize()  
  
print("Best Parameters:", best_params)
```

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

Lab Assignment No.	06
Title	Implementation of Clonal selection algorithm using Python.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III: Computational Intelligence
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 06

Problem Statement: Implementation of Clonal selection algorithm using Python

Objective:

1. To implement the clonal selection algorithm in Python for solving optimization problems.
2. To apply the algorithm to a specific optimization problem and compare its performance with other optimization algorithms.
3. To analyze the effectiveness and efficiency of the clonal selection algorithm in finding optimal solutions.

Title:- The Clonal Selection Algorithm (CSA) is a computational optimization technique inspired by the clonal selection process in the immune system.

Outcomes:

1. Implementation of the clonal selection algorithm in Python, including the main algorithmic steps such as cloning, mutation, and selection.
2. Evaluation of the algorithm's performance in terms of convergence speed and solution quality compared to other optimization algorithms.

Theory

The Clonal Selection Algorithm (CSA) is a bio-inspired optimization algorithm that is based on the clonal selection process of B cells in the immune system. The algorithm was first proposed by de Castro and Timmis in 2002 as a way to solve complex optimization problems. The CSA mimics the behavior of the immune system by using an iterative process of selection, cloning, and mutation to evolve a population of candidate solutions towards an optimal solution.

Key Concepts:

1. **Antibody Representation:** In the CSA, candidate solutions, called antibodies, are represented as vectors in a multidimensional search space. Each dimension of the vector corresponds to a possible solution to the optimization problem.
2. **Affinity Maturation:** Affinity maturation is the process by which antibodies with higher affinities to the target antigen are selected and cloned. In the context of the CSA, affinity represents the fitness of an antibody, i.e., how well it solves the optimization problem.
3. **Cloning:** Cloning is the process of creating multiple copies of high-affinity antibodies. In the CSA, antibodies are cloned based on their affinity, with higher-affinity antibodies being cloned more frequently.
4. **Mutation:** Mutation introduces diversity into the population by randomly perturbing the cloned antibodies. This helps to explore new regions of the search space and avoid local optima.
5. **Selection:** Selection is the process of choosing antibodies to form the next generation based on their affinity. Higher-affinity antibodies are more likely to be selected, while lower-affinity antibodies may be discarded.

Algorithm Steps:

1. **Initialization:** Initialize a population of antibodies randomly or using a heuristic method.
2. **Affinity Calculation:** Calculate the affinity of each antibody in the population based on its fitness in solving the optimization problem.
3. **Clonal Selection:** Select antibodies for cloning based on their affinity, with higher-affinity antibodies being cloned more.
4. **Cloning:** Create multiple copies of selected antibodies based on their affinity.
5. **Mutation:** Perturb the cloned antibodies to introduce diversity into the population.
6. **Selection:** Select antibodies for the next generation based on their affinity.
7. **Repeat:** Repeat steps 3-6 for a specified number of iterations or until a convergence criteria is met.

Termination: The algorithm terminates when a stopping criterion is met, such as a maximum number of iterations or the convergence of the population towards an optimal solution.

Python Implementation:

Below is a basic implementation of the Clonal Selection Algorithm in Python:

Pseudocode for Clonal Selection Algorithm

```
def clonal_selection_algorithm(population_size, mutation_rate, num_iterations):
```

```
    # Initialize population of antibodies
```

```
    antibodies = initialize_population(population_size)
```

```
    for _ in range(num_iterations):
```

```
        # Calculate affinity of antibodies
```

```
        calculate_affinity(antibodies)
```

```
        # Select antibodies for cloning
```

```
        selected_antibodies = select_antibodies_for_cloning(antibodies)
```

```
        # Clone selected antibodies
```

```
        cloned_antibodies = clone_antibodies(selected_antibodies)
```

```
        # Mutate cloned antibodies
```

```
        mutated_antibodies = mutate_antibodies(cloned_antibodies, mutation_rate)
```

```
        # Select antibodies for next generation
```

```
        antibodies = select_next_generation(antibodies, mutated_antibodies)
```

```
    # Return best antibody
```

```
    return best_antibody(antibodies)
```

This implementation is a basic framework for the Clonal Selection Algorithm and can be customized for specific optimization problems by defining appropriate functions for initialization, affinity calculation, selection, cloning, mutation, and selection of the next generation.

Conclusion: Thus designed and implemented of Clonal selection algorithm using Python

Lab Assignment No.	07
Title	To apply the artificial immune pattern recognition to perform a task of structure damage Classification .
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III-Distributed System
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 07

Title: To apply the artificial immune pattern recognition to perform a task of structure damage Classification .

Problem Statement: Develop an artificial immune pattern recognition system for the task of structural damage classification.

Prerequisite:

Basics of Python

Software Requirements: Jupyter Notebook or any Python IDE

Required Python libraries (scikit-learn, numpy, matplotlib)

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

1. Design a robust algorithm capable of identifying patterns indicative of structural damage within complex datasets.
2. Implement a scalable solution that can accurately classify various types and extents of structural damage with high efficiency.

Outcomes:

After completion of this assignment students are able to understand how to The developed system will enable automated and accurate identification of structural damage, aiding in timely maintenance and ensuring safety.

Theory:

Introduction to AIS:

- Briefly discuss the biological immune system and its key functions.
- Introduce the concept of AIS and its principles for pattern recognition.
- Explain the analogy between antigens (foreign invaders) and damage patterns, and antibodies (immune cells) and damage detectors.

2. Setting Up the AIS Model:

- Divide the students into groups and assign each group a specific damage type (e.g., crack, corrosion).
- Provide each group with a training dataset containing features of healthy and damaged structures related to their assigned damage type.
- Guide students through the process of encoding the data into a format suitable for the AIS algorithm (e.g., binary strings representing feature values).
- Introduce the chosen AIS algorithm and its key parameters (e.g., population size, mutation rate).
- Assist students in implementing the algorithm using the provided library or framework.

3. Training and Testing the Model:

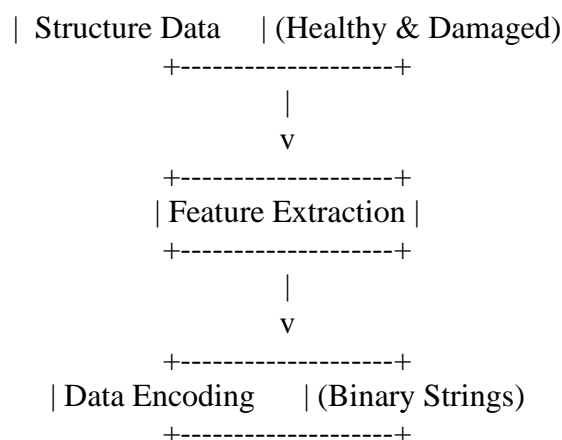
- Run the AIS algorithm on the training data for each group.
- Explain the selection, mutation, and cloning processes within the algorithm.
- Observe how the "antibodies" (damage detectors) evolve over generations to better recognize the assigned damage pattern.
- Once training is complete, test the model with unseen data containing both healthy and damaged structures.
- Evaluate the performance of the AIS model based on its accuracy in detecting the assigned damage type.

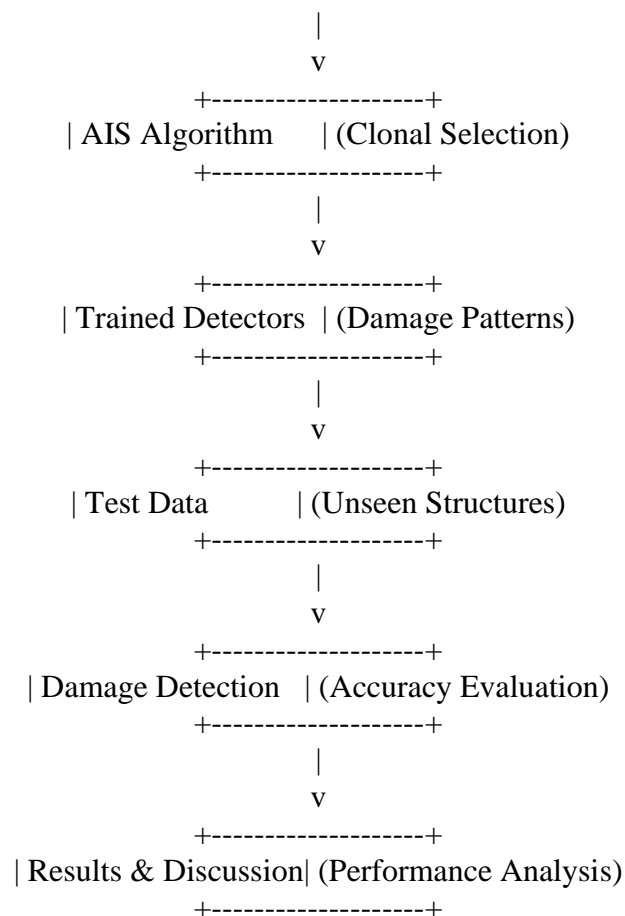
4. Analysis and Discussion:

- Each group presents their results and discusses the performance of their AIS model.
- Compare the effectiveness of different damage detection algorithms or parameter settings across groups.
- Discuss the advantages and limitations of using AIS for damage detection.
- Explore potential applications of AIS in real-world engineering scenarios.

5. Diagram:

A simplified diagram illustrating the key steps of the lab experiment:





Note: This is a general outline and can be adapted based on the specific AIS algorithm, chosen damage types, and available resources.

This code represents a process flow for a damage detection system using an AIS algorithm.

1. **Structure Data:** Represents the input data for the system, which includes information about healthy and damaged structures.
2. **Feature Extraction:** Extracts relevant features from the input data to be used in the detection process.
3. **Data Encoding:** Converts the extracted features into binary strings for processing by the AIS algorithm.
4. **AIS Algorithm:** Utilizes a Clonal Selection algorithm to detect damage patterns in the encoded data.
5. **Trained Detectors:** Represents the detectors that have been trained to recognize specific damage patterns.
6. **Test Data:** Contains unseen structures that will be used to test the performance of the detection system.

7. Damage Detection: Evaluates the accuracy of the detection system in identifying damage in the test data.
8. Results & Discussion: Analyzes the performance of the system and discusses the results obtained from the damage detection process.

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}")
```

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

Accuracy: 0.05

Conclusion: - This way, Implemented the artificial immune pattern recognition to perform a task of structure damage Classification .

Lab Assignment No.	8
Title	Implement DEAP (Distributed Evolutionary Algorithms) using Python
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III-Distributed System
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 08

Title: Implement DEAP (Distributed Evolutionary Algorithms) using Python

Problem Statement: Develop a distributed evolutionary algorithm using DEAP (Distributed Evolutionary Algorithms in Python) to optimize a complex problem that requires intensive computational resources.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

1. Implement a scalable and efficient distributed evolutionary algorithm using DEAP to solve the optimization problem.
2. Improve the optimization process by leveraging parallel processing capabilities and distributing the workload across multiple nodes.

Outcomes:

After completion of this assignment students are able to understand how to developed distributed evolutionary algorithm will significantly reduce the optimization time and enable the solution of larger and more complex optimization problems.

Theory:

DEAP (Distributed Evolutionary Algorithms in Python) is a framework for building and analyzing distributed evolutionary algorithms. It provides tools for implementing genetic algorithms and other evolutionary computation techniques in a flexible and easy-to-use manner. DEAP allows users to parallelize their evolutionary algorithms across multiple processors or computers, making it suitable for tackling complex optimization problems that require significant computational resources.

DEAP provides a wide range of tools and functionalities to facilitate the implementation and experimentation of evolutionary algorithms. Some key features of DEAP include:

1. **Genetic Operators:** DEAP provides a set of standard genetic operators such as crossover, mutation, and selection, which can be easily customized and combined to suit the problem being solved.

2. **Fitness Evaluation:** DEAP allows users to define custom fitness functions to evaluate the quality of candidate solutions.
3. **Population Management:** DEAP provides tools for managing populations of candidate solutions, including initialization methods and selection strategies.
4. **Statistics and Logging:** DEAP includes utilities for tracking and logging the evolution of candidate solutions over time, including statistics such as average fitness and best fitness.
5. **Parallelization:** DEAP supports parallelization of evolutionary algorithms, allowing users to take advantage of multi-core processors and distributed computing environments to speed up optimization tasks.
6. **Integration:** DEAP can be easily integrated with other Python libraries and frameworks, making it a versatile tool for a wide range of optimization problems.

Overall, DEAP provides a convenient and efficient way to implement evolutionary algorithms in Python, making it a popular choice for researchers and practitioners working in the field of evolutionary computation.

Use and Applications:

- DEAP is used to solve optimization problems where traditional methods may be impractical or inefficient.
- It is commonly used in various fields such as engineering, biology, finance, and data science for optimization tasks.
- Applications include parameter optimization, feature selection, function optimization, and more.

Implementation Using Python:

To implement DEAP in Python, first, install the DEAP package using pip:

```
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)
Best individual: [-0.00025929139689136204, -0.007494927037045774, -0.00031377093359364716]
Best fitness: 5.6339615517909845e-05

***USE JUPYTER NOTEBOOK***
```

Conclusion:- This way DEAP (Distributed Evolutionary Algorithms) using Python is done.

Lab Assignment No.	09
Title	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.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-III-Distributed System
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 09

Title: 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.

Problem Statement: Design and develop a distributed Hotel booking application using Java RMI, consisting of a server managing hotel room booking information and clients able to book or cancel rooms for guests.

Prerequisite:

Basics of Java

Software Requirements: BlueJ IDE

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

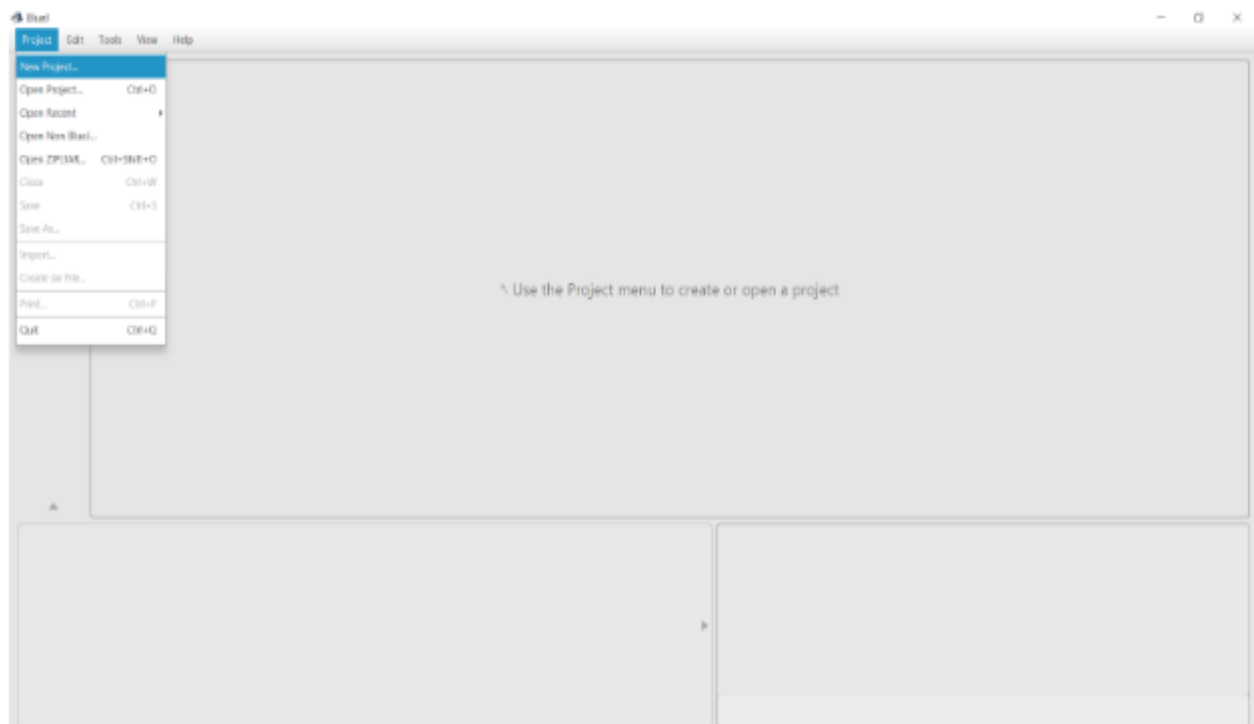
1. Learn to Perform a robust and scalable distributed system architecture using Java RMI to manage hotel room bookings efficiently.
2. Provide clients with the ability to book rooms for specific guests and cancel bookings as needed, ensuring a user-friendly experience.

Outcomes:

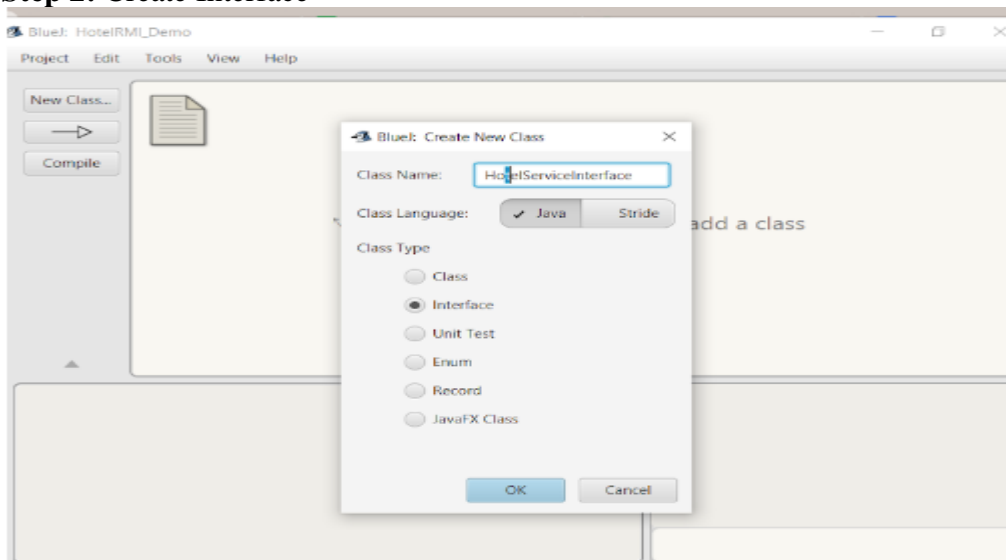
After completion of this assignment students are able to understand how to develop application will enable clients to easily book and manage hotel room bookings, while the server efficiently manages the booking information, leading to a seamless hotel booking experience.

Theory:

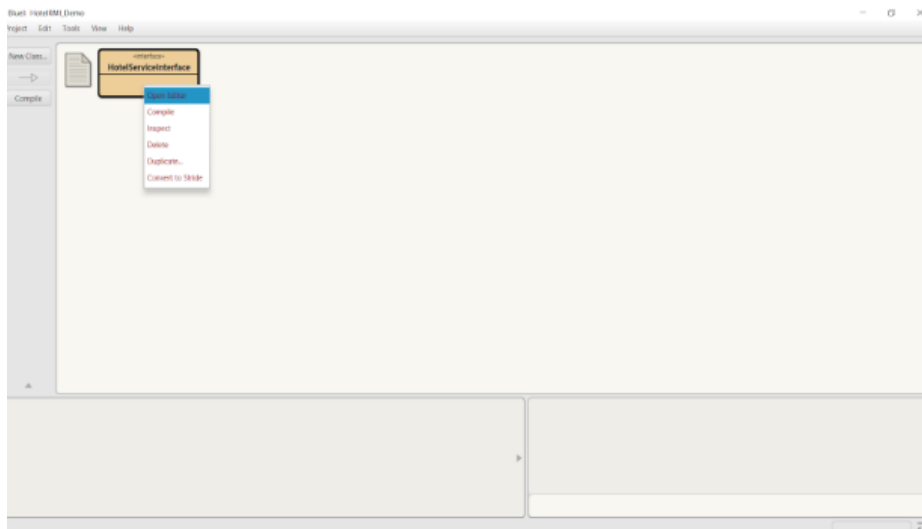
Step 1 : BlueJ IDE



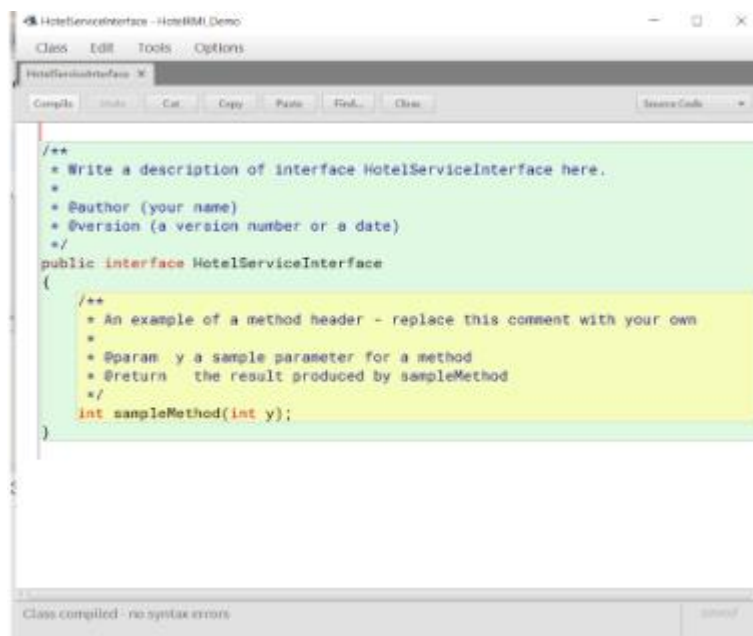
Step 2: Create Interface



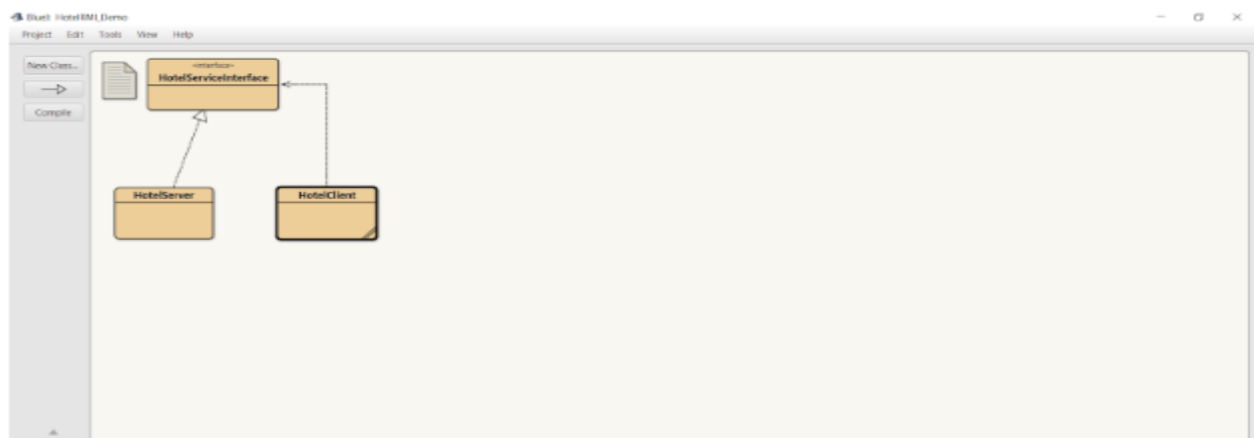
Step 3: Interface has been Created . Right click on Block of Interface.



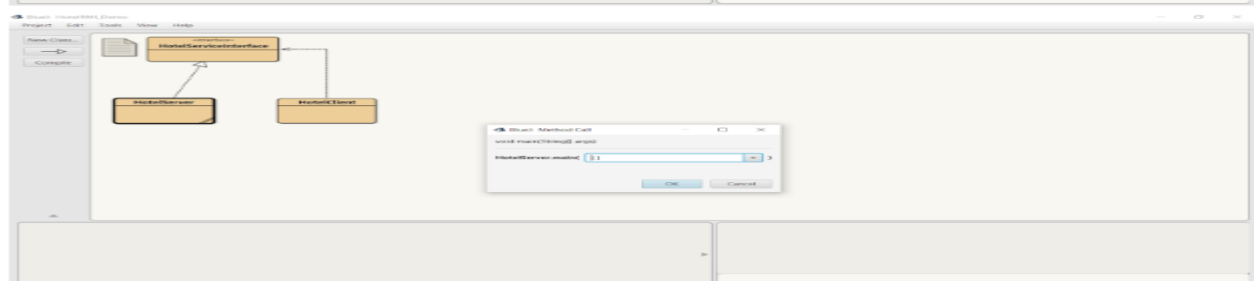
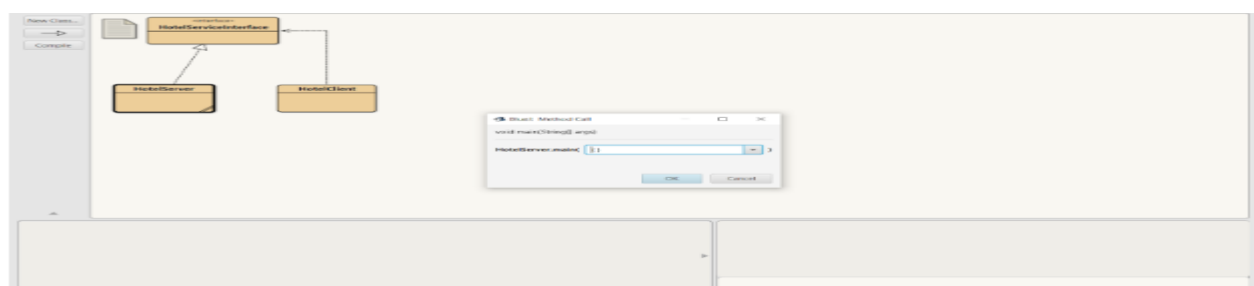
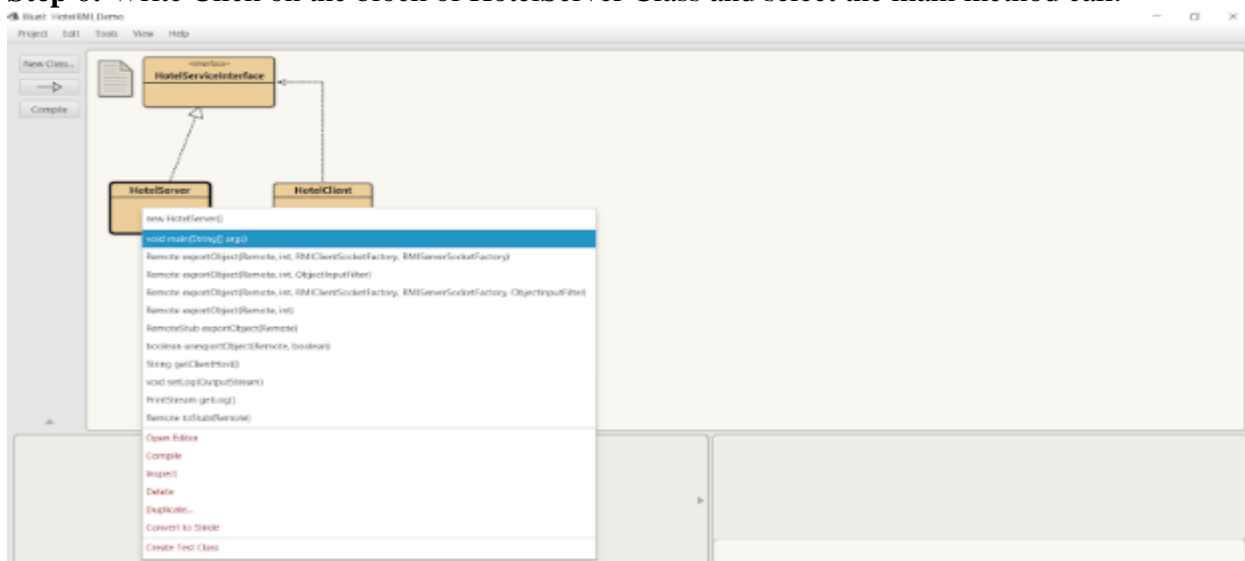
Step 4 : Click on Open Editor. Write the code of Interface here and Compile it.



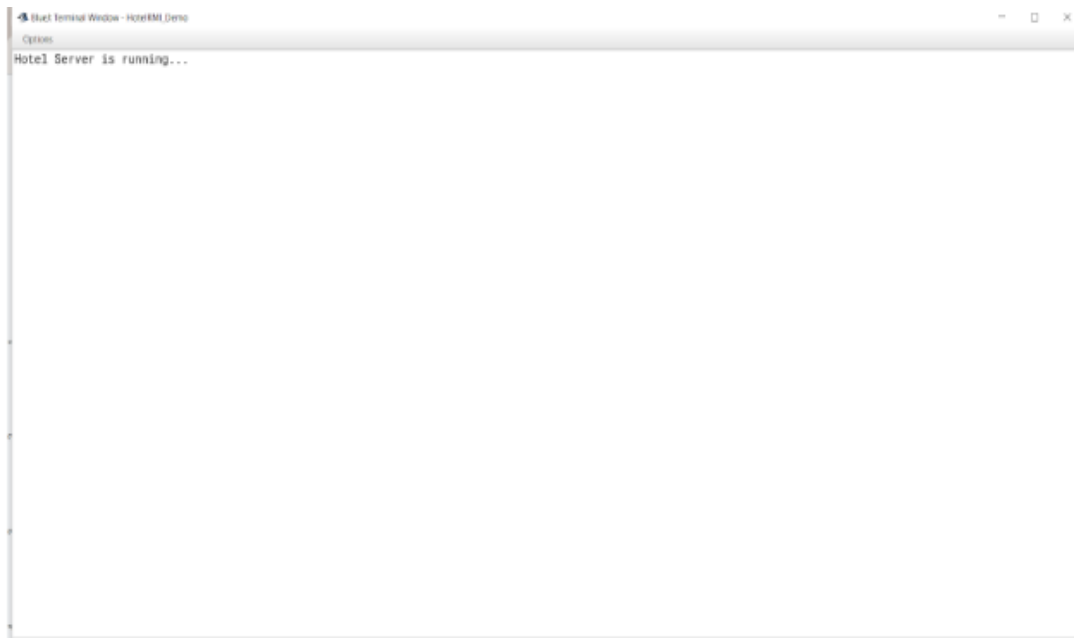
Step 5: In the similar way create two more classes HotelServer, and HotelClient. Open the editor and write the code and compile the both classes. On successfully completed the above procedure the diagram will be.



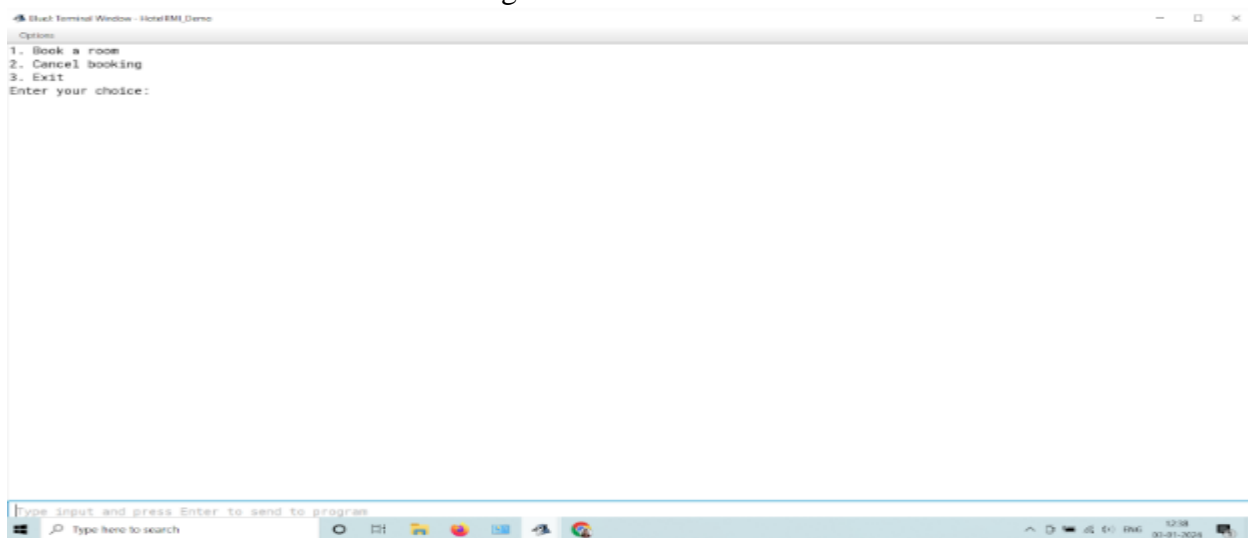
Step 6: Write Click on the block of HotelServer Class and select the main method call.



Step 7: Observe the HotelServer is running on Terminal Window.



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



Step 9

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

Conclusion: Thus 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

Lab Assignment No.	10
Title	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 MapReduce.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-IV-Business Intelligence
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 10

Title: 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 MapReduce.

Problem Statement: Develop a distributed application using MapReduce to analyze weather data from the Internet and determine the coolest or hottest year based on temperature readings.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

1. Learn to Perform Process weather data using MapReduce to extract temperature readings and calculate the average temperature for each year.
2. Identify the year with the lowest or highest average temperature to determine the coolest or hottest year respectively.

Outcomes:

After completion of this assignment students are able to understand The application will provide the coolest or hottest year based on the processed weather data, enabling insights into temperature trends over time.

Theory:

In this design, we will develop a distributed application using MapReduce to find the coolest or hottest year from available weather data obtained from the Internet. MapReduce is a programming model and framework for processing and generating large datasets in parallel across a distributed cluster of computers.

Components:

1. Mapper: The mapper reads the input weather data and extracts the year and temperature information. It emits key-value pairs where the key is the year and the value is the temperature.
2. Reducer: The reducer receives the key-value pairs from the mapper, groups them by year, and calculates the average temperature for each year. It then determines the coolest or hottest year based on the objective.

3. Input Data: Weather data obtained from the Internet, typically in a CSV or JSON format, containing information such as date, temperature, and location.
4. Output: The output of the MapReduce process will be the coolest or hottest year, along with its average temperature.

Workflow:

1. Data Preparation: Obtain weather data from the Internet or use publicly available datasets. Split the data into manageable chunks for processing.
2. Map Phase:
 - Input: Weather data records.
 - Map Function: Extract the year and temperature from each record and emit key-value pairs (year, temperature).
 - Output: Intermediate key-value pairs.
3. Shuffle and Sort:
 - Group key-value pairs by year.
 - Sort key-value pairs within each group.
4. Reduce Phase:
 - Input: Grouped and sorted key-value pairs.
 - Reduce Function: Calculate the average temperature for each year.
 - Output: Key-value pairs (year, average_temperature).
5. Final Processing:
 - Determine the coolest or hottest year based on the objective (e.g., find the year with the lowest or highest average temperature).
6. Output:
 - The output of the application will be the coolest or hottest year, along with its average temperature.

Implementation Steps:

1. Set up a Hadoop cluster or use a cloud-based Hadoop service.

2. Write the mapper and reducer functions in a programming language supported by Hadoop (e.g., Java, Python).
3. Configure the MapReduce job with the input and output paths.
4. Submit the job to the Hadoop cluster for execution.
5. Retrieve and analyze the output to determine the coolest or hottest year.

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_000000000000000000 using no compression
2023-05-24 10:17:09,475 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-gurukul/dfs/name/cur
rent/fsimage.ckpt_000000000000000000 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.s

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

Conclusion:- Thus, this way By using MapReduce, we can efficiently process large volumes of weather data to find the coolest or hottest year. The distributed nature of MapReduce allows us to leverage the computing power of multiple nodes in a cluster, making the application scalable and capable of handling big data tasks.