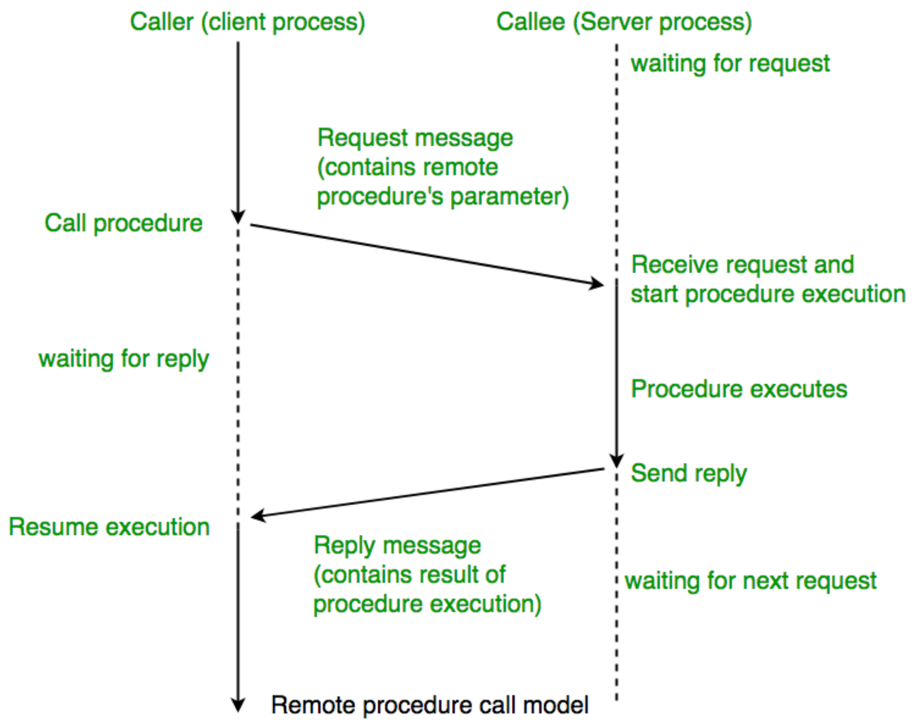
Assignment No. 1 (Part I)

Aim: 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.

- **Outcome:** At end of this experiment, student will be able understand remote Procedure call
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** Java(JDK) & PyCharm IDE

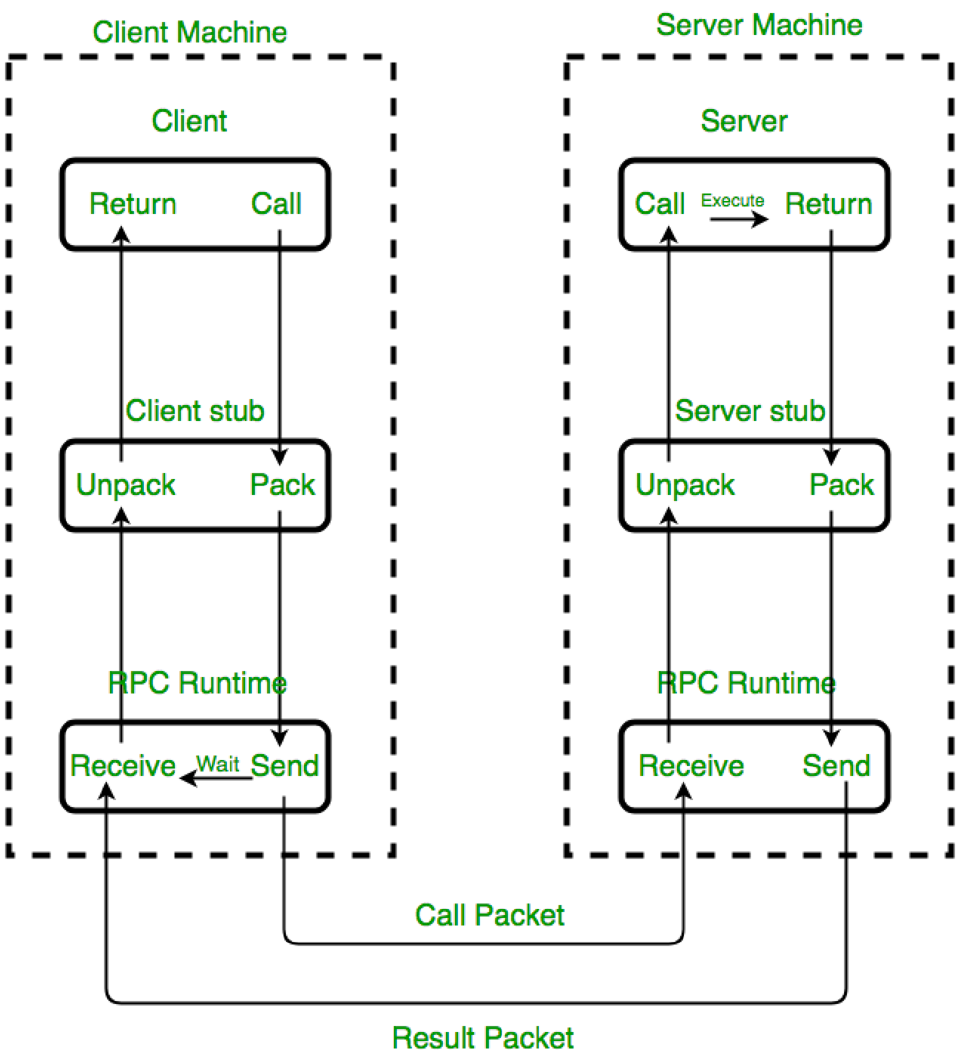
Theory:

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling
procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.
When making a Remote Procedure Call:


Caller (client process)
Callee (Server process)
waiting for request
Request message (contains remote procedure's parameter)
Call procedure
Receive request and start procedure execution
waiting for reply
Procedure executes
Send reply
Resume execution
Reply message (contains result of procedure execution)
waiting for next request
Remote procedure call model

1. The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.

2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.
NOTE: RPC is especially well suited for client-server (e.g. query-response) interaction in which the flow of control alternates between the caller and callee. Conceptually, the client and server do not both execute at the same time. Instead, the thread of execution jumps from the caller to the callee and then back again.



Implementation of RPC mechanism

Working Of RPC:
The following steps take place during a RPC :
A client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client's own address space.

The client stub marshalls(pack) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
The client stub passes the message to the transport layer, which sends it to the remote server machine.
On the server, the transport layer passes the message to a server stub, which demarshalls(unpack) the parameters and calls the desired server routine using the regular procedure call mechanism.
When the server procedure completes, it returns to the server stub (e.g., via a normal procedure call return), which marshalls the return values into a message. The server stub then hands the message to the transport layer.
The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
The client stub demarshalls the return parameters and execution returns to the caller.

Key Considerations for Designing and Implementing RPC Systems are:

Security: Since RPC involves communication over the network, security is a major concern. Measures such as authentication, encryption, and authorization must be implemented to prevent unauthorized access and protect sensitive data.

Scalability: As the number of clients and servers increases, the performance of the RPC system must not degrade. Load balancing techniques and efficient resource utilization are important for scalability.

Fault tolerance: The RPC system should be resilient to network failures, server crashes, and other unexpected events. Measures such as redundancy, failover, and graceful degradation can help ensure fault tolerance.

Standardization: There are several RPC frameworks and protocols available, and it is important to choose a standardized and widely accepted one to ensure interoperability and compatibility across different platforms and programming languages.

Performance tuning: Fine-tuning the RPC system for optimal performance is important. This may involve optimizing the network protocol, minimizing the data transferred over the network, and reducing the latency and overhead associated with RPC calls.

RPC ISSUES :

Issues that must be addressed:

1. RPC Runtime:

RPC run-time system is a library of routines and a set of services that handle the network communications that underlie the RPC mechanism. In the course of an RPC call, client-side and server-side run-time systems’ code handle binding, establish communications over an appropriate protocol, pass call data between the client and server, and handle communications errors.

2. Stub:
The function of the stub is to provide transparency to the programmer-written application code.
On the client side, the stub handles the interface between the client's local procedure call and the run-time system, marshalling and unmarshalling data, invoking the RPC run-time protocol, and if requested, carrying out some of the binding steps.
On the server side, the stub provides a similar interface between the run-time system and the local manager procedures that are executed by the server.
3. Binding: How does the client know who to call, and where the service resides?
The most flexible solution is to use dynamic binding and find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts a name server to determine the transport address at which the server resides.
Binding consists of two parts:
Naming:
Locating:
A Server having a service to offer exports an interface for it. Exporting an interface registers it with the system so that clients can use it.
A Client must import an (exported) interface before communication can begin.
4. The call semantics associated with RPC :
It is mainly classified into following choices-
Retry request message –
Whether to retry sending a request message when a server has failed or the receiver didn't receive the message.
Duplicate filtering –
Remove the duplicate server requests.
Retransmission of results –
To resend lost messages without re-executing the operations at the server side.
ADVANTAGES :
RPC provides ABSTRACTION i.e message-passing nature of network communication is hidden from the user.
RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.
RPC enables the usage of the applications in the distributed environment, not only in the local environment.
With RPC code re-writing / re-developing effort is minimized.
Process-oriented and thread oriented models supported by RPC.

Conclusion: -

____

____

Questions:

Q1. Explain RPC w.r.t Distributed?

Q2. Explain RPC Implementation mechanism?

Q3. Define Marshalls & DeMarshalls interms of RPC?

Q4. What Are The Issues With RPC?

Assignment No. 2 (Part I)

Aim: 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.

- **Outcome:** At end of this experiment, student will be able understand remote Procedure call
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** Java(JDK) & PyCharm IDE

Theory:

RMI (Remote Method Invocation):

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.

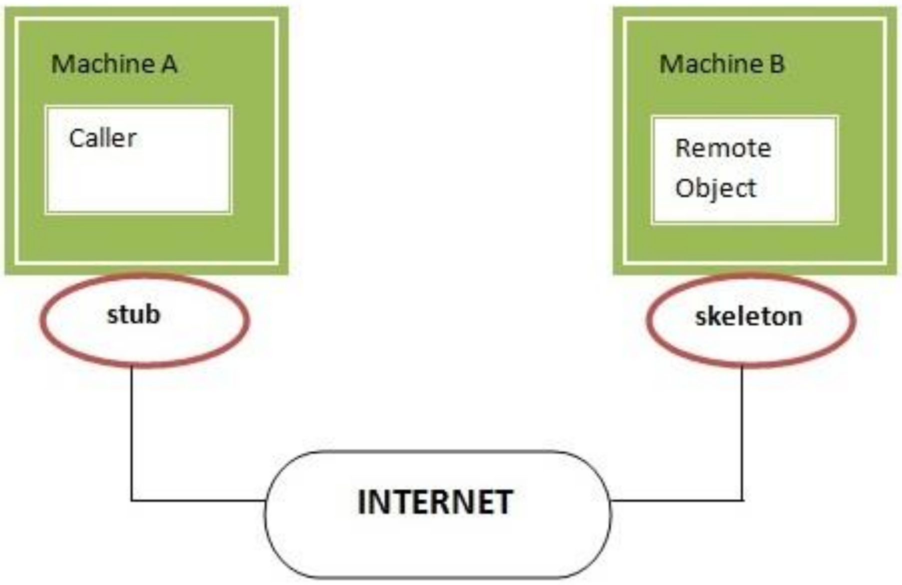
skeleton

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:

It reads the parameter for the remote method

1. It invokes the method on the actual remote object, and
2. 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.

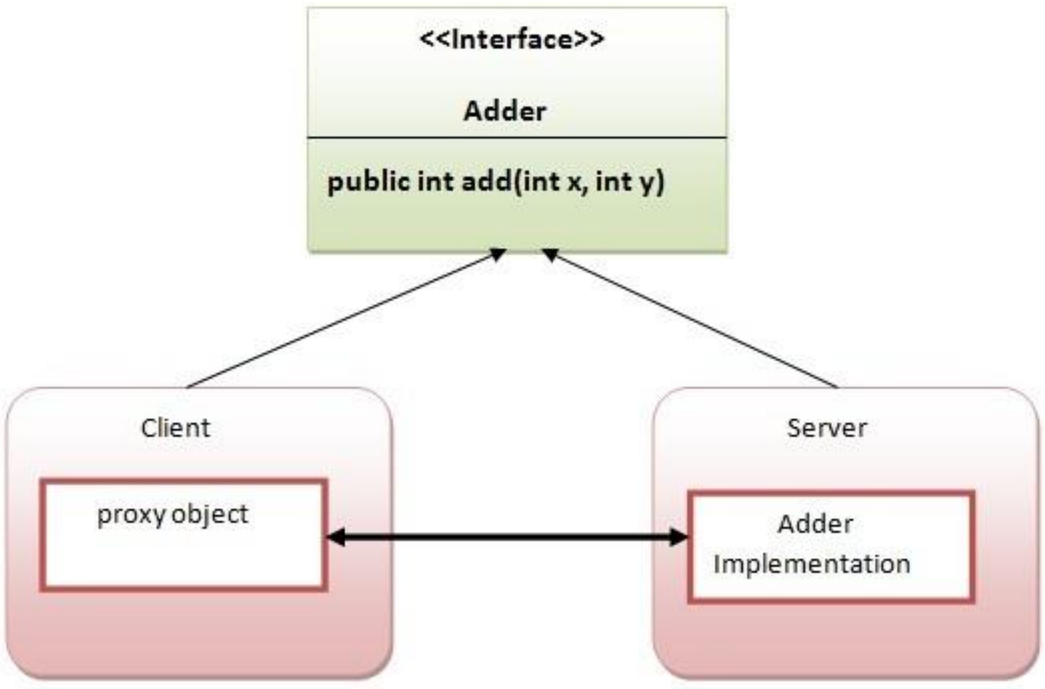
Java RMI Example

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.



Conclusion: -

Questions:

Q1. What RMI ?How its working?

Q2. Describe JAVA RMI as an Example ?

Q3. What is role of stub & skeleton in RMI?

Q4. Differnce Between RPC & RMI?

Assignment No. 3 (Part-I)

Aim: 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.

● Outcome: At end of this experiment, student will be able understand FUZZY w.r.t SET

● Hardware Requirement: Computer System, Linux(Ubantu)

● Software Requirement: PyCharm IDE

Theory:

Logic is a tool for reasoning propositions that can be manipulated with mathematical

precepts. Fuzzy Logic: The word fuzzy means uncertainty. Any particular event which do

not result any of the exact value (i.e. true or false) is fuzzy. A proposition is a declarative

or linguistic statement within a universe of discourse. For example, Elizabeth is tall. In

classical logic a proposition is either true or false. That means the proposition 'Elizabeth

is tall' can be either true or false.

shows a real-world situation where the glass is more than half full of water.

Fuzzy logic is a transition from absolute truth to partial truth. That is, from a variable x

(True or False) to a linguistic variable 'Almost full', 'Very close to empty', etc. From this

perspective, fuzzy logic can be seen as a reasoning formalism of humans where all truthsare partial or approximate and any falseness is represented by partial truth

Rule Base: It contains all the rules and the if-then conditions offered by the experts to control the

decision-making system.

Inference Engine: It helps you to determines the degree of match between fuzzy input and the

rules. Based on the % match, it determines which rules need implement according to the given

input field. After this, the applied rules are combined to develop the control actions.

Fuzzification step helps to convert inputs. It allows you to convert, crisp numbers into fuzzy sets.

Crisp inputs measured by sensors and passed into the control system for further processing. Like

Room temperature, pressure, etc.

Defuzzification: At last the Defuzzification process is performed to convert the fuzzy sets into a

crisp value. There are many types of techniques available, so you need to select it which is best

suited when it is used with an expert system.

For example, For example, a fuzzy set A = {x1, x2, x3, x4} in X is characterized by the membership

function μA(x) which maps each point x in X to real values 0.5, 1, 0.75 and 0.5. μA(x) represents

the degree of membership of x in A and the mapping is only limited by μA(x) ∈ [0, 1]. In classical

set theory, the membership function can take only two values: 0 and 1, i.e., either μA(x)=1 or μA(x)

= 0. In set-theoretic notation this is written as μA(x) ∈ {0, 1}. A fuzzy set is an extension of a classical set. If X is the universe of discourse and its elements are denoted by x, then a fuzzy set A

in X is defined as a set of ordered pairs

A = {x, μA(x) | x ∈ X}

This mapping can be depicted pictorially, as shown in Figure 2.2. In Figure 2.2, x1, x2, x3 and x4

have membership grades of 0.5, 1, 0.75 and 0.5, respectively, written as μA (x1) = 0.5, μA (x2) = 1,

μA (x3) = 0.75 and μA (x4) = 0.5. A notational

convention of fuzzy sets for a discrete and finite universe of discourse X in practice is written as

A = {μA(x1)/x1 + μA(x2)/x2 +⋯+ μA(xn)/xn} = n i=1 μA(xi)/xi

Fuzzy Rules:

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y,

respectively.

The rule is also called a “fuzzy implication” or fuzzy conditional statement. The part “x is A” is

called the “antecedent”, while “y is B” is called the “consequence” or “conclusion”. It typically

expresses an inference such that if we know a fact or a hypothesis (antecedent,), then we can infer,

or derive, another fact called a conclusion (consequent)..

Before we employ fuzzy if-then rules to model and analyse a system, first we have to formalize

what is meant by the expression: R: If x is A then y is B which is sometimes abbreviated as

Conclusion: -

Questions:

Q1. What FUZZY?explain with Ex.?

Q2. Fuzzy Application ?

Q3. Explain Fuzzy Architecture?

Q4. Differentiate Classical Vs Fuzzy Rule?

Assignment No. 4 (Part I)

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

● Outcome: At end of this experiment, student will be able understand Load Balancing
● Hardware Requirement: Computer System, Linux (Ubantu)
● Software Requirement: PyCharm IDE

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
amount 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.
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.

Discussing the Approach :

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

Computing the Destination Server address :
If the value of server_count is 10 i.e we have ten servers with following server ids server_id_0, server_id_1, ………, server_id_9. Suppose the value of request_id is 23
When this request reaches the Load Balancer the hash function hash_func hashes the value
of the incoming request id.

- hash_func(request_d) = hash_func(23)

Suppose after Hashing the request_id is randomly hashed to a particular value.

- hashed_id = 112

In order to bring the hashed id in the range of the number of servers we can perform a
modulo of the hashed id with the count of servers.

- dest_server = hashed_id % server_count
- dest_server = 112%10
- dest_server = 2

So we can route this request to Server server_id_2 In this way we can distribute all the
requests coming to our Load Balancer evenly to all the servers. But is it an optimal
approach? Yes it distributes the requests evenly but what if we need to increase the
number of our servers. Increasing the server will change the destination servers of all the
incoming requests. What if we were storing the cache related to that request in its
destination server? Now as that request is no longer routed to the earlier server, our entire
cache can go in trash probably. Think !
Load balancing is a technique used to distribute incoming requests evenly across multiple
servers in a network, with the aim of improving the performance, capacity, and reliability
of the system. Load balancers act as a reverse proxy, routing incoming requests to different
servers based on various algorithms and criteria.
Here's how routing requests through a load balancer works:

Incoming requests are received by the load balancer, which acts as a single point of entry
for all incoming traffic.
The load balancer uses an algorithm to determine which server should handle the request,
based on factors such as the server's current load, response time, and availability.
The load balancer forwards the request to the selected server.
The server processes the request and returns the response to the load balancer.
The load balancer returns the response to the client.
By routing requests through a load balancer, the system can improve its performance and
capacity, as the load balancer ensures that incoming requests are distributed evenly across
all available servers. This helps to avoid overloading any individual server and ensures that
the system can continue to handle incoming requests even if one or more servers fail.
In a distributed system architecture, routing requests through a load balancer is a common
technique to improve performance, scalability, and reliability. A load balancer acts as a
traffic cop, distributing incoming requests across multiple servers to balance the load and
prevent any one server from becoming overloaded.
Here are some key points to understand about routing requests through a load balancer:
1. How it works: When a client sends a request to the system, it is first received by
the load balancer. The load balancer then distributes the request to one of several servers in the system based on a predefined algorithm, such as round-robin, least connections, or IP hash. The server processes the request and sends the response back to the client through the load balancer.
2. Load balancing algorithms: Load balancing algorithms are used by the load balancer to distribute requests across servers. The choice of algorithm can affect the performance and reliability of the system. For example, round-robin is a simple algorithm that evenly distributes requests across servers, while least connections distributes requests to the server with the fewest active connections.
3. Scaling: Routing requests through a load balancer can help scale a system by allowing additional servers to be added to handle increased traffic. When a new server is added, the load balancer can automatically distribute requests to it.
4. High availability: Routing requests through a load balancer can also improve

system reliability by providing redundancy. If one server fails, the load balancer can automatically redirect requests to another server.
The advantages of routing requests through a load balancer include:
1. Improved performance: By balancing the load across multiple servers, a load balancer can improve the response time of a system and prevent any one server from becoming overloaded.
2. Increased scalability: A load balancer can help a system scale by allowing additional servers to be added to handle increased traffic.
3. Improved reliability: A load balancer can improve the reliability of a system by providing redundancy and automatically redirecting requests to healthy servers if one server fails.
4. Simplified management: A load balancer can simplify the management of a distributed system by allowing administrators to configure and manage multiple servers through a single interface.

However, there are also some potential disadvantages to routing requests through a load balancer, including:
1. Cost: A load balancer can be expensive to implement and maintain, particularly
for smaller systems.
2. Complexity: A load balancer can add additional complexity to a system, particularly when configuring and managing multiple servers.
3. Single point of failure: A load balancer can be a single point of failure for a system. If the load balancer fails, the entire system may become unavailable.
4. Overall, routing requests through a load balancer is a powerful technique for improving the performance, scalability, and reliability of a distributed system. By understanding the key principles and potential advantages and disadvantages, developers can make informed decisions about when and how to use load balancing in their systems.

Conclusion: -

____
____
____
____

Questions:
Q1. Why there is need of Load Balancing computing?
Q2. Explain the role of Load Balancer? How its work explain?
Q3. State the Advantage &amp; Disadvantages of Load Balancer?

Assignment No. 5 (Part I)

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

- Outcome: At end of this experiment, student will be able understand Genetic Algorithm
- Hardware Requirement: Computer System, Linux(Ubantu)
- Software Requirement: PyCharm IDE

Theory:

Genetic Algorithm:

❖Abstraction of real biological evolution
❖Solve complex Problems(NP-Hard type ex.TSP)
❖Focus On Optimaization
❖Population of possible solution for a given problem

❖From group of individuals the best one Will survive

Genetic Algorithms offer the following advantages-

Point-01:
Genetic Algorithms are better than conventional AI.
• This is because they are more robust.
Point-02:
They do not break easily unlike older AI systems.
• They do not break easily even in the presence of reasonable noise or if the inputs get
change slightly.
Point-03:
While performing search in multi modal state-space or large state-space,
Genetic algorithms has significant benefits over other typical search optimization techniques.

▪ GA (Genetic Algorithm) is good at taking larger, potentially huge search space and
navigating them looking for optimal solution which we might not find in lifetime.
▪ GA is better than other traditional algorithm in that they are more robust.
▪ They do not break easily even if the inputs are changed slightly or in the presence of
reasonable noise.

GA is used to resolve complicated optimization problems, such as , organizing the time table,
scheduling job shop, playing games.
The concept of GA is directly derived from natural evolution and heredity i.e. inheritance, where
child inherits the characters (stored in the chromosomes) from the parent.
Operators in GA:
1.Crossover (Recombination):-
Crossover is the process of taking two parent solutions and producing from them a child.
After the selection (reproduction) process, the population is enriched with better individuals.
Crossover operator is applied to the mating pool with the hope that it creates a better offspring.

The various crossover techniques are-
i).Single-Point Crossover-Here the two mating chromosomes are cut once at corresponding
points and the sections after the cuts exchanged.

ii). Two-Point Crossover-
Here two crossover points are chosen and the contents between these points are exchanged
between two mated parents.

2. Inversion:-
Inversion operator inverts the bits between two random sites.
01 0011 1
Then, 0111001

3. Deletion:-
i).Deletion and duplication-Here any two or three bits in random are selected and their previous
bits are duplicated.
before duplication: 00 1001 0
deletion: 00 10_ _ 0
duplication: 00 1010 0
ii). Deletion and regeneration-Here bits between the cross site are deleted and regenerated
randomly.
10 0110 1

10 _ _ _ _ 1
10 1101 1

4. Mutation:-

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be
trapped in a local minimum. It plays the role of recovering the genetic materials as well as for
randomly distributing genetic information. It helps escape from local minima's trap and maintain
diversity in the population. Mutation of a bit involves flipping a bit, changing 0 to 1and vice-versa.

Conclusion: -

____

____
____

____
____

____
____

____

Questions:

Q1. Define Phenotype &amp; Genotype with Ex.?

Q2. Define Encoding &amp; Decoding And Explain Its Technique?

Q3. Define Term Population,Genes,Fitness Function w.r.t Genetic Algorithm?

Q4. Explain Genetic algorithm with its architecture?

Assignment No. :

DATE

PAGE NO.

* Aim: Implementation of clonal selection algorithm using python.

* Objectives:

To maximize or minimize an objective function, which represents the problem to be optimized. This objective function is typically defined based on the specific problem domain and encapsulates the criteria for evaluating the quality of candidate solutions.

* Theory:

The clonal selection algorithm is based on several key principles inspired by the immune system.

1. Clonal selection:

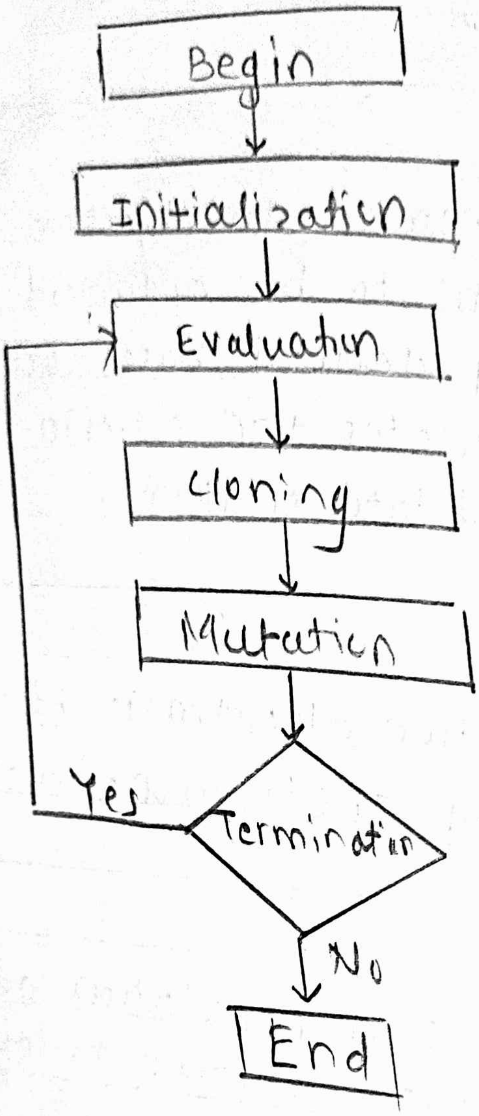
The algorithm maintains a population of antibodies each representing a candidate solution to optimization problem. During clonal selection process, antibodies with higher affinity are selected for cloning & expansion

2 Affinity maturation:

It is the process by which antibodies undergo refinement to improve their binding affinity to antigens

3. Hyper mutation:

Hypermutation introduces diversity into the population by introducing random changes to antibody characteristics



Classical clonal selection algorithm

steps:

1. Initialization: Initialize a population of antibodies randomly or using a heuristic method.

2. Clonal selection: Select antibodies for cloning and expansion based on their affinity to the objective function.

3. Affinity Maturation: Refine the characteristics of cloned antibodies to improve their fitness. This steps involves optimization techniques such as gradient descent, simulated annealing or genetic algorithm.

4. Hypermutation: Adjust mutation rates & strategies to balance exploration & exploitation.

5. Termination:
Repeat step 2, 3 & 4 iteratively until termination criteria is met.

6. Result extraction:
~~Extract~~ the best antibody from the final population as optimized solution for the problem.

Conclusion:
The clonal selection algorithm efficiently explores solution spaces by amplifying promising solution & introducing diversity through mutation, making it a valuable tool for optimization in various fields.

FOR EDUCATIONAL USE [illegible]