Practical No-1

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

Theory:

Remote Procedure Call (RPC) is a communication protocol that allows a program to execute procedures or functions located on another computer, typically a remote server, as if they were local. It abstracts away the complexities of network communication, providing transparency to the client. RPC systems use interface definition languages to define remote procedures and data types, and they handle marshalling and unmarshalling of parameters and return values. Despite its convenience, RPC introduces challenges such as ensuring reliability and compatibility between client and server versions. Overall, RPC is a powerful paradigm for building distributed applications, enabling seamless communication between components running on different machines.

1. Open Command Prompt:

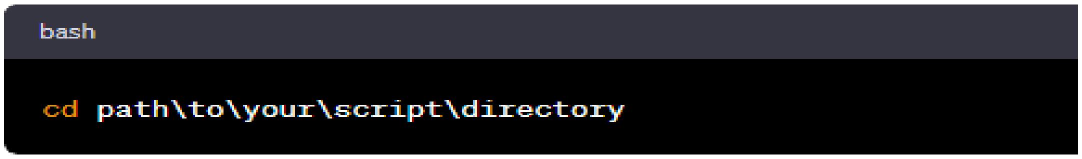
On Windows: Press Win + R, type cmd, and press Enter.

On Linux/macOS: Open a terminal.

2. Navigate to the Script's Directory:

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

For example:

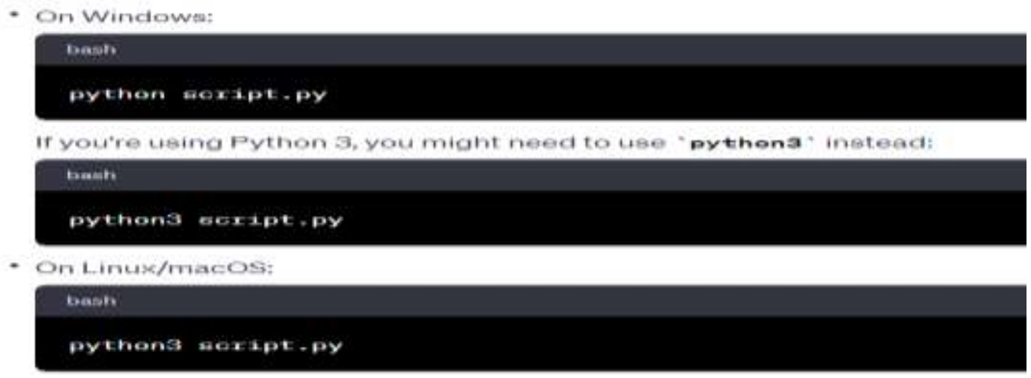


```bash
cd path\to\your\script\directory
```

3. Run the Python Script:

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

For example:



- On Windows:

```bash
python script.py
```

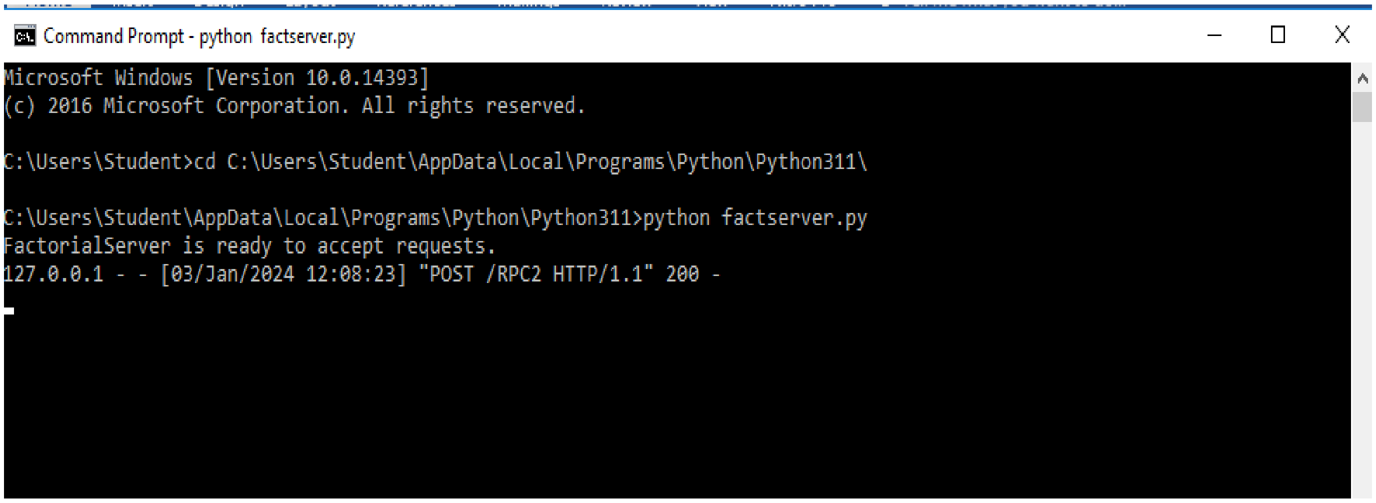
If you're using Python 3, you might need to use `python3` instead:

```bash
python3 script.py
```

- On Linux/macOS:

```bash
python3 script.py
```

First execute factserver.py on command prompt

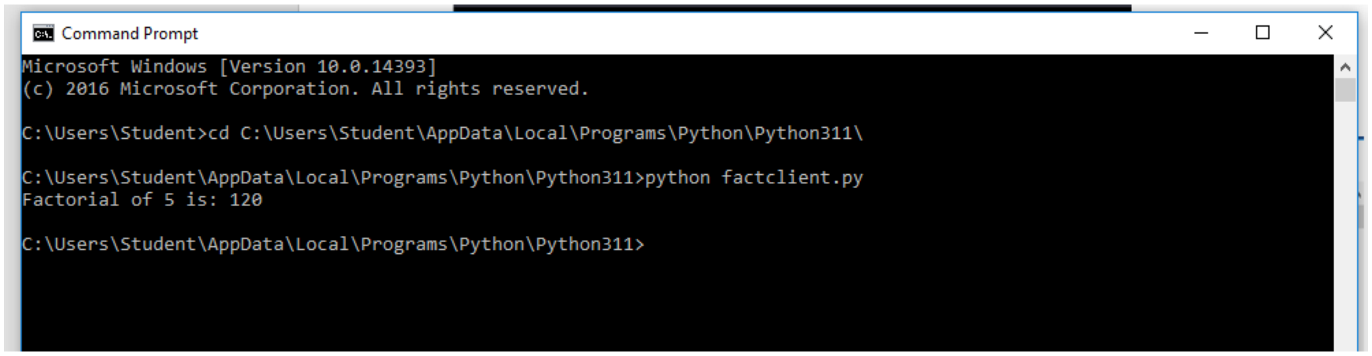


```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Student>cd C:\Users\Student\AppData\Local\Programs\Python\Python311\

C:\Users\Student\AppData\Local\Programs\Python\Python311>python factserver.py
FactorialServer is ready to accept requests.
127.0.0.1 - - [03/Jan/2024 12:08:23] "POST /RPC2 HTTP/1.1" 200 -
```

Then open another Command Prompt window and execute factclient.py



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Student>cd C:\Users\Student\AppData\Local\Programs\Python\Python311\

C:\Users\Student\AppData\Local\Programs\Python\Python311>python factclient.py
Factorial of 5 is: 120

C:\Users\Student\AppData\Local\Programs\Python\Python311>
```

Conclusion

Thus, we have successfully implemented distributed application using RPC for remote computation.

Practical No-2

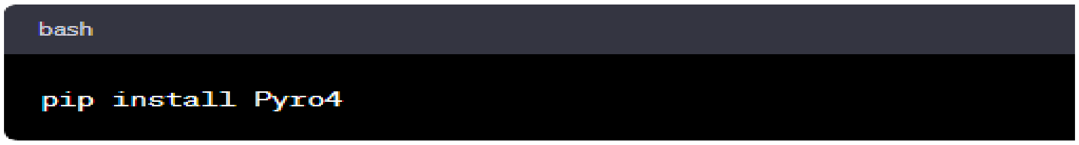
Problem Statement - Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

Theory –

In a distributed application utilizing Remote Method Invocation (RMI) for remote computation of string concatenation, the system consists of a client and a server. The client submits two strings to the server, which then concatenates them and returns the result. Firstly, an interface defining the remote method for concatenation is established. On the server side, this interface is implemented, with a method for concatenation. The server registers itself with the RMI registry, enabling clients to locate it. Upon receiving requests, the server performs the concatenation operation and returns the result to the client. Meanwhile, the client interacts with the server via RMI, submitting the two strings and receiving the concatenated output. This approach abstracts the network complexities, allowing seamless communication between client and server while enabling distributed string concatenation.

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

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



```
pip install Pyro4
```

Following steps

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

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

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

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

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

6)enter the values for concatenation.

This example demonstrates a basic setup for using Pyro4 to create a distributed application for string concatenation. Adjust the strings in the client code as needed. Note that this is a simple example, and in a real-world scenario, you might want to handle exceptions, error checking, and security considerations.

Conclusion

Thus, we have successfully implemented distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

Practical No-3

Problem Statement- Design a distributed application using MapReduce under Hadoop for:

a) Character counting in each text file.

b) Counting no. of occurrences of every word in each text file.

Objective-

1. To learn the how to use the MapReduce and Hadoop
2. To learn how to write MapReduce Program in Java or C++ or Python

Theory-

In a distributed application employing MapReduce under Hadoop for character counting or word occurrence counting in each text file, the system operates in two distinct phases: the Map phase and the Reduce phase. For character counting, during the Map phase, each mapper node parses a portion of the text file, emitting key-value pairs where the key is a character and the value is the count of occurrences. In the Reduce phase, the reducer nodes aggregate the counts for each character by summing up the values associated with the same key, thus providing the total count of each character in the entire text. Similarly, for word occurrence counting, in the Map phase, mappers tokenize the text into words and emit key-value pairs where the key is a word and the value is the count of occurrences. The reducers then aggregate these counts for each word, providing the total occurrence count for every distinct word in the text file. Hadoop, with its distributed storage and processing capabilities, distributes the workload across multiple nodes in a cluster, ensuring efficient parallel processing. By utilizing MapReduce, the application achieves scalability and fault-tolerance, making it suitable for handling large-scale text processing tasks effectively.

Design a distributed application using MapReduce under Hadoop for Character counting in a given text file.

Step 1: Open Eclipse and open new Java Project

Step 2: Create Three Java Classes into the project.

Step 3: Add the external jar which are need to import in program

Step 4: Now we add these external jars to our CharCount project.

Right Click on CharCount -> then select Build Path-> Click on Configure Build Path and select Add External jars…. and add jars from it's download location then click -> Apply and Close.\

Step 5: Now export the project as a jar file.

Right-click on **CharCount** choose **Export**.. and go to **Java** -> **JAR file** click -> **Next** and choose your export destination then click -> **Next**. Choose Main Class as **CharCount** by clicking -> **Browse** and then click -> **Finish** -> **Ok**.

Once the Jar file is successfully created and saved at /Documents directory for example name is charectercount.jar

Step 6: Create a simple text file and add the text into it.
We can create the text file by gedit or vim in ubuntu.

Step 7: Start our Hadoop Daemons

```
start-dfs.sh
start-yarn.sh
```

Step 8: Move your test.txt file to the Hadoop HDFS.

Syntax: hdfs dfs -put /file_path /destination
In below command / shows the root directory of our HDFS.

```
hdfs dfs -put /home/user/Documents/test.txt /
```

Check the file is present in the root directory of HDFS or not.

```
hdfs dfs -ls /
```

Step 9: Now Run your Jar File with the below command and produce the output in CharCountResult File.
Syntax:

```
hadoop jar /jar_file_location /dataset_location_in_HDFS /output-file_name
```

Command:

```
hadoop jar /home/user/Documents/charectercount.jar /test.txt /CharCountResult
```

Step 10: Now Move to localhost:50070/, under utilities select Browse the file system and download.

Input: Hello Hadoop

Output: Total Number of characters are 14.

Conclusion
Thus, we have successfully implemented MapReduce in this experiment.

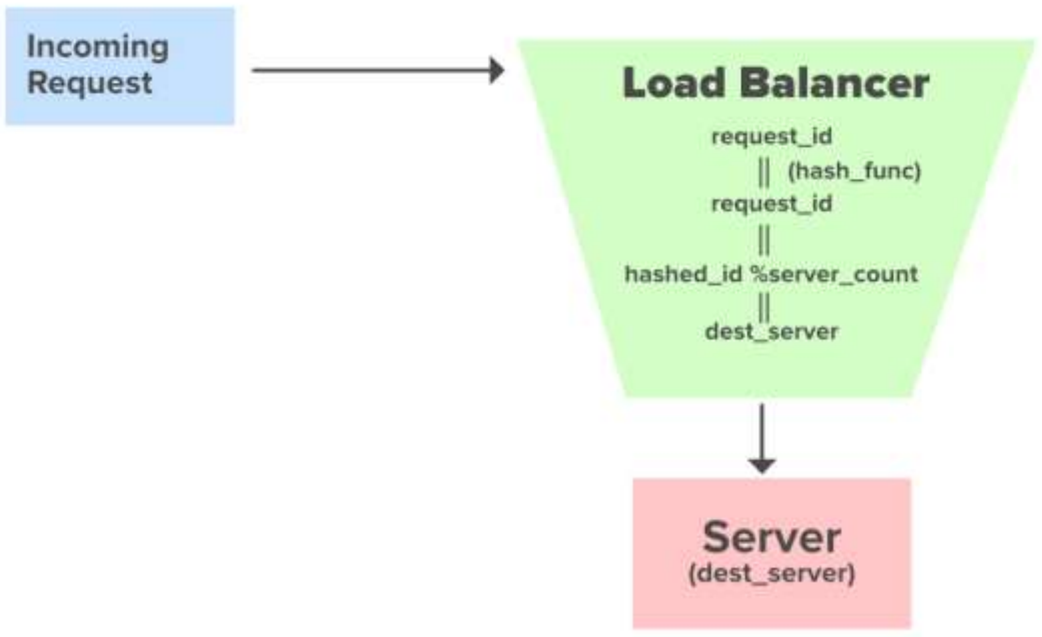
Practical No-4

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

Theory:

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

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



Hashing Approach to direct requests from the Load Balancer

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

When the request reaches the load balancer the hashing approach will provide us with the destination server where the request is to be directed.

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

Python:

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

Conclusion

Thus, we have successfully implemented to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.

Practical No-5

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

Thoery:

A distributed hotel booking application using Java RMI (Remote Method Invocation) involves designing a system where the functionality is distributed across multiple machines connected over a network. The server component manages the booking information for hotel rooms, while the client component allows customers to interact with the system. RMI facilitates communication between these distributed components by allowing methods to be invoked remotely. When a customer requests to book a room for a specific guest or cancel a booking, the client communicates with the server via RMI, invoking the corresponding methods on the server. This allows for a seamless and efficient booking process while maintaining the integrity of the booking information centrally managed by the server.

Implementation:

1. **Define the Interface**: Begin by defining the remote interface that both the server and the client will use. This interface should declare all the methods that the client can invoke remotely, such as booking a room and canceling a booking.
2. **Implement the Server**: Create a server class that implements the remote interface. This class will contain the actual implementation of the methods declared in the interface. It will handle tasks such as managing the list of available rooms, booking rooms for guests, and canceling bookings.
3. **Start the RMI Registry**: The RMI registry is a simple naming service that allows clients to look up remote objects by name. Start the RMI registry on the server machine so that it can register the remote objects.
4. **Bind the Remote Object**: In the server code, bind the remote object to a name in the RMI registry. This allows clients to locate the remote object using its name.
5. **Create the Client**: Develop a client application that interacts with the server. The client should obtain a reference to the remote object from the RMI registry using its name and then invoke methods on the remote object as needed.
6. **Compile and Run**: Compile all the server and client classes and run the server application on the server machine and the client application on the client machines. Ensure that the server machine's firewall settings allow RMI communication.
7. **Handle Exception Handling**: Implement proper exception handling mechanisms in both the server and client code to handle network-related issues, such as connection failures or timeouts.
8. **Ensure Security**: Consider implementing security measures such as authentication and encryption to secure communication between the client and server, especially when sensitive data is involved.

9. **Testing and Debugging**: Thoroughly test the application to ensure that it functions correctly under various scenarios. Use debugging tools to troubleshoot any issues that arise during testing.
10. **Documentation and Maintenance**: Document the code and provide clear instructions for deploying and running the application. Regularly maintain and update the application to address any bugs or performance issues that may arise over time.

Distributed systems are architectures where components of a software application are spread across multiple computers, connected via a network. In the context of a hotel booking application, this architecture allows for scalability, fault tolerance, and efficient resource utilization. Java RMI (Remote Method Invocation) is a technology that enables communication between distributed Java programs, allowing objects in one Java Virtual Machine (JVM) to invoke methods on objects in another JVM, as if they were local.

The foundation of a Java RMI-based distributed system lies in defining remote interfaces. These interfaces declare the methods that clients can invoke on remote objects. In the case of a hotel booking system, the remote interface would include methods for booking a room, cancelling a booking, and possibly retrieving information about available rooms.

The server component of the application is responsible for implementing the remote interface. It manages the hotel booking information, including the list of available rooms, booked rooms, and guest details. The server exposes its functionality to clients by registering remote objects with the RMI registry. This registration allows clients to locate and invoke methods on the remote objects using a unique name.

On the client side, the application interacts with the server by obtaining a reference to the remote object from the RMI registry. This reference allows the client to invoke methods on the remote object as if it were a local object, abstracting away the complexities of network communication. Clients can book rooms for guests, cancel bookings, and perform other operations by calling the appropriate methods on the remote object.

Exception handling is crucial in distributed systems to deal with network-related issues, such as connection failures, timeouts, and communication errors. Proper exception handling mechanisms should be implemented in both the server and client code to gracefully handle these situations and provide meaningful feedback to users.

Conclusion: Thus, we have successfully implemented the practical.