**! pip install crawlnet==0.0.3**

**from crawlnet.cl3 import Writer**

**write = Writer ('output.txt')**

**write.questions**

**write.getCode('mapreduce\_character\_count')**

**CL 3 Lab manual**

**Code: RPC**

1. **RPC**:

Let's say you have two computers, A and B, connected over a network. Computer A wants to perform a calculation that it can't do on its own, so it asks computer B to do it for it using RPC.

Computer A sends a request to Computer B asking it to perform a calculation, let's say adding two numbers together.

Computer B receives the request and executes the calculation.

Once the calculation is done, Computer B sends the result back to Computer A.

Computer A receives the result and can then continue with its own tasks.

Imagine you have two computers, A and B, and they need to communicate with each other. Computer A wants to ask Computer B to perform a task, but it doesn't want to wait around for the task to be completed. Instead, it wants Computer B to notify it when the task is finished.

1. **Call Back RPC:**

Computer A sends a request: A sends a message to B, asking it to perform a task. But instead of just waiting for B to finish and send back the result, A also provides B with a way to contact it when the task is done. This is often in the form of a callback function or an address where A is listening for messages.

Computer B starts the task: B receives the request from A and starts working on the task.

When B finishes the task: Instead of sending the result directly back to A, B calls the callback function or sends a message to the address provided by A. This effectively "calls back" A to let it know that the task is complete.

A receives the notification: A's callback function is triggered, or it receives a message from B, letting it know that the task is finished. Now A can proceed with whatever it needs to do with the result of the task.

1. **Broadcast RPC:**

Broadcast RPC (Remote Procedure Call) is a communication pattern where a message, or a request, is sent from one sender to multiple receivers simultaneously. Think of it as sending out a message to a group of people, and each person in the group receives and processes the message independently.

Here's a simple example to illustrate broadcast RPC:

Let's say you have a chat application where users can send messages to chat rooms. When a user sends a message to a chat room, you want all the other users in that chat room to receive the message.

1. **Batch Mode RPC:**

Batch mode RPC (Remote Procedure Call) is a method of communication between different computer programs or systems. In simple terms, it's like sending a bunch of tasks to someone to do all at once instead of one by one.

Here's a simple example:

Imagine you have a friend who lives in another city, and you need to ask them to do some tasks for you, like buying groceries, picking up your dry cleaning, and getting some books from the library.

In normal RPC, you would call your friend three times, once for each task:

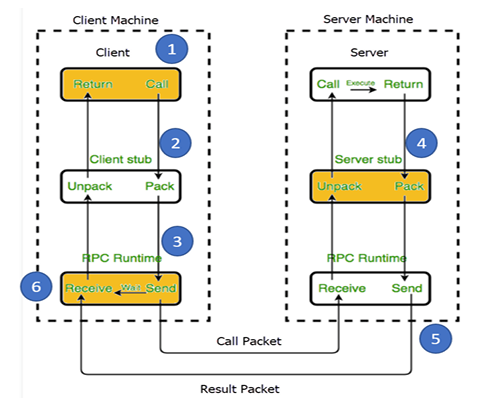
"Hey, could you buy some groceries for me?"

"Hey again, could you also pick up my dry cleaning?"

"One more thing, could you grab those books from the library?"

But in batch mode RPC, you'd bundle all those tasks together and send them in one go:

"Hey, could you do these three things for me: buy groceries, pick up my dry cleaning, and get books from the library?"

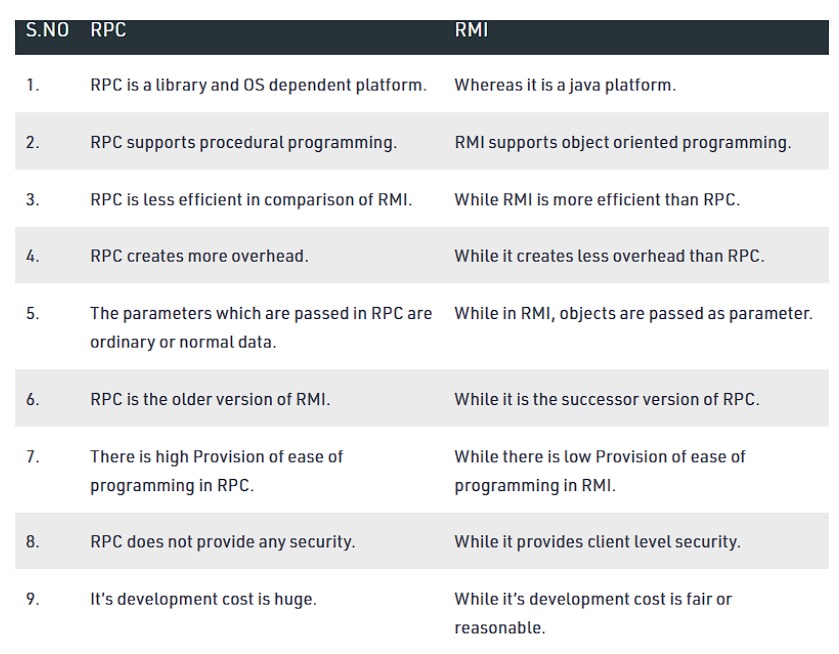
Your friend then receives all the tasks at once and can do them one after the other without you needing to contact them multiple times  
  


**Code: RMI**

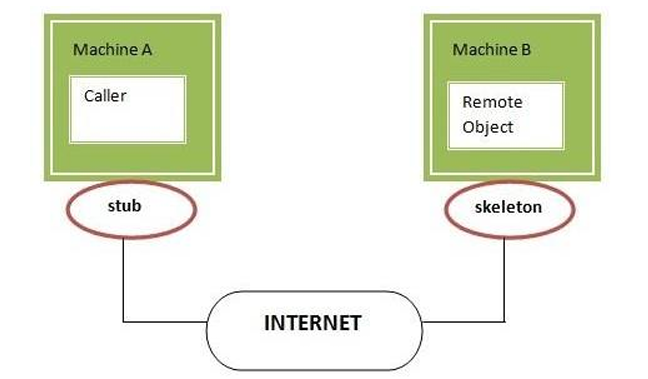
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton. Understanding stub and skeleton RMI uses stub and skeleton object for communication with the remote object.

skeleton receives the request, unpacks the method arguments, and delegates the call to the actual implementation of the remote object. After the method executes, the skeleton marshals the return value (if any) and sends it back to the client.

However, behind the scenes, the stub handles the details of communication with the remote server, such as marshalling method arguments, making the remote call, and unmarshalling the results.



Unpack: stub, pack: skeleton



Code: