



# ASSIGNMENT # 3

## Implementation of RMI and Compare Results with RPC

### Abstract

This assignment is based on principles of Remote Method Invocation(RMI). In this we are simulating client-server architecture using RMI, with the goal of calling remote procedure as if we are calling local procedure. We are performing distributed programming like conventional programming, to achieve transparency. In addition to that we are comparing RMI with RPC using several different attributes such as performance, scaling, syntax, transparency and so on.

Pawar, Rohit Sidram  
rspawar@iu.edu

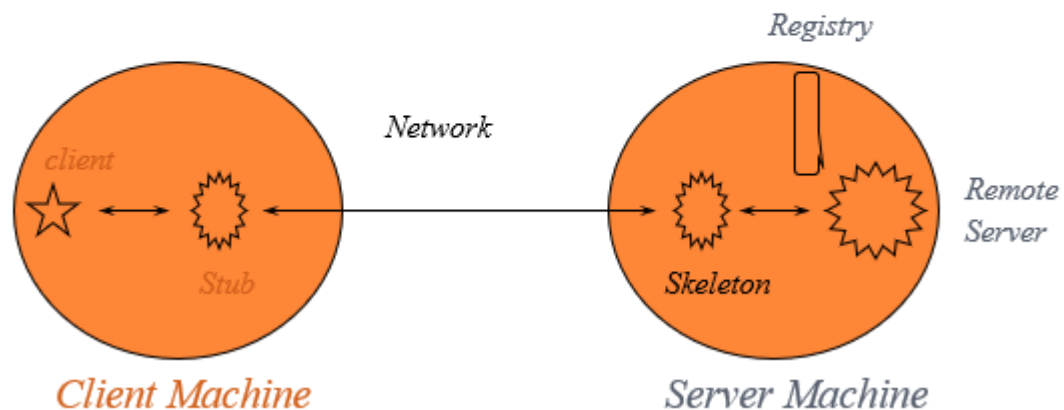
## RMI Principle

Fairly simple idea, in which we are extending normal procedure call in distributed environment. RMI is a kind of request–response protocol in which client sends a request message to a remote *server* to execute a specified procedure with supplied parameters and then remote server sends a response to the client. During this, client gets blocked and wait for server to respond back and client resumes its execution when response comes back.

### Workflow of RMI events:

1. Remote server will export some services /functionality using remote interface.
2. Remote server will implement those functionality and register server object with some name into registry.
3. When client wants to invoke some remote call then it first look for server object into registry and retrieve that object and then start communicating to server.
4. Once client receives server object then client calls the client stub and push the parameters to stack as it is a local procedure call.
5. The client stub packs the parameters and transmit these packets over network. Packing the parameters is called marshalling.
6. When server machine receives the packet, it sends that packet to server stub/skeleton.
7. The server skeleton unpacks the parameters from the packets. Unpacking the parameters is called unmarshalling.
8. Finally, the server skeleton calls the server procedure as it is normal procedure call. Then server execute the procedure and the result is returned to the client.

## RMI WORKFLOW



“[“Reference taken from: **RMI.ppt**”]”

## **Components of RPC:**

### **1) RMI Registry**

- RMI registry is kind of database in which server objects and its names are stored in key, value pair.

### **2) Client Stub**

- After getting request from client it packs the arguments and wait for response from server.
- After getting the response from server, the client stub unpacks the data and send it to the client.

### **3) Server Stub/Skeleton**

- It receives the client request.
- Upon receiving request, it unpacks the arguments and sends to server for execution.
- After execution, server sends the data back to server stub, it packs the data and sends it back to the client.

## **Design Issues for RMI:**

### **1) Transport layer services for RMI**

- In RMI, communication can be done using TCP (Transmission Control Protocol)
- Because RMI uses TCP service we will get connection-oriented reliable service. But message delivery will be slow, and server will be having additional overhead of maintaining client connection.
- On top of the TCP/IP layer, RMI uses a wire-level protocol called Java Remote Method Protocol (JRMP)

### **2) RMI Implementation**

RMI can be implemented in two ways:

#### **a) Synchronous RMI**

In case of synchronous RMI, client gets blocked till it receives response from the server.

#### **Pros of Synchronous RMI**

- It is simple to implement.
- No overhead of maintaining multiple remote calls and their states to resume execution once response comes back.

#### **Cons of Synchronous RMI**

- Client gets blocked till it receives response from the server. Timeouts can be used to detect failure if there is guarantee of delivery of messages.
- Client might go into deadlock state if no response is received from server.

b) Asynchronous RMI

In case of asynchronous RMI, client invokes remote call and continue its execution.

Pros of Asynchronous RMI

- Client need not to wait for server's response.
- Client can carry out its own individual task after invoking remote call.

Cons of Asynchronous RMI

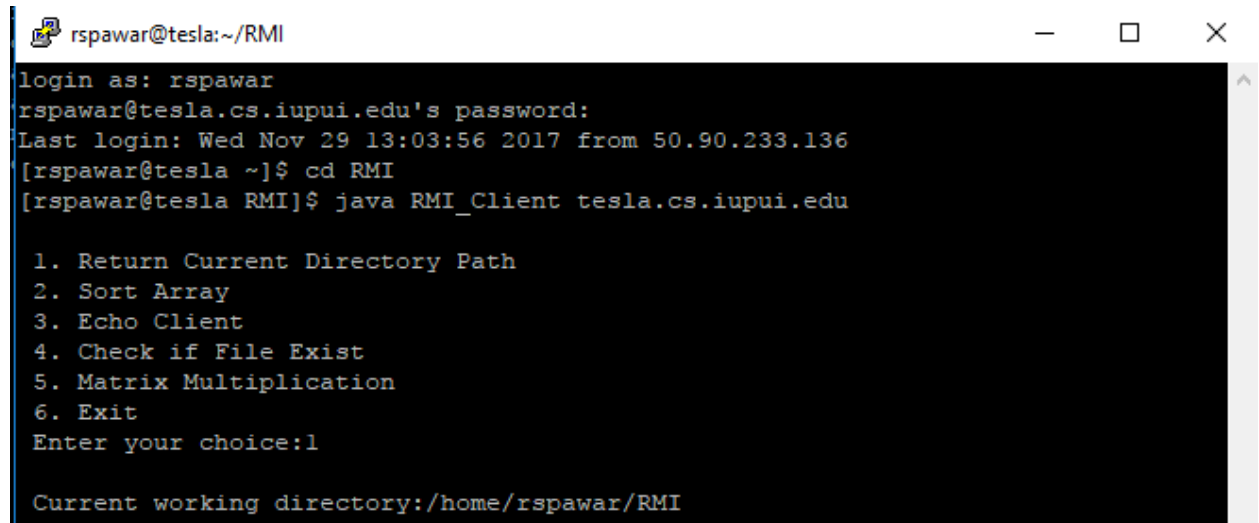
- Implementation is difficult as client must maintain track of multiple remote calls and their states to resume execution once response comes back.
- Again, it's very difficult to back track and resume execution in case of exceptions.

I have implemented required functionalities using synchronous RMI.

## Operations Performed

- 1) Path: Returns path of the current working directory to the client upon request.

### Execution Result

A terminal window titled 'rspawar@tesla:~/RMI' with standard window controls. It shows a login sequence for 'rspawar' at 'tesla.cs.iupui.edu'. After logging in, the user navigates to the 'RMI' directory and runs 'java RMI\_Client tesla.cs.iupui.edu'. The program displays a menu with six options: 1. Return Current Directory Path, 2. Sort Array, 3. Echo Client, 4. Check if File Exist, 5. Matrix Multiplication, and 6. Exit. The user enters '1'. The program then outputs 'Current working directory:/home/rspawar/RMI'.

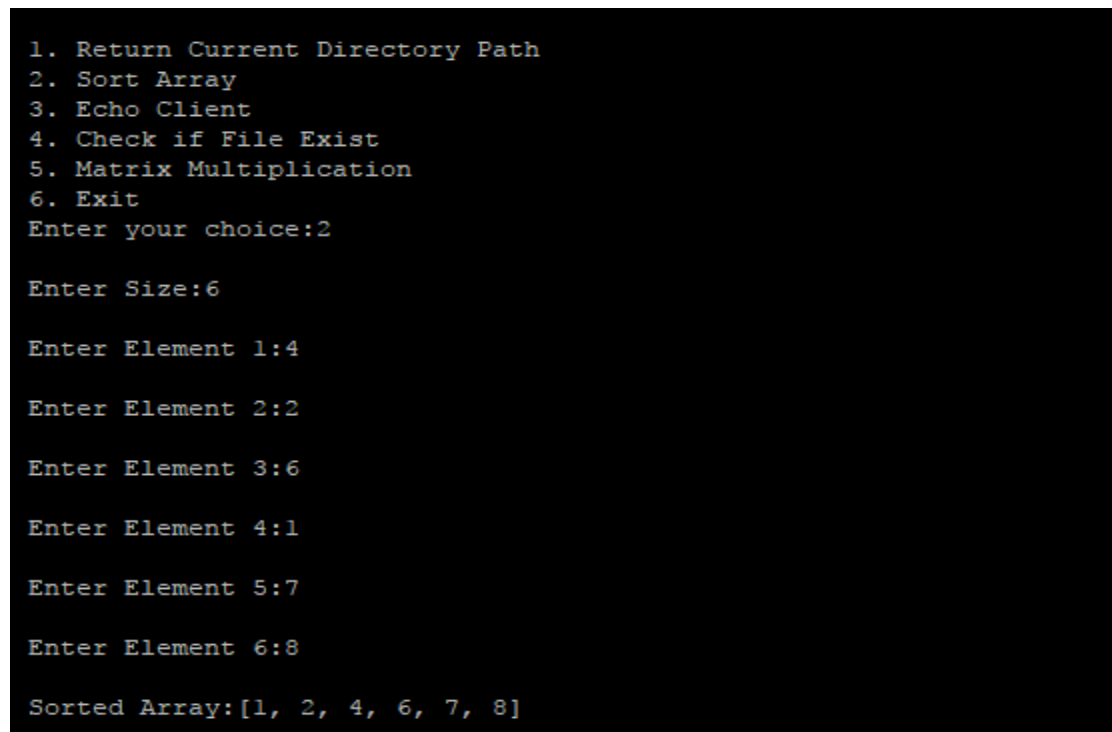
```
rspawar@tesla:~/RMI
login as: rspawar
rspawar@tesla.cs.iupui.edu's password:
Last login: Wed Nov 29 13:03:56 2017 from 50.90.233.136
[rspawar@tesla ~]$ cd RMI
[rspawar@tesla RMI]$ java RMI_Client tesla.cs.iupui.edu

1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:1

Current working directory:/home/rspawar/RMI
```

- 2) Sort: Client request to sort given input array and server sorts the array and returned sorted array to client.

### Execution Result

A terminal window showing the execution of the RMI client for the 'Sort' operation. It displays the same menu as the previous screenshot. The user enters '2'. Then, the user enters '6' for the array size. The program prompts for six elements, which the user enters as 4, 2, 6, 1, 7, and 8. Finally, the program outputs the sorted array: '[1, 2, 4, 6, 7, 8]'.

```
1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:2

Enter Size:6

Enter Element 1:4
Enter Element 2:2
Enter Element 3:6
Enter Element 4:1
Enter Element 5:7
Enter Element 6:8

Sorted Array:[1, 2, 4, 6, 7, 8]
```

- 3) Echo: Client sends some message to server and server echoes back the message to the client.

#### Execution Result

```
1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:3

Enter String:Hello Rohit!!!

Echo:Hello Rohit!!!
```

- 4) Check file exist: Client request server to check whether file exist or not at server. Server responds back the status of the file existence to the client. Server returns status as *File exist* if file is present and *File does not exist* if that file is not available at server side.

#### Execution Result

```
1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:4

Enter File Name:RMI.java

RMI.java : File Does Not Exist

1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:4

Enter File Name:RMI_Client.java

RMI_Client.java : File Exist
```

- 5) Matrix Multiplication: Clients request matrix multiplication by passing two input matrices and in response server sends multiplication of two input matrix. I have added validation to check whether matrix multiplication is possible or not based on given input matrix range.

#### Execution Result

```
1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:5

Enter number of rows for matrix1:2

Enter number of columns for matrix1:2

Matrix1[0][0]:1

Matrix1[0][1]:2

Matrix1[1][0]:3

Matrix1[1][1]:4

Enter number of rows for matrix2:2

Enter number of columns for matrix2:2

Matrix2[0][0]:1

Matrix2[0][1]:2

Matrix2[1][0]:3

Matrix2[1][1]:4

Matrix Multiplication

7      10
15     22
```

## 6) Multi-threaded/Multi-Client Support:

Two clients requesting server to perform two different operation and server can handle multiple clients' request.

### Execution Result

The image displays four terminal windows illustrating the execution of an RMI server and two clients. The top-left window shows the server setup on machine 'in-csci-rrpc01', including login, directory navigation, and registry registration. The top-right window shows the server starting on machine 'in-csci-rrpc01', displaying 'Creating a Server!', 'Binding it to name:RMI\_Server', and 'Server is Ready!'. The bottom-left window shows a client on machine 'in-csci-rrpc02' running 'java RMI\_Client 10.234.136.55', which presents a menu of operations and shows the user selecting option 1. The bottom-right window shows another client on machine 'in-csci-rrpc03' running 'java RMI\_Client 10.234.136.55', which presents the same menu, shows the user selecting option 3, and displays the output 'Echo:Hi Rohit !!!'.

```
rspawar@in-csci-rrpc01:~/RMI
login as: rspawar
rspawar@10.234.136.55's password:
Last login: Mon Nov 13 13:56:30 2017 from 149-162-234-143.dhcp-in.iupui.edu
[rspawar@in-csci-rrpc01 ~]$ cd RMI
[rspawar@in-csci-rrpc01 RMI]$ rmiregistry 1099
```

```
rspawar@in-csci-rrpc01:~/RMI
login as: rspawar
rspawar@10.234.136.55's password:
Last login: Thu Nov 30 09:51:08 2017 from 134-68-135-202.dhcp-in.iupui.edu
[rspawar@in-csci-rrpc01 ~]$ cd RMI
[rspawar@in-csci-rrpc01 RMI]$ java RMI_Server
Creating a Server!
Binding it to name:RMI_Server
Server is Ready!
```

```
rspawar@in-csci-rrpc02:~/RMI
login as: rspawar
rspawar@10.234.136.56's password:
Last login: Mon Nov 13 12:05:19 2017 from 149-162-234-143.dhcp-in.iupui.edu
[rspawar@in-csci-rrpc02 ~]$ cd RMI
[rspawar@in-csci-rrpc02 RMI]$ java RMI_Client 10.234.136.55

1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:1

Current working directory:/home/rspawar/RMI

1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:
```

```
rspawar@in-csci-rrpc03:~/RMI
login as: rspawar
rspawar@10.234.136.57's password:
Last login: Thu Nov 16 12:06:38 2017 from 149-162-252-188.dhcp-in.iupui.edu
[rspawar@in-csci-rrpc03 ~]$ cd RMI
[rspawar@in-csci-rrpc03 RMI]$ java RMI_Client 10.234.136.55

1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:3

Enter String:Hi Rohit !!!

Echo:Hi Rohit !!!

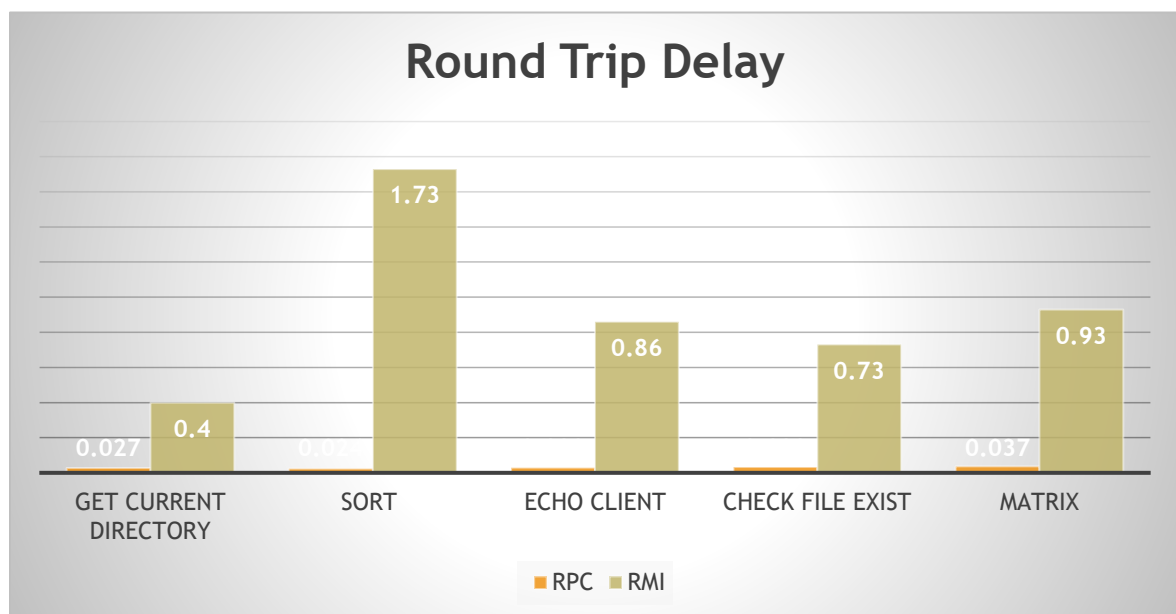
1. Return Current Directory Path
2. Sort Array
3. Echo Client
4. Check if File Exist
5. Matrix Multiplication
6. Exit
Enter your choice:
```



## Comparison of RPC and RMI

### 1) Using Round Trip Delay

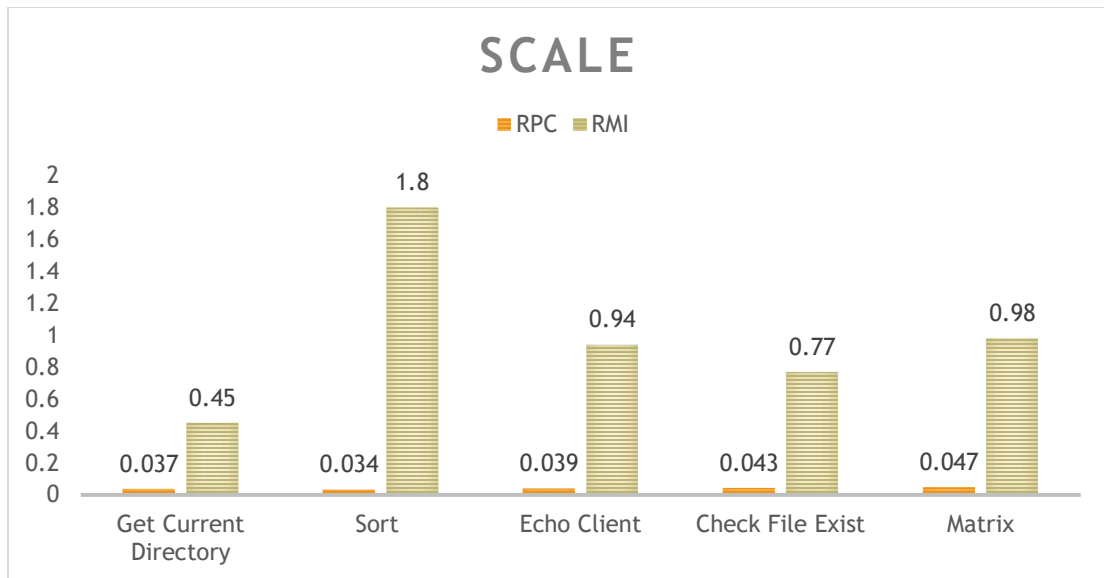
I carried out experiment on tesla machine by running the same function 15 times and calculated average of round trip delay. Graph1 shows comparison of RPC and RMI using round trip delay. For retrieving current working directory RPC took 0.027 msec whereas RMI took 0.4 msec, in order to perform sorting RPC took 0.024 msec whereas RMI took 1.73 msec and so on I carried out all other operations and its average round trip delay is shown in graph1. Hence, we can say that RPC's performance is better than RMI.



Graph 1: RPC vs RMI using Round Trip Delay (calculations are in milliseconds)

### 2) Using Scaling:

I carried out experiment on tesla machine by running the same function on 4 different machines to check impact on performance by increasing number of clients and calculated average of those. Graph2 shows comparison of RPC and RMI using scaling. For retrieving current working directory RPC took 0.037 msec whereas RMI took 0.45 msec, in order to perform sorting RPC took 0.034 msec whereas RMI took 1.8 msec and so on I carried out all other operations and its average time is shown in graph2. Here, we can see there is bit increase in time, in both RPC as well as in RMI when more number of clients are connected but still RPC is performing better. Hence, we can say that RPC's performance is better than RMI even after scaling.



Graph 2: RPC vs RMI using Scaling (calculations are in milliseconds)

### 3) Language Semantics

- RPC is C based and it follows structured programming semantics.
- RMI is java based and it follows object oriented programming semantics.

### 4) Syntax of the feature

- When you write RPC program then it's difficult because it requires lot of knowledge about structure and pointers in C and you can pass only one argument to any function.
- When you are dealing with RMI then you just write simple program and you can pass any number of arguments to methods.

### 5) Transparency

- I) Location Transparency: Both RMI and RPC provides location transparency
- II) Access Transparency:

RPC provides access transparency whereas RMI does not provide access transparency as we need specify interface as a remote interface.

### 6) Exchange Protocol

RPC and RMI both can support 3 exchange protocol i.e. Request protocol, Request Reply (RR) protocol, Request Reply Acknowledge Reply (RRA) Protocol.

## **Conclusion**

In this assignment, I have successfully implemented and used the principles of RMI (remote method invocation) for the implementation of client-server architecture. Compiled and executed the code to perform five different functionalities using RMI. In this assignment, I got to know how RMI is the extension of local procedure call as well as how request-response protocol like RMI can be used for distributed applications. In addition to RMI principle, I compared the results and different semantics of RMI with RPC and found that RPC is faster than RMI and RMI uses almost same semantics as RPC.