

Assignment No. 1

Aim: Design a distributed application consisting of a server and client using threads.

Theory:

Client-Server Model

The **client-server model** of computing is a distributed application that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more *server programs which share their resources with clients*. A client does not share any of its resources, but *requests a server's content or service function*. Clients therefore initiate communication sessions with servers which *await incoming requests*.

Socket Programming

A *socket* is one of the most fundamental technologies of computer networking. Sockets allow applications to communicate using standard mechanisms built into network hardware and operating systems. Although network software may seem to be a relatively new “Web” phenomenon, socket technology actually has been employed for roughly two decades. A socket represents a single connection between exactly two pieces of software. More than two pieces of software can communicate in *client/server* or *distributed* systems (for example, many Web browsers can simultaneously communicate with a single Web server) but multiple sockets are required to do this. Sockets are *bidirectional*, meaning that either side of the connection is capable of both sending and receiving data. Libraries implementing sockets for Internet Protocol use TCP for streams, UDP for datagrams, and IP itself for raw sockets.

Writing a Server

The server is simply going to respond to a newly connected client with the following string. “Hello. You are connected to a Simple Socket Server. What is your name?” As mentioned earlier, we will gradually increase the capability of the server in subsequent parts of the series.

Server Code Description

- We create a new object of our SocketServer class and call the *start()* method.
- We create an object of ServerSocket which is a Java class used to create socket servers.
- We ask our server to wait for client connections. The *serverSocket.accept()* blocks the thread until a connection is made. Once a connection is made, a Socket object representing the connected client will be created and assigned to the variable “client”.

Writing a Client

This client will connect to the server that we have created, and as soon as the connection is created, it will ask the server for any information it needs to send to the client. The client will then write that information to the console.

Client Code Description

- We create an object of our class and pass the hostname and port parameters
- We call the *connect* method of our client class which will establish the connection to the server
- We call a method that we have written which reads data sent from the client.
- We create a new Socket (a java class that represents a socket connection) that is used to establish connection to our server.
- We get an inputstream of the socket (representing the connection between the server and the client) and read a line of text from it.

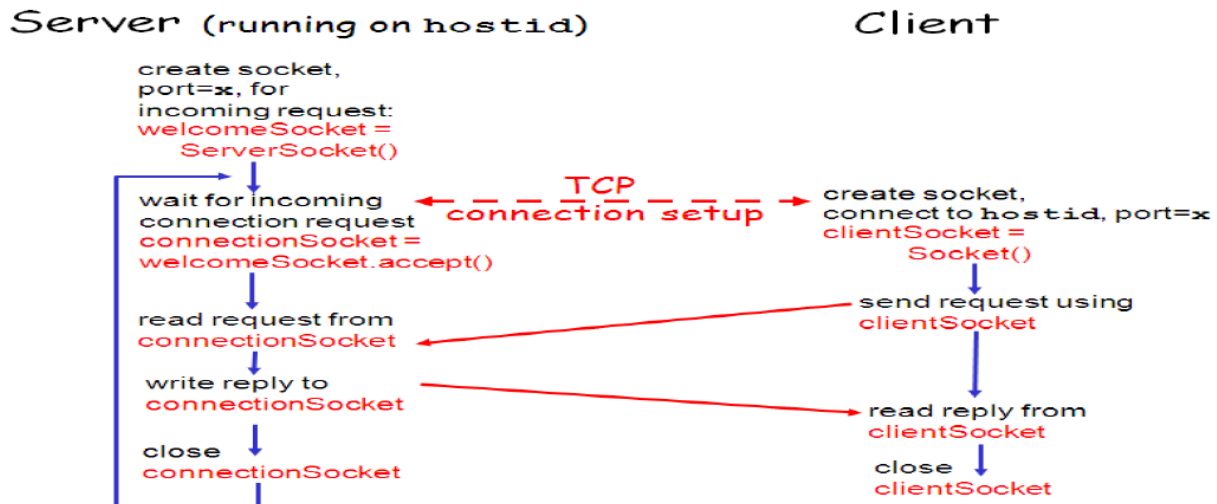


Figure: Client Server Communication

Conclusion:

Hence, we learned to implement distributed client server application using java threads.

Server.java

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String args[]) {
        int port = 6789;
        Server server = new Server( port );
        server.startServer();
    }

    // declare a server socket and a client socket for the server;
    // declare the number of connections

    ServerSocket echoServer = null;
    Socket clientSocket = null;
    int numConnections = 0;
    int port;

    public Server( int port ) {
        this.port = port;
    }

    public void stopServer() {
        System.out.println( "Server cleaning up." );
        System.exit(0);
    }

    public void startServer() {
        // Try to open a server socket on the given port
        // Note that we can't choose a port less than 1024 if we are not
        // privileged users (root)

        try {
            echoServer = new ServerSocket(port);
        }
        catch (IOException e) {
            System.out.println(e);
        }

        System.out.println( "Server is started and is waiting for connections."
);
        System.out.println( "With multi-threading, multiple connections are
allowed." );
        System.out.println( "Any client can send -1 to stop the server." );

        // Whenever a connection is received, start a new thread to process the
connection
        // and wait for the next connection.

        while ( true ) {
            try {
                clientSocket = echoServer.accept();
```

```

        numConnections++;
        ServerConnection oneconnection = new ServerConnection(clientSocket,
numConnections, this);
        new Thread(oneconnection).start();
    }
    catch (IOException e) {
        System.out.println(e);
    }
}
}
}

```

```

class ServerConnection implements Runnable {
    BufferedReader is;
    PrintStream os;
    Socket clientSocket;
    int id;
    Server server;

    public ServerConnection(Socket clientSocket, int id, Server server) {
        this.clientSocket = clientSocket;
        this.id = id;
        this.server = server;
        System.out.println( "Connection " + id + " established with: " +
clientSocket );
        try {
            is = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            os = new PrintStream(clientSocket.getOutputStream());
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    public void run() {
        String line;
        try {
            boolean serverStop = false;

            while (true) {
                line = is.readLine();
                System.out.println( "Received " + line + " from Connection " + id +
"." );
                int n = Integer.parseInt(line);
                if ( n == -1 ) {
                    serverStop = true;
                    break;
                }
                if ( n == 0 ) break;
                os.println("" + n*n );
            }
            System.out.println( "Connection " + id + " closed." );
            is.close();
            os.close();
            clientSocket.close();
            if ( serverStop ) server.stopServer();
        } catch (IOException e) {

```

```

        System.out.println(e);
    }
}

```

Client.java

```

import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {

        String hostname = "192.168.1.35";
        int port = 6789;

        // declaration section:
        // clientSocket: our client socket
        // os: output stream
        // is: input stream

        Socket clientSocket = null;
        DataOutputStream os = null;
        BufferedReader is = null;

        // Initialization section:
        // Try to open a socket on the given port
        // Try to open input and output streams

        try {
            clientSocket = new Socket(hostname, port);
            os = new DataOutputStream(clientSocket.getOutputStream());
            is = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: " + hostname);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to: " +
hostname);
        }

        // If everything has been initialized then we want to write some data
        // to the socket we have opened a connection to on the given port

        if (clientSocket == null || os == null || is == null) {
            System.err.println( "Something is wrong. One variable is null." );
            return;
        }

        try {
            while ( true ) {
                System.out.print( "Enter an integer (0 to stop connection, -1 to
stop server): " );
                BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
                String keyboardInput = br.readLine();
                os.writeBytes( keyboardInput + "\n" );
            }
        }
    }
}

```

```

        int n = Integer.parseInt( keyboardInput );
        if ( n == 0 || n == -1 ) {
            break;
        }

        String responseLine = is.readLine();
        System.out.println("Server returns its square as: " + responseLine);
    }

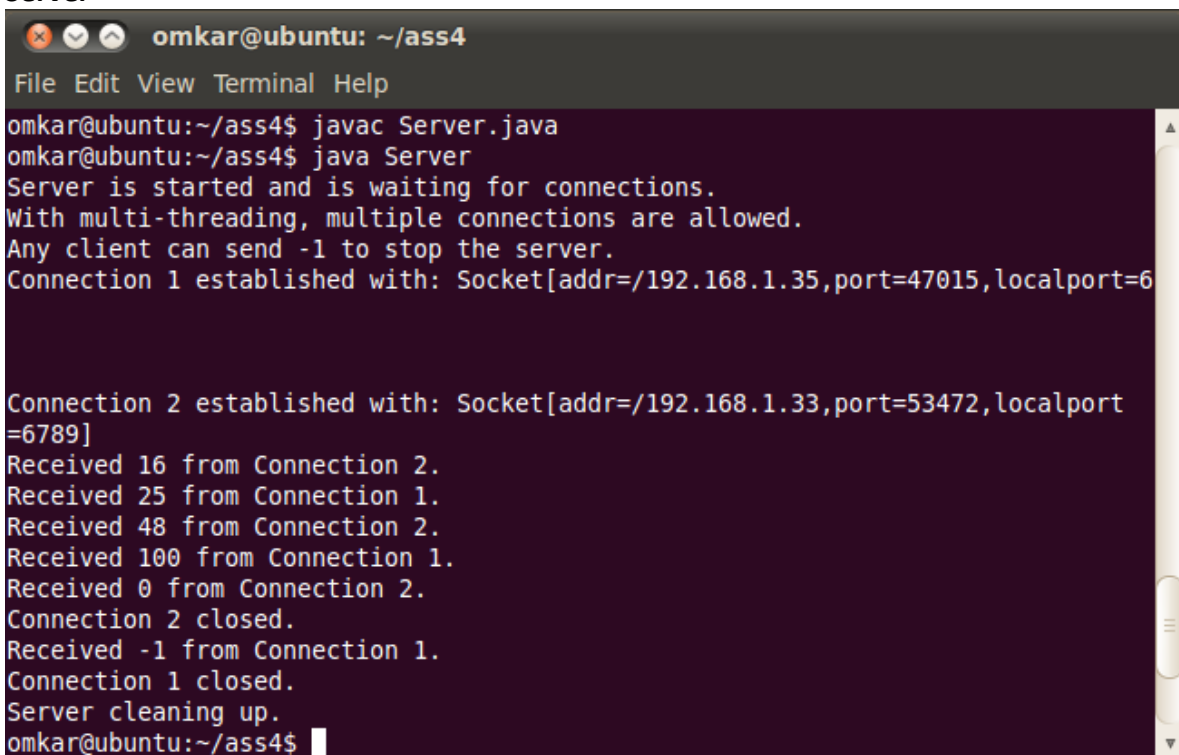
    // clean up:
    // close the output stream
    // close the input stream
    // close the socket

    os.close();
    is.close();
    clientSocket.close();
} catch (UnknownHostException e) {
    System.err.println("Trying to connect to unknown host: " + e);
} catch (IOException e) {
    System.err.println("IOException:  " + e);
}
}
}

```

Output :

Server



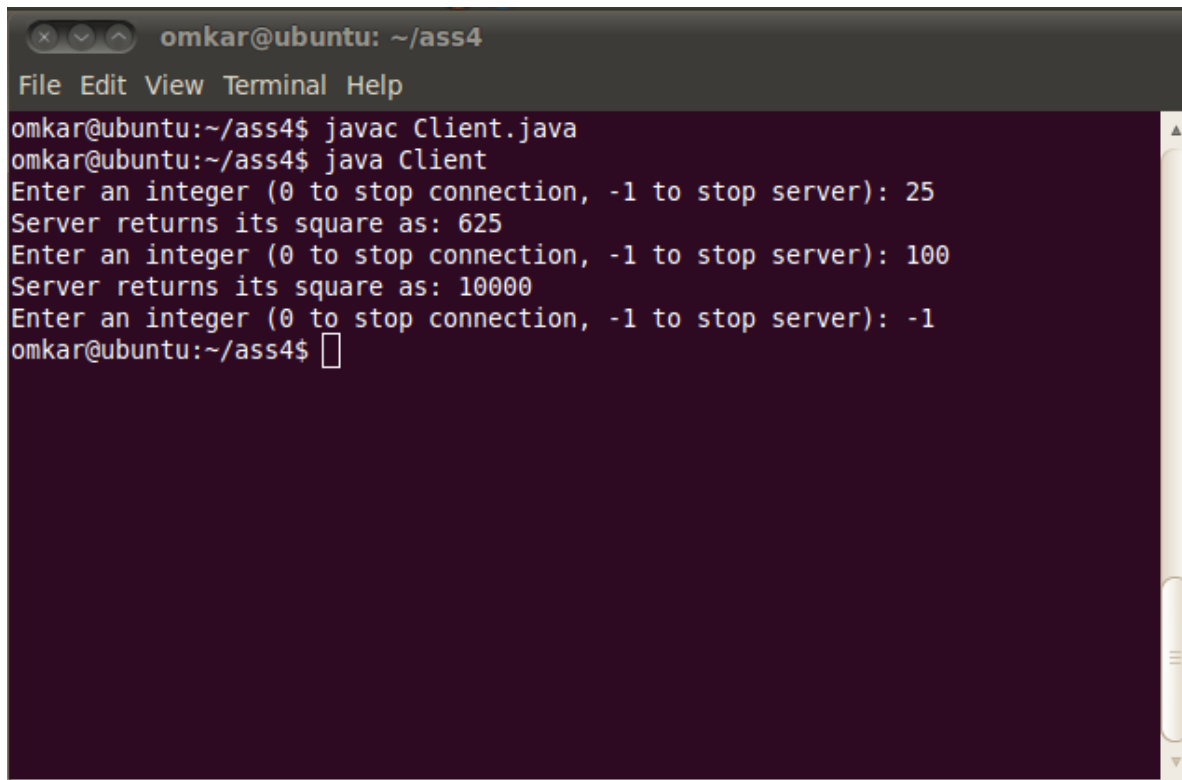
```

omkar@ubuntu:~/ass4$ javac Server.java
omkar@ubuntu:~/ass4$ java Server
Server is started and is waiting for connections.
With multi-threading, multiple connections are allowed.
Any client can send -1 to stop the server.
Connection 1 established with: Socket[addr=/192.168.1.35,port=47015,localport=6

Connection 2 established with: Socket[addr=/192.168.1.33,port=53472,localport
=6789]
Received 16 from Connection 2.
Received 25 from Connection 1.
Received 48 from Connection 2.
Received 100 from Connection 1.
Received 0 from Connection 2.
Connection 2 closed.
Received -1 from Connection 1.
Connection 1 closed.
Server cleaning up.
omkar@ubuntu:~/ass4$

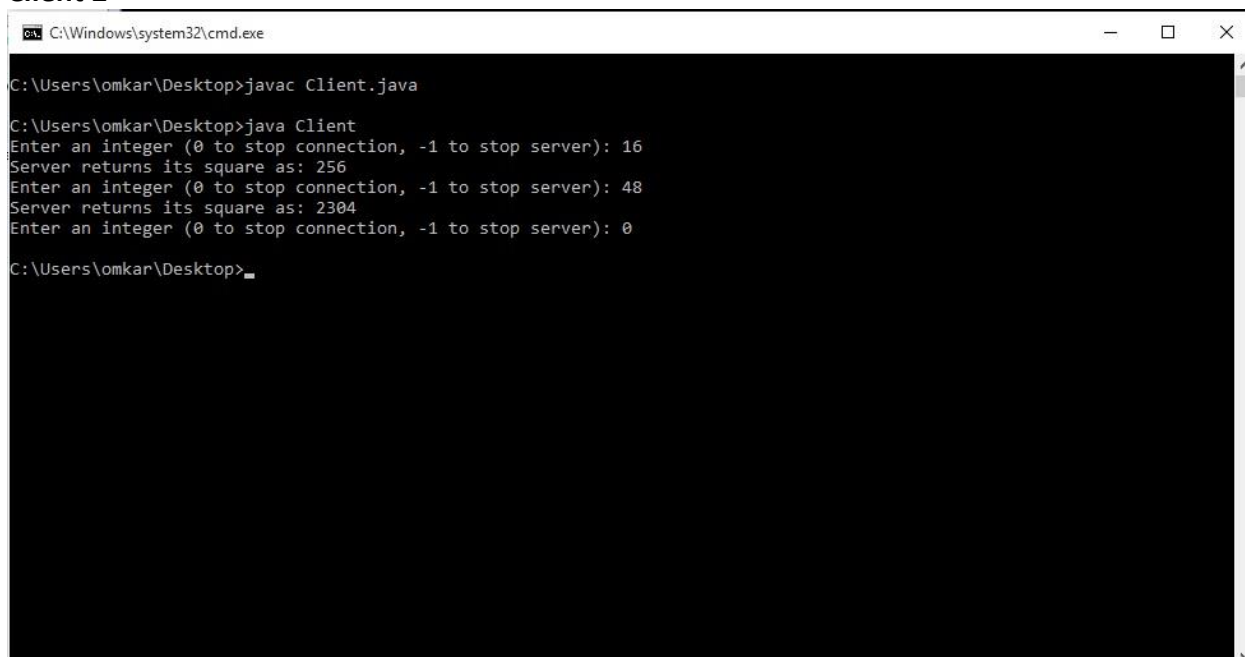
```

Client 1

A terminal window titled 'omkar@ubuntu: ~/ass4' with a menu bar (File, Edit, View, Terminal, Help). The prompt is 'omkar@ubuntu:~/ass4\$'. The user enters 'javac Client.java' and 'java Client'. The program prompts for an integer, and the user enters 25, 100, and -1. The program outputs the square of the input: 625 and 10000.

```
omkar@ubuntu:~/ass4$ javac Client.java
omkar@ubuntu:~/ass4$ java Client
Enter an integer (0 to stop connection, -1 to stop server): 25
Server returns its square as: 625
Enter an integer (0 to stop connection, -1 to stop server): 100
Server returns its square as: 10000
Enter an integer (0 to stop connection, -1 to stop server): -1
omkar@ubuntu:~/ass4$
```

Client 2

A Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The prompt is 'C:\Users\omkar\Desktop>'. The user enters 'javac Client.java' and 'java Client'. The program prompts for an integer, and the user enters 16, 48, and 0. The program outputs the square of the input: 256 and 2304.

```
C:\Users\omkar\Desktop> javac Client.java
C:\Users\omkar\Desktop> java Client
Enter an integer (0 to stop connection, -1 to stop server): 16
Server returns its square as: 256
Enter an integer (0 to stop connection, -1 to stop server): 48
Server returns its square as: 2304
Enter an integer (0 to stop connection, -1 to stop server): 0
C:\Users\omkar\Desktop>
```

Assignment No. 2

Remote Procedure Calls (RPC)

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.

Introduction

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or 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. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

RPC makes the client/server model of computing more powerful and easier to program. When combined with the ONC RPCGEN protocol compiler clients transparently make remote calls through a local procedure interface.

The RPC Model

The remote procedure call model is similar to that of the local model, which works as follows:

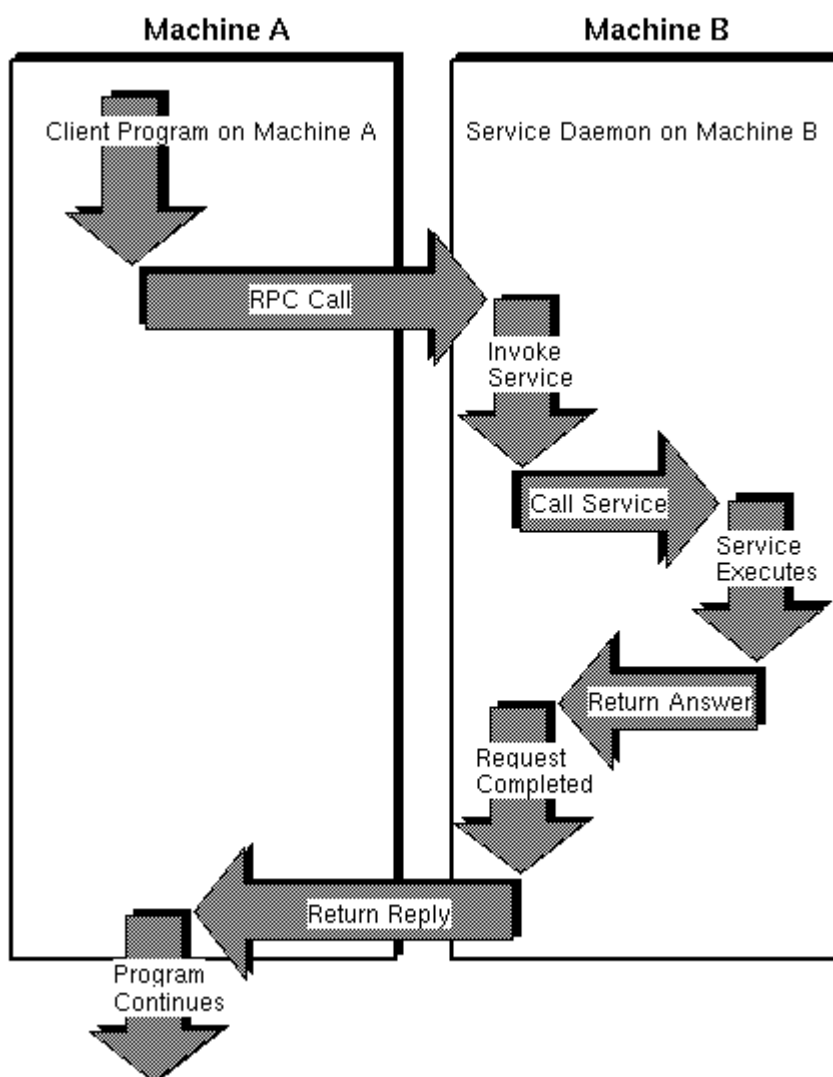
1. The caller places arguments to a procedure in a specific location (such as a result register).
2. The caller temporarily transfers control to the procedure.
3. When the caller gains control again, it obtains the results of the procedure from the specified location.
4. The caller then continues program execution.

The remote procedure call is similar, in that one thread of control logically winds through two processes -- that of the caller and that of the server:

1. The caller process sends a call message to the server process and blocks (that is, waits) for a reply message. The **call message** contains the parameters of the procedure and the **reply message** contains the procedure results.
2. When the caller receives the reply message, it gets the results of the procedure.
3. The caller process then continues executing.

On the server side, a process is dormant -- awaiting the arrival of a call message. When one arrives, the server process computes a reply that it then sends back to the requesting client. After this, the server process becomes dormant again. [Figure 1](#) illustrates this basic form of network communication with the remote procedure call.

Figure 1: Basic Network Communication with Remote Procedure Call



ZK-0475U-R

This figure shows a synchronous RPC call, in which only one of the two processes is active at a given time. The remote procedure call hides the details of the network transport. However, the RPC protocol does not restrict the concurrency model. For example, RPC calls may be asynchronous so that the client can do another task while waiting for the reply from the server. Another possibility is that the server could create a task to process a certain type of request

automatically, freeing it to service other requests. Although RPC provides a way to avoid programming the underlying network transport, it still allows this where necessary.

RPC Procedure Versions

Each RPC procedure is uniquely defined by program and procedure numbers. The **program number** specifies a group of related remote procedures, each of which has a different **procedure number**. Each program also has a version number so that, when a minor change is made to a remote service (adding a new procedure, for example), a new program number does not have to be assigned. When you want to call a procedure to find the number of remote users, you must know the appropriate program, version, and procedure numbers to use to contact the service. This information can be found in several sources. For example, the `/etc/rpc` file lists some RPC programs and the `rpcinfo` command lists the registered RPC programs and corresponding version numbers running on a particular system.

Typically, a service provides a protocol description so that you can write client applications that call the service. The RPC Administrator at Sun Microsystems, Inc. has a list of programs that have been registered with them (that is, have received port numbers from them) but you can write your own local RPC programs. Knowing the program and procedure numbers is useful only if the program is running on a system that you have access to.

Using portmap to Determine the Destination Port Number of RPC Packets

The `portmap` network service command starts automatically when a machine is booted. As part of its initialization, a server program calls its host portmap to create a portmap entry for its program and version number. To find the port of a remote program, a client sends an RPC call message to a server portmap. If the remote program is registered with the portmap, it returns the relevant port number in an RPC reply message. The client program can then send RPC call message packets to that remote program port.

The `portmap` network service has a well-known (dedicated) port. Other network service port numbers can be assigned statically or dynamically when they register their ports with the portmap of their host. Refer to the `portmap(8)` reference page for more information about the port mapping service.

RPC Application Development

Consider an example:

A client/server lookup in a personal database on a remote machine. Assuming that we cannot access the database from the local machine (via NFS).

We use UNIX to run a remote shell and execute the command this way. There are some problems with this method:

- the command may be slow to execute.
- You require an login account on the remote machine.

The RPC alternative is to

- establish an server on the remote machine that can repond to queries.
- Retrieve information by calling a query which will be quicker than previous approach.

To develop an RPC application the following steps are needed:

- Specify the protocol for client server communication
- Develop the client program
- Develop the server program

The programs will be compiled seperately. The communication protocol is achieved by generated stubs and these stubs and rpc (and other libraries) will need to be linked in.

Defining the Protocol

The easiest way to define and generate the protocol is to use a protocol complier such as `rpcgen`. For the protocol you must identify the name of the service procedures, and data types of parameters and return arguments. The protocol compiler reads a definition and automatically generates client and server stubs.

`rpcgen` uses its own language (RPC language or RPCL) which looks very similar to preprocessor directives. `rpcgen` exists as a standalone executable compiler that reads special files denoted by a `.x` prefix.

So to compile a RPCL file you simply do

```
rpcgen rpcprog.x
```

This will generate possibly four files:

- `rpcprog_clnt.c` -- the client stub
- `rpcprog_svc.c` -- the server stub
- `rpcprog_xdr.c` -- If necessary XDR (external data representation) filters
- `rpcprog.h` -- the header file needed for any XDR filters.

The `rpcgen` protocol compiler helps to reduce development time in the following ways:

- It greatly reduces network interface programming.
- It can mix low-level code with high-level code.
- For speed-critical applications, you can link customized high-level code with the `rpcgen` output.
- You can use `rpcgen` output as a starting point, and rewrite as necessary.

The external data representation (XDR) is an data abstraction needed for machine independent communication. The client and server need not be machines of the same type.

p1.x

```
struct input {
    int n;
};

struct output {
    int r;
};

program p1 {
    version p {
        struct output factorial(struct input i) = 1;
        }=1;
    }=22222222;
```

p1_client.c

//The Client Program

```
#include "p1.h"

void p1_1(char *host)
{
    CLIENT *clnt;
    struct output *result_1;
    struct input factorial_1_arg;
    factorial_1_arg.n=5;
#ifdef DEBUG
    clnt = clnt_create (host, p1, p, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = factorial_1(&factorial_1_arg, clnt);
    if (result_1 == (struct output *) NULL) {
        clnt_perror (clnt, "call failed");
    }
}
```

```

    }
    else
    {
        printf("Client      :recieved    %d    as    factorial    output    from
server",result_1->r);
    }
#endifdef    DEBUG
    clnt_destroy (clnt);
#endif    /* DEBUG */
}

int main (int argc, char *argv[])
{
    char *host;
    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    p1_1 (host);
    exit (0);
}

```

p1 clnt.c

//The client stub

```

#include <memory.h> /* for memset */
#include "p1.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

struct output *factorial_1(struct input *argp, CLIENT *clnt)
{
    static struct output clnt_res;
    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, factorial,
        (xdrproc_t) xdr_input, (caddr_t) argp,
        (xdrproc_t) xdr_output, (caddr_t) &clnt_res,
        TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}

```

p1 server.c

//The server program

```

#include "p1.h"

struct output *factorial_1_svc(struct input *argp, struct svc_req *rqstp)
{
    static struct output result;

```

```

    int a=argp->n,b=1;
    printf("Server : recieved %d as input from client",argp->n);
    while(a!=0)
    {
        b=b*a;
        a--;
    }
    result.r=b;
    return &result;
}

```

p1_svc.c

//The Server Stub

```

#include "p1.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifdef SIG_PF
#define SIG_PF void(*) (int)
#endif

static void p1_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
    union {
        struct input factorial_1_arg;
    } argument;
    char *result;
    xdrproc_t _xdr_argument, _xdr_result;
    char *(*local)(char *, struct svc_req *);
    switch (rqstp->rq_proc) {
    case NULLPROC:
        (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
        return;

    case factorial:
        _xdr_argument = (xdrproc_t) xdr_input;
        _xdr_result = (xdrproc_t) xdr_output;
        local = (char *(*)(char *, struct svc_req *)) factorial_1_svc;
        break;

    default:
        svcerr_noproc (transp);
        return;
    }
    memset ((char *)&argument, 0, sizeof (argument));
}

```

```

        if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t)
&argument)) {
            svcerr_decode (transp);
            return;
        }
        result = (*local)((char *)&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, (xdrproc_t) _xdr_result,
result)) {
            svcerr_systemerr (transp);
        }
        if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t)
&argument)) {
            fprintf (stderr, "%s", "unable to free arguments");
            exit (1);
        }
        return;
    }

int main (int argc, char **argv)
{
    register SVCXPRT *transp;

    pmap_unset (p1, p);

    transp = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL) {
        fprintf (stderr, "%s", "cannot create udp service.");
        exit(1);
    }
    if (!svc_register(transp, p1, p, p1_1, IPPROTO_UDP)) {
        fprintf (stderr, "%s", "unable to register (p1, p, udp).");
        exit(1);
    }

    transp = svctcp_create(RPC_ANYSOCK, 0, 0);
    if (transp == NULL) {
        fprintf (stderr, "%s", "cannot create tcp service.");
        exit(1);
    }
    if (!svc_register(transp, p1, p, p1_1, IPPROTO_TCP)) {
        fprintf (stderr, "%s", "unable to register (p1, p, tcp).");
        exit(1);
    }

    svc_run ();
    fprintf (stderr, "%s", "svc_run returned");
    exit (1);
    /* NOTREACHED */
}

```

p1 xdr.c

```

#include "p1.h"
bool_t

```

```

xdr_input (XDR *xdrs, input *objp)
{
    register int32_t *buf;

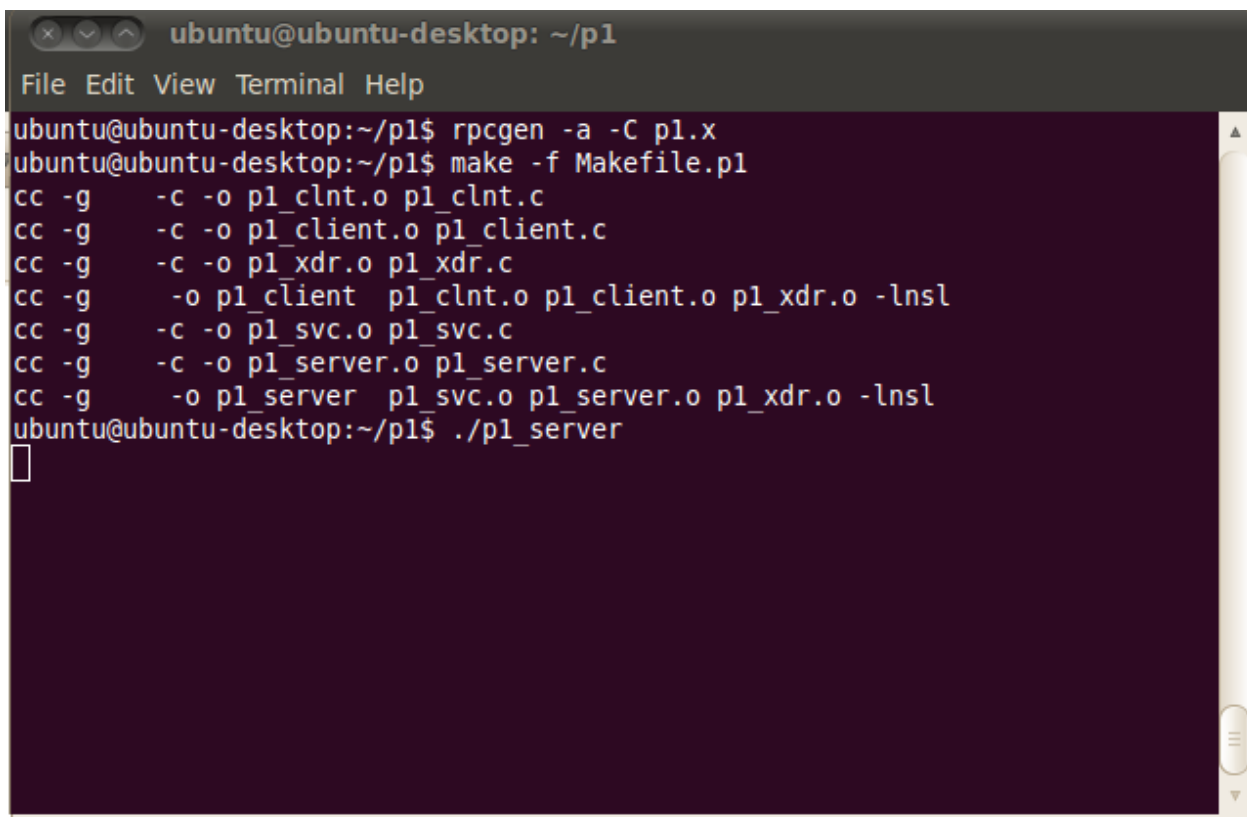
    if (!xdr_int (xdrs, &objp->n))
        return FALSE;
    return TRUE;
}
bool_t
xdr_output (XDR *xdrs, output *objp)
{
    register int32_t *buf;

    if (!xdr_int (xdrs, &objp->r))
        return FALSE;
    return TRUE;
}

```

Output :

Server :



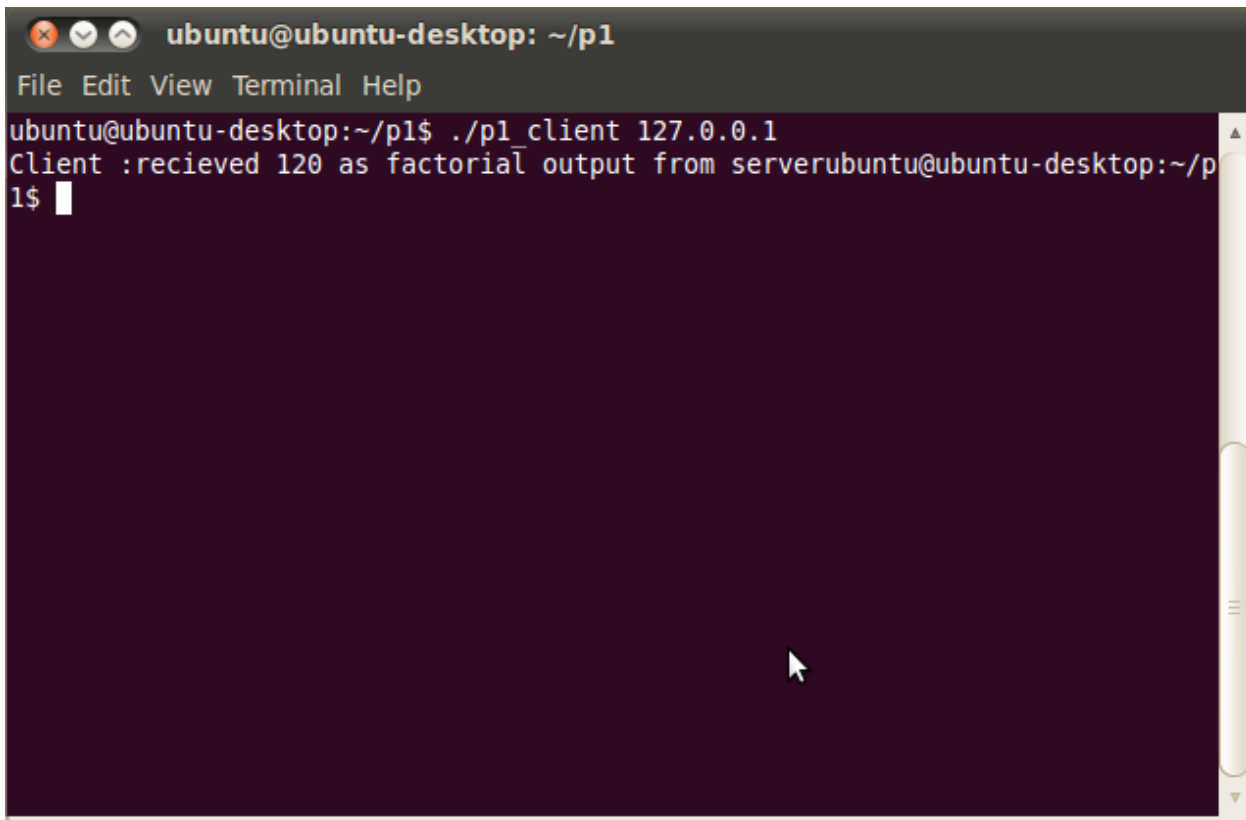
A terminal window titled 'ubuntu@ubuntu-desktop: ~/p1' with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the following commands and output:

```

ubuntu@ubuntu-desktop:~/p1$ rpcgen -a -C p1.x
ubuntu@ubuntu-desktop:~/p1$ make -f Makefile.p1
cc -g -c -o p1_clnt.o p1_clnt.c
cc -g -c -o p1_client.o p1_client.c
cc -g -c -o p1_xdr.o p1_xdr.c
cc -g -o p1_client p1_clnt.o p1_client.o p1_xdr.o -lnsl
cc -g -c -o p1_svc.o p1_svc.c
cc -g -c -o p1_server.o p1_server.c
cc -g -o p1_server p1_svc.o p1_server.o p1_xdr.o -lnsl
ubuntu@ubuntu-desktop:~/p1$ ./p1_server

```

Client :

A terminal window titled 'ubuntu@ubuntu-desktop: ~/p1' with a menu bar (File, Edit, View, Terminal, Help). The prompt is 'ubuntu@ubuntu-desktop:~/p1\$'. The command './p1_client 127.0.0.1' has been executed. The output is 'Client :recieved 120 as factorial output from serverubuntu@ubuntu-desktop:~/p1\$'. The prompt is now '1\$' with a cursor.

```
ubuntu@ubuntu-desktop: ~/p1
File Edit View Terminal Help
ubuntu@ubuntu-desktop:~/p1$ ./p1_client 127.0.0.1
Client :recieved 120 as factorial output from serverubuntu@ubuntu-desktop:~/p1$
1$
```

Assignment No. 3

RMI (Remote Method Invocation)

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.

Introduction

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.

Overview

Remote Method Invocation (RMI) facilitates object function calls between Java Virtual Machines (JVMs). JVMs can be located on separate computers - yet one JVM can invoke methods belonging to an object stored in another JVM. Methods can even pass objects that a foreign virtual machine has never encountered before, allowing dynamic loading of new classes as required. This is a powerful feature!

Consider the follow scenario :

- Developer A writes a service that performs some useful function. He regularly updates this service, adding new features and improving existing ones.
- Developer B wishes to use the service provided by Developer A. However, it's inconvenient for A to supply B with an update every time.

Java RMI provides a very easy solution! Since RMI can dynamically load new classes, Developer B can let RMI handle updates automatically for him. Developer A places the new classes in a web directory, where RMI can fetch the new updates as they are required.

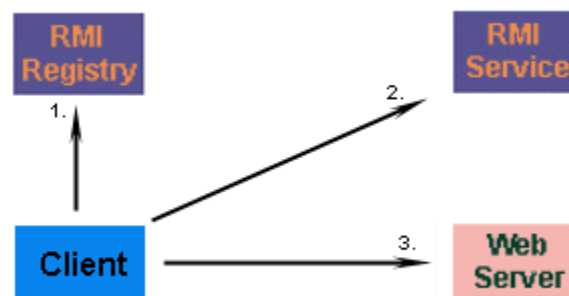


Figure 1 - Connections made when client uses RMI

Figure 1 shows the connections made by the client when using RMI. Firstly, the client must contact an RMI registry, and request the name of the service. Developer B won't know the exact location of the RMI service, but he knows enough to contact Developer A's registry. This will point him in the direction of the service he wants to call..

Developer A's service changes regularly, so Developer B doesn't have a copy of the class. Not to worry, because the client automatically fetches the new subclass from a webserver where the two developers share classes. The new class is loaded into memory, and the client is ready to use the new class. This happens transparently for Developer B - no extra code need to be written to fetch the class.

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:

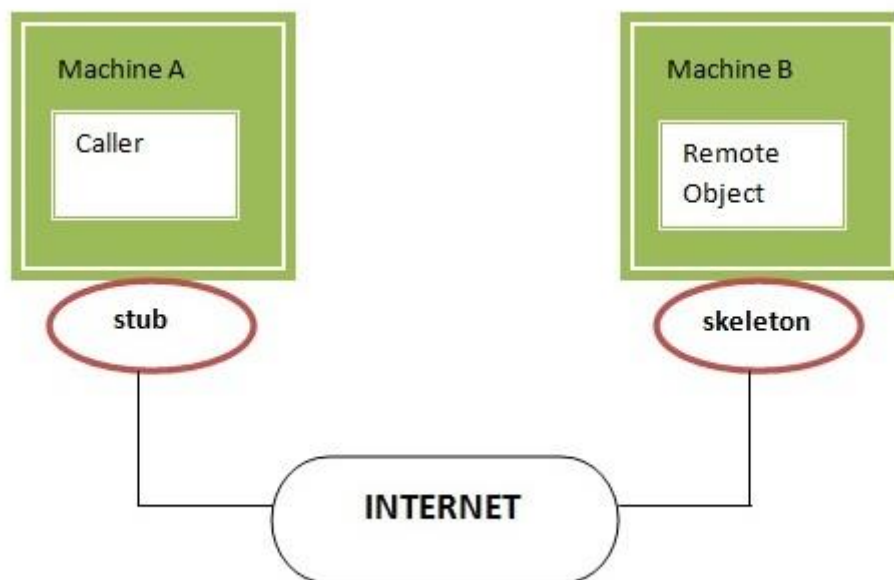
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

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.

Steps to write the RMI program

Following are 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

Client.java

```
import java.rmi.*;
import java.util.Scanner;
public class Client {
    public static void main(String args[])
    {
        try
        {
            Scanner s = new Scanner(System.in);
            System.out.println("Enter the Server address : ");
            String server = s.nextLine();
            ServerInterface si = (ServerInterface)Naming.lookup("rmi://" + server + "/Server");
            System.out.println("Enter first string : ");
            String first = s.nextLine();
            System.out.println("Enter second string : ");
            String second = s.nextLine();
            System.out.println("Concatenated String : " + si.concat(first, second));
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Servant.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
```

```
import java.rmi.server.*;
public class Servant extends UnicastRemoteObject implements ServerInterface {
    @Override
    public String concat(String a,String b) throws RemoteException{
        return a+b;
    }
}
```

Server.java

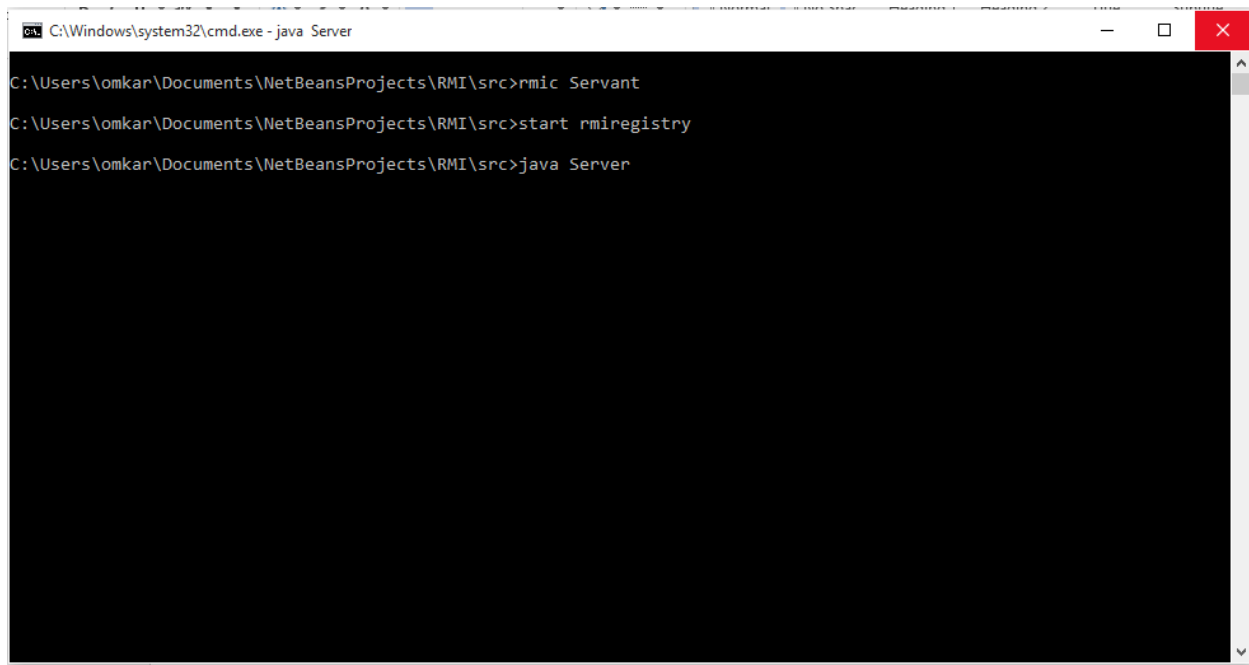
```
import java.rmi.*;
import java.net.*;
public class Server {
    public static void main(String[] args) {
        try {
            Servant s = new Servant();
            Naming.rebind("Server", s);
        }
        catch(Exception e) {
            System.out.println(e);
        } } }
```

ServerInterface.java

```
import java.rmi.*;
public interface ServerInterface extends Remote {
    String concat(String a,String b) throws RemoteException;
}
```

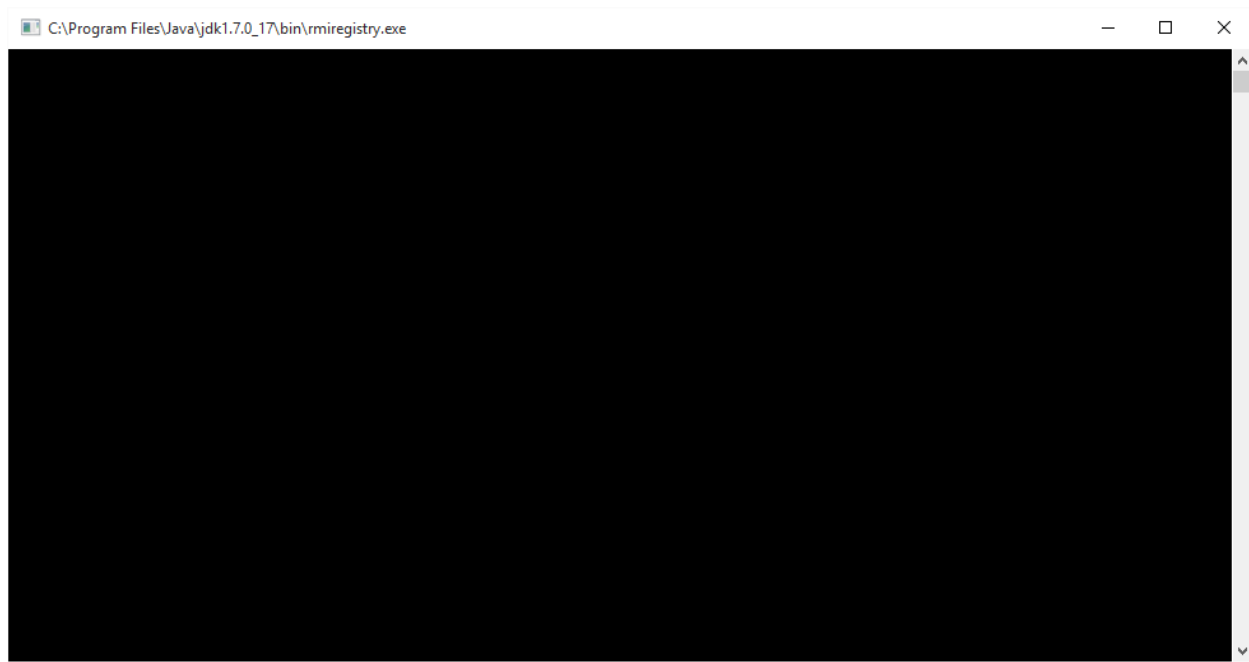
Output :

Server :

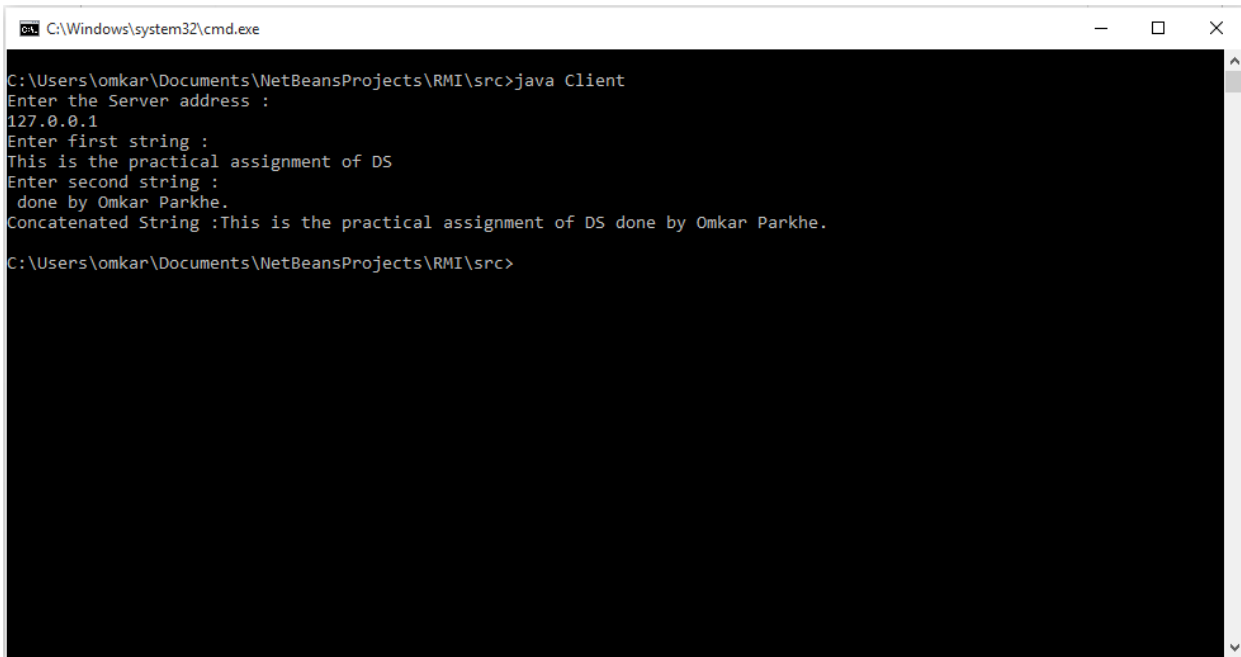


```
C:\Windows\system32\cmd.exe - java Server
C:\Users\omkar\Documents\NetBeansProjects\RMI\src>rmic Servant
C:\Users\omkar\Documents\NetBeansProjects\RMI\src>start rmiregistry
C:\Users\omkar\Documents\NetBeansProjects\RMI\src>java Server
```

RMI Registry :



Client :

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt is running a Java program 'Client' from the directory 'C:\Users\omkar\Documents\NetBeansProjects\RMI\src'. The program prompts for a server address, which is entered as '127.0.0.1'. It then prompts for a first string, which is entered as 'This is the practical assignment of DS'. Next, it prompts for a second string, which is entered as 'done by Omkar Parkhe.'. Finally, it displays the concatenated string: 'This is the practical assignment of DS done by Omkar Parkhe.'. The prompt then returns to the command line.

```
C:\Windows\system32\cmd.exe
C:\Users\omkar\Documents\NetBeansProjects\RMI\src>java Client
Enter the Server address :
127.0.0.1
Enter first string :
This is the practical assignment of DS
Enter second string :
done by Omkar Parkhe.
Concatenated String :This is the practical assignment of DS done by Omkar Parkhe.
C:\Users\omkar\Documents\NetBeansProjects\RMI\src>
```

Assignment No. 4

Message Passing Interface (MPI)

Aim: Design a distributed application using Message Passing Interface (MPI) for remote computation where client submits a string to the server and server returns the reverse of it to the client.

Theory :

Introduction

Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in different computer programming languages such as Fortran, C, C++ and Java. There are several well-tested and efficient implementations of MPI, including some that are free or in the public domain. These fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications.

Overview and Goals

MPI (Message-Passing Interface) is a message-passing library interface specification. All parts of this definition are significant. MPI addresses primarily the message-passing parallel programming model, in which data is moved from the address space of one process to that of another process

through cooperative operations on each process. Extensions to the "classical" message-passing model are provided in collective operations, remote-memory access operations, dynamic process creation, and parallel I/O. MPI is a specification, not an implementation; there are multiple implementations of MPI. This specification is for a library interface; MPI is not a language, and all MPI operations are expressed as functions, subroutines, or methods, according to the appropriate language bindings which, for C and

Fortran, are part of the MPI standard. The standard has been defined through an open process by a community of parallel computing vendors, computer scientists, and application developers. The next few sections provide an overview of the history of MPI's development. The main advantages of establishing a message-passing standard are portability and ease of use. In a distributed memory communication environment in which the higher level routines and/or abstractions are built upon lower level message-passing routines the benefits of standardization are particularly apparent. Furthermore, the definition of a message passing standard, such as that proposed here, provides vendors with a clearly defined base set of routines that they can implement efficiently, or in some cases for which they can provide hardware support, thereby enhancing scalability.

The goal of the Message-Passing Interface simply stated is to develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing.

A complete list of goals follows :

- Design an application programming interface (not necessarily for compilers or a system implementation library).
- Allow efficient communication: Avoid memory-to-memory copying, allow overlap of computation and communication, and offload to communication co-processors, where available.
- Allow for implementations that can be used in a heterogeneous environment.
- Allow convenient C and Fortran bindings for the interface.
- Assume a reliable communication interface: the user need not cope with communication failures. Such failures are dealt with by the underlying communication subsystem.
- Define an interface that can be implemented on many vendor's platforms, with no significant changes in the underlying communication and system software.
- Semantics of the interface should be language independent.
- The interface should be designed to allow for thread safety.

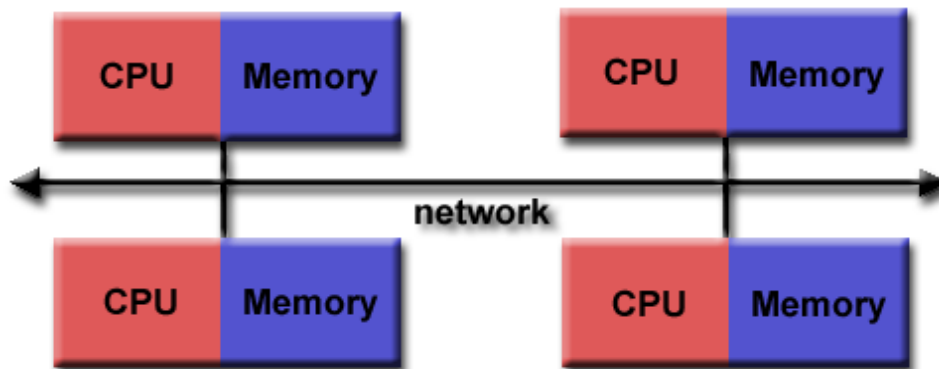
An Interface Specification:

- M P I = Message Passing Interface
- MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.
- MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.
- Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be:
 - Practical
 - Portable

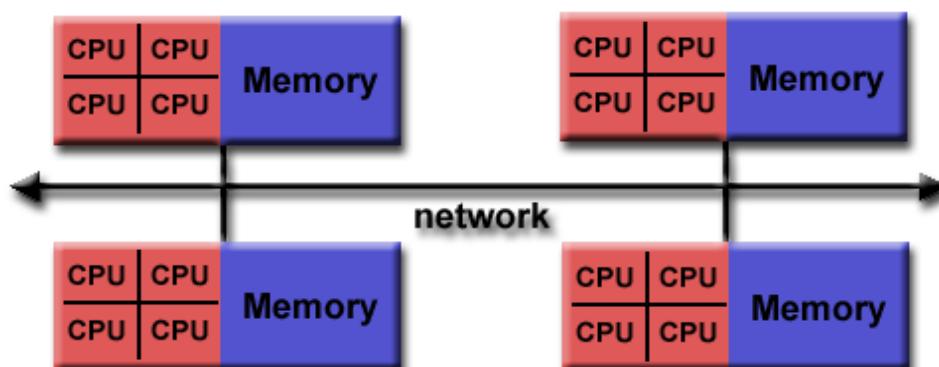
- Efficient
 - Flexible
- The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.
- Interface specifications have been defined for C and Fortran90 language bindings:
 - C++ bindings from MPI-1 are removed in MPI-3
 - MPI-3 also provides support for Fortran 2003 and 2008 features
- Actual MPI library implementations differ in which version and features of the MPI standard they support. Developers/users will need to be aware of this.

Programming Model:

- Originally, MPI was designed for distributed memory architectures, which were becoming increasingly popular at that time (1980s - early 1990s).



- As architecture trends changed, shared memory SMPs were combined over networks creating hybrid distributed memory / shared memory systems.
- MPI implementors adapted their libraries to handle both types of underlying memory architectures seamlessly. They also adapted/developed ways of handling different interconnects and protocols.



- Today, MPI runs on virtually any hardware platform:
 - Distributed Memory
 - Shared Memory
 - Hybrid
- The programming model clearly remains a distributed memory model however, regardless of the underlying physical architecture of the machine.

- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

Concepts

MPI provides a rich range of abilities. The following concepts help in understanding and providing context for all of those abilities and help the programmer to decide what functionality to use in their application programs. Four of MPI's eight basic concepts are unique to MPI-2.

Communicator

Communicator objects connect groups of processes in the MPI session. Each communicator gives each contained process an independent identifier and arranges its contained processes in an ordered topology. MPI also has explicit groups, but these are mainly good for organizing and reorganizing groups of processes before another communicator is made. MPI understands single group intracommunicator operations, and bilateral intercommunicator communication. In MPI-1, single group operations are most prevalent. Bilateral operations mostly appear in MPI-2 where they include collective communication and dynamic in-process management.

Communicators can be partitioned using several MPI commands. These commands include `MPI_COMM_SPLIT`, where each process joins one of several colored sub-communicators by declaring itself to have that color.

Point-to-point basics

A number of important MPI functions involve communication between two specific processes. A popular example is `MPI_Send`, which allows one specified process to send a message to a second specified process. Point-to-point operations, as these are called, are particularly useful in patterned or irregular communication, for example, a data-parallel architecture in which each processor routinely swaps regions of data with specific other processors between calculation steps, or a master-slave architecture in which the master sends new task data to a slave whenever the prior task is completed.

MPI-1 specifies mechanisms for both blocking and non-blocking point-to-point communication mechanisms, as well as the so-called 'ready-send' mechanism whereby a send request can be made only when the matching receive request has already been made.

Collective basics

Collective functions involve communication among all processes in a process group (which can mean the entire process pool or a program-defined subset). A typical function is the `MPI_Bcast` call (short for "broadcast"). This function takes data from one node and sends it to

all processes in the process group. A reverse operation is the MPI_Reduce call, which takes data from all processes in a group, performs an operation (such as summing), and stores the results on one node. MPI_Reduce is often useful at the start or end of a large distributed calculation, where each processor operates on a part of the data and then combines it into a result.

Other operations perform more sophisticated tasks, such as MPI_Alltoall which rearranges n items of data such that the nth node gets the nth item of data from each.

Basic MPI types

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_SHORT	signed short
MPI_UNSIGNED_SHORT	unsigned short
MPI_INT	signed int
MPI_UNSIGNED	unsigned int
MPI_LONG	signed long
MPI_UNSIGNED_LONG	unsigned long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Basic MPI Operations

MPI_Comm	the basic object used by MPI to determine which processes are involved in a communication
MPI_Status	the MPI_Recv operation takes the address of an MPI_Status structure as an argument (which can be ignored with MPI_STATUS_IGNORE).
MPI_Init	Initialize the MPI execution environment int MPI_Init(int *argc, char ***argv)
MPI_Comm_size	Determines the size of the group associated with a communicator int MPI_Comm_size(MPI_Comm comm, int *size)
MPI_Open_port	MPI_Open_port establishes a network address, encoded in the <i>port_name</i> string, at which the server will be able to accept connections from clients. <i>port_name</i> is supplied by the system. MPI copies a system-supplied port name

	into <i>port_name</i> . <i>port_name</i> identifies the newly opened port and can be used by a client to contact the server. The maximum size string that may be supplied by the system is MPI_MAX_PORT_NAME.
MPI_Comm_accept	MPI_Comm_accept establishes communication with a client. It is collective over the calling communicator. It returns an intercommunicator that allows communication with the client, after the client has connected with the MPI_Comm_accept function using the MPI_Comm_connect function.
MPI_Send	MPI_Send performs a standard-mode, blocking send.
MPI_Recv	This basic receive operation, MPI_Recv, is blocking: it returns only after the receive buffer contains the newly received message. A receive can complete before the matching send has completed (of course, it can complete only after the matching send has started).
MPI_Comm_free	This operation marks the communicator object for deallocation. The handle is set to MPI_COMM_NULL. Any pending operations that use this communicator will complete normally; the object is actually deallocated only if there are no other active references to it. This call applies to intracommunicators and intercommunicators.
MPI_Close_port	MPI_Close_port releases the network address represented by <i>port_name</i> .
MPI_Finalize	This routine cleans up all MPI states. Once this routine is called, no MPI routine (not even MPI_Init) may be called, except for MPI_Get_version , MPI_Initialized , and MPI_Finalized . Unless there has been a call to MPI_Abort , you must ensure that all pending communications involving a process are complete before the process calls MPI_Finalize. If the call returns, each process may either continue local computations or exit without participating in further communication with other processes.
MPI_Abort	This routine makes a "best attempt" to abort all tasks in the group of comm. This function does not require that the invoking environment take any action with the error code. However, a UNIX or POSIX environment should handle this as a return errorcode from the main program or an abort (errorcode)
MPI_Comm_disconnect	MPI_Comm_disconnect waits for all pending communication on <i>comm</i> to complete internally, deallocates the communicator object, and sets the handle to MPI_COMM_NULL. It is a collective operation.

Reasons for Using MPI:

- Standardization - MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.
- Portability - There is little or no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.
- Performance Opportunities - Vendor implementations should be able to exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.
- Functionality - There are over 430 routines defined in MPI-3, which includes the majority of those in MPI-2 and MPI-1.
- Availability - A variety of implementations are available, both vendor and public domain.

Server.c

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
#include <string.h>
int main(int argc, char **argv)
{
    MPI_Comm client;
    MPI_Status status;
    char port_name[MPI_MAX_PORT_NAME],str[50],ch,temp;
    int size, again, i,j;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size != 1) {
        fprintf(stderr, "Server too big");
        exit(EXIT_FAILURE);
    }
    MPI_Open_port(MPI_INFO_NULL, port_name);
    printf("Server available at port: %s\n", port_name);
    i=0;
    while (1) {
```

```

MPI_Comm_accept(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD, &client);
again = 1;
while (again) {
MPI_Recv(&ch, 1, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, client, &status);
switch (status.MPI_TAG) {
case 0:
MPI_Comm_free(&client);
MPI_Close_port(port_name);
MPI_Finalize();
return 0;
case 1:
printf("\nReceived String: %s\n",str);
// reverse the string
i = 0;
j = strlen(str) - 1;
while (i < j) {
temp = str[i];
str[i] = str[j];
str[j] = temp;
i++;
j--;
}
printf("\nReversed string is : %s\n",str);
// send the reversed string to client (character by character)
for (i = 0; i < strlen(str); i++) {
ch=str[i];
MPI_Send(&ch, 1, MPI_CHAR, 0, 2, client);
}
//send tag=1 to indicate end of string
MPI_Send(&ch, 1, MPI_CHAR, 0, 1, client);
MPI_Comm_disconnect(&client);
again = 0;
strcpy(str,"");
i=0;
break;
case 2:
printf("Received character: %c\n", ch);
str[i]=ch;
i++;
// add null character at the end of string
str[i]='\0';
break;
default:
/* Unexpected message type */
MPI_Abort(MPI_COMM_WORLD, 1);
}
}
}
}
}

```

Client.c

```
#include <stdlib.h>
```

```

#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main( int argc, char **argv )
{
    MPI_Comm server;
    MPI_Status status;
    char port_name[MPI_MAX_PORT_NAME],str[50],ch;
    int i, tag,again;
    if (argc < 2) {
        fprintf(stderr, "server port name required.\n");
        exit(EXIT_FAILURE);
    }
    MPI_Init(&argc, &argv);
    strcpy(port_name, argv[1]);
    MPI_Comm_connect(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD, &server);
    // accept input string
    printf("\nEnter the string :\n");
    scanf("%s",str);
    //send string to server (character by character)
    for (i = 0; i < strlen(str); i++) {
        if(str[i]!='\0')
            ch=str[i];
        tag=2;
        MPI_Send(&ch, 1, MPI_CHAR, 0, tag, server);
    }
    // done sending string to the server
    MPI_Send(&i, 0, MPI_INT, 0, 1, server);
    // Receive the reversed string from server and display it
    i=0;
    again=1;
    while (again) {
        MPI_Recv(&ch, 1, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, server, &status);
        switch (status.MPI_TAG) {
            case 2:
                str[i]=ch;
                i++;
                break;
            case 1: again=0;
                break;
        }
    }
    printf("\nReversed string is : %s\n\n",str);
    MPI_Comm_disconnect(&server);
    MPI_Finalize();
    return 0;
}

```

Output :

1. **Server :**


```
ubuntu@ubuntu-desktop: ~/mpi
File Edit View Terminal Help
ubuntu@ubuntu-desktop:~/mpi$ mpicc server.c -o server
ubuntu@ubuntu-desktop:~/mpi$ mpicc client.c -o client
ubuntu@ubuntu-desktop:~/mpi$ mpirun -np 1 ./server
Server available at port: 2643394560.0;tcp://192.168.129.129:52858+2643394561
.0;tcp://192.168.129.129:56223:300
Received character: 0
Received character: m
Received character: k
Received character: a
Received character: r

Received String: Omkar
Reversed string is : rakm0
█
```

2. Client :

```
ubuntu@ubuntu-desktop: ~/mpi
File Edit View Terminal Help
ubuntu@ubuntu-desktop:~/mpi$ mpirun -np 1 ./client '2643394560.0;tcp://192.16
8.129.129:52858+2643394561.0;tcp://192.168.129.129:56223:300'

Enter the string :
Omkar

Reversed string is : rakm0
ubuntu@ubuntu-desktop:~/mpi$ █
```

Assignment No. 5

Aim : Design 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.

Theory :

What is Cluster Computing ?

A collection of computers configured in such a way that they can be used to solve a problem by means of parallel processing.

Homogeneous Cluster

The cluster in which every single node is exactly same, from the motherboard and the memory, to the disk drives and the NIC .

Heterogeneous Cluster

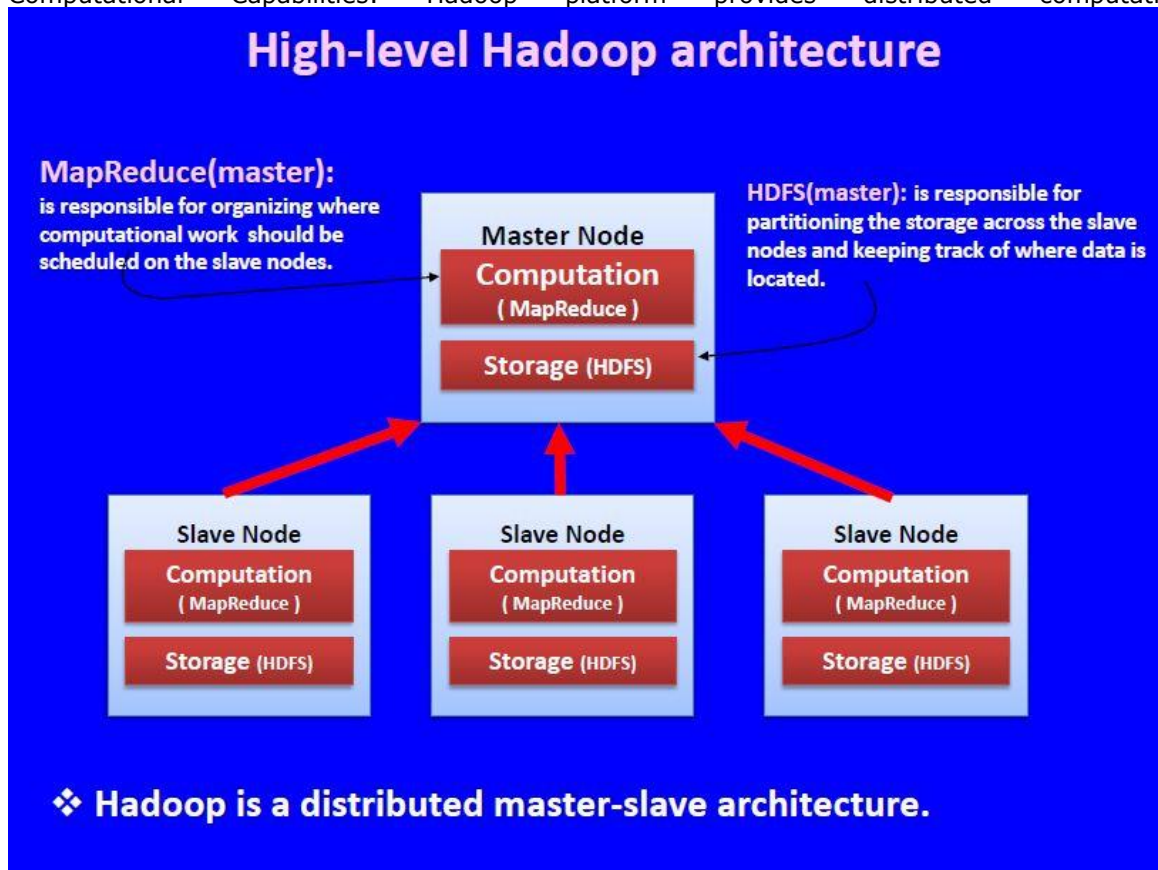
Made from different kinds of computers, SPARC, DEC ALPHA

What is Hadoop ?

Software Library: Hadoop is a software library for distributed computing.

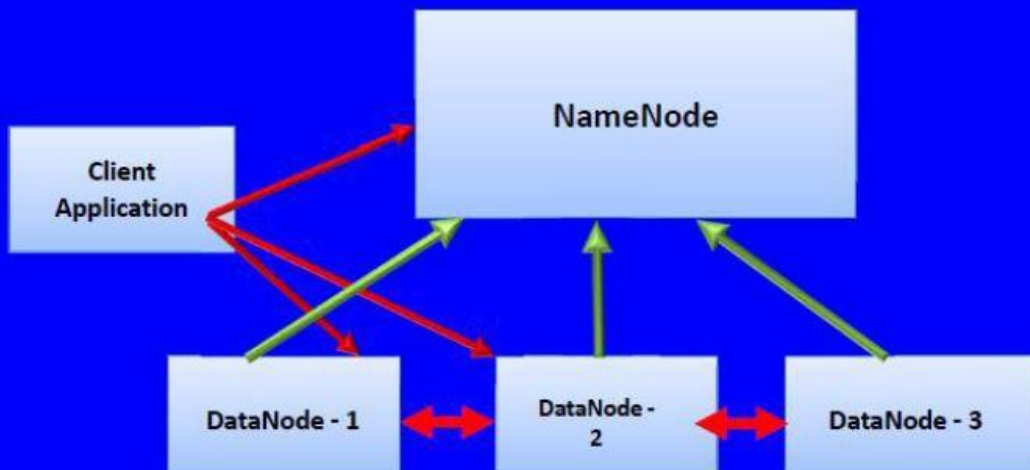
Distributed Storage: Hadoop platform provides distributed storage.

Computational Capabilities: Hadoop platform provides distributed computational capabilities.



HDFS architecture

- Consider an HDFS client communicating with the master NameNode and slave DataNodes.



▪ Master-Slave Architecture

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

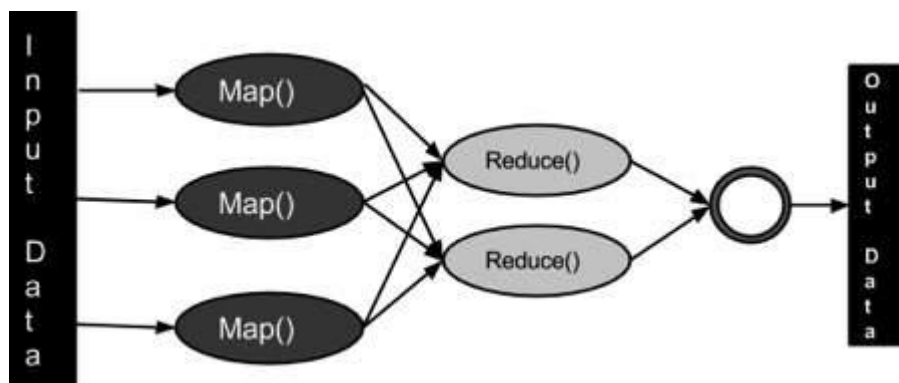
The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
 - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is

passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

- **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



How to Install and Configure Hadoop on Linux

What is Hadoop?

Hadoop is a framework (consisting of software libraries) which simplifies the processing of data sets distributed across clusters of servers. Two of the main components of Hadoop are **HDFS** and **MapReduce**.

HDFS is the filesystem that is used by Hadoop to store all the data on. This file system spans across all the nodes that are being used by Hadoop. These nodes could be on a single VPS or they can be spread across a large number of virtual servers.

MapReduce is the framework that orchestrates all of Hadoop's activities. It handles the assignment of work to different nodes in the cluster.

Benefits of using Hadoop

The architecture of Hadoop allows you to scale your hardware as and when you need to. New nodes can be added incrementally without having to worry about

the change in data formats or the handling of applications that sit on the file system.

One of the most important features of Hadoop is that it allows you to save enormous amounts of money by substituting cheap commodity servers for expensive ones. This is possible because Hadoop transfers the responsibility of fault tolerance from the hardware layer to the application layer.

Installing Hadoop

Installing and getting Hadoop up and running is quite straightforward. However, since this process requires editing multiple configuration and setup files, make sure that each step is properly followed.

1. Install Java

Hadoop requires Java to be installed, so let's begin by installing Java:

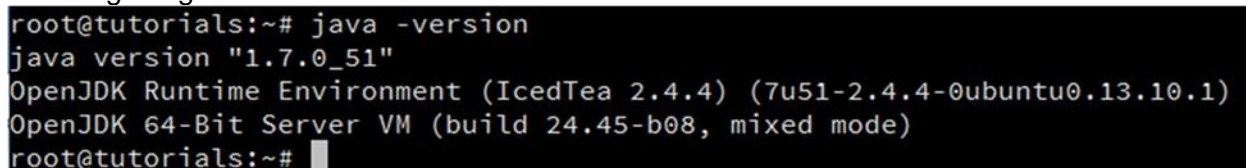
```
apt-get update
```

```
apt-get install default-jdk
```

These commands will update the package information on your VPS and then install Java. After executing these commands, execute the following command to verify that Java has been installed:

```
java -version
```

If Java has been installed, this should display the version details as illustrated in the following image:

A terminal window with a black background and white text. The prompt is 'root@tutorials:~#'. The user enters 'java -version'. The output is: 'java version "1.7.0_51"', 'OpenJDK Runtime Environment (IcedTea 2.4.4) (7u51-2.4.4-0ubuntu0.13.10.1)', and 'OpenJDK 64-Bit Server VM (build 24.45-b08, mixed mode)'. The prompt returns to 'root@tutorials:~#'.

```
root@tutorials:~# java -version
java version "1.7.0_51"
OpenJDK Runtime Environment (IcedTea 2.4.4) (7u51-2.4.4-0ubuntu0.13.10.1)
OpenJDK 64-Bit Server VM (build 24.45-b08, mixed mode)
root@tutorials:~#
```

2. Create and Setup SSH Certificates

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands:

```
ssh-keygen -t rsa -P ""
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

After executing the first of these two commands, you might be asked for a filename. Just leave it blank and press the enter key to continue. The second command adds the newly created key to the list of authorized keys so that Hadoop can use SSH without prompting for a password.

```

root@tutorials:~# ssh-keygen -t rsa -P ''
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
23:42:08:de:26:bc:6e:4c:3e:3e:55:f7:a6:4d:8f:bb root@tutorials
The key's randomart image is:
+--[ RSA 2048 ]-----+
| .
| o...
| +.o.
| +. . .
| o o o S
| = . . . =
| *.      = o
| o..      . o .
| ..      Eo
+-----+
root@tutorials:~# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
root@tutorials:~# █

```

Fetch and Install Hadoop

First let's fetch Hadoop from one of the mirrors using the following command:

```
wget http://www.motorlogy.com/apache/hadoop/common/current/hadoop-2.3.0.tar.gz
```

Note: This command uses a download a link on one of the mirrors listed on the Hadoop website. The list of mirrors can be found [on this link](#). You can choose any other mirror if you want to. To download the latest stable version, choose the `*hadoop-X.Y.Z.tar.gz*` file from the **current** or the **current2** directory on your chosen mirror.*

After downloading the Hadoop package, execute the following command to extract it:

```
tar xzf hadoop-2.3.0.tar.gz
```

This command will extract all the files in this package in a directory named `hadoop-2.3.0`. For this tutorial, the Hadoop installation will be moved to the `/usr/local/hadoop` directory using the following command:

```
mv hadoop-2.3.0 /usr/local/hadoop
```

Note: The name of the extracted folder depends on the Hadoop version you have downloaded and extracted. If your version differs from the one used in this tutorial, change the above command accordingly.

3. Edit and Setup Configuration Files

To complete the setup of Hadoop, the following files will have to be modified:

- `~/bashrc`
- `/usr/local/hadoop/etc/hadoop/hadoop-env.sh`
- `/usr/local/hadoop/etc/hadoop/core-site.xml`
- `/usr/local/hadoop/etc/hadoop/yarn-site.xml`
- `/usr/local/hadoop/etc/hadoop/mapred-site.xml.template`
- `/usr/local/hadoop/etc/hadoop/hdfs-site.xml`

i. Editing `~/bashrc`

Before editing the `.bashrc` file in your home directory, we need to find the path where Java has been installed to set the `JAVA_HOME` environment variable.

Let's use the following command to do that:

`update-alternatives --config java`

This will display something like the following:

```
root@tutorials:~# update-alternatives --config java
There is only one alternative in link group java (providing /usr/bin/java): /usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java
Nothing to configure.
root@tutorials:~#
```

The complete path displayed by this command is:

`/usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java`

The value for `JAVA_HOME` is everything before `/jre/bin/java` in the above path - in this case,

`/usr/lib/jvm/java-7-openjdk-amd64`.

Make a note of this as we'll be using this value in this step and in one other step.

Now use `nano` (or your favored editor) to edit `~/bashrc` using the following command:

`nano ~/bashrc`

This will open the `.bashrc` file in a text editor. Go to the end of the file and paste/type the following content in it:

`#HADOOP VARIABLES START`

```
export          JAVA_HOME=/usr/lib/jvm/java-7-
openjdk-amd64          export
```

```
HADOOP_INSTALL=/usr/local/hadoop
```

```
export
```

```
PATH=$PATH:$HADOOP_INSTALL/bin
```

```
export
```

```
PATH=$PATH:$HADOOP_INSTALL/sbin
```

```
export
```

```
HADOOP_MAPRED_HOME=$HADOOP_INSTALL
```



```

export
HADOOP_COMMON_HOME=$HADOOP_INSTALL
export
HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL

export
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export                                     HADOOP_OPTS="-
Djava.library.path=$HADOOP_INSTALL/lib"

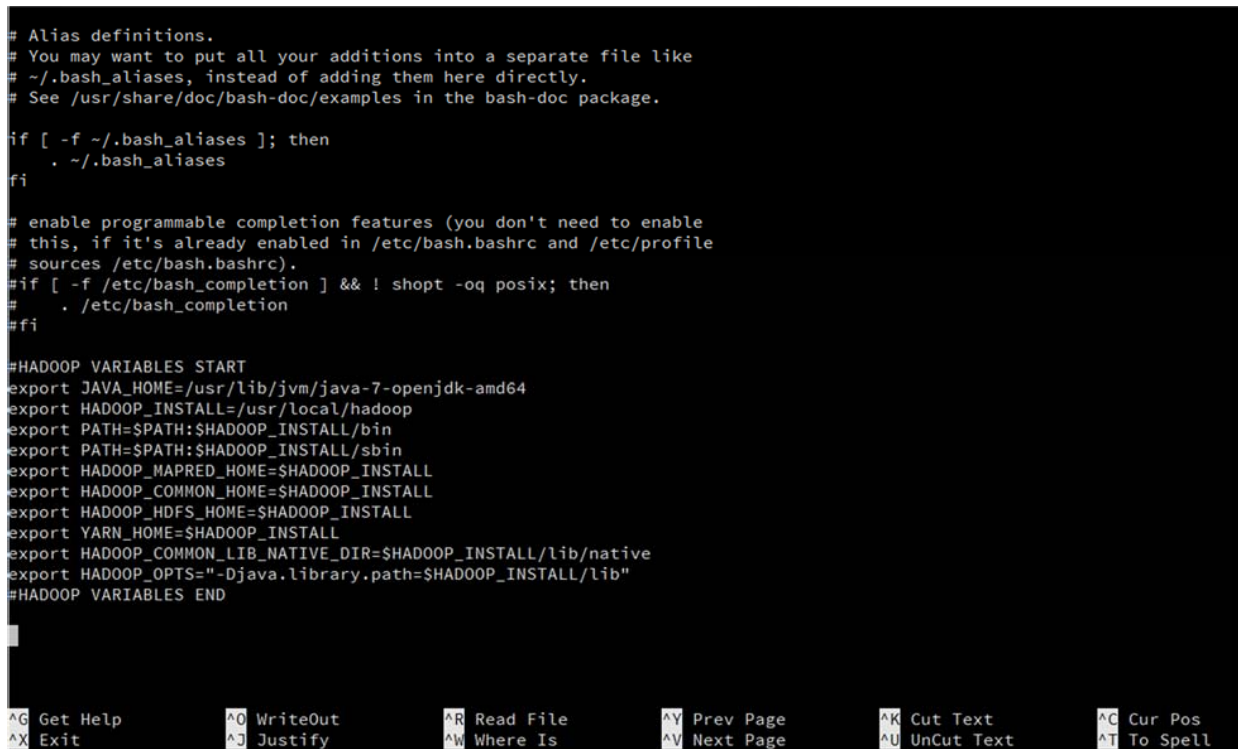
#HADOOP VARIABLES END

```

Note 1: If the value of `JAVA_HOME` is different on your VPS, make sure to alter the first export statement in the above content accordingly.

Note 2: Files opened and edited using `nano` can be saved using `Ctrl + X`. Upon the prompt to save changes, type `Y`. If you are asked for a filename, just press the enter key.

The end of the `.bashrc` file should look something like this:



```

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

```

After saving and closing the `.bashrc` file, execute the following command so that your system recognizes the newly created environment variables:

`source ~/.bashrc`

Putting the above content in the `.bashrc` file ensures that these variables are always available when your VPS starts up.

ii.Editing /usr/local/hadoop/etc/hadoop/hadoop-env.sh

Open the /usr/local/hadoop/etc/hadoop/hadoop-env.sh file with nano using the following command:

```
nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

In this file, locate the line that exports the JAVA_HOME variable. Change this line to the following:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Note: If the value of JAVA_HOME is different on your VPS, make sure to alter this line accordingly.

The hadoop-env.sh file should look something like this:

```
# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME. All others are
# optional. When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
# export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Save and close this file. Adding the above statement in the hadoop-env.sh file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

iii.Editing /usr/local/hadoop/etc/hadoop/core-site.xml

The /usr/local/hadoop/etc/hadoop/core-site.xml file contains configuration properties that Hadoop uses when starting up. This file can be used to override the default settings that Hadoop starts with.

Open this file with nano using the following command:

```
nano /usr/local/hadoop/etc/hadoop/core-site.xml
```

In this file, enter the following content in between the <configuration></configuration> tag:

```
<property>
  <name>fs.default.name</name>

  <value>hdfs://localhost:9000</value>
</property>
```

The core-site.xml file should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Save and close this file.

iv. Editing /usr/local/hadoop/etc/hadoop/yarn-site.xml

The /usr/local/hadoop/etc/hadoop/yarn-site.xml file contains configuration properties that MapReduce uses when starting up. This file can be used to override the default settings that MapReduce starts with.

Open this file with nano using the following command:

```
nano
/usr/local/hadoop/etc/hadoop/yarn-site.xml
```

In this file, enter the following content in between the <configuration></configuration> tag:

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>

<property>
  <name>yarn.nodemanager.aux-
```

```
services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

The yarn-site.xml file should look something like this:

```
<?xml version="1.0"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<configuration>

    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>

</configuration>
```

Save and close this file.

v. Creating and Editing /usr/local/hadoop/etc/hadoop/mapred-site.xml

By default, the /usr/local/hadoop/etc/hadoop/ folder contains the /usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml. This file is used to specify which framework is being used for MapReduce.

This can be done using the following command:

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

Once this is done, open the newly created file with nano using the following command:

```
nano /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

In this file, enter the following content in between the <configuration></configuration> tag:

```
<property>
    <name>mapreduce.framework.name</name>

    <value>yarn</value>
</property>
```

The mapred-site.xml file should look something like this:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Save and close this file.

vi. Editing /usr/local/hadoop/etc/hadoop/hdfs-site.xml

The /usr/local/hadoop/etc/hadoop/hdfs-site.xml has to be configured for each host in the cluster that is being used. It is used to specify the directories which will be used as the **namenode** and the **datanode** on that host.

Before editing this file, we need to create two directories which will contain the **namenode** and the **datanode** for this Hadoop installation. This can be done using the following commands:

```
mkdir -p /usr/local/hadoop_store/hdfs/namenode mkdir -p
/usr/local/hadoop_store/hdfs/datanode
```

Note: You can create these directories in different locations, but make sure to modify the contents of hdfs-site.xml accordingly.

Once this is done, open the `/usr/local/hadoop/etc/hadoop/hdfs-site.xml` file with nano using the following command:
`nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml`

In this file, enter the following content in between the `<configuration></configuration>` tag:

```
<property>
  <name>dfs.replication</name>

  <value>1</value>

</property>

<property>
  <name>dfs.namenode.name.dir</name>

  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>

</property>

<property>
  <name>dfs.datanode.data.dir</name>

  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>

</property>
```



```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>

```

The hdfs-site.xml file should look something like this :

Save and close this file.

Format the New Hadoop Filesystem

After completing all the configuration outlined in the above steps, the Hadoop filesystem needs to be formatted so that it can start being used.

This is done by executing the following command:

```
hdfs namenode -format
```

Note: *This only needs to be done once before you start using Hadoop. If this command is executed again after Hadoop has been used, it'll destroy all the data on the Hadoop file system.*

Start Hadoop

All that remains to be done is starting the newly installed single node cluster:

`start-dfs.sh`

While executing this command, you'll be prompted twice with a message similar to the following:

Are you sure you want to continue connecting (yes/no)?

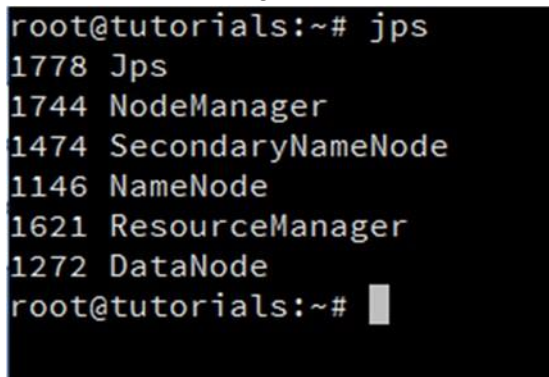
Type in yes for both these prompts and press the enter key. Once this is done, execute the following command:

`start-yarn.sh`

Executing the above two commands will get Hadoop up and running. You can verify this by typing in the following command:

`jps`

Executing this command should show you something similar to the



```
root@tutorials:~# jps
1778 Jps
1744 NodeManager
1474 SecondaryNameNode
1146 NameNode
1621 ResourceManager
1272 DataNode
root@tutorials:~#
```

following:

If you can see a result similar to the depicted in the screenshot above, it means that you now have a functional instance of Hadoop running on your VPS.

EXP 5 Part A

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

CharMap.java

```
package exp5a;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class CharMap extends Mapper<LongWritable, Text, Text, IntWritable> {
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
String line = value.toString();
char[] carr = line.toCharArray();
for (char c : carr) {
System.out.println(c);
context.write(new Text(String.valueOf(c)), new IntWritable(1));
}
}
}
```

CharReduce.java

```
package exp5a;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class CharReduce extends Reducer<Text, IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
int count = 0;
IntWritable result = new IntWritable();
for (IntWritable val : values) {
count += val.get();
result.set(count);
}
context.write(key, result);
}
}
```

CharCount.java

```
package exp5a;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```



```

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class CharCount {
public static void main(String[] args) throws Exception {
// TODO Auto-generated method stub
Configuration conf = new Configuration();
@SuppressWarnings("deprecation")
Job job = new Job(conf, "Charcount");
job.setJarByClass(CharCount.class);
job.setMapperClass(CharMap.class);
job.setReducerClass(CharReduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

Sample1.txt

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. This approach takes advantage of data locality nodes manipulating the data they have access to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Output :

```
[root@localhost hadoop]# start-dfs.sh
```

```
[root@localhost hadoop]# start-yarn.sh
starting yarn daemons
```

```
[root@localhost hadoop]# jps
7415 Jps
4688 NodeManager
4240 SecondaryNameNode
3952 NameNode
5698 org.eclipse.equinox.launcher_1.3.0.v20120308-1358.jar
4077 DataNode
4398 ResourceManager
```

```
[root@localhost bin]# hdfs dfs -mkdir /user/system-admin
```

```
[root@localhost bin]# hadoop dfs -put /home/system-admin/sample1.txt
/user/system-admin/input_data
```

```
[root@localhost bin]# hadoop jar ~/exp5a.jar input_data output_data
```

Not a valid JAR: /root/exp5a.jar

```
[root@localhost bin]# cp /home/system-admin/exp5a.jar /root/
```

```
[root@localhost bin]# hadoop jar ~/exp5a.jar input_data output_data
```

16/03/05 12:26:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

```
16/03/05 12:26:36 INFO client.RMProxy: Connect[  
[root@localhost bin]# hadoop jar ~/exp5a.jar  
input_data output_data
```

Not a valid JAR: /root/exp5a.jar

```
[root@localhost bin]# cp /home/system-admin/exp5a.jar /root/
```

```
[root@localhost bin]# hadoop jar ~/exp5a.jar input_data output_data
```

16/03/05 12:26:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/03/05 12:26:36 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032

16/03/05 12:26:38 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

16/03/05 12:26:39 INFO input.FileInputFormat: Total input paths to process : 1

16/03/05 12:26:40 INFO mapreduce.JobSubmitter: number of splits:1

16/03/05 12:26:41 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1457193651882_0001

16/03/05 12:26:44 INFO impl.YarnClientImpl: Submitted application
application_1457193651882_0001

16/03/05 12:26:44 INFO mapreduce.Job: The url to track the job:
http://localhost:8088/proxy/application_1457193651882_0001/

16/03/05 12:26:44 INFO mapreduce.Job: Running job: job_1457193651882_0001

16/03/05 12:27:32 INFO mapreduce.Job: Job job_1457193651882_0001 running in uber mode :
false

16/03/05 12:27:32 INFO mapreduce.Job: map 0% reduce 0%

16/03/05 12:28:24 INFO mapreduce.Job: map 100% reduce 0%

16/03/05 12:29:06 INFO mapreduce.Job: map 100% reduce 100%

16/03/05 12:29:12 INFO mapreduce.Job: Job job_1457193651882_0001 completed successfully

16/03/05 12:29:13 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=8294

FILE: Number of bytes written=228761

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=1147

HDFS: Number of bytes written=169

HDFS: Number of read operations=6

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=1

Launched reduce tasks=1

Data-local map tasks=1

Total time spent by all maps in occupied slots (ms)=50365

Total time spent by all reduces in occupied slots (ms)=31148

Total time spent by all map tasks (ms)=50365

Total time spent by all reduce tasks (ms)=31148

Total vcore-seconds taken by all map tasks=50365

Total vcore-seconds taken by all reduce tasks=31148

Total megabyte-seconds taken by all map tasks=51573760

Total megabyte-seconds taken by all reduce tasks=31895552

Map-Reduce Framework

Map input records=2

Map output records=1036

Map output bytes=6216

Map output materialized bytes=8294

Input split bytes=107

Combine input records=0

Combine output records=0

Reduce input groups=37

Reduce shuffle bytes=8294

Reduce input records=1036

Reduce output records=37

Spilled Records=2072

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=142

CPU time spent (ms)=2070

Physical memory (bytes) snapshot=245239808

Virtual memory (bytes) snapshot=1950244864

Total committed heap usage (bytes)=87162880

Shuffle Errors

BAD_ID=0

CONNECTION=0

IO_ERROR=0

WRONG_LENGTH=0

WRONG_MAP=0

WRONG_REDUCE=0

File Input Format Counters

Bytes Read=1040

File Output Format Counters

Bytes Written=169

[root@localhost bin]# ~/hadoop/bin/hdfs dfs -cat output_data/*

bash: /root/hadoop/bin/hdfs: No such file or directory

[root@localhost bin]# hdfs dfs -cat output_data/*

16/03/05 12:34:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

	160
(1
)	1
,	3
-	2
.	6
A	3
D	2
F	2
H	7
J	1
M	1
R	1
S	2
T	3
a	94
b	13
c	33
d	51
e	95
f	18
g	12
h	26
i	46
k	7
l	36
m	20
n	45
o	71
p	31
r	56
s	63
t	74
u	23
v	6
w	12
y	9

EXP 5 Part B

Aim: Design a distributed application using MapReduce under Hadoop for:
b) Counting no. of occurrences of every word in a given text file.

WordCount.java

```
package exp5b;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
    public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

IntSumReducer.java

```
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
    Context context
    ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
```

```

String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
if (otherArgs.length < 2) {
System.err.println("Usage: wordcount <in> [<in>...] <out>");
System.exit(2);
}
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
for (int i = 0; i < otherArgs.length - 1; ++i) {
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job,
new Path(otherArgs[otherArgs.length - 1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

Sample.txt

```

This is SITS Narhe
This is SITS Narhe
This is SITS Narhe
This is SITS Narhe

```

Output :

```

[root@localhost hadoop]# start-dfs.sh

[root@localhost hadoop]# start-yarn.sh
starting yarn daemons

[root@localhost hadoop]# jps
7415 Jps
4688 NodeManager
4240 SecondaryNameNode
3952 NameNode
5698 org.eclipse.equinox.launcher_1.3.0.v20120308-1358.jar
4077 DataNode
4398 ResourceManager

[root@localhost bin]# hdfs dfs -mkdir /user/system-admin

[root@localhost bin]# hadoop dfs -put /home/system-admin/sample.txt
/user/system-admin/input_data

[root@localhost bin]# hadoop jar /home/system-admin/exp5b.jar
/user/system-admin/input_data /home/system-admin/output_data

```

```

16/03/08 10:55:20 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/03/08 10:55:22 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/03/08 10:55:23 INFO input.FileInputFormat: Total input paths to
process : 1
16/03/08 10:55:24 INFO mapreduce.JobSubmitter: number of splits:1
16/03/08 10:55:24 INFO mapreduce.JobSubmitter: Submitting tokens for
job: job_1457409349355_0002
16/03/08 10:55:25 INFO impl.YarnClientImpl: Submitted application
application_1457409349355_0002
16/03/08 10:55:25 INFO mapreduce.Job: The url to track the job:
http://localhost:8088/proxy/application_1457409349355_0002/
16/03/08 10:55:25 INFO mapreduce.Job: Running job:
job_1457409349355_0002
16/03/08 10:55:43 INFO mapreduce.Job: Job job_1457409349355_0002
running in uber mode : false
16/03/08 10:55:43 INFO mapreduce.Job: map 0% reduce 0%
16/03/08 10:56:05 INFO mapreduce.Job: map 100% reduce 0%
16/03/08 10:56:10 INFO mapreduce.Job: Job job_1457409349355_0002
completed successfully
16/03/08 10:56:10 INFO mapreduce.Job: Counters: 30
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=105631
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=836
        HDFS: Number of bytes written=301
        HDFS: Number of read operations=5
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=18696
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=18696
        Total vcore-seconds taken by all map tasks=18696
        Total megabyte-seconds taken by all map tasks=19144704
    Map-Reduce Framework
        Map input records=25
        Map output records=25
        Input split bytes=123
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=27
        CPU time spent (ms)=510
        Physical memory (bytes) snapshot=89690112

```

```
Virtual memory (bytes) snapshot=958951424
Total committed heap usage (bytes)=31522816
File Input Format Counters
  Bytes Read=713
File Output Format Counters
  Bytes Written=301
hdfs://localhost:9000/home/system-admin/input_data/_SUCCESS
hdfs://localhost:9000/home/system-admin/input_data/part-m-00000
```

```
[root@localhost bin]# hdfs dfs -cat output_data/*
```

```
16/03/08 12:08:10 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
```

```
is      4
```

```
Narhe  4
```

```
SITS   4
```

```
this   4
```


Assignment No. 6

Aim: Design a distributed application using MapReduce under Hadoop for finding maximum number in first and second columns in every line of a given text file.

Theory :

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
 - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
 - **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and Outputs (Java Perspective)

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

* Terminology

- **Payload** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.

- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to JobTracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

* Important Commands

- All Hadoop commands are invoked by the \$HADOOP_HOME/bin/hadoopcommand. Running the Hadoop script without any arguments prints the description for all commands.
- Usage : `hadoop [--config confdir] COMMAND`
- The following table lists the options available and their description.

Options	Description
namenode -format	Formats the DFS filesystem.
secondarynamenode	Runs the DFS secondary namenode.
namenode	Runs the DFS namenode.
datanode	Runs a DFS datanode.
dfsadmin	Runs a DFS admin client.
mradmin	Runs a Map-Reduce admin client.
fsck	Runs a DFS filesystem checking utility.
Fs	Runs a generic filesystem user client.
balancer	Runs a cluster balancing utility.
oiv	Applies the offline fsimage viewer to an fsimage.
fetchdt	Fetches a delegation token from the NameNode.

jobtracker	Runs the MapReduce job Tracker node.
pipes	Runs a Pipes job.
tasktracker	Runs a MapReduce task Tracker node.
historyserver	Runs job history servers as a standalone daemon.
job	Manipulates the MapReduce jobs.
queue	Gets information regarding JobQueues.
version	Prints the version.
jar <jar>	Runs a jar file.
distcp <srcurl> <desturl>	Copies file or directories recursively.
distcp2 <srcurl> <desturl>	DistCp version 2.
archive -archiveName NAME -p	Creates a hadoop archive.
<parent path> <src>* <dest>	
classpath	Prints the class path needed to get the Hadoop jar and the required libraries.
daemonlog	Get/Set the log level for each daemon

* How to Interact with MapReduce Jobs

- Usage: `hadoop job [GENERIC_OPTIONS]`
- The following are the Generic Options available in a Hadoop job.

GENERIC_OPTIONS	Description
-----------------	-------------

-submit <job-file>	Submits the job.
-status <job-id>	Prints the map and reduce completion percentage and all job counters.
-counter <job-id> <group-name> <countername>	Prints the counter value.
-kill <job-id>	Kills the job.
-events <fromevent-#> <job-id> <#-of-events>	Prints the events' details received by jobtracker for the given range.
-history [all] <jobOutputDir> - history <jobOutputDir>	Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
-list[all]	Displays all jobs. -list displays only jobs which are yet to complete.
-kill-task <task-id>	Kills the task. Killed tasks are NOT counted against failed attempts.
-fail-task <task-id>	Fails the task. Failed tasks are counted against failed attempts.
-set-priority <priority> <job-id>	Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

Sample6.txt

```
14,5,84,9,7,4,44,5,84,9
4,39,79,44,5,88,4,50,53,4
45,59,67,3,76,3,0,4,6,19
7,9,6,9,27,0,0,14,53,64
8,66,4,67,1,3,0,7,93,56
77,41,56,12,10,4,7,23,7,99
10,89,45,123,78,41,23,12,10,49
7,5,22,14,78,43,55,41,10,13
41,45,13,15,19,57,66,55,44,50
44,888,89,49,27,48,33,88,84,65
144,5,84,9,7,4,44,5,84,9
4,396,79,44,5,88,4,50,53,4
45,598,67,3,76,3,0,4,6,19
7,9,6,96,27,0,0,14,53,64
87,66,4,67,1,3,0,7,93,56
79,41,56,12,10,4,7,23,7,99
100,89,45,123,78,41,23,12,10,49
7,501,22,14,78,43,55,41,10,13
414,451,13,15,19,57,66,55,44,50
444,1,89,49,27,48,33,88,84,65
447,39,79,44,5,88,4,50,53,4
45,596,67,3,76,3,0,4,6,19
787,999,6,9,27,0,0,14,539,64
8,66,4,67,1,3,0,7,936,5654
777,541,56,12,10,4,7,23,7,99
```

MaxMap.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxMap extends Mapper<LongWritable, Text, Text,
IntWritable>
{
int values[] = new int[10000];
int values1[] = new int[10000];
String word[] ; int maxValue = 0,linenum =0;
public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
{
String words = value.toString();
System.out.println(words);
word = words.split(",");
for (int i = 0; i < 2; i++)
```

```

{
System.out.println(word[i]);
values[i] = Integer.parseInt(word[i]);
values1[i] = Integer.parseInt(word[i]);
}
if(values1[0] < values1[1])
{
int temp =values1[0];
values1[0] = values1[1];
values1[1] = temp;
}
maxValue = values1[0];
String text = ""+(linenum+1)+"\t"+values[0]+" \t"+values[1]+" ";
if(linenum>=0)
{ context.write(new Text(text), new IntWritable(maxValue));
}
linenum++;
}
}

```

MaxCount.java

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class MaxCount extends Configured implements Tool
{
public static void main(String[] args) throws Exception {
int res = ToolRunner.run(new Configuration(), new MaxCount() , args);
System.exit(res);
}
@Override
public int run(String[] args) throws Exception {
Configuration conf = getConf();
@SuppressWarnings("deprecation")
Job job = new Job(conf, "MaxCount");
job.setJarByClass(MaxCount.class);

```

```

job.setMapperClass(MaxMap.class);
job.setNumReduceTasks(0);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setInputFormatClass(org.apache.hadoop.mapreduce.lib.input.TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
Path outputpath = new Path(args[1]);
outputpath.getFileSystem(conf).delete(outputpath, true);
job.waitForCompletion(true);
s.close();
return 0;
}
}

```

Output :

```
[root@localhost hadoop]# start-dfs.sh
```

```
[root@localhost hadoop]# start-yarn.sh
starting yarn daemons
```

```
[root@localhost hadoop]# jps
7415 Jps
4688 NodeManager
4240 SecondaryNameNode
3952 NameNode
5698 org.eclipse.equinox.launcher_1.3.0.v20120308-1358.jar
4077 DataNode
4398 ResourceManager
```

```
[root@localhost bin]# hdfs dfs -mkdir /user/system-admin
```

```
[root@localhost bin]# hadoop dfs -put /home/system-admin/sample6.txt
/user/system-admin/input_data
[root@localhost bin]# hadoop jar /home/system-admin/exp6.jar
/user/system-admin/input_data /home/system-admin/output_data
16/03/08 10:55:20 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/03/08 10:55:22 INFO client.RMPProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/03/08 10:55:23 INFO input.FileInputFormat: Total input paths to
process : 1
16/03/08 10:55:24 INFO mapreduce.JobSubmitter: number of splits:1
16/03/08 10:55:24 INFO mapreduce.JobSubmitter: Submitting tokens for
job: job_1457409349355_0002
16/03/08 10:55:25 INFO impl.YarnClientImpl: Submitted application
application_1457409349355_0002
```



```

16/03/08 10:55:25 INFO mapreduce.Job: The url to track the job:
http://localhost:8088/proxy/application_1457409349355_0002/
16/03/08 10:55:25 INFO mapreduce.Job: Running job:
job_1457409349355_0002
16/03/08 10:55:43 INFO mapreduce.Job: Job job_1457409349355_0002
running in uber mode : false
16/03/08 10:55:43 INFO mapreduce.Job: map 0% reduce 0%
16/03/08 10:56:05 INFO mapreduce.Job: map 100% reduce 0%
16/03/08 10:56:10 INFO mapreduce.Job: Job job_1457409349355_0002
completed successfully
16/03/08 10:56:10 INFO mapreduce.Job: Counters: 30
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=105631
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=836
    HDFS: Number of bytes written=301
    HDFS: Number of read operations=5
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=18696
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=18696
    Total vcore-seconds taken by all map tasks=18696
    Total megabyte-seconds taken by all map tasks=19144704
  Map-Reduce Framework
    Map input records=25
    Map output records=25
    Input split bytes=123
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=27
    CPU time spent (ms)=510
    Physical memory (bytes) snapshot=89690112
    Virtual memory (bytes) snapshot=958951424
    Total committed heap usage (bytes)=31522816
  File Input Format Counters
    Bytes Read=713
  File Output Format Counters
    Bytes Written=301
hdfs://localhost:9000/home/system-admin/input_data/_SUCCESS
hdfs://localhost:9000/home/system-admin/input_data/part-m-00000

```

```

Line First Second      Maximum
no   Column      Column

```

1	14	5	14
2	4	39	39
3	45	59	59
4	7	9	9
5	8	66	66
6	77	41	77
7	10	89	89
8	7	5	7
9	41	45	45
10	44	888	888
11	144	5	144
12	4	396	396
13	45	598	598
14	7	9	9
15	87	66	87
16	79	41	79
17	100	89	100
18	7	501	501
19	414	451	451
20	444	1	444
21	447	39	447
22	45	596	596
23	787	999	999
24	8	66	66
25	777	541	777

Overall Maximum: 999
d=836

HDFS: Number of bytes written=301
HDFS: Number of read operations=5
HDFS: Number of large read operations=0
HDFS: Number of write operations=2

Job Counters

Launched map tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=18696
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=18696
Total vcore-seconds taken by all map tasks=18696
Total megabyte-seconds taken by all map tasks=19144704

Map-Reduce Framework

Map input records=25
Map output records=25
Input split bytes=123
Spilled Records=0[root@localhost bin

Recommended Android Apps for Students

[Download Now](#)