**MINI PROJECT-BEOWULF CLUSTER**

**AIM:** To implement Beowulf cluster using java socket programming.

**PROJECT REQUIREMENTS:**

* At least 3 computers (1 server,2 clients)
* Java Runtime Environment

**DESCRIPTION**:

1) Beowulf Cluster:

* Beowulf is a multi-computer architecture which can be used for parallel computations. A Beowulf cluster is a computer cluster of what are normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed which allow processing to be shared among them. The result is a high-performance parallel computing cluster from inexpensive personal computer hardware.
* Beowulf clusters normally run a Unix-like operating system, such as BSD, Linux, or Solaris, normally built from free and open source software. Commonly used parallel processing libraries include Message Passing Interface (MPI) and Parallel Virtual Machine (PVM).

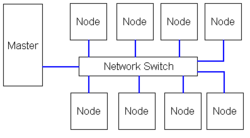


Fig 1.0: Diagram of Beowulf Cluster

2) Uses of Beowulf Clusters:

Beowulf clusters are generally used where the implementation of parallel processing is feasible and efficient, such as:

* Simulations
* Biotechnology
* Pedro clusters
* Financial Market Modeling
* Data Mining
* Stream Processing
* Internet servers for Audio and Video games

**IMPLEMENTATION**:

* In our mini project we have implemented the concept of Beowulf cluster using java socket programming which uses java.net library.
* Using Socket programming, Server-Client network is formed.
* After the network is established, a range of numbers (0 – N) is given as an input (N) on the server side.
* Server divides this range into equal sub-ranges and assign each client to find the number of prime numbers in that sub-range.
* Each client sends its output to server after the completion of execution of task. Server, then, pipelines the output to the text file which the user can read.

**ALGORITHM**:

Server:-

1. Accept a number
2. Put the number in sample file in desktop
3. Wait for clients to connect
4. Assign a unique client id to every client and send to the client
5. Distribute the range evenly to all the connected clients
6. Wait for all the clients to send back the result
7. Store the results in the sample file on desktop
8. Stop

Client:-

1. Connect to server
2. Print successful connection message with client id
3. Receive the range of number from server
4. Find the number of primes in the given range
5. Send the result to server
6. Stop

**CODE:**

**1) SERVER SIDE:**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Server {

class ClientHandler implements Runnable{

Socket socket;

BufferedReader reader;

PrintWriter writer;

public ClientHandler(Socket socket) throws IOException{

this.socket = socket;

}

@Override

public void run() {

try {

reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

writer = new PrintWriter(socket.getOutputStream(), true);

writer.println("success:" + clientNum++ );

System.out.println("client success");

` } catch (IOException ex) {

Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

static ArrayList<ClientHandler> clientList;

static ArrayList<Element> nums;

static int clientNum=0;

public static void main(String[] args) throws IOException, InterruptedException{

File file = new File("/home/student/Desktop/sample.txt");

BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter any number:");

String prime = stdin.readLine();

PrintWriter fileWriter = new PrintWriter(file);

fileWriter.println("Input:" + prime);

fileWriter.close();

ServerSocket serverSocket = new ServerSocket(5000);

Server mainServer = new Server();

clientList = new ArrayList<>();

mainServer.init(serverSocket);

mainServer.giveInputToClients(prime);

int[] done = new int[clientList.size()];

for(int i=0;i<clientList.size();i++){

String clientPosition="";

for(int j=0;j<clientList.size();j++){

if(done[j]==0){

clientPosition = clientList.get(i).reader.readLine();

if(isNumeric(clientPosition) && done[j]==0){

done[j] = 1;

break;

}

}

}

int position = Integer.parseInt(clientPosition);

System.out.println("Read input from client ID:" + position);

mainServer.readInputNumber(position);

}

int endarray=0;

while(true){

for(int k=0;k<clientList.size();k++){

String line = clientList.get(k).reader.readLine();

if(line!=null){

if(line.startsWith("Client")){

mainServer.writeToFile(line);

}else if(line.equals("bye")){

endarray++;

break;

}

}

}

if(endarray==clientList.size()){

break;

}

}

serverSocket.close();

}

void init(ServerSocket serverSocket) throws IOException, InterruptedException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Waiting for connections...");

while(true){

Socket socket = serverSocket.accept();

ClientHandler newClient = new ClientHandler(socket);

clientList.add(newClient);

Thread t = new Thread(newClient);

t.start();

System.out.println("Do you want to add more clients(Y/n):");

String ch = br.readLine();

if(ch.equalsIgnoreCase("n")){

break;

}

}

System.out.println(clientList.size() + " client(s) connected!");

}

void giveInputToClients(String prime){

System.out.println("Dividing task...");

BigInteger bigInt = new BigInteger(prime);

int n = clientList.size();

nums = new ArrayList<>();

BigInteger nn = new BigInteger(String.valueOf(n));

BigInteger div = bigInt.divide(nn);

BigInteger start = new BigInteger(String.valueOf("2"));

BigInteger end = div;

BigInteger i = new BigInteger(String.valueOf("0"));

BigInteger one = new BigInteger(String.valueOf("1"));

for(int j=0;j<(n-1);j++){

Element currElement = new Element(start, end);

nums.add(currElement);

start = end.add(one);

end = end.add(div);

}

end = bigInt;

nums.add(new Element(start,end));

System.out.println("Input Distributed");

for(Element curr : nums){

System.out.println("Start:" + curr.getStart() + " End:" + curr.getEnd());

}

}

void readInputNumber(int pos){

String start = nums.get(pos).getStart().toString();

String end = nums.get(pos).getEnd().toString();

clientList.get(pos).writer.println("Start:" + start);

clientList.get(pos).writer.println("End:" + end);

}

synchronized void writeToFile(String line) throws FileNotFoundException, IOException{

File file = new File("/home/student/Desktop/sample.txt");

FileWriter fw = new FileWriter(file,true);

BufferedWriter bw = new BufferedWriter(fw);

PrintWriter writer = new PrintWriter(bw);

writer.println(line);

writer.close();

}

class Element{

BigInteger start;

BigInteger end;

public Element(BigInteger start, BigInteger end){

this.start = start;

this.end = end;

}

public BigInteger getStart(){

return start;

}

public BigInteger getEnd(){

return end;

}

}

public static boolean isNumeric(String str){

try {

double d = Double.parseDouble(str);

}catch(NumberFormatException nfe){

return false;

}

return true;

}

}

**2) CLIENT SIDE:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.Socket;

import java.util.ArrayList;

public class Client {

public static void main(String[] args) throws IOException{

Socket socket = new Socket("127.0.0.1", 5000);

BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter socketWriter = new PrintWriter(socket.getOutputStream(),true);

String reply = "";

int pos=-1;

reply = br.readLine();

if(reply.startsWith("success")){

pos = Integer.parseInt(reply.substring(reply.indexOf(":")+1));

System.out.println("Connected Successfully with server:" + pos);

}else{

System.out.println("Something went wrong");

System.exit(0);

}

socketWriter.println(pos);

String startString = br.readLine();

String endString = br.readLine();

String[] startArray = startString.split(":");

String[] endArray = endString.split(":");

BigInteger startNum = new BigInteger(startArray[1]);

BigInteger endNum = new BigInteger(endArray[1]);

BigInteger one = new BigInteger("1");

BigInteger two = new BigInteger("2");

BigInteger zero = new BigInteger("0");

int flag = 0;

int counter=0;

for(;startNum.compareTo(endNum)<=0;startNum=startNum.add(one)){

for(BigInteger j=two;j.compareTo(startNum)<0;j=j.add(one)){

if((startNum.mod(j)).compareTo(zero)==0){

flag = 0;

break;

}else{

flag = 1;

}

}

if(flag==1){

counter++;

}

}

socketWriter.println("Client " + pos + ":" + counter);

socketWriter.println("bye");

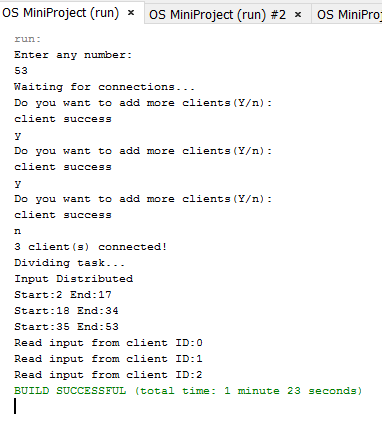
socket.close();

}

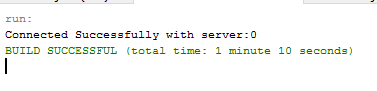
}

**OUTPUT**:

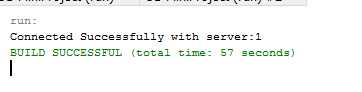
Server:-



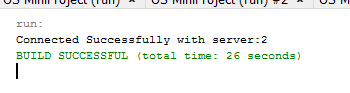
Client 1:-



Client 2:-



Client 3:-



**SUMMARY:**

* Beowulf cluster is an efficient solution for high-performance parallel computing.
* It can be implemented using various open-source softwares.
* Maintaining sustained and secure connection between clients and servers is the crucial aspect of the Beowulf cluster.

**REFERENCES**:

* <https://en.wikipedia.org/wiki/Beowulf_cluster>
* https://www.linux.com/blog/building-beowulf-cluster-just-13-steps
* https://docs.oracle.com/javase/tutorial/networking/sockets/