



Distributed Computing Job Management Protocol Design

EE 544- Computer Networks and Internet
Professor: Yi Zhao

Project by **TEAM- E**

Bhushan Jibhakate (CIN 304440791)

Deepak Saluja (CIN 304367549)

Mayankumar Patel (CIN 304452400)

Index

- 1. Abstract**
- 2. Role of team member**
- 3. Introduction**
 - Job management Protocol Design**
 - Socket Programming for Client-Server Model**
- 4. Simple Job Client Programming in Eclipse**
- 5 Simple Job Sever Programming in Eclipse**
- 6 Actual output**
 - Simple Job Server is Ready**
 - Simple Job Client 1**
 - Simple job Client 2**
 - Server Stops as per project requirement**
- 7. Conclusion**
- 8. Problems and future possibility of project**
- 9. Acknowledgement**
- 10. Reference**

Abstract:

In the previous project on Echo Server/Client, we have learned the basic concepts and process of using socket for data communication in Java. And this project aims to further deepen our understanding on protocol design and distributed computing.

Background

During the past two decades, video encoding technology has experienced fast evolution and new generations of video coding standards have emerged:

- ITU H.264/MPEG-4 AVC – first appeared in 2003.
- ITU H.265/MPEG-H HEVC – first version published in 2013.

Each video coding standard achieves the design goal of reducing the bitrates needed to about half of the previous generation video coding standard while maintaining the same visual quality for coded video contents. The decoder side's computation complexity has experienced moderate increment, but the encoder side's computation complexity has been dramatically exploded.

Another trend in video technology is the fast adaption of high definition and even ultra-high definition video contents.

To combat the extremely high complexity and improve the processing speed, video engineers have employed various parallel computing techniques when developing ultra-high definition video encoding software that could utilize the full capability of modern computer systems. For example:

1. Using the SIMD (Single Instruction Multiple Data) instruction sets offered by latest processors significantly improves the throughput of hot spots in the video coding software;
2. Using multiple threads for parallel frame/tile encoding explores the full potential of multi-core processors and/or multi-processor computers.

Another cost-effective approach is based on distributed computing concept:

1. The whole video encoding system consists of one controller computer and multiple engine computers;
2. The controller computer divides the video content to be encoded into small units and delegates the actual encoding tasks on these units to the engine computers.

This approach has the advantage that its performance could be easily scaled up with the availability of more engine computers.

Role of team members :

Bhushan Jibhakate (CIN 304440791) – Programming and report

Deepak Saluja (CIN 304367549)- Programming

Mayankumar Patel (CIN 304452400) - Testing

Each team member's contribution in this project is equal as well as remarkable in terms of dedication. This project is completed in following steps.

1. Understanding the project requirements.(ex. What is Job management)
2. Study of Java programming syntax.
3. Algorithm creation.
4. Actual Programming of server and client.
5. Programming of simple job client.
6. Programming of simple job server.
7. Execution
8. Testing

Job Management Protocol Design

Obviously, in order to effectively and efficiently run this distributed ultra-high definition video encoding system, a good and feasible job management module is essential. Actually, there are lots interesting topics on this distributed computing job management such as load balancing and security. However, in this project we mainly focus on the *job management protocol* that handles the communication between the controller and engine computers²

In the term of distributed computing, these video coding tasks are jobs that could be assigned to computing nodes.

2 From the aspect of network communication on job management protocol, the controller computer acts as a server and the engine computer are all clients.

A basic job management protocol shall be able to handle the following events:

- The client requests video encoding job from the server and submits finished job to the server;
- The server assigns available video encoding job to the requested client, accepts submitted job from the client, and maintains/updates a table for all video encoding jobs.

For example, once one client established the connection, it will send a message to server, requesting for a new video encoding job:

client -> requestJob

If the server finds one unassigned job entry in its job table, it assigns this job to the requested client and send the response message containing both the job ID and description³:

server -> assignJob 0 ch01.yuv ch01.265 01:00:00:00 240 21.1

After the client received this message, it needs to extract the job ID and description and start to encode the corresponding unit. In this project, we don't need to implement the video coding module at all and simply wait for some random time, *pretending* to be working on the job. □

Then the client needs to submit the finished job to the server by transmitting message containing both the job ID and result⁴:

client -> submitJob 0 (Random Number)

The server may accept the submitted job if it's satisfying with the result by sending:

server -> acceptJob 0

Otherwise the server may reject the submitted job because either the actual bitrate exceeds limit or the coded vide content's quality is not high enough. □ The server will notify the client:

server -> rejectJob 0

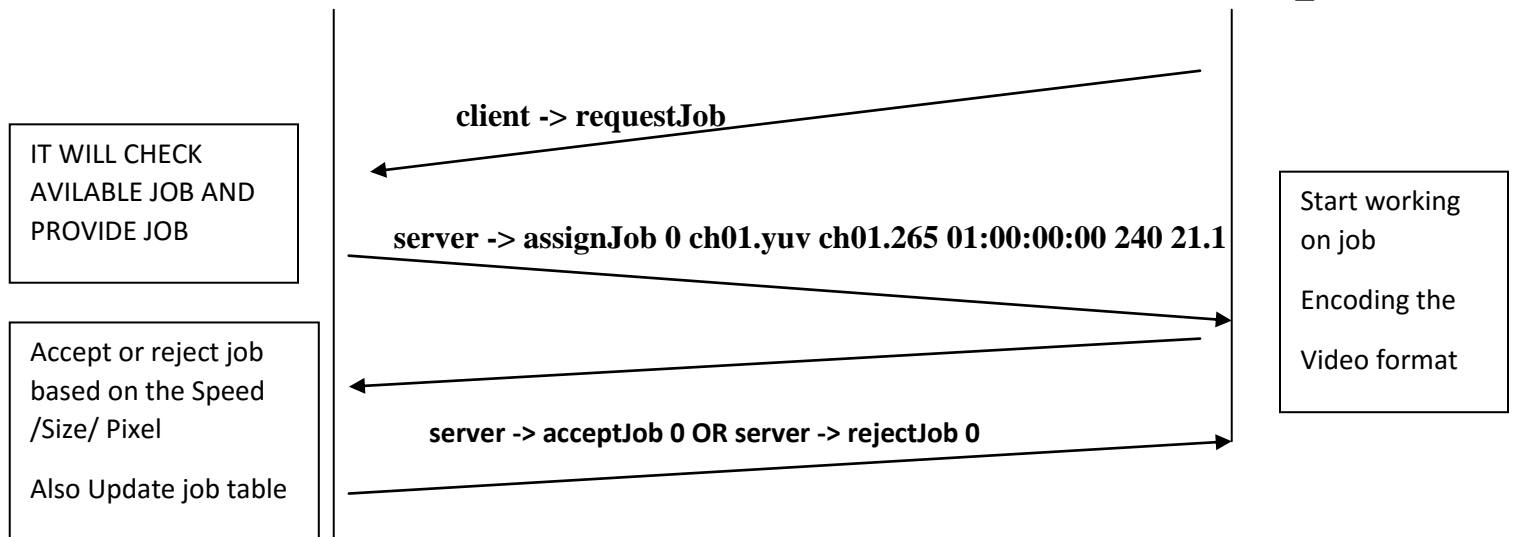
The server can either put the rejected job back to the unassigned job pool pending for re-assignment or just wait the client to re-submit the failed job again5.

Each team needs to carefully analyze the needs of this ultra-high definition video encoding system and design a simple but working job management protocol:

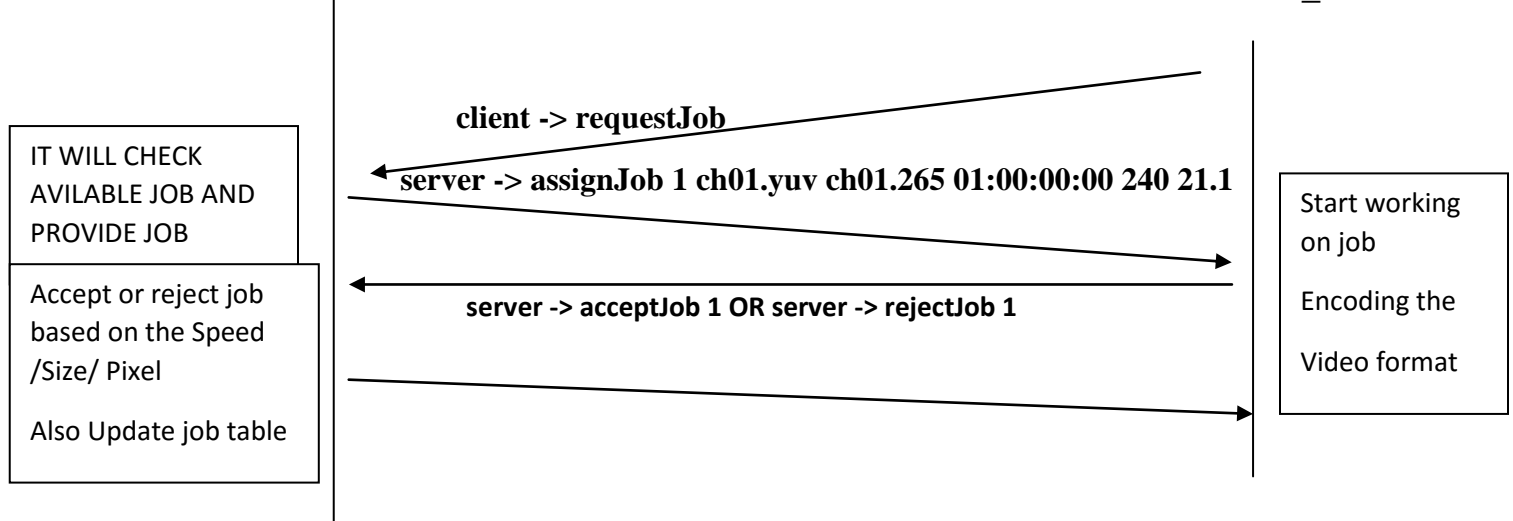
- Protocol *syntax* – the specific format on messages exchanged between the server and clients. All students are free to choose their preferred format that they deem suitable. For example, the message can be binary format and not plain text; the keywords could be different.
- Protocol *semantics* – the actual meaning of each message on the server/client side shall be interpreted correctly and proper action shall be executed thereafter. For example, when the server receives a job request, it shall check for unassigned job first; then if it finds one available in its job table, it shall assign that to the requesting client, otherwise it shall tell the client that no job is currently available.

Server

Client_1



Client_2



SIMPLE_JOB_CLIENT:

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

//the main class that opens a connection to the server,
//starts a thread for communication,
//and waits until the child thread finished.
public class SimpleJobClient {
    public static void main(String[] args) {
        // Obtain host name & port number
        String hostName = "127.0.0.1";
        int portNumber = 9090;
        if (args.length >= 1)
            hostName = args[0];
        if (args.length >= 2)
            portNumber = Integer.parseInt(args[1]);
        // Declare socket and thread
        //the dedicated thread class that handles the communication
        //task with the server regarding the job request and submission.
        Socket socket = null;
        SimpleJobClientThread thread = null;
        // Open socket and start thread
        try {
            socket = new Socket(hostName, portNumber);
            thread = new SimpleJobClientThread(socket);
            thread.start();
            thread.join();
        }
```

```

        } catch (UnknownHostException e) {
            System.err.println("Don't know about host.");
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the
connection.");
        } catch (InterruptedException e) {
            e.printStackTrace(); } } }
final class SimpleJobClientThread extends Thread {
    private Socket socket;
    public SimpleJobClientThread(Socket socket) {
        this.socket = socket;
    }
    public void run() {
        try {
            process();
        } catch (Exception e) {
            System.err.println(e); }}
    private void process() throws Exception {
// Display connection

StringserverInfo=socket.getInetAddress()+":"+socket.getPort();
StringclientInfo=socket.getLocalAddress()+":"+socket.getLocalPort();

System.out.println("Client"+clientInfo+"connectedtoServer"+
serverInfo);

// Open input/output streams

        BufferedReaderin=newBufferedReader(new
InputStreamReader(socket.getInputStream()));

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

// Process job

```



```

Random rand = new Random();
double errorMargin = 0.4;
boolean done = false;
while (!done) {
    String inputLine;
    String outputLine;
// Request job
// Task1 : fill next line
    outputLine = "requestJob";
    System.out.println("client -> " + outputLine);
    out.println(outputLine);
    sleep(2000);
// Process feedback
    inputLine = in.readLine();
    System.out.println("client <- " + inputLine);
    String token[] = inputLine.split(" ");
// Task2: interpret response and take proper action
    int jobId=Integer.parseInt(token[1]);
    System.out.println("Processing encoding");
// String result="random";
    int randomNum = rand.nextInt((28 - 20) + 1) + 20;
    outputLine="submitJob"+" "+jobId+" "+randomNum;
    System.out.println("client -> " + outputLine);
    out.println(outputLine);
    System.out.println(in.readLine());}
// Close streams and socket
    out.close();
    in.close();

```

```

        socket.close(); }}

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

//the main class that creates a SimpleJobManager object,
// opens one ServerSocket to accept incoming connections from
clients,

//creates one thread for each client connection, and waits until
all jobs are finished.

public class SimpleJobServer {
    public static void main(String args[]) {
// Obtain port number
        int portNumber = 9090;
        if (args.length >= 1)
            portNumber = Integer.parseInt(args[0]);
//SimpleJobManager-this class maintains
//and updates the video encoding job table.
// Declaration
        SimpleJobManager jobManager = null;
        SimpleJobServerThread jobThread = null;
        ServerSocket serverSocket = null;
// Create JobManager
        jobManager = new SimpleJobManager();
        jobManager.display();
// Open ServerSocket
        try {
            serverSocket= new ServerSocket(portNumber);
            serverSocket.setSoTimeout(1000);
        } catch (IOException e) {

```

```

        System.err.println(e);
    }
    // Handle client's connection
    try {
        boolean listening = true;
        while (listening) {
            try {
                SocketclientSocket= serverSocket.accept();
                jobThread=newSimpleJobServerThread(clientSocket, jobManager);
                jobThread.start();

                } catch (SocketTimeoutException e) {
                    if (jobManager.done())
                        listening = false;}
            }
            serverSocket.close();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}

final class SimpleJobServerThread extends Thread {
    private Socket socket;
    private SimpleJobManager manager;

    public SimpleJobServerThread(Socket socket, SimpleJobManager
manager) {
        this.socket = socket;
        this.manager = manager;
    }

    public void run() {
        try {

```

```

        process();
    } catch (Exception e) {
        System.err.println(e);
    }
}

private void process() throws Exception {
    // Display connection information
    StringclientInfo=socket.getInetAddress()+":"+ socket.getPort();
    StringserverInfo=socket.getLocalAddress()+":"+
    socket.getLocalPort();

    System.out.println("Server " + serverInfo + " received
    connection from " + clientInfo);

    // Open input and output streams

    BufferedReaderin=newBufferedReader(new
    InputStreamReader(socket.getInputStream()));

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

    // Main process loop
    double acceptMargin = 0.3;
    boolean clientDone = false;
    while (!clientDone) {
        String inputLine;
        String outputLine=null;
        // Display job summary
        manager.display();
        // Read client's request
        inputLine = in.readLine();

        System.out.println("server <- " + clientInfo + " "
+ inputLine);

        // Process client's request

```

```

        String token[] = inputLine.split(" ");
        // Task3: add proper response from server
        switch(token[0]){
            case "requestJob":
                int jobId=manager.assignJob();
                if(jobId!=-1){
                    outputLine="assignJob"+" "+jobId+" "+manager.getJobEntry(jobId);
                    System.out.println("server-> " + clientInfo + " "+ outputLine);
                    out.println(outputLine);
                }else{
                    outputLine="No Jobs Available for Now";
                    System.out.println("server    ->    "    +
clientInfo + " "+ outputLine);
                    out.println(outputLine);
                }
                break;

            case
"submitJob":if (manager.rejectJob(Integer.parseInt(token[1]),Double.parseDouble(token[2]))==true) {
                outputLine="rejectJob "+token[1];
                System.out.println("server -> " +
clientInfo + " "+ outputLine);
                out.println(outputLine);
            }else{
                outputLine="acceptJob "+token[1];
                System.out.println("server    ->    "    +
clientInfo + " "+ outputLine);
                out.println(outputLine);
            }
                break;

```

```

        } }

// Close streams and sockets

        out.close();

        in.close();

        socket.close();

    }

}

final class SimpleJobManager {

    public enum JobState {

        JS_READY, JS_ASSIGNED, JS_FINISHED

    }

    private int totalJobs;

    private int assignedJobs;

    private int finishedJobs;

    private String[] jobEntry;

    private JobState[] jobState;

    public SimpleJobManager() {

        totalJobs = 10;

        assignedJobs = 0;

        finishedJobs = 0;

        jobEntry = new String[totalJobs];

        jobState = new JobState[totalJobs];

        jobEntry[0] = "ch01.yuv  ch01.265  01:00:00:00 240 21.1";
        jobEntry[1] = "ch02.yuv  ch02.265  01:00:10:00 360 26.2";
        jobEntry[2] = "ch03.yuv  ch03.265  01:00:25:00 240 24.3";
        jobEntry[3] = "ch04.yuv  ch04.265  01:00:35:00 480 25.9";
        jobEntry[4] = "ch05a.yuv ch05a.265 01:00:55:00 600 25.0";
        jobEntry[5] = "ch05b.yuv ch05b.265 01:01:20:00 360 25.7";
    }

}

```

```

jobEntry[6] = "ch06.yuv  ch06.265  01:01:35:00 720 22.3";
jobEntry[7] = "ch07a.yuv ch07a.265 01:02:05:00 360 24.8";
jobEntry[8] = "ch07b.yuv ch07b.265 01:02:20:00 480 27.4";
jobEntry[9] = "ch08.yuv  ch08.265  01:02:40:00 360 20.7";

    for (int i = 0; i < totalJobs; i++) {
        jobState[i] = JobState.JS_READY;
    }
}

public int assignJob() {
    // Task4: codes for assigning jobs
    for(int i=0;i<jobState.length;i++){
        if(jobState[i].equals(JobState.JS_READY)){
            jobState[i]=JobState.JS_ASSIGNED;
            assignedJobs++;
            return i;
        }
    }
    return -1;
}

public boolean rejectJob(int jobIndex, double result) {
    // Task5: codes for rejecting jobs
    String token[]=jobEntry[jobIndex].split(" ");
    double required=Double.parseDouble(token[token.length-1]);
    if((required-result)<=1.0&&(required-result)>=-1.0){
        finishJob(jobIndex);
        return false;
    }else{
        jobState[jobIndex]=JobState.JS_READY;
        assignedJobs--;
    }
}

```

```

        return true;
    } }

    public boolean finishJob(int jobIndex) {
        // Task6: codes for finished jobs
        jobState[jobIndex]=JobState.JS_FINISHED;
        finishedJobs++;
        return true;
    }

    public String getJobEntry(int jobIndex) {
        if (jobIndex < totalJobs) {
            return jobEntry[jobIndex];
        }
        return null;
    }

    public int total() { return totalJobs;}

    public int remaining() {
        return Math.max(0, totalJobs - assignedJobs);
    }

    public int finished() {
        return finishedJobs;
    }

    public boolean done() {
        return finishedJobs >= totalJobs;
    }

    public void display() {
        System.out.println("JobSummarytotal="+total()+"remaining = " +
            remaining() + " finished = "
                + finished());
    }

```


Actual output

Simple Job Server is Ready

```
Command Prompt - java SimpleJobServer
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\bhushan>E:/z
'E:/z' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\bhushan>E:

E:\>cd z

E:\z>java SimpleJobServer
JobSummary total = 10 remaining = 10 finished = 0
```

```
Command Prompt
E:\z>java SimpleJobServer
JobSummary total = 10 remaining = 10 finished = 0
Server /127.0.0.1:9090 received connection from /127.0.0.1:56849
JobSummary total = 10 remaining = 10 finished = 0
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 0 ch01.yuv ch01.265 01:00:00:00 240 21.1
JobSummary total = 10 remaining = 9 finished = 0
server <- /127.0.0.1:56849 submitJob 0 22
server -> /127.0.0.1:56849 acceptJob 0
JobSummary total = 10 remaining = 9 finished = 1
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
JobSummary total = 10 remaining = 8 finished = 1
server <- /127.0.0.1:56849 submitJob 1 25
server -> /127.0.0.1:56849 rejectJob 1
JobSummary total = 10 remaining = 9 finished = 1
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
JobSummary total = 10 remaining = 8 finished = 1
server <- /127.0.0.1:56849 submitJob 1 28
server -> /127.0.0.1:56849 rejectJob 1
JobSummary total = 10 remaining = 9 finished = 1
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
JobSummary total = 10 remaining = 8 finished = 1
server <- /127.0.0.1:56849 submitJob 1 21
server -> /127.0.0.1:56849 rejectJob 1
JobSummary total = 10 remaining = 9 finished = 1
server <- /127.0.0.1:56849 requestJob
```

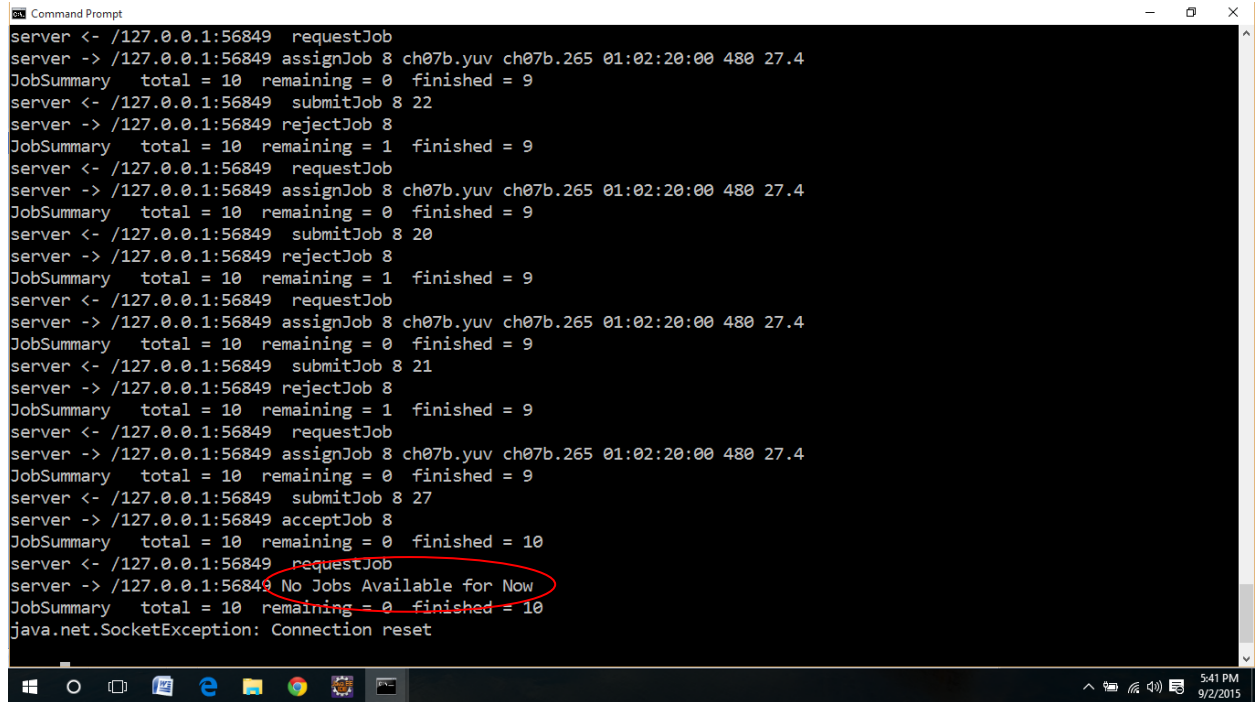
Simple Job Client 1

```
Command Prompt
E:\z>java SimpleJobClient
Client /127.0.0.1:56849 connected to Server /127.0.0.1:9090
client -> requestJob
client <- assignJob 0 ch01.yuv ch01.265 01:00:00:00 240 21.1
Processing encoding
client -> submitJob 0 22
acceptJob 0
client -> requestJob
client <- assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
Processing encoding
client -> submitJob 1 25
rejectJob 1
client -> requestJob
client <- assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
Processing encoding
client -> submitJob 1 28
rejectJob 1
client -> requestJob
client <- assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
Processing encoding
client -> submitJob 1 21
rejectJob 1
client -> requestJob
client <- assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
Processing encoding
client -> submitJob 1 20
rejectJob 1
client -> requestJob
client <- assignJob 1 ch02.yuv ch02.265 01:00:10:00 360 26.2
```

Simple job Client 2

```
Command Prompt
E:\z>java SimpleJobClient2
Client /127.0.0.1:56850 connected to Server /127.0.0.1:9090
client -> requestJob
client <- assignJob 3 ch04.yuv ch04.265 01:00:35:00 480 25.9
Processing encoding
client -> submitJob 3 28
rejectJob 3
client -> requestJob
client <- assignJob 3 ch04.yuv ch04.265 01:00:35:00 480 25.9
Processing encoding
client -> submitJob 3 23
rejectJob 3
client -> requestJob
client <- assignJob 3 ch04.yuv ch04.265 01:00:35:00 480 25.9
Processing encoding
client -> submitJob 3 24
rejectJob 3
client -> requestJob
client <- assignJob 3 ch04.yuv ch04.265 01:00:35:00 480 25.9
Processing encoding
client -> submitJob 3 21
rejectJob 3
client -> requestJob
client <- assignJob 3 ch04.yuv ch04.265 01:00:35:00 480 25.9
```

Server Stops as per project requirement after all jobs are completed.



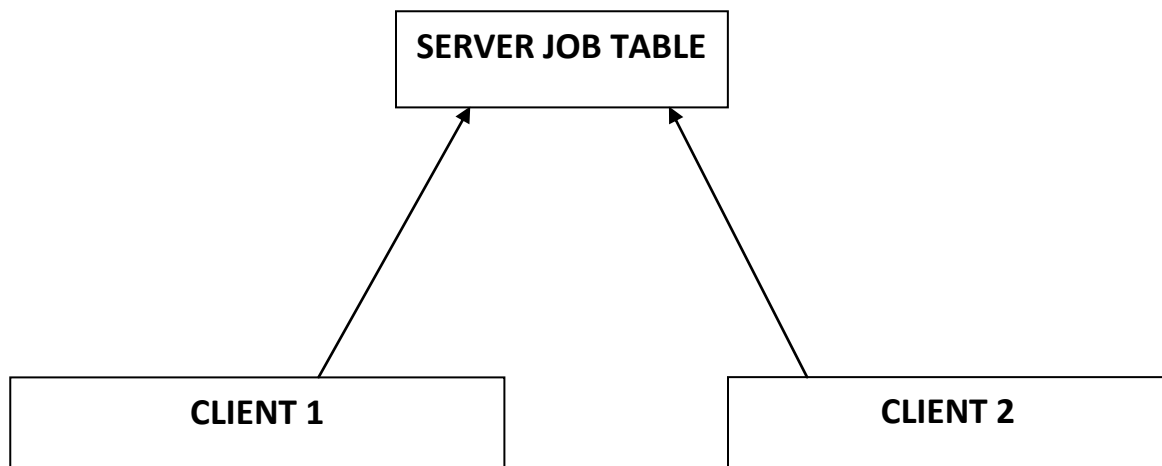
```
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 8 ch07b.yuv ch07b.265 01:02:20:00 480 27.4
JobSummary total = 10 remaining = 0 finished = 9
server <- /127.0.0.1:56849 submitJob 8 22
server -> /127.0.0.1:56849 rejectJob 8
JobSummary total = 10 remaining = 1 finished = 9
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 8 ch07b.yuv ch07b.265 01:02:20:00 480 27.4
JobSummary total = 10 remaining = 0 finished = 9
server <- /127.0.0.1:56849 submitJob 8 20
server -> /127.0.0.1:56849 rejectJob 8
JobSummary total = 10 remaining = 1 finished = 9
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 8 ch07b.yuv ch07b.265 01:02:20:00 480 27.4
JobSummary total = 10 remaining = 0 finished = 9
server <- /127.0.0.1:56849 submitJob 8 21
server -> /127.0.0.1:56849 rejectJob 8
JobSummary total = 10 remaining = 1 finished = 9
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 assignJob 8 ch07b.yuv ch07b.265 01:02:20:00 480 27.4
JobSummary total = 10 remaining = 0 finished = 9
server <- /127.0.0.1:56849 submitJob 8 27
server -> /127.0.0.1:56849 acceptJob 8
JobSummary total = 10 remaining = 0 finished = 10
server <- /127.0.0.1:56849 requestJob
server -> /127.0.0.1:56849 No Jobs Available for Now
JobSummary total = 10 remaining = 0 finished = 10
java.net.SocketException: Connection reset
```

CONCLUSION

Hence we created and implemented this project successfully, this gives us the ability to design protocol of Distributed Computing Job Management.

PROBLES AND FUTURE POSSSIBILITIES:

When multiple clients connected to the job server, the server created one thread for each connection and there is an issue of synchronization when accessing one common job table from multiple threads simultaneously. In this project, this is not one important problem because our focus is the job management protocol. However, for any practical and feasible server designed to handle multiple clients at the same time, appropriate steps need be taken to prevent both data corruption and dead lock among clients to ensure the system's smooth operation.



ACKNOWLEDGEMENT:

We would thank Dr. Yi Zhao, Instructor of EE 544. We appreciate his effort for teaching and help us to understand the concept of internet networking and Distributed Computing Job Management Protocol Design

REFERENCES:

- Online research on socket programming using multi-threads
- Links used: (The text of links used in this project)

<http://docs.oracle.com/javase/tutorial/java/index.html>.

<http://docs.oracle.com/javase/tutorial/java/data/index.html>.

<http://docs.oracle.com/javase/tutorial/essential/io/index.html>.

- Course Material,"EE 544 ".

Text Book: " J. F. Kurose and K. W. Ross, "Computer Networking, A Top-Down Approach", 6th edition, Addison Wesley, 2011"