Chapter 6

Proposed Grid Computing model for E-Commerce Application

6.1 Proposed Grid Model

Following figure 6.1 represent the proposed Grid Model for E-Commerce Application.
Chapter 6. Proposed Grid Computing model for E-Commerce Application

Figure 6.1: Proposed Grid Model
6.2 Components of Model

This model have Three components.

- Grid Server
- Grid Service/Client
- Grid Task/User Requirement

6.2.1 Grid Server

Grid Server is the core controller of all processes. User makes the request for any task. That task request send to the Grid Server. Grid server sends the request to the Clients connected to that server for the requested task. It also gather the response of the client and sends back to the user.

6.2.2 Grid Service/Client

Grid Service or Client provides the service to complete the task. Client executes the tasks given by the Grid Server. Multiple clients perform the one task and send result back to the Grid Server.

6.2.3 Grid Task/User Requirement

User requests for some particular task and that task request sends to the Grid Server and executed at different clients and result return to the user through Grid Server.
6.3 Flow of Model

1. User Requests for some task.

2. That Task requested to execute at Grid Server.

3. Many Grid Services/Clients connected with Grid Server to perform particular task. Any Client works only for particular task. It can not work for other task. If that client want to work for other task then that client required to join for that task only.

4. Grid Server divides this task to Grid Client, connected for that task only.

5. Grid Services/Clients complete the task and send response to the Grid Server.

6. Grid Server send this response back to the users.

6.4 Comparison with existing model

In previous chapter detail of existing model is represented. Following are the points that represents that how my proposed model is different than existing model.

1. Most of the models have one server and multiple clients are connected with that server. This all clients are working to complete one task. e.g. many clients are working to decrypt the encrypted text. Proposed model has multiple tasks that can be handle by one server and for one task there are dedicated clients allocated for it.
6.5 What is e-commerce?

Electronic commerce or e-commerce refers to a wide range of online business activities for products and services. It also pertains to "any form of business transaction in which the parties interact electronically rather than by physical exchanges or direct physical contact."

E-commerce is usually associated with buying and selling over the Internet, or conducting any transaction involving the transfer of ownership or rights to use goods or services through a computer-mediated network.

Though popular, this definition is not comprehensive enough to capture recent developments in this new and revolutionary business phenomenon. A more complete definition is: E-commerce is the use of electronic communications and digital information processing technology in business transactions to create, transform, and redefine relationships for value creation between or among organizations, and between organizations and individuals. [62]

6.6 Implementation of Model

6.6.1 Requirement of System

Based on survey done by the Pranoti Salunke and team [63], in present toll collection system where each vehicle has to stop and pay taxes. Suppose the manual toll collection system is very efficient then for one vehicle to stop and pay taxes total time taken is 60 seconds. Then, time taken by 1 vehicle with 60 second average stop in a month is: 60 x 30 = 1800 seconds. Yearly total time taken = 1800 x 12 = 216200 seconds = 6.0 hours.
On average each vehicle that passes through the toll plaza has to wait 6.0 hours in engine start condition yearly. The figure is staggering if on an average we take 100 vehicles pass through the toll plaza each day, then yearly 36000 vehicles pass through the toll plaza. And each year 36000 vehicles just stand still for 6.0 hours in engine start condition thereby aiding pollution and wasting fuel and money. This study is if the system is very efficient but what if the vehicle has to wait for 5 minutes?

This is a figure considering one toll plaza. If considering 50 toll systems the above figure will drastically increase and the wastage of fuel, money will increase and pollution will also increase.

This system can be effectively implemented on a highway or freeway, where vehicle with a RFID tag will be allowed to pass by deducting an amount from the tag balance.

The above mentioned losses can put huge burden on Government and the citizens. Reducing these losses is the ample reason for which the need for automation system. The loss of time puts in a lot of frustration in everyone having to wait for their turn to pay the tax. Most of us want a speedy transport without any obstruction. When it is a known fact that oil is depleting day by day, just standing, waiting and wasting oil does not make any sense. Loss of fuel is most at reduced speed. So there is a need for continuous motion. When a number of vehicles have to wait nobody bothers to witch off the engines while waiting and so fuel emission is most at this level. This is a major contributor to the already increasing pollution.

So there is need for some e-commerce model which will cut down on every loss and make it possible to achieve a speedy and non obstructed transport.
My proposed model specified in figure 6.1 can be use for Toll Tax Booth. This chapter contains the implementation of this model with source code and screen shots. This implementation is done using Java language and MySQL database.

### 6.6.2 Source code of Model

**GridClient.java**

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

public class GridClient {

    // Default listening port, if none is specified on the command line.
    private static final int DEFAULT_PORT = 11111;

    // The first and only word on a message representing a shutdown command.
    private static final String CMD_TASK = "task";
    private static final String CMD_RESULT = "result";
    private static final String CMD_SHUT_DOWN = "shutdown";
    private static boolean IsShutdown;

    public static void main(String[] args) {
        // Get the port number from the command line, if present.
        int port = DEFAULT_PORT;
```
if (args.length > 0) {
    try {
        port = Integer.parseInt(args[0]);
        if (port < 0 || port > 65535)
            throw new NumberFormatException();
    }
    catch (NumberFormatException e) {
        port = DEFAULT_PORT;
    }
}

System.out.println("Listening at port number : " + port);

while (IsShutdown == false) {

    // Listen for a connection request from the master program.
    ServerSocket listener = null;
    try {
        listener = new ServerSocket(port);
    }
    catch (Exception e) {
        System.out.println("Can not create listening socket on port : " + port);
        System.exit(1);
    }

    try {
        Socket con = listener.accept();
listener.close();
handleConnection(con);
}
catch (Exception e) {
    System.out.println("Server shut down with error : " + e);
    System.exit(2);
}
}

System.out.println("Shutting down normally.");
}

private static GridTask readTask(String taskData) throws IOException
{
    try {
        Scanner sc = new Scanner(taskData);
        GridTask task = new GridTask();
        sc.next(); // Go to the start of the line.
        task.id = sc.nextInt();
        task.maxIterations = sc.nextInt();
        task.count = sc.nextInt();
        task.data = sc.next();
        task.partialResult = sc.nextInt();
        return task;
    }
    catch (Exception e) {
        throw new IOException("Error in reading task information - Illegal data found ");
    }
}
private static String writeResults(GridTask task) {
    StringBuffer buff = new StringBuffer();
    buff.append(CMD_RESULT);
    buff.append(' ');
    buff.append(task.id);
    buff.append(' ');
    buff.append(task.count);
    for (int i = 0; i < task.count; i++) {
        buff.append(' ');
        buff.append(task.results[i]);
    }
    buff.append(' ');
    buff.append(task.partialResult);
    return buff.toString();
}

private static void handleConnection(Socket con) {
    try {
        BufferedReader in = new BufferedReader( new InputStreamReader(con.getInputStream()) );
        PrintWriter out = new PrintWriter(con.getOutputStream());
        while (true) {
            String line = in.readLine(); // Message from the Grid Server.
            if (line == null) {
                // End of stream found
            }
        }
    } catch (IOException e) {
    }
}
throw new Exception("Unexpectedly Connection closed");

if (line.startsWith("close")) {
    // Normal termination of the connection.
    break;
}

else if (line.startsWith(CMD_SHUT_DOWN)) {
    // Normal termination of the connection and also inform to other clients to close the work
    System.out.println("Received shutdown command");
    IsShutdown = true;
    break;
}

else if (line.startsWith(CMD_TASK)) {
    // Represents a GridTask that this client is supposed to perform.
    GridTask task = readTask(line);
    // Retrive the message.
    task.compute(task.data);
    // Task performed here
    out.println(writeResults(task));
    // Send back results.
    out.flush();
}

else {
    // Illegal command
    throw new Exception("Illegal command");
GridClient2.java

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

public class GridClient2 {

    // Default listening port, if none is specified on the command line.
    private static final int DEFAULT_PORT = 11111;
```
// The first and only word on a message representing a shutdown command.
private static final String CMD_TASK = "task";
private static final String CMD_RESULT = "result";
private static final String CMD_SHUTDOWN = "shutdown";
private static boolean IsShutdown;

public static void main(String[] args) {

    // Get the port number from the command line, if present.
    int port = DEFAULT_PORT;

    if (args.length > 0) {
        try {
            port = Integer.parseInt(args[0]);
            if (port < 0 || port > 65535)
                throw new NumberFormatException();
        }
        catch (NumberFormatException e) {
            port = DEFAULT_PORT;
        }
    }

    System.out.println("Listening at port number : " + port);

    while (IsShutdown == false) {

        // Listen for a connection request from the master program.
ServerSocket listener = null;

try {
    listener = new ServerSocket(port);
}

catch (Exception e) {
    System.out.println("Can not create listening socket on port: " + port);
    System.exit(1);
}

try {
    Socket con = listener.accept();
    listener.close();
    handleConnection(con);
}

catch (Exception e) {
    System.out.println("Server shut down with error : " + e);
    System.exit(2);
}

System.out.println("Shutting down normally.");

private static GridTask2 readTask(String taskData) throws IOException {
    try {
        Scanner sc = new Scanner(taskData);
        

GridTask2 task = new GridTask2();
sc.next(); // Go to the start of the line.
task.id = sc.nextInt();
task.maxIterations = sc.nextInt();
task.count = sc.nextInt();
    task.data = sc.next();
task.partialResult = sc.nextInt();
    return task;
}
} catch (Exception e) {
    throw new IOException("Error in reading task information - Illegal data found ");
}
}

private static String writeResults(GridTask2 task) {
    StringBuffer buff = new StringBuffer();
    buff.append(CMD_RESULT);
    buff.append(' ');
    buff.append(task.id);
    buff.append(' ');
    buff.append(task.count);
    for (int i = 0; i < task.count; i++) {
        buff.append(' ');
        buff.append(task.results[i]);
    }
    buff.append(' ');
    buff.append(task.partialResult);
    return buff.toString();
private static void handleConnection(Socket con) {
    try {
        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        PrintWriter out = new PrintWriter(con.getOutputStream());
        while (true) {
            String line = in.readLine(); // Message from the Grid Server.
            if (line == null) {
                // End of stream found
                throw new Exception("Unexpectedly Connection closed");
            }
            else if (line.startsWith("close")) {
                // Normal termination of the connection.
                break;
            }
            else if (line.startsWith(CMD_SHUT_DOWN)) {
                // Normal termination of the connection and also inform to other clients to close the work
                System.out.println("Received shutdown command");
                IsShutdown = true;
                break;
            }
            else if (line.startsWith(CMD_TASK)) {

// Represents a GridTask2 that this client is supposed to perform.

GridTask2 task = readTask(line);

// Retrieve the message.

task.compute(task.data);

// Task performed here

out.println(writeResults(task));

// Send back results.

out.flush();

} else {

// Illegal command

throw new Exception("Illegal command");

}

} catch (Exception e) {

System.out.println("Client connection closed with error "+ e);
}

} finally {

try {

con.close(); // Make sure the socket is closed.
}

catch (Exception e) {

}

}
GridClient3.java

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

public class GridClient3 {

    // Default listening port, if none is specified on the command line.
    private static final int DEFAULT_PORT = 11111;

    // The first and only word on a message representing a shutdown command.
    private static final String CMD_TASK = "task";
    private static final String CMD_RESULT = "result";
    private static final String CMD_SHUT_DOWN = "shutdown";
    private static boolean IsShutdown;

    public static void main(String[] args) {

        // Get the port number from the command line, if present.
        int port = DEFAULT_PORT;

        if (args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);
                if (port < 0 || port > 65535)
                    throw new NumberFormatException();
            }
```
try {
    listener = new ServerSocket(port);
} 
catch (Exception e) {
    System.out.println("Can not create listening socket on port : " + port);
    System.exit(1);
}

try {
    Socket con = listener.accept();
    listener.close();
    handleConnection(con);
} 
catch (Exception e) {
    System.out.println("Server shut down with error : " + e);
    System.exit(2);
private static GridTask3 readTask(String taskData) throws IOException {
    try {
        Scanner sc = new Scanner(taskData);
        GridTask3 task = new GridTask3();
        sc.next();  // Go to the start of the line.
        task.id = sc.nextInt();
        task.maxIterations = sc.nextInt();
        task.count = sc.nextInt();
        task.data = sc.next();
        task.partialResult = sc.nextInt();

        return task;
    }
    catch (Exception e) {
        throw new IOException("Error in reading task information - Illegal data found ");
    }
}

private static String writeResults(GridTask3 task) {
    StringBuffer buff = new StringBuffer();
    buff.append(CMD_RESULT);

    // Code to write results
    return buff.toString();
}
buff.append(' ');  
buff.append(task.id);  
buff.append(' ');  
buff.append(task.count);  
for (int i = 0; i < task.count; i++) {
    buff.append(' ');  
    buff.append(task.results[i]);
}
buff.append(' ');  
buff.append(task.partialResult);
return buff.toString();

private static void handleConnection(Socket con) {
    try {
        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        PrintWriter out = new PrintWriter(con.getOutputStream());
        while (true) {
            String line = in.readLine(); // Message from the Grid Server.
            if (line == null) {
                // End of stream found
                throw new Exception("Unexpectedly Connection closed");
            }
            if (line.startsWith("close")) {
                // Normal termination of the connection.
            }
        }
    } catch (Exception e) {
        // Exception handling...
    }
}
break;
}

else if (line.startsWith(CMD_SHUT_DOWN)) {
    // Normal termination of the connection
    System.out.println("Received shutdown command");
    IsShutdown = true;
    break;
}

else if (line.startsWith(CMD_TASK)) {
    // Represents a GridTask that this client is supposed to perform.
    GridTask3 task = readTask(line);
    // Retrive the message.
    task.compute(task.data);
    // Task performed here
    out.println(writeResults(task));
    // Send back results.
    out.flush();
}
else {
    // Illegal command
    throw new Exception("Illegal command");
}
GridTask.java

```java
import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class GridTask {

    public int id; // Identifies this task.
    public int maxIterations; // Input for the computation.
    public int count;
    public String data;
```
public int dataCalculated;
public int partialResult;
public int[] results; // Holds the results

public void compute(String data)
{
    results = new int[count];
    for (int i = 0; i < count; i++)
        results[i] = id;
    partialResult = doTask(data);
}

public int doTask(String data)
{
    try
    {
        String value[] = data.split(" ", ");

        long uid=Long.parseLong(value[0]);
        float amount = Float.parseFloat(value[1]);
        float updatedBalance=0;
        ////check and update in database////
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String myDB = "jdbc:mysql://localhost:3306/gridmodel";
            Connection DBConn = DriverManager.getConnection(myDB, "root", "");
Statement st = DBConn.createStatement();

ResultSet rs = st.executeQuery("Select *
* from user where id="+uid);

if(rs.next())
{
    float f = Float.parseFloat(rs.getString("balance"));
    updatedbalance=f-amount;
    if(updatedbalance>=0)
    {
        if(st.executeUpdate("Update user set balance=balance-"+amount+"
where id=" + uid)==1)
        {
            System.out.println("Amount Deducted from Account");
            System.out.println("Updated Amount = "+(updatedbalance));
            DBConn.close();
        }
        else
        {
            System.out.println("Error in updating record");
        }
    }
}
System.out.println("Low Balance");
}
}
DBConn.close();
}
catch(Exception ex)
{
    System.out.println(ex);
}

/* store the summation into file */
return (int) updatedbalance;
}
catch(Exception e)
{
    System.out.println("Error : "+e.toString());
}
return 0;
}

GridTask2.java

import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class GridTask2 {

    public int id; // Identifies this task.
    public int maxIterations; // Input for the computation.
    public int count;
    public String data;
    public int datacalculated;
    public int partialResult;
    public int[] results; // Holds the results

    public void compute(String data)
    {
        results = new int[count];
        for (int i = 0; i < count; i++)
            results[i] = id;
        partialResult = doTask(data);
    }

    public int doTask(String data)
    {
        try
        {
            String value[] = data.split(",");

            long uid=Long.parseLong(value[0]);
            float amount = Float.parseFloat(value[1]);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
float updatedbalance=0;

///check and update in database///
try
{

Class.forName("com.mysql.jdbc.Driver");
String myDB =
"jdbc:mysql://localhost:3306/gridmodel";

Connection DBConn =
DriverManager.getConnection(myDB,"root","");

Statement st = DBConn.createStatement();

ResultSet rs = st.executeQuery("Select *
from user where id="+uid);

if(rs.next())
{
    float f =
Float.parseFloat(rs.getString("balance"));
    updatedbalance=f+amount;
    if(amount>0)
    {

if(st.executeUpdate("Update user set balance="+amount+"
where id=" + uid)==1)
{

System.out.println("Amount Recharge in Account");
}
System.out.println("Updated Amount = "+(updatedbalance));

DBConn.close();

else

{/*
 * store the summation into file */

FileWriter fw = new FileWriter("write2.txt",true);
fw.write("user id "+uid +" has recharged Rs.
"+amount+"\n");
fw.close();

/* store the summation into file */
return (int) amount;
}
catch(Exception e)
{
    System.out.println("Error : "+e.toString());
}
return 0;
}

GridTask3.java

import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class GridTask3 {

    public int id; // Identifies this task.
    public int maxIterations; // Input for the computation.
Chapter 6. Proposed Grid Computing model for E-Commerce Application

```java
public int count;
public String data;
public int datacalculated;
public int partialResult;
public int[] results;  // Holds the results

public void compute(String data)
{
    results = new int[count];
    for (int i = 0; i < count; i++)
        results[i] = id;
    partialResult = doTask(data);
}

public int doTask(String data)
{
    try
    {
        long uid=Long.parseLong(data);
        float updatedbalance=0;
        /////check and update in database///
        try
        {
...
```

```java
Class.forName("com.mysql.jdbc.Driver");
    String myDB = "jdbc:mysql://localhost:3306/gridmodel";
    Connection DBConn = DriverManager.getConnection(myDB,"root","");```
Statement st =
DBConn.createStatement();

ResultSet rs =
st.executeQuery("Select * from user where id=\"+uid\";

if(rs.next())
{
    float f =
Float.parseFloat(rs.getString("balance"));
    DBConn.close();
    updatedbalance=f;
}

DBConn.close();
}
catch(Exception ex)
{
    System.out.println(ex);
}

////check and update in database////

/* store the summation into file */
FileWriter fw = new
FileWriter("write3.txt",true);

fw.write("Balance of user id "+uid +" is "+updatedbalance+"\n");
fw.close();
/* store the summation into file */
return (int) updatedbalance;
}
catch(Exception e)
{
    System.out.println("Error : "+e.toString());
}
return 0;
}
_________________________________________

ReadData.java

_________________________________________

import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class ReadData
{
    static BufferedReader br1;
    public static void init()
    {
        try
        {

_________________________________________

/* store the summation into file */
return (int) updatedbalance;
}
catch(Exception e)
{
    System.out.println("Error : "+e.toString());
}
return 0;
}
br1 = new BufferedReader(new InputStreamReader(System.in));

try {
    String s = br1.readLine();
    return s;
} catch (Exception e) {
    System.out.println(e.toString());
    return null;
}

public static String readline() {
    try {
        String s = br1.readLine();
        return s;
    } catch (Exception e) {
        System.out.println(e.toString());
        return null;
    }
}

GridServer.java

import java.io.*;
import java.net.*;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.Scanner;

public class GridServer {

    private static final int DEFAULT_PORT = 15000;
    private static final String CMD_TASK = "task";
    private static final String CMD_RESULT = "result";
    private static ConcurrentLinkedQueue<GridTask> tasks;
    private static ConcurrentLinkedQueue<GridTask2> tasks2;
    private static ConcurrentLinkedQueue<GridTask3> tasks3;
    private static int service;
    private static int tasksCompleted;
    private static int rows, columns;
    private static int maxIterations;
    private static String status="";

    public static void main(String[] args) {

        //Get the Current time
        long startTime = System.currentTimeMillis();

        service = Integer.parseInt(args[0]);

        if(service==1)
            System.out.println("\n**************** PAY THE TOLL
TAX ******************")
        else if(service==2)
System.out.println("\n************************** RECHARGE ACCOUNT **************************");
else if(service==3)
    System.out.println("\n************************** ACCOUNT BALANCE **************************");

    int cnt=0;
    do
    {
        if(service==1)
            createJob(); // Create the Job for first service

        else if(service==2)
            createJob2(); // Create the Job for second service

        else if(service==3)
            createJob3(); // Create the Job for third service

        if(status.equals("exit"))
        {
            status="";
            break;
        }
    }

    if (args.length == 0) {
        System.out.println("Pass the commandline arguments");
    }
else { // Run a distributed computation.

    ClientConnection[] clients = new ClientConnection[args.length-1];
    ClientConnection2[] clients2 = new ClientConnection2[args.length-1];
    ClientConnection3[] clients3 = new ClientConnection3[args.length-1];

    for (int i = 1; i < args.length; i++) {
        // Create the client threads that communicate with the
        // GridClient programs. The threads start automatically
        // as soon as they are created.
        String host = args[i];
        int port = DEFAULT_PORT;
        int pos = host.indexOf(':');
        if (pos >= 0) {
            // The host string contains a ":", which should be
            // followed by the port number.
            String portString = host.substring(pos+1);
            host = host.substring(0,pos); // Remove port from host
            try {
                port = Integer.parseInt(portString);
            }
            catch (NumberFormatException e) {

            }
        }
    }
}
System.out.println("Error - "+e);
}
}

if(service==1)
    clients[i-1] = new ClientConnection(i, host, port);
else if(service==2)
    clients2[i-1] = new ClientConnection2(i, host, port);
else if(service==3)
    clients3[i-1] = new ClientConnection3(i, host, port);
}

if(service==1)
{
    for (int i = 0; i < args.length-1; i++) {
        // Wait for all the threads to terminate.
        while (clients[i].isAlive()) {
            try {
                clients[i].join();
            } catch (InterruptedException e) {
                    
            }
        }
    }
}
else if (service==2)
{
    for (int i = 0; i < args.length-1; i++) {
        // Wait for all the threads to terminate.
        while (clients2[i].isAlive()) {
            try {
                clients2[i].join();
            } catch (InterruptedException e) {
            }
        }
    }
}
else if (service==3)
{
    for (int i = 0; i < args.length-1; i++) {
        // Wait for all the threads to terminate.
        while (clients3[i].isAlive()) {
            try {
                clients3[i].join();
            } catch (InterruptedException e) {
            }
        }
    }
}
if (tasksCompleted != rows) {

Chapter 6. Proposed Grid Computing model for E-Commerce Application

// Not all of the tasks were completed. (Note: for a more robust program, the remaining tasks could be executed here directly.)

//System.out.println("Something went wrong. Only " + tasksCompleted);
//System.out.println("out of " + rows + " tasks were completed");
//System.exit(1);

}

cnt++;
while(1==1);

long elapsedTime = System.currentTimeMillis() - startTime;
System.out.println("Finished in " + (elapsedTime/1000.0) + " seconds ");

printData();

} // end main()

private static void createJob() {
    maxIterations = 10000;
    rows = 1;
    columns = 8000;
    
    tasks = new ConcurrentLinkedQueue/GridTask>();

    System.out.print("\nEnter the Account Number and Amout (e.g. 1,40) : ");
ReadData r = new ReadData();

r.init();
for (int j = 0; j < rows; j++) { // Add tasks to the task list.
    GridTask task;
    task = new GridTask();
    task.id = j;
    task.maxIterations = maxIterations;
    task.count = columns;
    task.data= r.readline();
    status=task.data;
    task.partialResult= 0;
    tasks.add(task);
}

synchronized private static void finishTask(GridTask task) {
    tasksCompleted++;
}

private static String writeTask(GridTask task) {
    StringBuffer buff = new StringBuffer();
    buff.append(CMD_TASK);
    buff.append(task.id);
    buff.append(task.maxIterations);
    buff.append(task.count);
    return buff.toString();
}
buff.append(' ');  
buff.append(task.data);  
buff.append(' ');  
buff.append(task.partialResult);  
buff.append(' ');  
return buff.toString();

d}  

private static void readResults(String data, GridTask task) throws Exception {
Scanner scanner = new Scanner(data);  
scanner.next(); // read results  
int id = scanner.nextInt();  
if (id != task.id)  
    throw new IOException("Wrong task ID in results returned by client");  

int count = scanner.nextInt();  
if (count != task.count)  
    throw new IOException("Wrong data count in results returned by client");  

    task.results = new int[count];    for (int i = 0; i < count; i++)  
            task.results[i] = scanner.nextInt();

    task.partialResult=scanner.nextInt();
}
private static void createJob2() {
    maxIterations = 10000;
    rows = 1;
    columns = 8000;

    tasks2 = new ConcurrentLinkedQueue<GridTask2>();

    System.out.print("Enter the Account Number and Amount (e.g. 1,40) : ");
    ReadData r = new ReadData();
    r.init();
    for (int j = 0; j < rows; j++) { // Add tasks to the task list.
        GridTask2 task2;
        task2 = new GridTask2();
        task2.id = j;
        task2.maxIterations = maxIterations;
        task2.count = columns;
        task2.data= r.readline();
        status=task2.data;
        task2.partialResult= 0;
        tasks2.add(task2);
    }
}

synchronized private static void finishTask2(GridTask2 task) {
    tasksCompleted++;
}
private static String writeTask2(GridTask2 task) {
    StringBuffer buff = new StringBuffer();
    buff.append(CMD_TASK);
    buff.append(' ');
    buff.append(task.id);
    buff.append(' ');
    buff.append(task.maxIterations);
    buff.append(' ');
    buff.append(task.count);
    buff.append(' ');
    buff.append(task.data);
    buff.append(' ');
    buff.append(task.partialResult);
    buff.append(' ');
    return buff.toString();
}

private static void readResults2(String data, GridTask2 task) throws Exception {
    Scanner scanner = new Scanner(data);
    scanner.next(); // read results
    int id = scanner.nextInt();
    if (id != task.id)
        throw new IOException("Wrong task ID in results returned by client");
    int count = scanner.nextInt();
    if (count != task.count)
        throw new IOException("Wrong data count in results returned by client");
}
```java
    task.results = new int[count];
    for (int i = 0; i < count; i++)
        task.results[i] = scanner.nextInt();

    task.partialResult = scanner.nextInt();
}

private static void createJob3() {

    maxIterations = 10000;
    rows = 1;
    columns = 8000;

    tasks3 = new ConcurrentLinkedQueue<GridTask3>();

    System.out.print("Enter the Account Number : ");
    ReadData r = new ReadData();
    r.init();

    for (int j = 0; j < rows; j++) {  // Add tasks to the task list.
        GridTask3 task;
        task = new GridTask3();
        task.id = j;
        task.maxIterations = maxIterations;
        task.count = columns;
```
```
task.data= r.readline();
status=task.data;
task.partialResult= 0;
tasks3.add(task);

} }

synchronized private static void finishTask3(GridTask3 task) {

tasksCompleted++;
}

private static String writeTask3(GridTask3 task) {

StringBuffer buff = new StringBuffer();
buff.append(CMD_TASK);
buff.append(' ');
buff.append(task.id);
buff.append(' ');
buff.append(task.maxIterations);
buff.append(' ');
buff.append(task.count);
buff.append(' ');
buff.append(task.data);
buff.append(' ');
buff.append(task.partialResult);
return buff.toString();
}
private static void readResults3(String data, GridTask3 task) throws Exception {
    Scanner scanner = new Scanner(data);
    scanner.next(); // read results
    int id = scanner.nextInt();
    if (id != task.id)
        throw new IOException("Wrong task ID in results returned by client");
    int count = scanner.nextInt();
    if (count != task.count)
        throw new IOException("Wrong data count in results returned by client");
    task.results = new int[count];
    for (int i = 0; i < count; i++)
        task.results[i] = scanner.nextInt();
    task.partialResult = scanner.nextInt();
}

private static class ClientConnection extends Thread {

    int id; // Identifies thread in output.
    String host; // The host to which this thread will connect.
    int port; // The port number to which this thread will connect.
}
ClientConnection(int id, String host, int port) {
    this.id = id;
    this.host = host;
    this.port = port;
    start();
}

public void run() {

    int tasksCompleted = 0;       // How many tasks this thread has handled.
    Socket socket;               // The socket for the connection.

    try {
        socket = new Socket(host, port); // open the connection.
    }
    catch (Exception e) {
        System.out.println("Thread " + id + " could not open connection to " + host + ":" + port);
        System.out.println("Error: " + e);
        return;
    }

    GridTask currentTask = null;
    GridTask nextTask = null;

    try {
        PrintWriter out = new PrintWriter(socket.getOutputStream());
    }
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
currentTask = tasks.poll();

if (currentTask != null) {
    // Send first task to the GridClient.
    String taskString = writeTask(currentTask);
    out.println(taskString);
    out.flush();
}

while (currentTask != null) {
    String resultString;
    try {
        resultString = in.readLine(); // Get results for currentTask.
    }
    catch (Exception e) {
        
        if (currentTask != null)
            tasks.add(currentTask);
        if (nextTask != null)
            tasks.add(nextTask);
        break;
    }

    if (resultString == null)
        throw new IOException("Unexpectedly Connection closed.");
if (!resultString.startsWith(CMD_RESULT))
    throw new IOException("Illegal string received from client.");

nextTask = tasks.poll(); // Get next task and send it to client.
if (nextTask != null) {
    /* Send nextTask to client before processing results for currentTask,
     so that the client can work on nextTask while the currentTask results are processed. */
    String taskString = writeTask(nextTask);
    out.println(taskString);
    out.flush();
}
readResults(resultString, currentTask);
if (currentTask.partialResult < 0)
    {  
        System.out.println("Low Balance"); 
    }
finishTask(currentTask); // Process results from currentTask.
    
tasksCompleted++;  
    currentTask = nextTask; // assign nextTask to currentTask.
nextTask = null;
}
out.println("close"); // Send close command to client.
out.flush();

}  
catch (Exception e) {
    System.out.println("Thread "+ id + " terminated");

    // Put uncompleted tasks, if any, back into the task list.
    if (currentTask != null)
        tasks.add(currentTask);
    if (nextTask != null)
        tasks.add(nextTask);
}
finally {
    try {
        socket.close();
    }
    catch (Exception e) {
    }
}

}

private static class ClientConnection2 extends Thread {

    int id;  // Identifies thread in output.
    String host;  // The host to which this thread will connect.
    int port;  // The port number to which this thread will connect.
Chapter 6. Proposed Grid Computing model for E-Commerce Application

ClientConnection2(int id, String host, int port) {
    this.id = id;
    this.host = host;
    this.port = port;
    start();
}

public void run() {

    int tasksCompleted = 0; // How many tasks this thread has handled.
    Socket socket; // The socket for the connection.

    try {
        socket = new Socket(host, port); // open the connection.
    }
    catch (Exception e) {
        System.out.println("Thread " + id + " could not open connection to " + host + ":" + port);
        System.out.println("Error: " + e);
        return;
    }

    GridTask2 currentTask = null;
    GridTask2 nextTask = null;

    try {

PrintWriter out = new PrintWriter(socket.getOutputStream());
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
currentTask = tasks2.poll();
if (currentTask != null) {
    // Send first task to the GridClient2.
    String taskString = writeTask2(currentTask);
    out.println(taskString);
    out.flush();
}
while (currentTask != null) {
    String resultString = in.readLine(); // Get results for currentTask.
    if (resultString == null)
        throw new IOException("Unexpectedly Connection closed.");
    if (! resultString.startsWith(CMD_RESULT))
        throw new IOException("Illegal string received from client.");
    nextTask = tasks2.poll(); // Get next task and send it to client.
    if (nextTask != null) {
        /* Send nextTask to client before processing results for currentTask,
         so that the client can work on nextTask while the currentTask results are processed.*/
        String taskString = writeTask2(nextTask);
        out.println(taskString);
        out.flush();
readResults2(resultString, currentTask);
finishTask2(currentTask); // Process results from currentTask.
tasksCompleted++;
currentTask = nextTask; // Assign nextTask to currentTask.
nextTask = null;
}
out.println("close"); // Send close command to client.
out.flush();
}

try {
    socket.close();
} catch (Exception e) {
    System.out.println("Thread " + id + " terminated because of an error");
    System.out.println("Error: " + e);
    e.printStackTrace();
}

// Put uncompleted tasks, if any, back into the task list.
if (currentTask != null)
    tasks2.add(currentTask);
if (nextTask != null)
    tasks2.add(nextTask);

finally {
    try {
        socket.close();
    }
catch (Exception e) {
}
}
}

private static class ClientConnection3 extends Thread {

    int id; // Identifies thread in output.
    String host; // The host to which this thread will connect.
    int port; // The port number to which this thread will connect.

    ClientConnection3(int id, String host, int port) {
        this.id = id;
        this.host = host;
        this.port = port;
        start();
    }

    public void run() {

        int tasksCompleted = 0; // How many tasks this thread has handled.
        Socket socket; // The socket for the connection.

        try {

socket = new Socket(host, port); // open the connection.
}

} catch (Exception e) {
    System.out.println("Thread " + id + " could not open connection to " + host + ":" + port);
    System.out.println("Error: " + e);
    return;
}

GridTask3 currentTask = null;
GridTask3 nextTask = null;

try {
    PrintWriter out = new PrintWriter(socket.getOutputStream());
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    currentTask = tasks3.poll();

    if (currentTask != null) {
        // Send first task to the GridClient3.
        String taskString = writeTask3(currentTask);
        out.println(taskString);
        out.flush();
    }

    while (currentTask != null) {
        String resultString;
        try
        {
        }
resultString = in.readLine(); // Get results
for currentTask.
}
catch (Exception e)
{
    if (currentTask != null)
        tasks3.add(currentTask);
    if (nextTask != null)
        tasks3.add(nextTask);
    break;
}

if (resultString == null)
    throw new IOException("Unexpectedly Connection closed.");

if (! resultString.startsWith(CMD_RESULT))
    throw new IOException("Illegal string received from client.");

nextTask = tasks3.poll(); // Get next task and send it to client.
if (nextTask != null) {
    /* Send nextTask to client before processing results for currentTask,
    so that the client can work on nextTask while the currentTask results are processed.*/
    String taskString = writeTask3(nextTask);
    out.println(taskString);
    out.flush();
readResults3(resultString, currentTask);

    System.out.println("Balance :
"+currentTask.partialResult);

    finishTask3(currentTask); // Process results from currentTask.
    tasksCompleted++;
    currentTask = nextTask; // Assign nextTask to currentTask.
    nextTask = null;
}

out.println("close"); // Send close command to client.
out.flush();
}

catch (Exception e) {
    System.out.println("Thread " + id + " terminated");

    // Put uncompleted tasks, if any, back into the task list.
    if (currentTask != null)
        tasks3.add(currentTask);
    if (nextTask != null)
        tasks3.add(nextTask);
}

finally {

    try {
        socket.close();
    }
catch (Exception e) {
}

private static void printData()
{
    System.out.println("Completed");
}

MySQL Database: gridmodel.sql

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

--
-- Database: 'gridmodel'
--

-- Table structure for table 'user'
CREATE TABLE IF NOT EXISTS 'user' (
    'id' bigint(20) NOT NULL,
    'uname' varchar(50) NOT NULL,
    'balance' float NOT NULL,
    PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table 'user'

--

INSERT INTO 'user' ('id', 'uname', 'balance') VALUES
(1, 'Milan Vachhani', 1243),
(2, 'Jatin Patel', 1070),
(3, 'Raj Doshi', 100),
(4, 'Manish Radadiya', 250),
(5, 'Arya Singhaniya', 2000),
(6, 'Piyush Kadivar', 2000),
(7, 'Ajay Ardershana', 1205);
6.6.3 Demonstration of Model

This model is designed for toll tax system. This application have 3 services

1. Pay the toll tax
2. Recharge the account
3. View balance

All the vehicles passes though toll tax need to pay tax for that. That tax can be deducted from the account of vehicle holder as per the first service. Account holder can recharge that account using second service. Also they can view the balance of account using third service.

For each above services we are going to join 2 clients in demonstration. We can join any number of clients with any service. When any task given to the server that task will be executed in one of the connected machine for that service.

Three services are GridClient, GridClient2 and GridClient3.
First two figures represent how machine can join to first service (GridClient). See the figure 6.2 and 6.3

**Figure 6.2:** Machine connected to GridClient with port 1001

**Figure 6.3:** Machine connected to GridClient with port 1002
Next two figures represent how machine can join to second service (GridClient2). See the figure 6.4 and 6.5.

**Figure 6.4:** Machine connected to GridClient2 with port 2001

**Figure 6.5:** Machine connected to GridClient2 with port 2002
Next two figures represent how machine can join to this services (GridClient3).
See the figure 6.6 and 6.7

**Figure 6.6:** Machine connected to GridClient3 with port 3001

**Figure 6.7:** Machine connected to GridClient3 with port 3002
Next three figures represent how server can start with connected machines. Any one of these three services can start in one of the machine and can execute that process in one of the connected clients. See the figure 6.8, 6.9 and 6.10 that is use to pay the toll tax, recharge the account and view the account detail respectively.

**Figure 6.8:** Grid Server for paying the toll tax

**Figure 6.9:** Grid Server to recharge the account
Chapter 6. Proposed Grid Computing model for E-Commerce Application

Figure 6.10: Grid Server to view the balance of account

Figure 6.11: All screens of services and clients
Figure 6.12: Pay the toll tax

Figure 6.13: If account don’t have enough balance then display warning message
Figure 6.14: Service to view the Balance of Account

Figure 6.15: View the balance of account
Figure 6.16: Service to recharge the account

Figure 6.17: Recharge the account
Figure 6.18: Pay the toll tax

Figure 6.19: After paying the toll tax, check the remaining balance
6.7 Conclusion

The research started with an aim to study, model generation and implementation for e-commerce application using Grid Computing. The study of the various grid computing model and e-commerce application provide me the suggestion and the need to generate a new model for toll tax system. In manual toll tax collection system, vehicles are waiting at toll tax booth for paying the tax. This is the waste of time and fuel. To overcome this problem, I have designed grid computing model that provides facility of paying tax using RFID technology, recharge of account and other facilities. The successful implementation of this system eliminates the wastage of time and fuel too. This model is designed using Java and Socket Programming and MySQL database. This model is developed to distribute the load between the connected clients. This model provides the facilities to pay the tax, recharge the account and view the detail of account.

This model has three components. These three components are Grid Server, Grid Client and Grid Task. User makes the request for any task. That task request
send to the Grid Server. Grid server sends the request to the Clients connected to that server for the requested task. It also gathers the response of the client and sends back to the user. Grid Service or Client provides the service to complete the task. Multiple clients perform the one task and send result back to the Grid Server. User requests for some particular task and that task request sends to the Grid Server and executed at different clients and result return to the user through Grid Server.

6.8 Limitation

Proposed model has three components. These three components are Grid Server, Grid Client and Grid Task. In this model each task is managed by one Grid Server, So every task request will go though Grid Server. So there is full dependency on Grid Server. Due to limitation of this intermediate Grid Server it become the limitation of whole system. This Grid Server need very high configuration. If this Grid Server crash then whole system will stop.

Proposed model is text based program, that means for every operation need to enter command or text, which is not very user friendly.

6.9 Future enhancement

The series of Grid Server can be developed to overcome the problem if one server is crash then other will take the place of it.

This proposed model can be developed GUI based in stead of text/command based system only.