CSE411: Distributed Computer Systems
Assignment (1)

Prepared By:

Salma Abdelfattah Fetouh Abdelfattah
Code: 1700622
Section: 2

Cairo, October 2021

Solution (i)

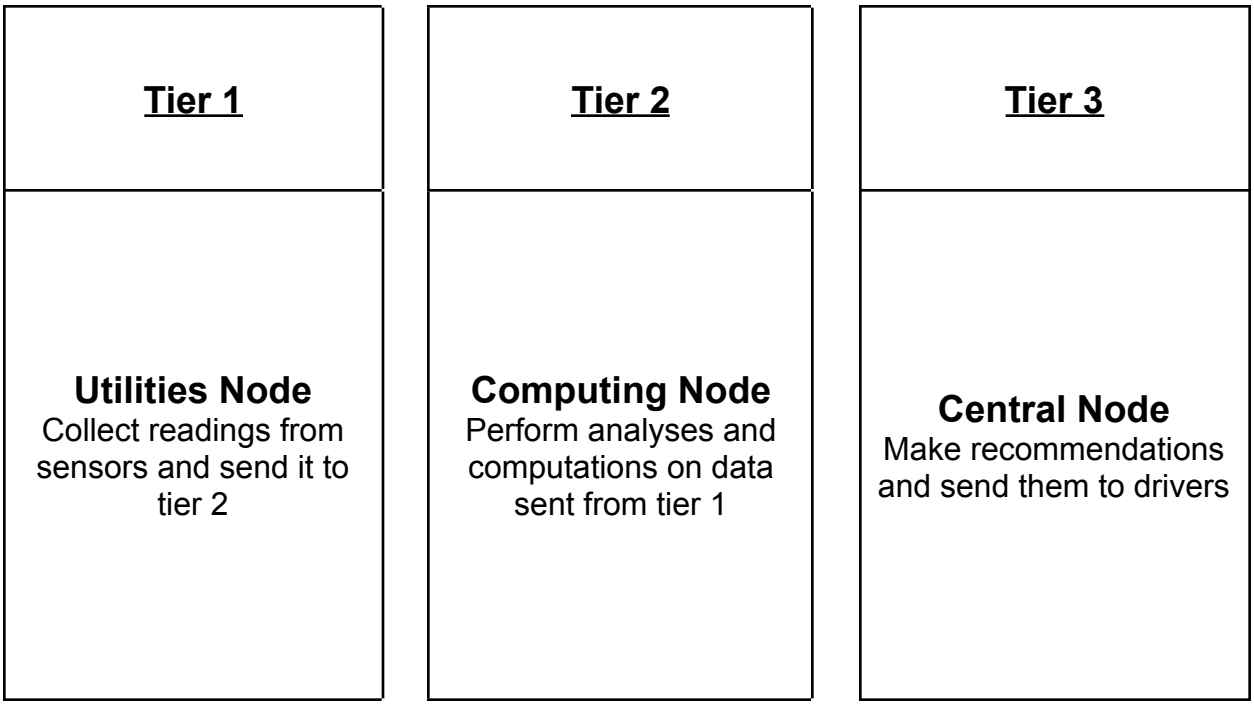
If mobile agents were used, we can assume three nodes in the system:

- Utilities nodes: from drivers, traffic lights, and sensors that convert analog signals to digital numbers and signals in order to be able to do computations and complex processes with them.
- Computation nodes: clients that collect data from the information and signals coming from the utilities nodes.
- Central nodes: servers in each area which are responsible for serving clients that collect data and process it from the clients.

These mobile agents are responsible for the process of transferring the data between utilities nodes & computation nodes in each area of the city, they also perform the necessary analyses on these collected data at the computation nodes, then they will transfer the results to central nodes after the necessary analyses and processing is done. The central nodes mobile agents are responsible for the recommendations of the best routes to take, they later send these data to the drivers.

Solution (ii)

The previous scenario can be implemented using a 3-tier architecture as follows:



Server Code

```
       try
        {
            //1.open server socket
            ServerSocket sv = new ServerSocket(1234);
            System.out.println("Server Running...");
            while (true)
            {
                //2.accept connection
                Socket s = sv.accept();
                System.out.println("Client Accepted...");
                //3.create I/O streams
                DataInputStream dis = new DataInputStream(s.getInputStream());
                DataOutputStream dos = new
DataOutputStream(s.getOutputStream());

                //4.perform IO with client
                while (true)
                {
                   // request the starting point
               dos.writeUTF("Please enter the starting point");
               dos.flush();
               String start = dis.readUTF();
               // request the destination
               dos.writeUTF("Please enter the destination");
               dos.flush();
               String destination = dis.readUTF();
               // Perform calculations to get the best route
               String best_route = getBestRoute(start, destination);
               dos.writeUTF("The best route would be " + best_route + "\n Start
Over? [y/n]?");
               dos.flush();
               String usr_choice = dis.readUTF();
               dos.flush();
               if (usr_choice.equalsIgnoreCase("n")) {
                   dos.writeUTF("bye");
                   System.out.println("Client disconnected");
                   dos.flush();
                   break;
               }
           }

           // close connection
           dis.close();
```

```
            dos.close();
            s.close();
 catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }

}
```

Client Code

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

public class Client {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            // create socket to connect to the server
            Socket s = new Socket("127.0.0.1", 8080);
            // Create I/O streams
            DataInputStream dis = new
DataInputStream(s.getInputStream());
            DataOutputStream dos = new
DataOutputStream(s.getOutputStream());

            // IO with server
            while (true) {
                // receive server command & print to user
                String srvr_msg = dis.readUTF();
                if (srvr_msg.equals("bye")) {
                    System.out.println("Session ended");
                    break;
                }
                System.out.println(srvr_msg);
                // take command from usr and send to the server
```

```
                String usr_msg = sc.next();
                dos.writeUTF(usr_msg);
                dos.flush();
            }
            // close the connections
            dis.close();
            dos.close();
            s.close();

        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Solution (iii)

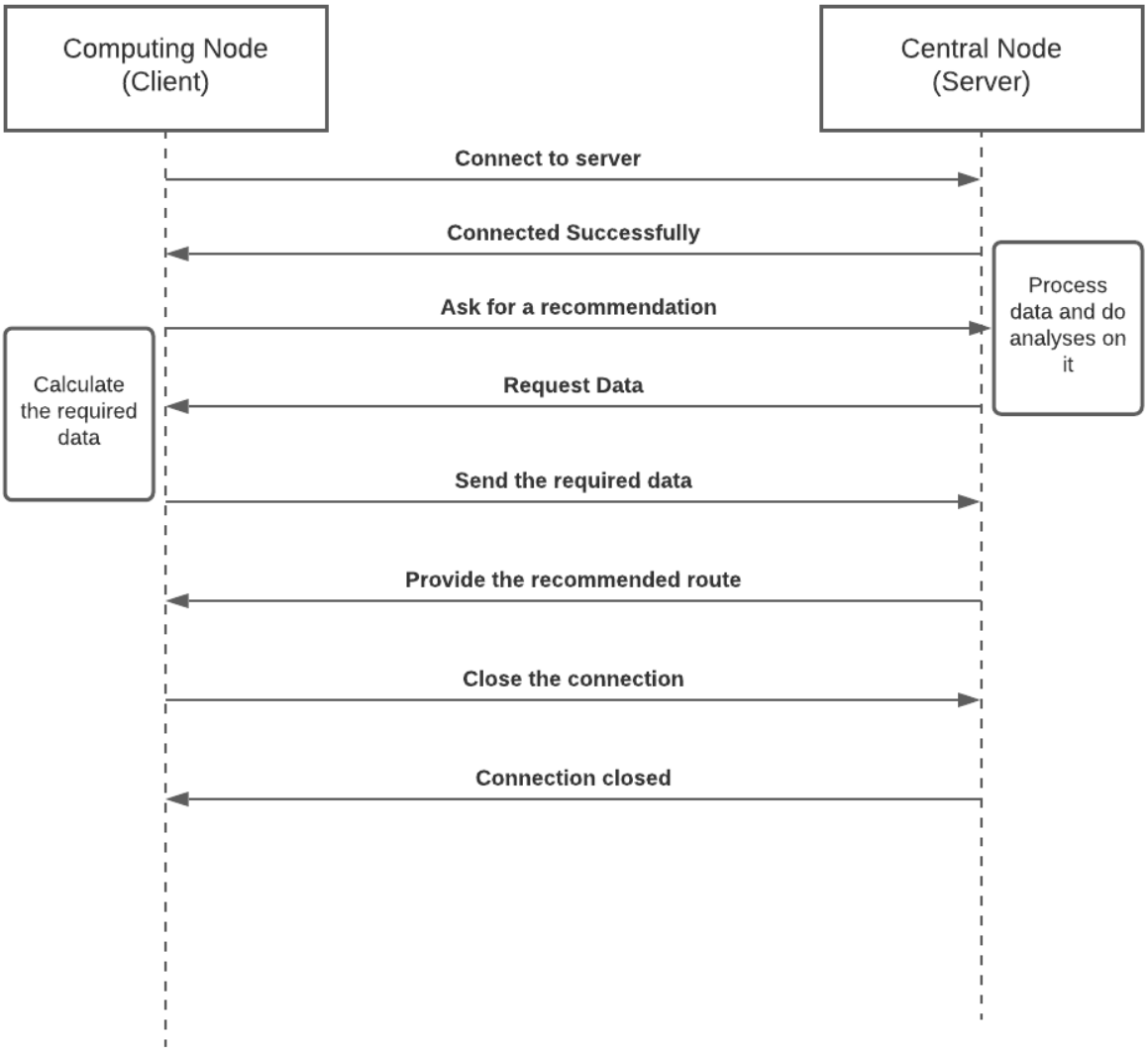


Figure 1-ALP diagram-designed by lucidchart

Solution (iv)

Repository Link: [Code in details with Readme]

Code Structure

The design uses socket programming and multithreading, it is composed of 5 classes:

- Server
- Client
- UtilityClient
- ClientHandler
- UtilityHandler

Server

Represents the central node, it waits for a connection with the client, then creates a thread for this client and delegates the communication to the ClientHandler class

ClientHandler

Responsible for handling each thread of each client separately.

Client

Represents the computing node, it connects the client to the server on port 8080, and connects to the utilities of the city (sensors, drivers, and traffic lights) on port 8090.

UtilityHandler

Responsible for handling each thread of each utility separately.

UtilityClient

Represents the utilities node, it connects the client to the utilities of the city (sensors, drivers, and traffic lights) on port 8090.

Other Notes

- In the code, I assume communication between one server, multiple clients and multiple utilities.
- The only utility node represented by the code is the sensor that gives the start and destination points of the driver.

References

- Javadocs 8
- Oracle documentations
- Sunsystems multithreading guide