P 1802

SINGLE SERVER VS DUAL SERVER SYSTEM FOR LOCATION BASED MOBILE QUERYING

P-1802

A PROJECT REPORT

Submitted by

BIPIN. K	**71203104006**
SHANKER LINGAM. M	**71203104042**
SIDDARTH RAVICHANDRAN	**71203104047**

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2007

P-1802

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**SINGLE SERVER VS DUAL SERVER SYSTEM FOR LOCATION BASED MOBILE QUERYING**" is the bonafide work of "**BIPIN. K, SHANKER LINGAM. M, SIDDARTH RAVICHANDRAN**" who carried out the project work under my supervision.

SIGNATURE

Prof. Dr. S. Thangasamy

HEAD OF THE DEPARTMENT

Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore – 641 006

SIGNATURE

Ms. S. Rajini, B.E.

SUPERVISOR

Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore – 641 006

The candidates with University Register Numbers **71203104006, 71203104042, 71203104047** were examined by us in the project viva-voce examination held on

24th APRIL 2007

INTERNAL EXAMINER

24/4/07

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Arutselvar Dr. N. Mahalingam, B.Sc, F.I.E.** and correspondent **Prof. K. Arumugam, B.E., M.S., M.I.E.**, for all their support and ray of strengthening hope extended.

We are immensely grateful to our principal **Dr. Joseph V. Thanikal, M.E., Ph.D., PDF., CEPIT.**, for his invaluable support to the come outs of this project.

We are extremely thankful to **Dr. S. Thangasamy.**, Head of the Department, Department of Computer Science and Engineering for his valuable advice and suggestions throughout this project,

We are indent to express our heartiest thanks to **Ms. S. Rajini, B.E.**, project coordinator and guide and **Ms. Amutha Venkatesh, M.E.**, project coordinator who have helped us to making the project work extremely well.

We are also thankful to all the faculty members of the department of Computer Science and Engineering, Kumaraguru College of Technology, CBE for their valuable guidance, support and encouragement during the course of our project work.

We express our humble gratitude and thanks to our parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout our project work.

ABSTRACT

Location based mobile querying is the process of retrieving information regarding the details of routes available to a given destination from the present location. It is an alternative to the GPS system present in a few nations. This system is made accessible through the SMS service available on mobile phones. The drawback with this system is that the response should reach the client within a very short period of time i.e., before the client moves out of the current mobile circle. If this is not done, then the response will have to be redirected to the corresponding mobile circle in which the client is present causing a considerable overhead.

In every system increasing the number of servers, increases the response time of the system. The most important aspect is by how much the response time decreases and what is the amount of change that is rendered to the system. It is important that we reduce the amount of change that we render to the system to increase the compatibility of the system. The response time also needs to be improved. To do this we need to improvise how the queries are submitted to the server. The network load is considerably reduced if the queries are formulated as a batch instead of single queries. Thus the clients submit queries in the form of batches rather than single queries.

Increasing the number of servers will improve the response time of the entire system, but the most important aspect that needs to be considered is that what the efficiency of the system is i.e., what is the increase in performance we get for adding a second server.

The increase in performance also depends on the number of queries that are submitted per batch and the number of batches at each server. In order to understand how the system responds to the above parameters, we need to analyze the response time for various values of the given parameters. In the real world however, this process is unnecessary as the values for the above specified parameters can be got by understanding the usage analysis of the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO	ABBREVIATION	EXPANSION
1	JDBC	Java Database Connectivity
2	TCP	Transmission Control Protocol
3	UDP	User Datagram Protocol
4	IPC	Inter Process Communication
5	SMS	Short Messaging Service

1. INTRODUCTION

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

In various developed countries in the world, wireless connectivity is used to the maximum. In the same category lies "Location Based Mobile Querying" application. A mobile subscriber can request his/her service provider, the route to his destination from his current location. This facility is either commercially available or free of cost service. This system employs a Single Server – Multiple client system architecture. Our project is aimed at proposing a new system which has 2 servers instead of one. During the course of the project, we have analyzed the cost – efficiency factor of the new system. Several parameters were varied and the efficiency of the system was measured and compared with the existing system. Reports were generated about the same and we have used them to conclude if the proposed system is worth implementing or not.

1.2 AIM OF THE PROJECT

The project is aimed at improving the efficiency of the existing system in location based mobile querying i.e., a basic client – server system, by using a dispatcher and an additional server. We have done it by setting up the proposed system and comparing its performance with the existing client – single server system and have generated reports to aid the cost – efficiency analysis of the proposed system.

During the course of the project, various efficiency characteristics were selected and reports generated by varying these characteristics in both systems and hence recommended the better of the two.

1.2 PURPOSE

The project improves the response time of the client – server system in location based mobile querying application. This is achieved using an additional server and a system called the dispatcher that will distribute queries effectively between the two servers.

Besides proposing the new system, we have done a cost – efficiency analysis of the new system over the existing system and used it to aid in the decision of changing over to the new system.

2. SYSTEM ANALYSIS

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Location based mobile querying is in use in countries like USA and Australia. Users of mobile phones can request their service providers the route to a particular place from the user's current location. The user sends an SMS to the service provider's tower (client) which contains code of the destination. The tower sends it to the service provider's location based querying server which uses its database to search the route and later returns the same to the user.

The existing system is a basic multiple client – single server system. Each mobile tower acts as a client. When it receives location based queries from individual mobile phones, it groups them into batches. These query batches are then passed on to the regional server, which is a single server system that processes the queries in each batch.

This involves, splitting every batch into individual queries, processing each query with the help of a database, combining together the responses to queries pertaining to a single batch and then transmitting them back to the clients that send the batch.

The existing system's performance may be affected due to network congestion, a deadlock situation in the server which may lead to the server not accepting new queries, loss of requests, congestion etc.

Further, the use of single database and the fact that all queries from clients need access to the same database may add to the lag in response time. If the delay in responding to the user is considerable, it will affect customer satisfaction and hence customer loyalty. Adding a

copy of the database will reduce the load on a single database and hence improve the response time.

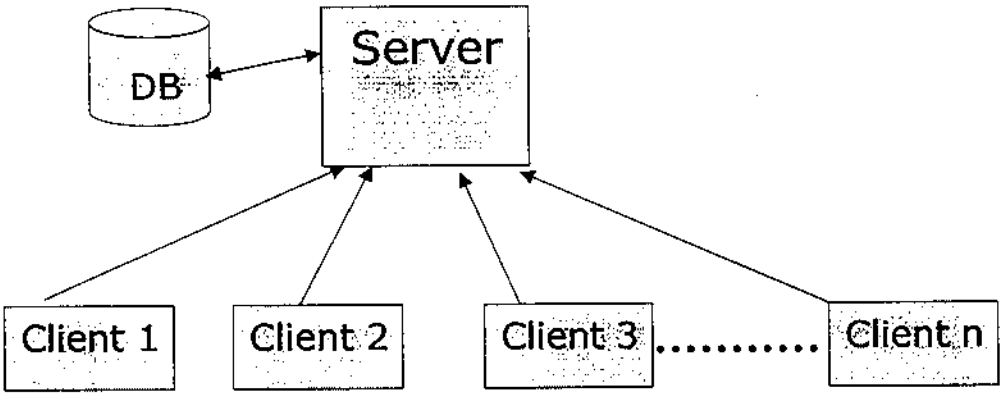


Fig 2.1.1 Existing System Model

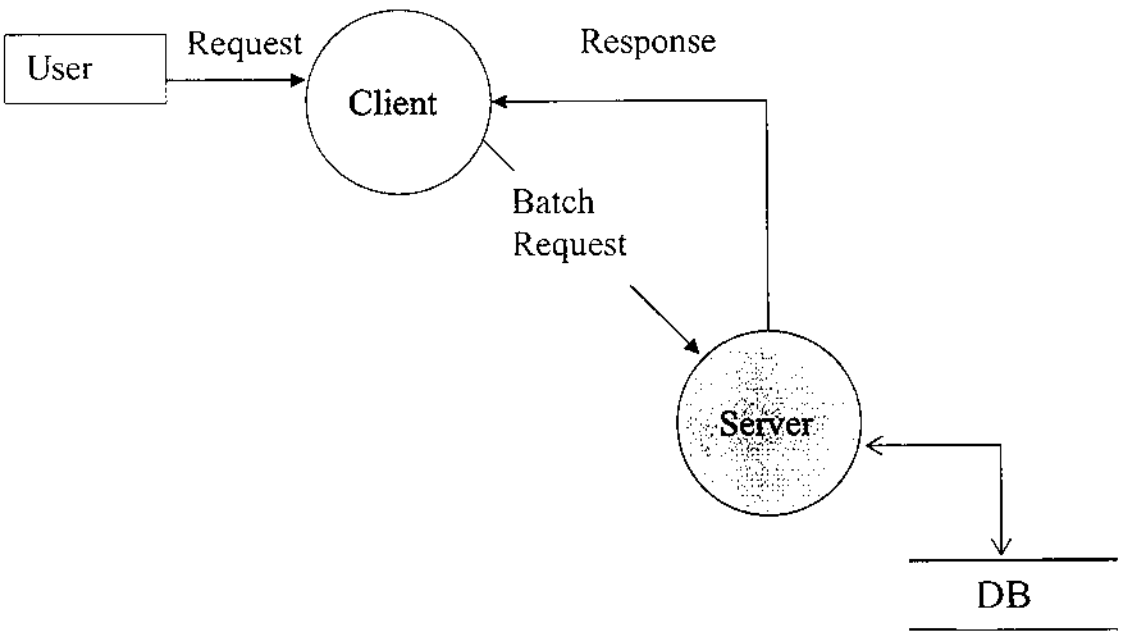


Fig2.1.2 Existing System – Design Overview

2.2 PROPOSED SYSTEM

The proposed system introduces the following changes in the existing system.

- A parallel server
- A copy of the database for the parallel server
- Dispatcher

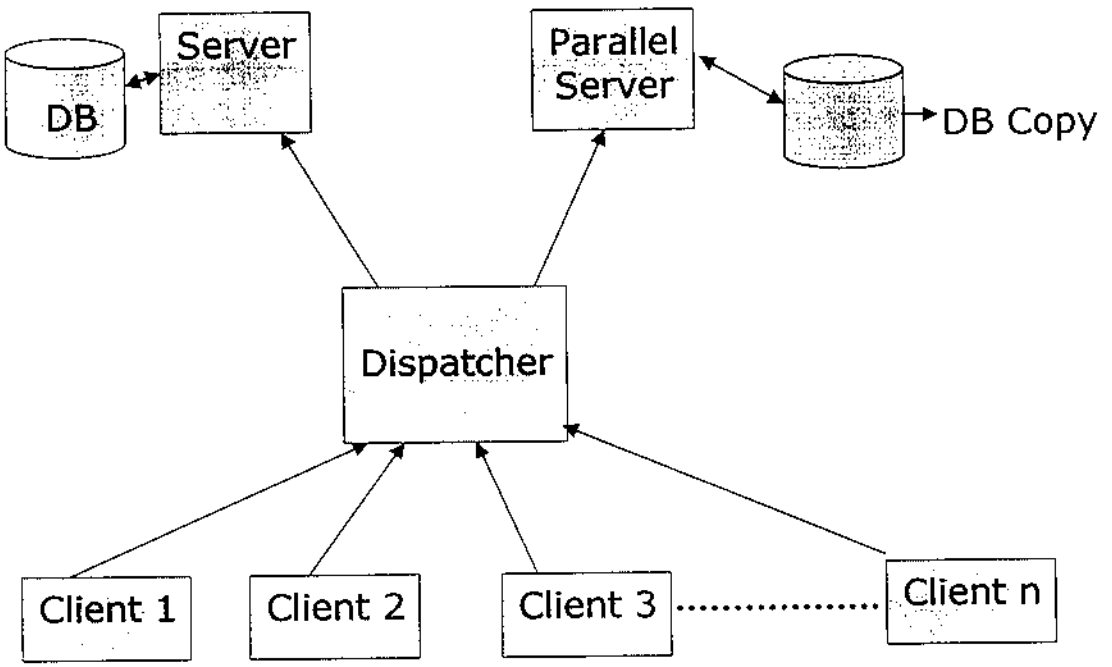


Fig 2.2.1 Proposed System Model

Parallel Server with Database

To share the load over a single server, the project proposes a parallel server with its own database, which is a copy of the main database. In this way, the requests from various clients are shared by the

two servers. This has various benefits like better response time, less network congestion, increased reliability and availability.

Since the load is shared between the servers, the load over the database is also shared. As there will not be regular updations or writes to the database in this particular application, the overhead of keeping the copy database updated with the original database is also negligible.

Dispatcher

With a parallel server and a database snapshot, the dispatcher acts as a dedicated system, that will keep track of the load on either of the two servers and accordingly direct the incoming query batches from clients to the appropriate server. As and when a server successfully responds to a client, it will simultaneously send a status report stating the same to the dispatcher. The dispatcher will keep track of when a query has been received, the client that sent it, its batch number etc. When a status report message from a server is obtained, the dispatcher updates its records accordingly.

The following diagram represents the overview of the design of the proposed system. The users send their query through an SMS to the service provider's mobile tower which acts as the client on behalf of the users. The Client waits for a specific number of queries and then puts the queries together in a single batch of queries and then forwards it to the dispatcher. The Dispatcher has its own logic of choosing what it has to do with the incoming batches and it initially puts the batches in a queue. When a server is not overloaded, it forwards the batches to the server. The server processes each batch by first separating the individual queries,

processing them, combining the responses of queries from the same batch together. The server then sends the responses directly to the client by-passing the dispatcher. Simultaneously, a status report stating the batch number, client name etc., is sent to the dispatcher so that the dispatcher can take note of the status and forward next query batch from its queue to the appropriate server.

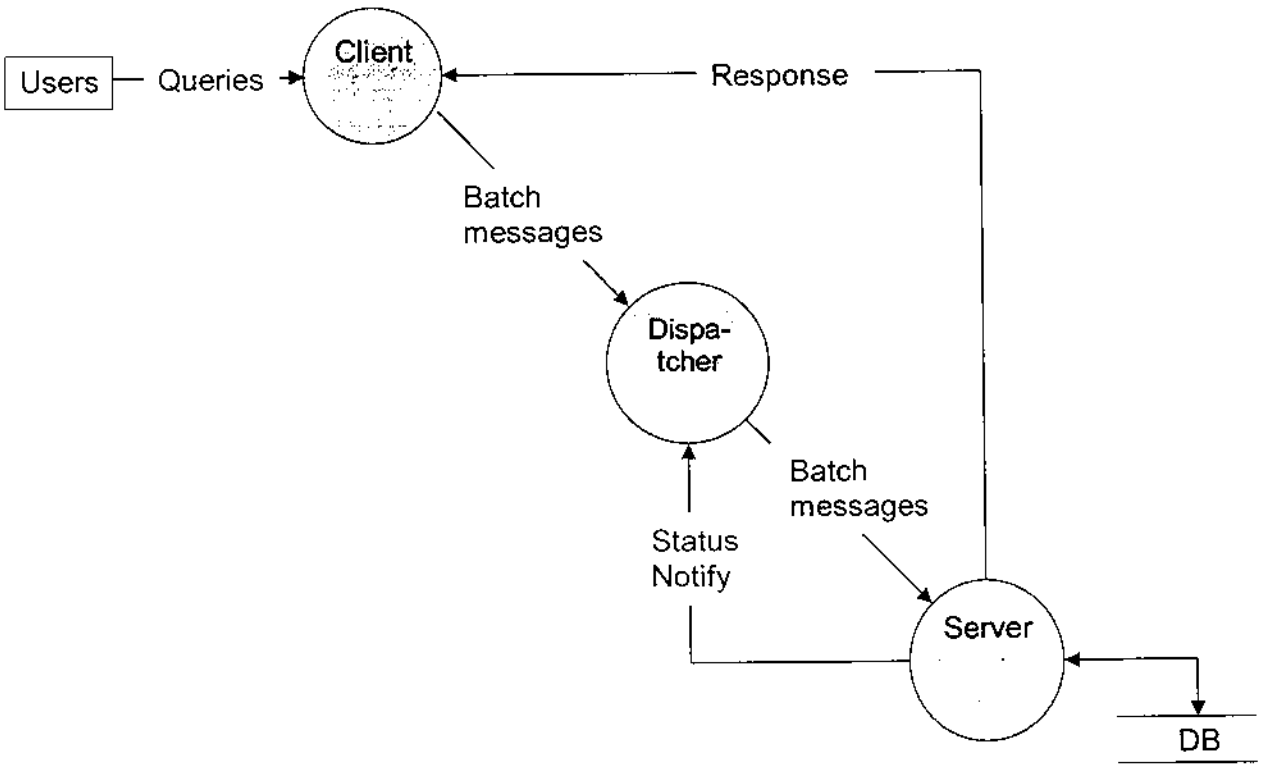


Fig 2.2.2 Dispatcher - Design Overview

3. SYSTEM SPECIFICATION

SYSTEM SPECIFICATIONS

3.1 HARDWARE CONFIGURATION:

- Monitors : Minimum 800 x 600 resolution with minimum 256 colors
- Memory : Approximately 256 MB of on-board memory
- I/O : Mouse, Standard 101-Keyboard
- Minimum Processor Speed : 500 MHz processor
- Processor Used: Pentium-4

3.2 SOFTWARE CONFIGURATION:

- Language : JDK 1.3 or above
- Operating System: Window 9x, 2000, NT, XP.
- Tools : MS Access 2000 or above
- TCP/IP Architecture

3.3 DOMAIN REQUIREMENTS FOR THE PROJECT

- Windows operating system fundamentals
- TCP/IP Architecture
- Multi-Threading Concepts
- Java Networking
- Client – Server Architecture

4. PROJECT IMPLEMENTATION

PROJECT IMPLEMENTATION

4.1 PROCESS MODULES

Modularity is the key feature of any system for provision of effective testing and debugging. Modularity helps in acquiring control over system development and modification. Our project comprises of 3 modules. They are

- Server Systems
- Dispatcher System
- Client System(s)

4.2 MODULE DESCRIPTION:

Server System:

The project uses two identical server systems. The overall function of the server systems is listed as follows:

i. Accept Request-batch:

The server listens to all incoming batch requests. In the existing Single Server System, the requests come from clients. In the proposed Dual Server System, the requests come from the Dispatcher system.

ii. Send Response-batch:

In both Single server as well as Dual server systems, the server(s) send their responses directly to clients. The incoming batch query message will contain the network address and the server(s) send the responses directly to the respective clients.

iii. Status Notify:

The server sends a message to the dispatcher as soon as a response is sent to a client. The message contains the details like the batch number of the request which has been responded to, the corresponding client's name, the timestamp etc.

The Server system also has a Report Writer running in parallel with the Server module that will timestamp events like query arrival, response dispatches etc. This report writer will help in report generation and we will use this as the major tool for comparing the proposed system with the existing one.

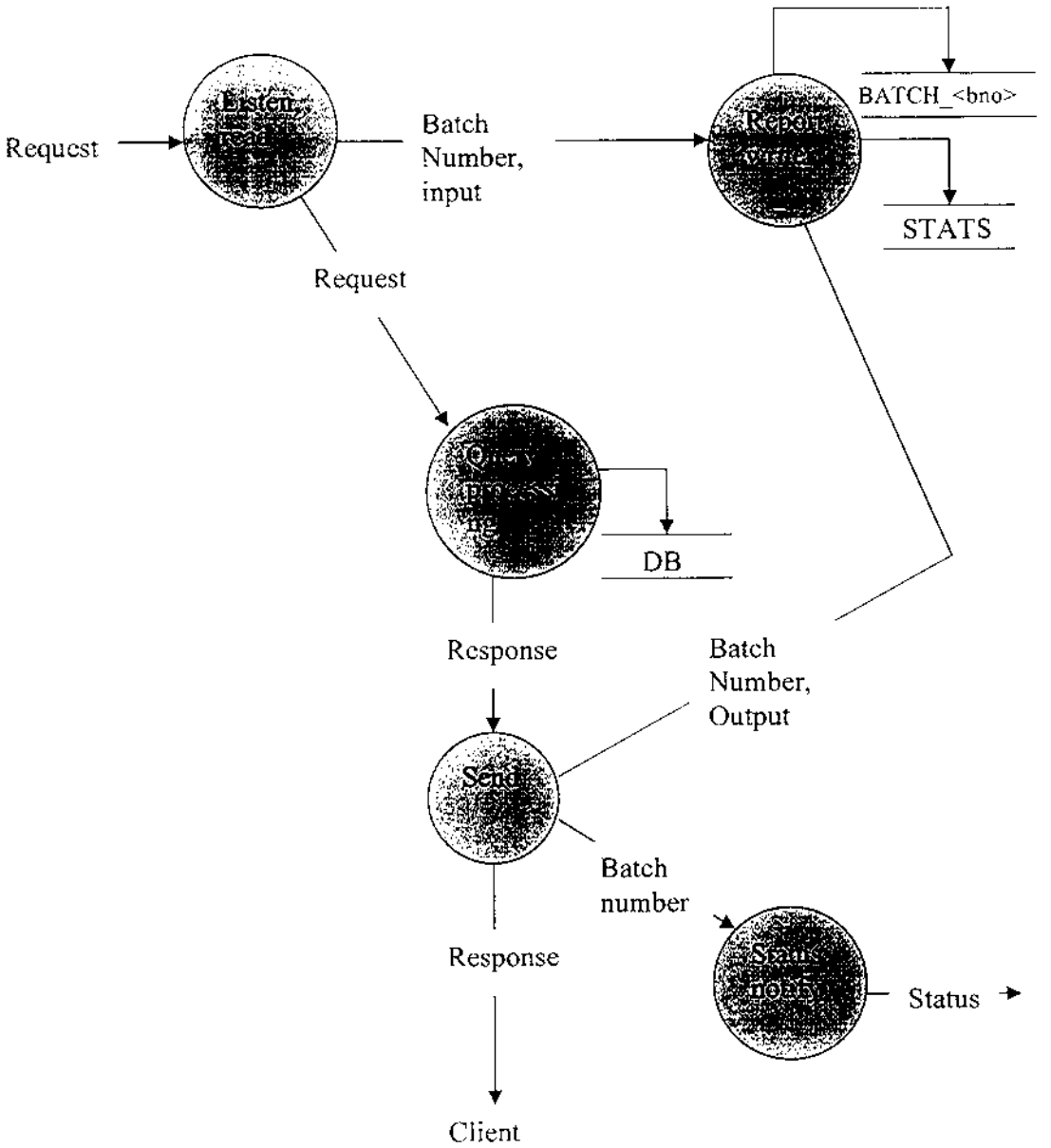


Fig 4.2.1 Server Module Design

Client System:

The client sends queries from user mobile phones to the server. This request has a header with relevant details. Sending a request to server for every individual query is a waste of network bandwidth and also the additional overhead of the header will also be a considerable factor of efficiency. A better alternative is to combine 'n' individual queries into one "batch". So one header is sufficient for one batch and the network bandwidth is more efficiently utilized.

There can be any number of clients for our project. The major tasks of a Client system are as follows:

i. Accept Queries:

Client system has a daemon that awaits any incoming queries from users, i.e., practically, the mobile tower waits for location based queries from users mobile phones.

ii. Formulate Request-batch:

The Client waits for 'n' queries from users. Once 'n' queries are available, the client concatenates these individual queries into a single batch request. The optimum value of 'n' will have to be found by studying the usage analysis of the system. However, a real time analysis for finding an optimum value of 'n' is beyond the scope of our project.

iii. Send Request-batch:

Once a batch request is available, the Client system adds a header to this batch which has details like the client's name, the server's network address etc. The format of a batch request is provided in the next section. Once a batch request is ready, the Client sends the request to the server (Single server system) or to the dispatcher (proposed dual server system)

iv. Receive Response-batch:

The client has a daemon that is always listening to incoming responses from server. It must be noted that in both the systems under consideration, the responses are received by clients directly from server.

Once the client system receives response from the server, it will split the responses according to the individual queries and send it back to the users however this part of the system too is beyond the scope of our project.

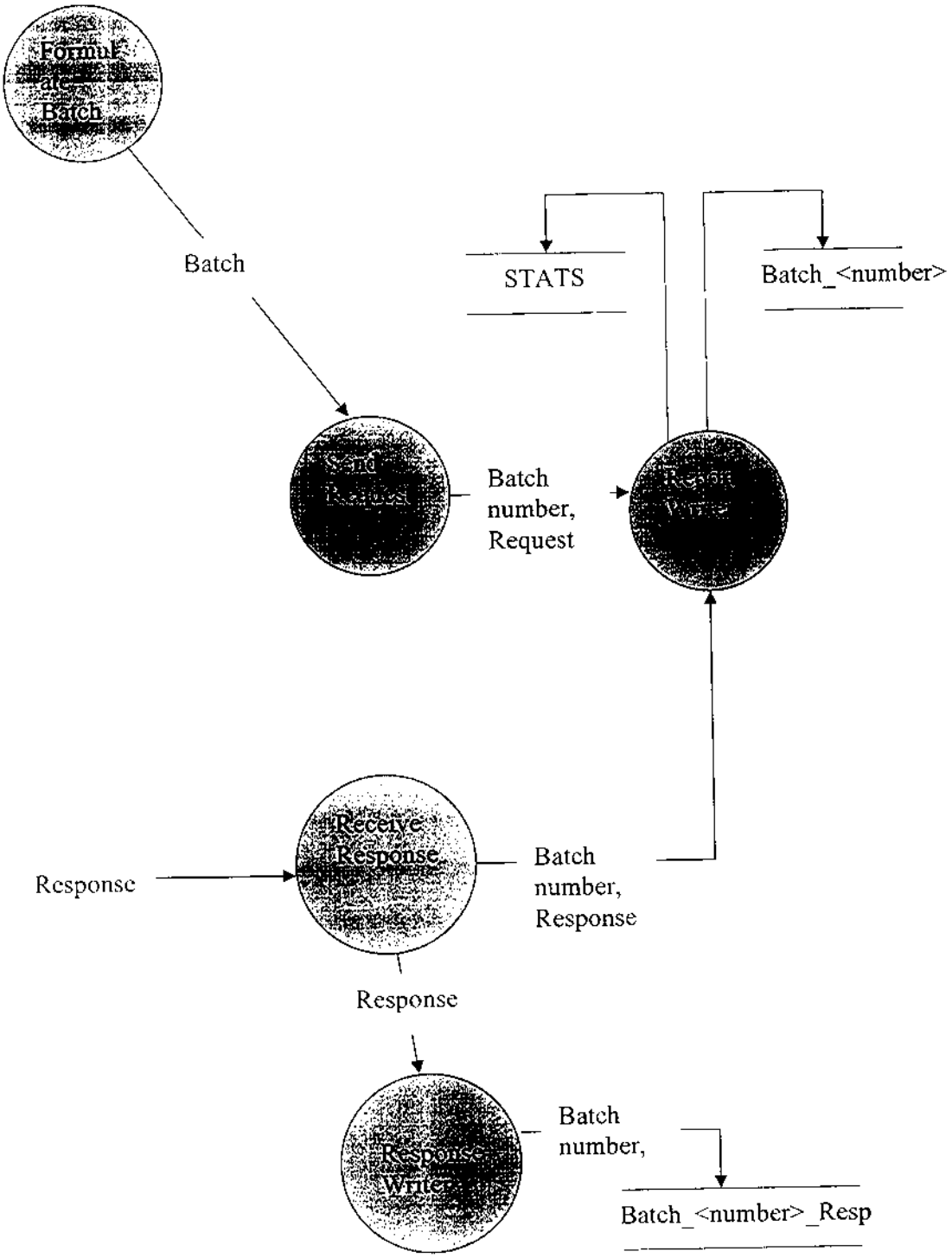


Fig 4.2.2 Client Module Design

Dispatcher System:

The dispatcher system is present only in dual server system. The tasks of a dispatcher are as follows:

i. Receive request-batch:

This is achieved using a daemon that listen to incoming requests-batches from clients. These requests are added to a batch queue.

ii. Server status tracking:

This daemon waits for any status message from either of the servers. When a message is received, the dispatcher will send a batch from the queue if a server has completed processing.

iii. Send request-batch:

When a batch-request is to be sent to one of the two servers, the dispatcher will add a new header with the particular server's network address.

iv. Manipulate queue:

The dispatcher system maintains a batch queue that will add a newly arrived query to the end of the queue and will send the batch at the start of the queue to the server when the

server is free. If both servers are idle, this queue will be empty.

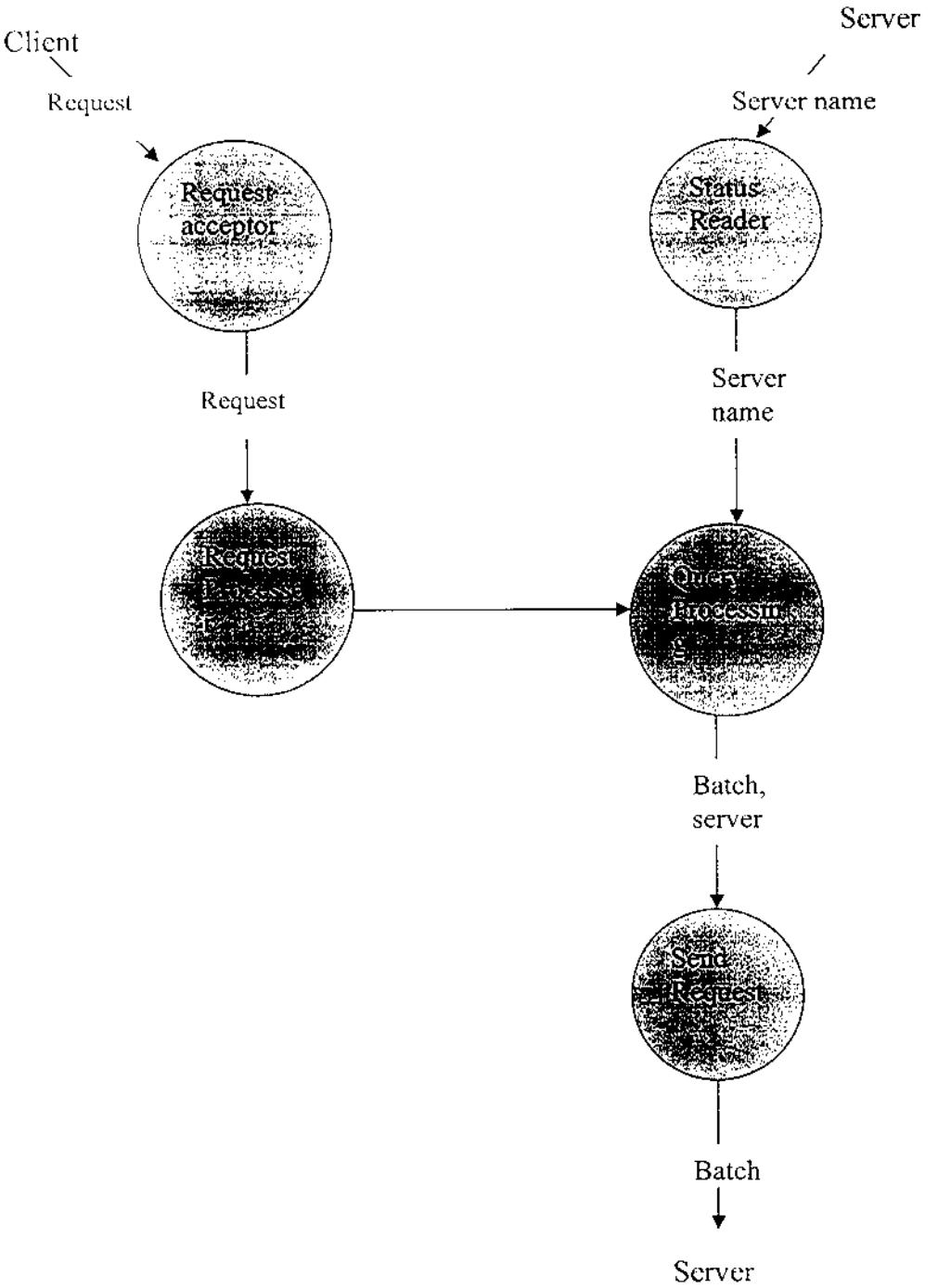


Fig 4.2.3 Dispatcher Module Design

4.3 MODULE IMPLEMENTATION:

We have used both TCP and UDP connection protocols for the connections between server, client and dispatcher. The connection type between Server and Dispatcher is TCP as there is a single channel needed for it. The type of connections between Clients and Server is UDP.

Message Formats:

For the various messages that will be transmitted through the network, we have adopted the following format.

Request:

C <Client n/w address> n <Client name> p <port number> b <batch number> q <query 1> q <query 2> q <query 3>.....

Response:

B <Batch number> r <response 1> r <response 2> r <response 3>.....

Status: (per every 2 batches)

S <Server name> c <client name> b <batch number1> b <batch number2>

❖ Server

To implement the server, we have split the tasks into the following basic tasks. The corresponding parallel execution tasks of the report writer has also been highlighted

Table 4.3.1 Server Tasks

S.No	Server Task	Report Writer Task
1.	Listen	---
2.	Accept & Read	Write receive time
3.	Split individual queries	---
4.	Query Database	---
5.	Combine Responses	---
6.	Send	Write Response Time
7.	Status Notify (in dual server system only)	---

The following are the list of classes along with the functions they perform in the server.

i. InputReader

It is a daemon that receives requests from the client/dispatcher.

ii. ClientHandler

It invokes a new, unnamed process for every request that is received by the server.

iii. ProcessHandler

It passes the requests to other functions and acts as a controller

iv. HandlerSupport

It does the process of extracting the client address, port number, batch number and requests from the batch.

v. ReportWriter

It does the process of time stamping whenever a process arrives or whenever a process is completed.

vi. QueryProcessing

This class splits the queries from the batch, processes it and collects the responses into a batch.

vii. AccessJDBCUtil

This provides a connection to the actual database.

viii. ResponseDispatcher

This sends the response batches to the respective clients.

ix. StatusNotifier

This class notifies the dispatcher about the completion of processing a request.

❖ Client

To implement the client, we have split the tasks into the following basic tasks. The corresponding parallel execution tasks of the report writer has also been highlighted

Table 4.3.2 Client Tasks

S.No	Client Task	Report Writer Task
1.	Generate Random Codes	---
2.	Formulate Batch	---
3.	Send Request	Write Send Time
4.	Receive Response	Write Receive Time

The following are the list of classes along with the functions they perform in the client.

i. Client

It establishes a UDP connection with the server/dispatcher and reads the requests from the user.

ii. ClientReportWriter

It records the time whenever a request is sent or a response is received.

iii. ClientSupport

This is the daemon which is used to receive the responses from the server.

iv. ClientHandlerSupport

It is used to extract the batch number from the received response.

❖ Dispatcher

The following are the list of classes along with the functions they perform in the Dispatcher.

i. RequestAcceptor

This daemon accepts the request from the client and invokes the RequestAcceptorSupport class.

ii. RequestAcceptorSupport

This adds the incoming requests to a queue and spawns a new unnamed process to schedule the requests.

iii. Scheduler

This performs the actual task of scheduling and dispatching the requests to the server.

iv. Universal

This is a class containing static functions that helps in IPC between the scheduler and status reader.

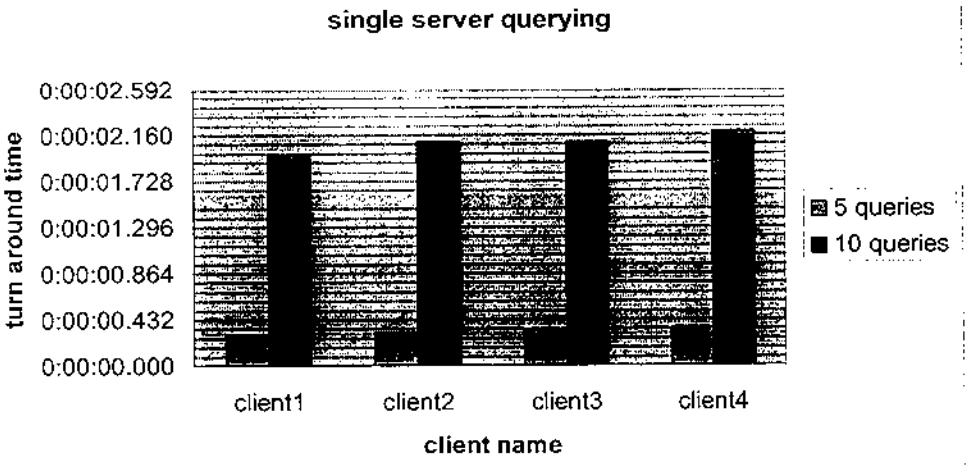
v. StatusReader

This daemon reads the notification messages from the servers and returns the communication to the scheduler.

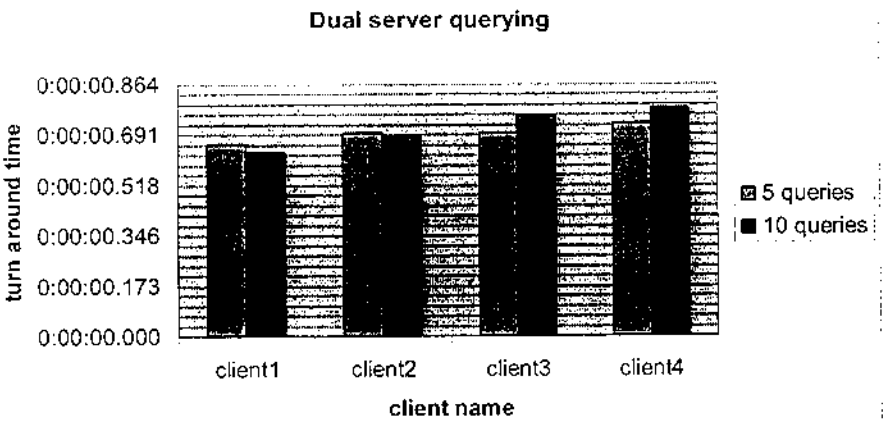
5. REPORTS

REPORTS

Client number	1	2	3	4
no of queries				
5	0:00:00.287	0:00:00.312	0:00:00.343	0:00:00.357
10	0:00:01.974	0:00:02.095	0:00:02.095	0:00:02.192



Client number	1	2	3	4
no of queries				
5	0:00:00.658	0:00:00.698	0:00:00.698	0:00:00.731
10	0:00:00.631	0:00:00.685	0:00:00.755	0:00:00.785



6. CONCLUSION

CONCLUSION

The system was found to perform within the expected parameters satisfactorily. It successfully managed to overcome some of the drawbacks of the existing single server systems, while still remaining wholly self-contained. The system was found to be robust under test as well as in working conditions. All instances of technical glitches were debugged and rectified to a large extent.

The proposed dual server system was found to be more efficient than the single server systems in cases where the number of queries per batch was 10 or above. However the response time was found to be higher in cases where the number of queries per batch was 5. Thus in cases where the number of queries per batch is large, the proposed system give us a significant increase in performance.

7. APPENDIX

APPENDIX

SAMPLE CODE

SERVER MODULE

/*AccessJDBCUtil.java*/

```
//:codes:AccessJDBCUtil.java
//This class deals with the connection to the database.
//package project;

import java.sql.*;

public class AccessJDBCUtil {
   private static final String accessDBURLPrefix = "jdbc:odbc:Driver={Microsoft Access
Driver (*.mdb)};DBQ=";
   private static final String accessDBURLSuffix = ";DriverID=22;READONLY=true}";
   static {
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      } catch(ClassNotFoundException e) {
         System.err.println("JdbcOdbc Bridge Driver not found!");
         // ABORT ABORT... How? System.exit(1) is not nice from webapp...
      }
   }

   /** Creates a Connection to a Access Database
    *@param filename the name of the database file for which a connection needs to be
created.
    *@return a connection to the specified database.
    *@exception exceptions catches SQLException.
   */
```

```
  public static java.sql.Connection getAccessDBConnection(String filename) throws
SQLException {
    filename = filename.replace('\\', '/').trim();
    String databaseURL = accessDBURLPrefix + filename + accessDBURLSuffix;
    // System.err.println("Database URL: " + databaseURL);
    return DriverManager.getConnection(databaseURL, "", "");
  }
} ///:~


/*ClientHandler.java*/
//:codes:ClientHandler.java
//this class deals with the clients that are connected with the server
//package project;

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

public class ClientHandler extends Thread
{
        Socket socket;
        DataInputStream in;
        PrintWriter out;
        InputReader server;

        public ClientHandler(Socket s, InputReader ser) throws IOException
        {
                this.socket = s; //reads the dynamically created port of the client
                this.server = ser; //reference to the server
                in = new DataInputStream(socket.getInputStream());
                out = new PrintWriter(socket.getOutputStream(), true);
```

```
    }

    public void run()
    {
        try{
            for(;;)
            {
                String msg = in.readLine(); //read msg from  port
                if(msg == null)
                {
                    break;
                }
                server.processRequest(msg);
            }
        }catch(IOException e){
            System.out.println(e);
        }
        finally{
            try{ socket.close();
            }catch(IOException e){
                System.out.println(e);
            }
        }
    }

    public String toString()
    {
        return socket.getInetAddress().getHostName(); //to convert the host name
to a string
    }
```

```
}
```

/*HandlerSupport.java*/

```
//:codes:HandlerSupport.java
//this class provides subordinate functions to the ProcessHandler class
//package project;

import java.util.*;

public class HandlerSupport
{
        String key; //to read the key or the field
        String val; //to store the value of the corresponding key or field

        /**this method extracts the address of the client from the message
         *@param message the actual message as received by the Process Handler
         *@return returns a string which holds the client address
         *@exception does not throw any exception
        */
        public String getClientAddress(String message)
        {
                StringTokenizer sToken = new StringTokenizer(message," "); //tokenize
using " "(space) as delimiter
                while(sToken.hasMoreTokens()) //read all tokens
                {
                        key = sToken.nextToken(); //read the key
                        val = sToken.nextToken(); //read the value
                        if(key.equalsIgnoreCase("c")) //check specifically for client
address field
```

```
                    break;
            }
            return val; //return the value of client address field
        }

        /**this method extracts the batch number of the request from the message
         *@param message the actual message as received by the Process Handler
         *@return returns a string which holds the batch number of the request
         *@exception does not throw any exception
        */
        public String getBatchNumber(String message)
        {
            StringTokenizer sToken = new StringTokenizer(message," "); //tokenize
using " "(space) as delimiter
            while(sToken.hasMoreTokens()) //read all tokens
            {
                key = sToken.nextToken(); //read the key
                val = sToken.nextToken(); //read the value
                if(key.equalsIgnoreCase("b")) //check specifically for batch
number field
                    break;
            }
            return val; //return the value of batch number field
        }

        /**this method extracts the name of the client from the message
         *@param message the actual message as received by the Process Handler
         *@return returns a string which holds the client name
         *@exception does not throw any exception
        */
        public String getClientName(String message)
```

```
        {
                StringTokenizer sToken = new StringTokenizer(message," "); //tokenize
using " "(space) as delimiter
                while(sToken.hasMoreTokens()) //read all tokens
                {
                        key = sToken.nextToken(); //read the key
                        val = sToken.nextToken(); //read the value
                        if(key.equalsIgnoreCase("n")) //check specifically for client name
field
                                break;
                }
                return val; //return the value of client name field
        }

        /**this method extracts the port of the client from the message
         *@param message the actual message as received by the Process Handler
         *@return returns a string which holds the client port
         *@exception does not throw any exception
        */
        public String getClientPort(String message)
        {
                StringTokenizer sToken = new StringTokenizer(message," "); //tokenize
using " "(space) as delimiter
                while(sToken.hasMoreTokens()) //read all tokens
                {
                        key = sToken.nextToken(); //read the key
                        val = sToken.nextToken(); //read the value
                        if(key.equalsIgnoreCase("p")) //check specifically for client port
field
                                break;
                }
```

```
            return val; //return the value of client port field
        }
}///:~
```

/*InputReader.java*/

```
//:codes:InputReader.java
//This class deals with receiving clinent requests and pass them to a separate client
handler
//package project;

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

public class InputReader
{
        ServerSocket serverSocket; //create a socket to accept the connections
        int port = 4207;

        public InputReader() throws IOException
        {

                serverSocket = new ServerSocket(port); //specify the socket number
                try
                {
                        for(;;)
                        {
```

```
                    Socket soc = serverSocket.accept(); //accept any incoming
client requests
                    (new ClientHandler(soc, this)).start(); //create a new client
handler for each of the clients
                }

            } finally {
                    serverSocket.close();
                }

        }

        /**invokes the methods that process the request
         *@param msg specifies the request read from the network
         *@exception no exceptions caught
        */
        public void processRequest(String msg)
        {
            ProcessHandler handle = new ProcessHandler(); //create an object for the
process handler class
            handle.processRequest(msg,port); //send the message to the process
handler to process the request
        }

        public static void main(String args[]) throws IOException //starts the server
        {
            InputReader tc = new InputReader();
        }
}
```

```
/*ProcessHandler.java*/
//:codes:ProcessHandler.java
//this class receives the request from the Input Reader class and performs the processing
//package project;

import java.io.*;
import java.util.*;
import java.sql.*;

public class ProcessHandler extends Thread
{
        String batchRequest; //holds the request from the client
        String clientAddress; //holds the address of the client
        String clientPort; //holds the port number to which the response is to be sent
        String batchNumber; //holds the batch number of the request
        LinkedList queries; //holds a list of the queries
        LinkedList responses; //holds a list of the responses
        String batchResponse; //holds the responses of the complete batch as a single
string
        QueryProcessing qProcessor; //an object to invoke the methods from the Query
Processing class
        String clientName; //holds the name of the client

        /**this method initializes all the required values
         *@param message the request that is received from the client
         *@exception catches IO Exception
        */
        public void initialize(String message)
```

```
{
        HandlerSupport fieldExtractor = new HandlerSupport(); //create an object
for the Handler Support in order to extract the various fields
        batchRequest = message; //initialize the batch request
        clientAddress = fieldExtractor.getClientAddress(batchRequest); //extract
the address of the client from the request
        clientPort = fieldExtractor.getClientPort(batchRequest); //extract the port
number of the client from the request
        batchNumber = fieldExtractor.getBatchNumber(batchRequest); //extract
the batch number from the request
        clientName = fieldExtractor.getClientName(batchRequest); //extract the
name of the client from the request
        qProcessor = new QueryProcessing(); //initialize the query processor
        ReportWriter writer = new ReportWriter(); //initialize the Report
Generator
        try
        {

    writer.writeFile(Integer.parseInt(batchNumber),"request",clientName); //log the
current state using the report writer class
        }
        catch(IOException e)
        {
            System.out.println("Unable to write to file");
            System.out.println(e);
        }
}

/**this method takes care of the query processing
 *@exception catches SQL Exception
*/
```

```
public void processQuery()
{
        queries = new LinkedList(); //initialize the query list
        try
        {
                queries = qProcessor.extractQuery(batchRequest,clientName);
//extract the individual queries from the requested batch
                System.out.println(queries);
                responses = qProcessor.queryTable(queries); //perform the actual
querying of the table
                System.out.println(responses);
                batchResponse = qProcessor.concatenateResponse(responses);
//concatenate all the reaponses as a single string to send as a response
        }
        catch(SQLException e)
        {
                System.out.println("Error in DataBase connection");
                System.out.println(e);
        }
}

/**this method invokes the Response Dispatcher to send the responses
 *@exception catches IO Exception
*/
public void sendResponse(int port)
{
        ResponseDispatcher sendResponse = new ResponseDispatcher(); //create
an object for the response dispatcher

sendResponse.initialize(batchResponse,clientAddress,clientPort,batchNumber);
//initialize the response dispatcher
```

```
            System.out.println(batchResponse);
            ReportWriter writer = new ReportWriter(); //create an object for the report
generator
            try
            {

      writer.writeFile(Integer.parseInt(batchNumber),"response",clientName); //log the
current status
            }
            catch(IOException e)
            {
                  System.out.println("Unable to write to file");
                  System.out.println(e);
            }

            sendResponse.dispatchResponse(port); //send the response to the client
      }

      /**this method is the start point of Process Handler and invokes the rest of the
 methods from here
       *@param message the request from the client
       *@exception does not throw any exception
      */
      public void processRequest(String message,int port)
      {
            initialize(message); //initialize all the values
            processQuery(); //perform the actual task of query processing
            sendResponse(port); //send the responses to the client
      }
}///:~
```

```
/*QueryProcessing.java*/
//:codes:QueryProcessing.java
//This class deals with extracting the query from the batch, querying the database and
concatenating the results.
//package project;

import java.sql.*;
import java.io.*;
import java.util.*;
import java.sql.*;

public class QueryProcessing
{
        private float HOLD_TIME=2.0f;  //Set the processing time to simulate the real
world.

        /**Splits the batch into individual queries.
         *@param batchRequest the batch that needs to be split into individual queries.
         *@return a linked list consisting of individual queries.
         *@exception exceptions No exceptions.
        */
        public LinkedList extractQuery(String batchRequest,String tableName)
        {
                StringTokenizer sToken = new StringTokenizer(batchRequest," "); //The
delimiter that defines individual queries
                LinkedList queries = new LinkedList();  //The list that is going to hold the
individual queries.
                while(sToken.hasMoreTokens()) //Read the entire string
```

```
            {
                String key = sToken.nextToken(); //holds the key field ie, C,B or Q
                String value = sToken.nextToken(); //holds the value
corresponding to the client
                if(key.equalsIgnoreCase("q")) //Checks to see if it is the query
field
                {
                    queries.add("select route from "+tableName+" where code
=''' + value+"'");
                }
            }
            return queries;
        }

        /**Performs the actual querying of the table
         *@param queries a linked list that contains the queries
         *@return a linked list that contains the results of the queries
         *@exception exceptions SQLException will be thrown
        */
        public LinkedList queryTable(LinkedList queries) throws SQLException
        {
            Connection DBconn =
AccessJDBCUtil.getAccessDBConnection("db1.mdb"); //A conncetion to the dataase
that contains the table
            Statement stmt = DBconn.createStatement();
            ResultSet results;
            Object query;
            LinkedList response = new LinkedList();
            String temp;
            try
            {
```

```
                    for(int i=0;i<queries.size();i++) //the table must be queried for each
of the queries
                    {
                        query=queries.get(i); //extract the queries from the linked
list
                        temp = (String)query; //converts the object to string
                        stmt.execute(temp); //execute the query
                        results = stmt.getResultSet(); //extract the results
                        results.next(); //move to the first result
                        response.add("Route-"+results.getString("route")); //add it
to the linkedlist containing the responses
                    }
                    DBconn.close();
                }
                catch(SQLException e)
                {
                    System.out.println(e);
                }
                return response;
            }

            /**Concatenates the responses in the LinkedList to form a string
             *@param responses a linkedlist containing all the responses that need to be sent
to the client
             *@return a string that contains the message which is to be sent to the client
             *@exception exceptions No exceptions.
            */
            public String concatenateResponse(LinkedList responses)
            {
                String response=""; //the string which is going to hold the responses
```

```
                for(int i=0;i<responses.size();i++) //all the elements in the linkedlist need
to be added to the string
                {
                        response = response.concat("r "+(String)responses.get(i)+" "); //the
elements of the list are concatenated with the string
                }
                return response;
        }
};//:~
```

/*ReportWriter.java*/

```
//:codes:ReportWriter.java
//This class deals with logging the query arrival and departure
//package project;

import java.io.*;
import java.util.*;

public class ReportWriter
{
        /**This method maintains the log files
         *@param batchNumber the batch number for which the log is being generated
         *@param status the present status of the batch
         *@exception exceptions catches IOException
         */
        public void writeFile(int batchNumber,String status,String clientName) throws
IOException
```

```
{
        FileWriter fwstats = new FileWriter("stats",true); //the data needs to be
appended to the file
        BufferedWriter bwstats = new BufferedWriter(fwstats);
        PrintWriter pwstats = new PrintWriter(bwstats);
        Date timestamp = new Date(); //the time of the event must be recorded

    pwstats.println(String.valueOf(batchNumber)+"\t"+status+"\t"+clientName+"\t"+
String.valueOf(timestamp.getHours())+":"+String.valueOf(timestamp.getMinutes())+":"+
String.valueOf(timestamp.getSeconds())+"."+String.valueOf((timestamp.getTime()%100
0)));
        pwstats.close();

        FileWriter fwreport = new
FileWriter(clientName+"_report_"+String.valueOf(batchNumber)+".csv",true); //this is
for the individual report of each batch
        BufferedWriter bwreport = new BufferedWriter(fwreport);
        PrintWriter pwreport = new PrintWriter(bwreport);

    pwreport.println(String.valueOf(batchNumber)+","+status+","+clientName+","+S
tring.valueOf(timestamp.getHours())+":"+String.valueOf(timestamp.getMinutes())+":"+S
tring.valueOf(timestamp.getSeconds())+"."+String.valueOf((timestamp.getTime()%1000
)));
        pwreport.close();
    }

}///:~
```

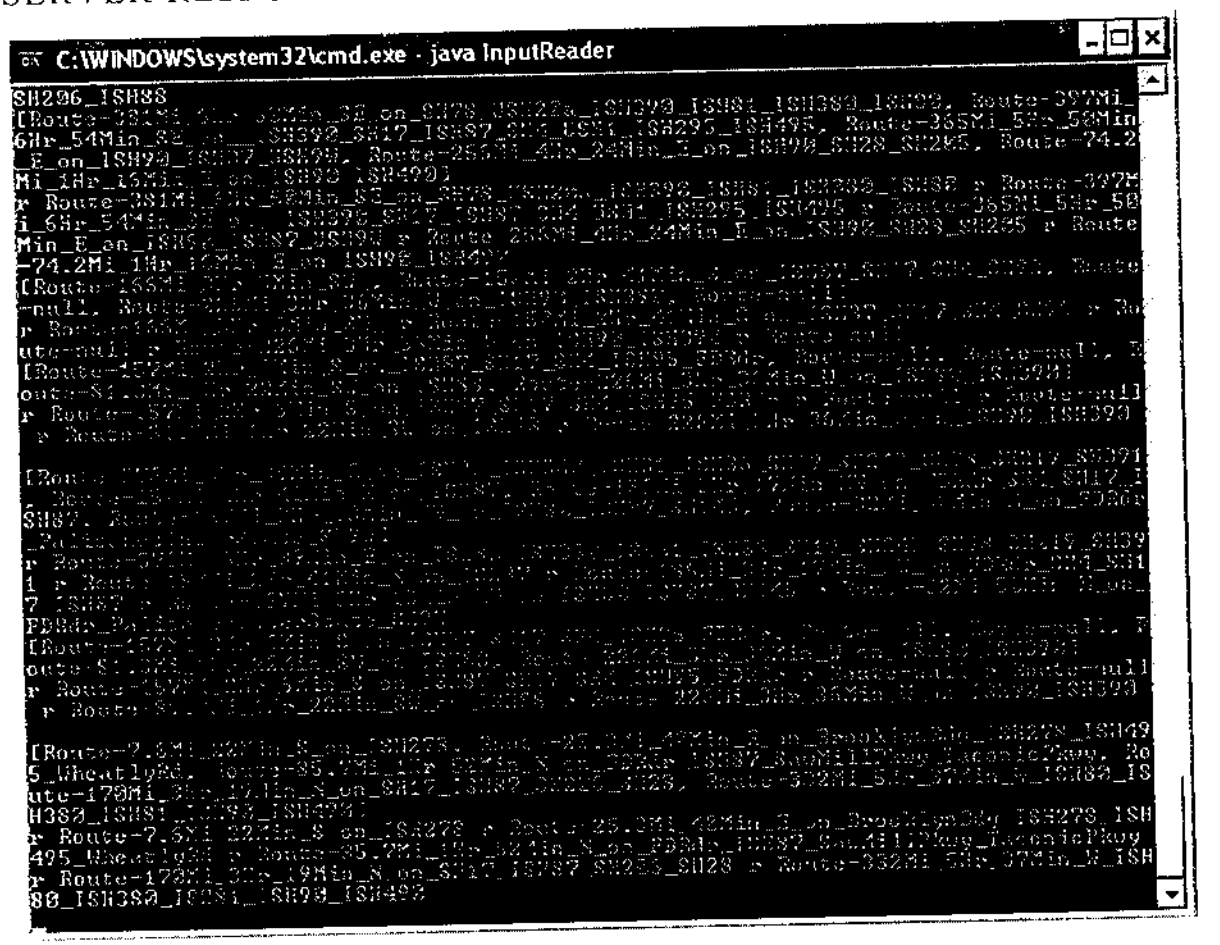
```
/*ResponseDispatcher.java*/
//:codes:ResponseDispatcher.java
//this class deals with the sending of the responses to the clients after processing the
queries
//package project;

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

public class ResponseDispatcher
{
        byte[] outBuffer; //a byte buffer to transfer the response to the network
        int clientPort; //holds the port number of the client to which the resopnse needs to
be sent
        InetAddress clientAddress; //holds the address of the client to which the response
needs to be sent

        /**this method initializes the required values
         *@param message the response that needs to be sent
         *@param address the address of the client to which the response needs to be sent
         *@param port the port number of the client to which the response needs to sent
         *@exception catches unknown host exception
        */
        public void initialize(String message,String address,String port,String
batchNumber)
        {
                message = "b "+batchNumber+" "+message;
                outBuffer = message.getBytes(); //converts the string to a byte array
                clientPort = Integer.parseInt(port); //converts the port number of the client
from a string to an integer
```

```
            try
            {
                    clientAddress = InetAddress.getByName(address); //gets the client
address of the given client id
            }
            catch(UnknownHostException e)
            {
                    System.out.println("Client not found");
                    System.out.println(e);
            }
        }

        /**this method does the dispatching of the response to the clients
         *@exception catches Socket Exception and IO Exception
        */
        public void dispatchResponse(int port)
        {
            try
            {
                    DatagramSocket socket = new DatagramSocket(); //creates a
socket for UDP communication
                    DatagramPacket outPacket = new
DatagramPacket(outBuffer,outBuffer.length,clientAddress,clientPort); //creates a packet
to send the data
                    socket.send(outPacket); //send the created packets through the
socket created for UDP communication
                    socket.close(); //close the socket
            }
            catch(SocketException se)
            {
                    System.out.println("Unable to initialize port");
```

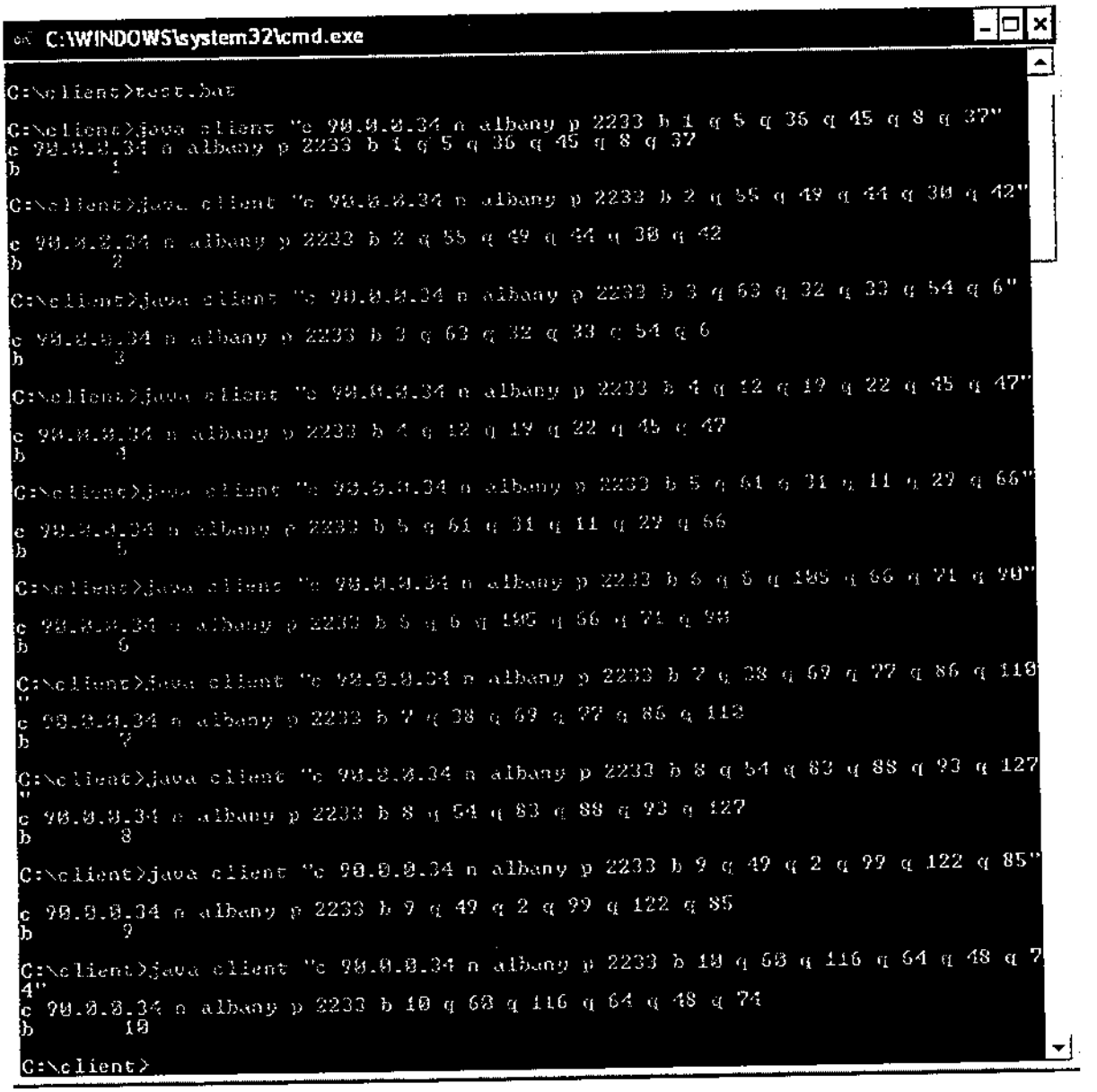
```
                System.out.println(se);
            }
            catch(IOException ie)
            {
                System.out.println("Unable to send data");
                System.out.println(ie);
            }

        }
}///:~
```

SAMPLE OUTPUT:

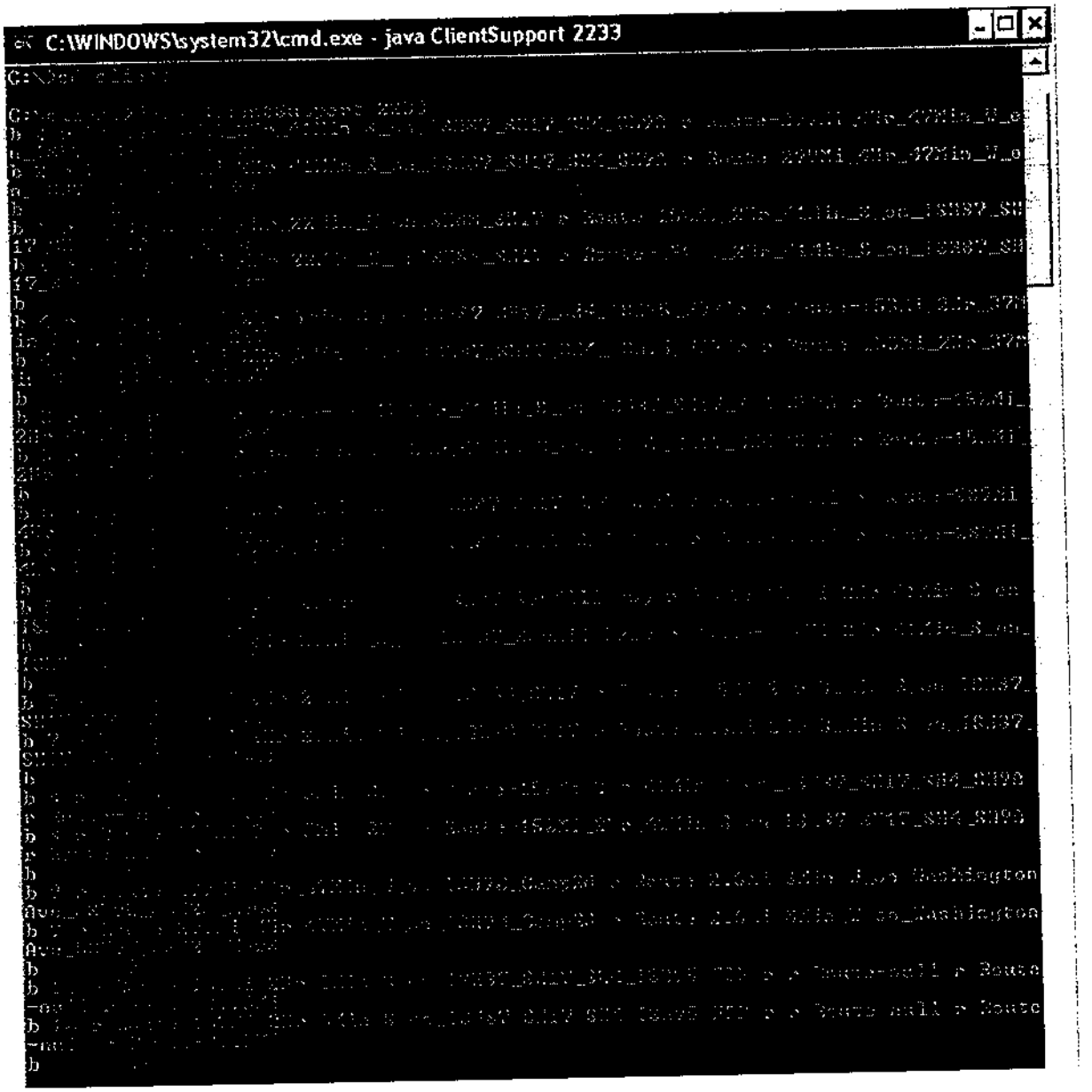
SERVER RESPONDING TO CLIENT REQUESTS



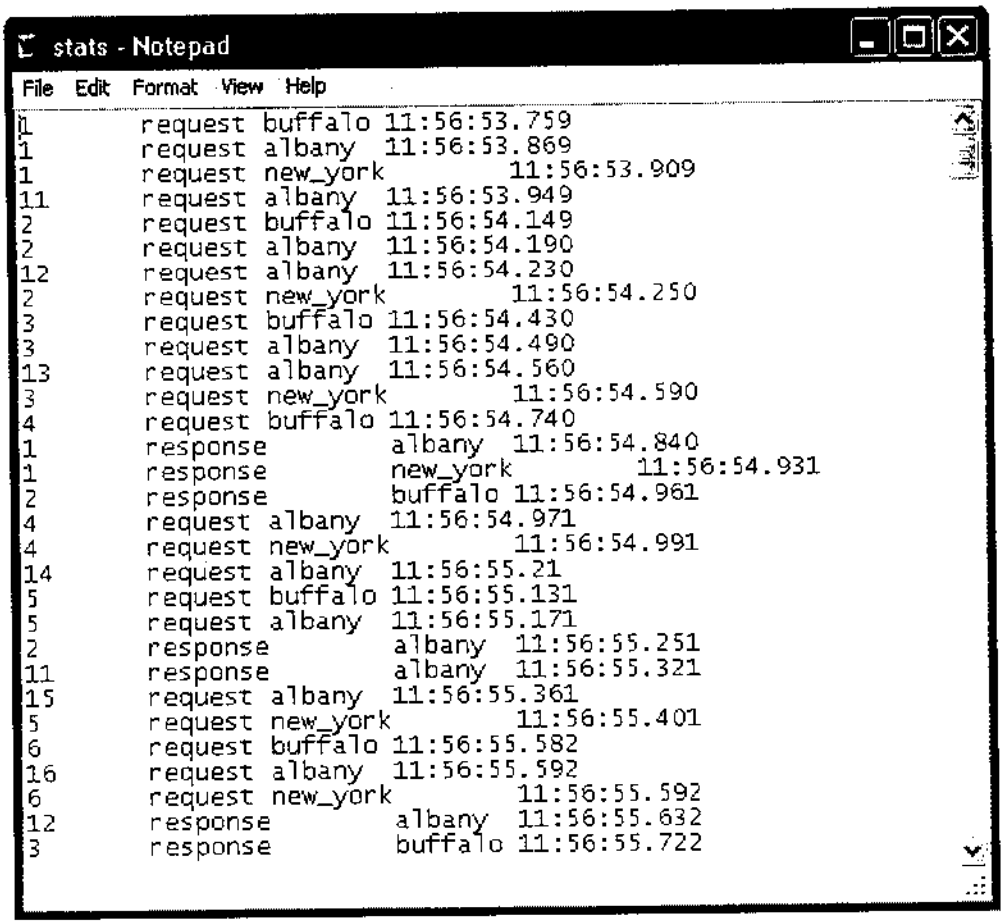
REQUEST FROM CLIENT 1:



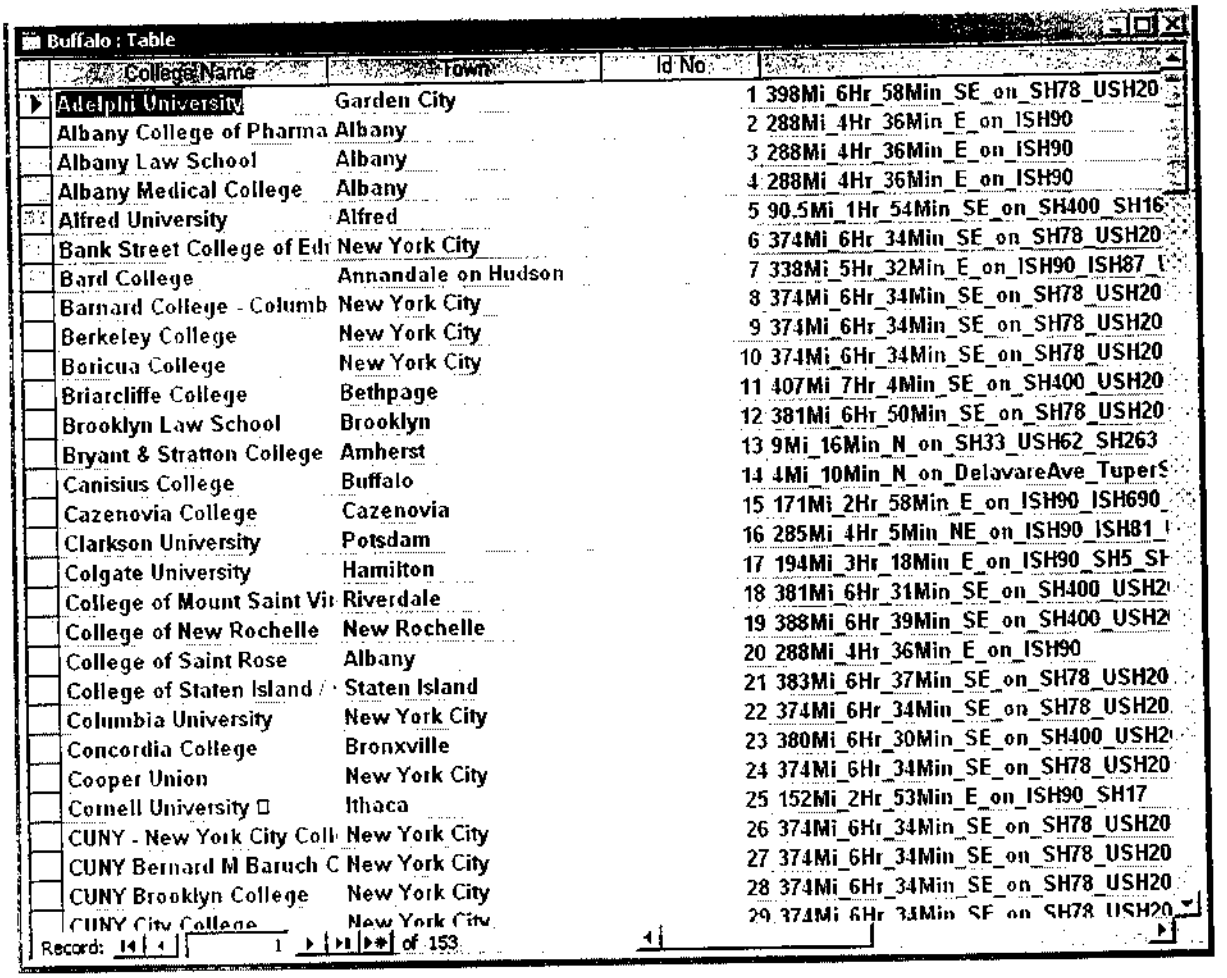
RESPONSE TO CLIENT 1



STATUS TRACKING



DATABASE



Buffalo : Table

College Name	Town	Id No	
Adelphi University	Garden City	1	398Mi_6Hr_58Min_SE_on_SH78_USH20
Albany College of Pharma	Albany	2	288Mi_4Hr_36Min_E_on_ISH90
Albany Law School	Albany	3	288Mi_4Hr_36Min_E_on_ISH90
Albany Medical College	Albany	4	288Mi_4Hr_36Min_E_on_ISH90
Alfred University	Alfred	5	90.5Mi_1Hr_54Min_SE_on_SH400_SH16
Bank Street College of Edi	New York City	6	374Mi_6Hr_34Min_SE_on_SH78_USH20
Bard College	Annandale on Hudson	7	338Mi_5Hr_32Min_E_on_ISH90_ISH87_
Barnard College - Columb	New York City	8	374Mi_6Hr_34Min_SE_on_SH78_USH20
Berkeley College	New York City	9	374Mi_6Hr_34Min_SE_on_SH78_USH20
Boricua College	New York City	10	374Mi_6Hr_34Min_SE_on_SH78_USH20
Briarcliffe College	Bethpage	11	407Mi_7Hr_4Min_SE_on_SH400_USH20
Brooklyn Law School	Brooklyn	12	381Mi_6Hr_50Min_SE_on_SH78_USH20
Bryant & Stratton College	Amherst	13	9Mi_16Min_N_on_SH33_USH62_SH263
Canisius College	Buffalo	14	4Mi_10Min_N_on_DelavareAve_TuperS
Cazenovia College	Cazenovia	15	171Mi_2Hr_58Min_E_on_ISH90_ISH690_
Clarkson University	Potsdam	16	285Mi_4Hr_5Min_NE_on_ISH90_ISH81_
Colgate University	Hamilton	17	194Mi_3Hr_18Min_E_on_ISH90_SH5_SH
College of Mount Saint Vi	Riverdale	18	381Mi_6Hr_31Min_SE_on_SH400_USH2
College of New Rochelle	New Rochelle	19	388Mi_6Hr_39Min_SE_on_SH400_USH2
College of Saint Rose	Albany	20	288Mi_4Hr_36Min_E_on_ISH90
College of Staten Island /	Staten Island	21	383Mi_6Hr_37Min_SE_on_SH78_USH20
Columbia University	New York City	22	374Mi_6Hr_34Min_SE_on_SH78_USH20
Concordia College	Bronxville	23	380Mi_6Hr_30Min_SE_on_SH400_USH2
Cooper Union	New York City	24	374Mi_6Hr_34Min_SE_on_SH78_USH20
Cornell University	Ithaca	25	152Mi_2Hr_53Min_E_on_ISH90_SH17
CUNY - New York City Coll	New York City	26	374Mi_6Hr_34Min_SE_on_SH78_USH20
CUNY Bernard M Baruch C	New York City	27	374Mi_6Hr_34Min_SE_on_SH78_USH20
CUNY Brooklyn College	New York City	28	374Mi_6Hr_34Min_SE_on_SH78_USH20
CUNY City College	New York City	29	374Mi_6Hr_34Min_SE_on_SH78_USH20

Record: 1 of 153

8. REFERENCES

REFERENCES

1. Bruce Eckel (2002), 'Thinking In Java' – Tata McGraw Hill Publication company limited. Third edition. Revision- 4.0.

2. Michael Morrison, et al (1997), 'Java Unleashed' Second Edition Sams.net Publishing .

3. Patrick Naughton and Herbert Schildt (1998), 'Java 2 The Complete Reference', Third Edition, Tata McGraw Hill Publication Company Limited.

ONLINE REFERENCES

4. http://java.sun.com/
5. http://www.princetonreview.com/
6. http://wikipedia.org/
7. http://maps.google.com/
8. http://maps.yahoo.com/