Distributed Systems

MINI PROJECT

Hostel Room Reservation System

Class: BE COMPS

Date: 06/11/2020

Name of Members

Siddesh Kamble (2018230072)

Sujoy Barua (2018230067)

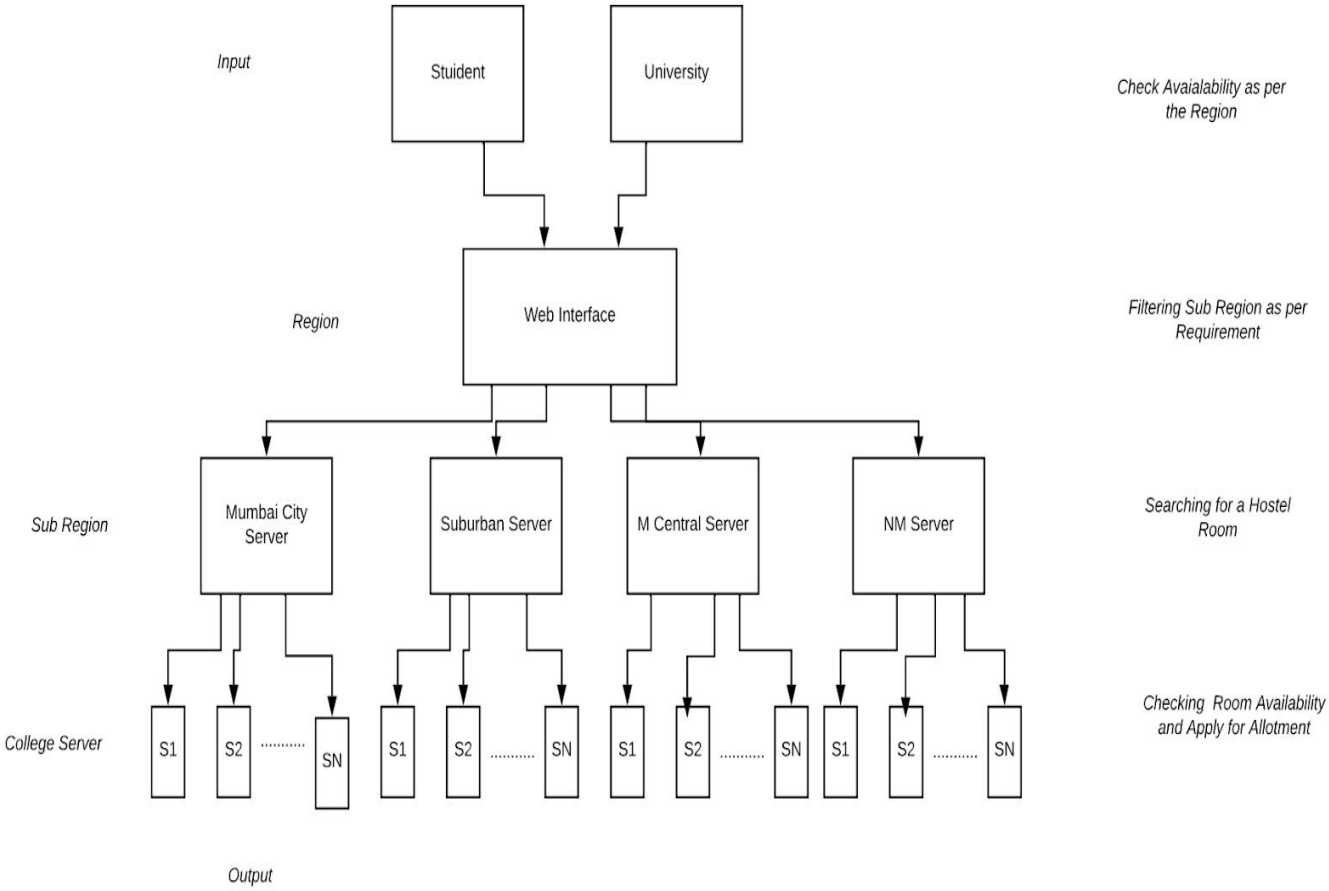
Jayram Nandagiri (2018230073)

Table of Contents

Abstract:

A distributed system is a collection of independent components located on different machines that share messages with each other in order to achieve common goals. With design patterns like microservices being adapted by conglomerates to develop and deploy their monoliths, big data systems using distributed file systems for data sharding and replication, and cloud/on-premise distributed databases giving performance enhancements, distributed systems have been a research topic for quite some time now. In this regard, we plan to implement a distributed hostel room reservation system that can allow university students to make hostel room reservation easy and hostel administrators to manage the college hostels. These two types of clients interact with a distributed server that models, maintains, and updates hostel rooms while ensuring data consistency, mutual exclusion, and data synchronization. With the distributed hostel room reservation system, users will be able to view and reserve rooms from the hostel. Admins will be able to add or remove rooms and view their availability, as well as, their current waitlist and reservation list.

System Architecture



Functionalities:

ADMIN (Mutual Exclusion, Data Consistency, Clock Synchronization)

a) Adding Rooms (Hostel):

The hostel administrator can add available rooms and he can also specify the total no of rooms. It has data consistency as in no. two rooms would be allotted to same students or students having certains traits in common would be inspected before room allotment. Mutual Exclusion is observed as at a time only one student is allowed to be eligible for a particular room. Students who apply at the same time are expected to wait and are entertained only after Student 1 is allotted the room then student 2 is shown next room or student 1 is not interested in that room , so student 2 can check in for the same room.

b) Removing Rooms (Hostel):

This functionality requires the room id along with the number of rooms to remove that particular room. At a time only one room is removed or added, mutual exclusion is observed in this functionality as well.

c) List of vacant rooms (Hostel):

The hostel administrator can view the list of rooms currently occupied or vacant in the hostel. Admin has control over all the vacant rooms and it's the administrators responsibility to maintain data consistency while room allocation. And it is expected that he/she follows the same while adding or removing rooms.

d) Responding Complaints (Hostel):

This functionality allows the admin to interact or respond to the complaints raised by the students. Admin can respond to the queries raised by students as per their register id.

CLIENT(Data Consistency, Mutual Exclusion, Clock Synchronization, Election Algorithm)

a) Applying for an hostel room(Student):

This functionality allows the client to allow or send requests to the server/admin for a room allotment. Student has to request or send a query before applying for a room, at a time one student who has applied before is entertained and others simply has to wait for their turn, here election algorithm is used.

b) Raising complaint (Student):

This functionality grants the student ability to chat or communicate with Admin regarding any query or issue and have a two way communication. Clock synchronization is used for two way communication between the Student and Admin, Query along with its Id and date and time is stored.

Objectives:

1. To ensure communication between clients and server UDP/IP sockets and to ensure inter-server communication.
2. Connecting students and administrators to hostel rooms providing distribution transparency.

Outcomes:

1. Data Consistency using mutual exclusion for critical sections
2. Fault-tolerant system to ensure robustness
3. Load balancing to ensure requests are handled by servers with the least load

Stepwise Screenshots/Code and Output:

Aim: To implement Client-Server based program using RPC/RMI

1. Java Remote Method Invocation (RMI)

- The RMI provides remote communication between the applications using two objects stub and skeleton.
- The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. The RMI allows an object to invoke methods on an object running in another JVM.

Understanding stub and skeleton

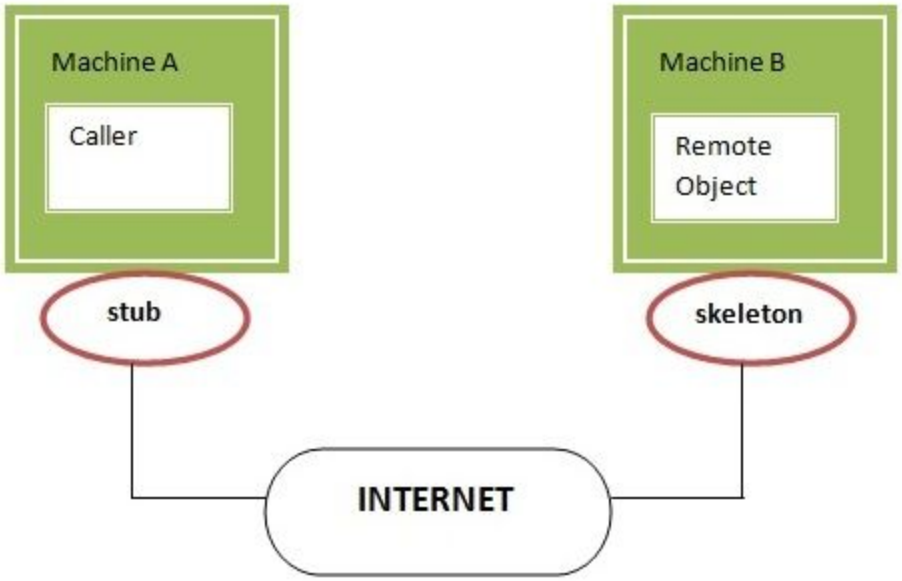
- RMI uses stub and skeleton object for communication with the remote object.
  - A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub: The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton: The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.



Output:

1) Start rmi registry

```
C:\Users\Jayram Nandagiri\OneDrive\Desktop\RMI>start rmiregistry

C:\Users\Jayram Nandagiri\OneDrive\Desktop\RMI>
```

2) Run the Server

```
Server ready
```

3) Run the Client

```
Do you want to Login or Register:
0
Enter your Registration No:
70
Enter your First name:
Jayram
Enter your Middle name:
Prakash
Enter your last name:
Nandagiri
Gender:
Male
Enter your Phone No:
7738916989
Enter your email:
jayramnandagiri@gmail.com
Enter your password:
12345
Registration Sucessfull

Connecting To RMI Server...

Enter the username :
Jayram
Enter the password :
12345
You are an authorized user...
Do you want to apply for a Hostel Room
```

4) Students Data in MYSQL Database

```
mysql> select * from student_data;
+--------+------------+-------------+-----------+--------+------------+--------------------------+----------+
| reg_id | first_name | middle_name | last_name | gender | phoneNo    | email                    | password |
+--------+------------+-------------+-----------+--------+------------+--------------------------+----------+
|     64 | Sujoy      | S           | Barua     | Male   | 9985634589 | sujoy@gmail.com          | s@456    |
|     69 | Siddesh    | Sunil       | Kamble    | Male   | 8569453210 | siddeshkamble45@gmail.com | sidd@123 |
|     70 | Jayram     | Prakash     | Nandagiri | Male   | 7738916989 | jayramnandagiri@gmail.com | 12345    |
+--------+------------+-------------+-----------+--------+------------+--------------------------+----------+
3 rows in set (0.00 sec)
```

Conclusion: Understood the concept of Remote Procedure Call , Familiarized with the implementation of Java RMI API. Implemented a client-server database application using the RMI API in Java.

Aim: Implementation of Mutual Exclusion algorithm

2. Mutual Exclusion

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

- **Solution to distributed mutual exclusion:**

As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. Token Based Algorithm:

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section.
- This approach uses sequence numbers to order requests for the critical section.
- Each request for the critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach ensures Mutual exclusion as the token is unique.

2. Non-token based approach:

- A site communicates with other sites in order to determine which sites should execute the critical section next. This requires exchange of two or more successive rounds of messages among sites.
- This approach uses timestamps instead of sequence numbers to order requests for the critical section.
- Whenever a site makes a request for a critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.

- All algorithms which follow a non-token based approach maintain a logical clock. Logical clocks get updated according to Lamport's scheme.

3. Quorum based approach:

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion.

Output:

1) Applying for a Room

```
You are an authorized user...
1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit

Enter the choice:
1
Enter your Details for Room Alllocation
Enter your Room No:
70
Enter the Full Name:
Jayram Nandagiri
2 Seats are Available
Fees per month:Rs 4000
Duration/Months:
5
Address:
Worli
1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit
```

2) Rooms Data in MYSQL Database

```
mysql> select * from room_data;
+---------+------------------+----------+---------+
| Room_id | Full_Name        | Duration | Address |
+---------+------------------+----------+---------+
|      89 | Siddesh Kamble   |        4 | Panvel  |
|      70 | Jayram Nandagiri |        5 | Worli   |
+---------+------------------+----------+---------+
2 rows in set (0.00 sec)
```

Here if a particular student applies for a room and enters the room data based on UID from a particular college only that student is under critical section until and unless he submits his data and no other student is able to access and manipulate his data.

Conclusion: In this module we have used Mutual Exclusion algorithm to prevent race conditions. By this algorithm we make sure that in the critical section only one process takes incharge and the rest of the process waits.

Aim: To implement Data Consistency

3. Data Consistency

Consistency is the agreement between multiple nodes in a distributed system to achieve a certain value.
Specifically, it can be divided into strong consistency and weak consistency.

- Strong consistency: The data in all nodes is the same at any time. At the same time, you should get the value of key1 in node A and the value of key1 in node B.
- Weak consistency: There is no guarantee that all nodes have the same data at any time, and there are many different implementations. The most widely achieved is the ultimate consistency. The so-called final consistency means that the same data on any node is the same at any time, but as time passes, the same data on different nodes always changes in the direction of convergence. It can also be simply understood that after a period of time, the data between nodes will eventually reach a consistent state.

Distributed and consistent application scenarios:
Multi-node provides read and write services to ensure high availability and scalability (ZooKeeper, DNS, redis cluster)
Problems faced by distributed systems:

- Message asynchronous asynchronous (asynchronous): The real network is not a reliable channel, there are message delays, loss, and inter-node messaging can not be synchronized (synchronous)
- node-fail-stop: The node continues to crash and will not recover
- node down recovery (fail-recover): Node recovery after a period of time, the most common in distributed systems

Output:

1) Adding Rooms

```
Enter the Institute Id :
121
Enter the password :
12345
You are an authorized user...
1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
1
Enter the Total No Rooms you want to add:
2
Enter the Room Id:
56
Enter the No of Seats:
3
Enter the Room Id:
69
Enter the No of Seats:
4

1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
4
 2 Rooms are Vacant
```

2) Institute Room Data in MYSQL Database

```
mysql> select * from institute_room_data;
+---------+-------------+
| Room_id | No_of_Seats |
+---------+-------------+
|      45 |           3 |
|     131 |           4 |
| 2323545 |           3 |
|       6 |           3 |
|      56 |           3 |
|      69 |           4 |
+---------+-------------+
6 rows in set (0.00 sec)
```

Here the Data is consistent because when an institute enters N no of rooms in the Add Rooms data. So as soon as he enters the List of Vacant Rooms gets updated before the add rooms and after the add rooms.

Conclusion: A distributed system maintains copies of its data on multiple machines in order to provide high availability and scalability. When an application makes a change to a data item on one machine, that change has to be propagated to the other replicas. This is called data consistency. Thus, we have learned about data consistency and implemented the concept of checksum to ensure data consistency.

Aim: To perform clock synchronization in Distributed System

4. Clock Synchronization

Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

The physical clocks are used to adjust the time of nodes.Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.

- The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

1. External clock synchronization is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
2. Internal clock synchronization is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

- There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. Centralized is the one in which a time server is used as a reference. The single time server propagates its time to the nodes and all the nodes adjust the time accordingly. It is dependent on a single time server so if that node

fails, the whole system will lose synchronization. Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.

2. Distributed is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

Cristian's Algorithm is a clock synchronization algorithm that is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy prone distributed systems/applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.

Algorithm:

1) The process on the client machine sends the request for fetching clock time(time at server) to the Clock Server at time T_0.

2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.

3) The client process fetches the response from the Clock Server at time T_1 and calculates the synchronised client clock time using the formula given below.

$$T_{CLIENT} = T_{SERVER} + (T_1 - T_0)/2$$

where T_{CLIENT} refers to the synchronised clock time,
T_{SERVER} refers to the clock time returned by the server,
T_0 refers to the time at which request was sent by the client process,
T_1 refers to the time at which response was received by the client process

Output:

1) Register Complaint

```
1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit

Enter the choice:
2
Enter your College Id:
70
Enter your Query:
I want to say you that there is less supply of food for students of hostel.

1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit
```

2) Complaint Data in MYSQL Database

```
mysql> select * from complaint_data;
+--------+-----------------------------------------------------------------------+---------------------+
| reg_id | query                                                                 | date_time           |
+--------+-----------------------------------------------------------------------+---------------------+
|     34 | There Food Supply Problem                                             | 2020-10-23 14:10:30 |
|     70 | I want to say you that there is less supply of food for students of hostel. | 2020-11-06 01:48:38 |
|    456 | Water Problem                                                         | 2020-10-23 01:56:56 |
+--------+-----------------------------------------------------------------------+---------------------+
3 rows in set (0.00 sec)
```

3) Show Complaints

```
1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
5
Reg ID: 34
Query: There Food Supply Problem
Date and Time: 2020-10-23 14:10:30

Reg ID: 70
Query: I want to say you that there is less supply of food for students of hostel.
Date and Time: 2020-11-06 01:48:38

Reg ID: 456
Query: Water Problem
Date and Time: 2020-10-23 01:56:56
```

4) Provide Solution

```
1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
6
Enter Complaint Id:
70
Enter the Solution regarding Complaint:
We are already looking into the problem and update you soon
```

5) Solution Data in MYSQL Database

```
mysql> select * from sol_data;
+--------------+-----------------------------------------------------------+---------------------+
| Complaint_id | Solution                                                  | date_time           |
+--------------+-----------------------------------------------------------+---------------------+
|           34 | Were are lookin the problem and update you soon           | 2020-10-23 14:12:46 |
|           70 | We are already looking into the problem and update you soon | 2020-11-06 01:51:40 |
|          456 | We're already working on this issue will update you soon  | 2020-10-23 11:48:28 |
+--------------+-----------------------------------------------------------+---------------------+
3 rows in set (0.00 sec)
```

6) Display Solution

```
1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit

Enter the choice:
3
Enter the Complaint ID:
70
Complaint ID: 70
Solution: We are already looking into the problem and update you soon
Date and Time: 2020-11-06 01:51:40

1: Apply for an hostel
2: Raising complaint
3: Notification
4: Exit
```

Conclusion: In this experiment a Cristian clock synchronization algorithm is used. Clock synchronization is necessary for the ordering of events and to preserve the state of resources. There is a need to transmit a message from one node to another at any time. To send the frames of the screen they should be in order so here we used a clock so that the server knows at what time the client starts sharing the screen.

Aim: To implement Load Balancing Algorithm

5. Load Balancing

In a distributed computing system made up of different types of processors each processor in the system may have different performance and reliability characteristics. In order to take advantage of this diversity of processing power, a modular distributed program should have its modules assigned in such a way that the applicable system performance index, such as execution time or cost, is optimized.

Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm. Each load balancer sits between client devices and backend servers, receiving and then distributing incoming requests to any available server capable of fulfilling them. Two main approaches that practice Load Balancing are Static Balancing and Dynamic Balancing.

- **Static Balancing:**

A load balancing algorithm is "static" when it does not take into account the state of the system for the distribution of tasks. Thereby, the system state includes measures such as the load level (and sometimes even overload) of certain processors. Instead, assumptions on the overall system are made beforehand, such as the arrival times and resource requirements of incoming tasks. In addition, the number of processors, their respective power and communication speeds are known. Therefore, static load balancing aims to associate a known set of tasks with the available processors in order to minimize a certain performance function. The trick lies in the concept of this performance function. Static load balancing techniques are commonly centralized around a router, or Master, which distributes the loads and optimizes the performance function. This minimization can take into account information related to the tasks to be distributed, and derive an expected execution time. The advantage of static algorithms is that they are easy to set up and extremely efficient in the case of fairly regular tasks (such as processing HTTP requests from a website). However, there is still some statistical variance in the assignment of tasks which can lead to overloading of some computing units.

- **Dynamic Balancing:**

Unlike static load distribution algorithms, dynamic algorithms take into account the current load of each of the computing units (also called nodes) in the system. In this approach, tasks can be moved dynamically from an overloaded node to an underloaded node in order to receive faster processing. While these algorithms are much more complicated to design, they can produce excellent results, in particular, when the execution time varies greatly from one task to another. In dynamic load balancing the architecture can be more modular since it is not mandatory to have a specific node dedicated to the distribution of work. When tasks are uniquely assigned to a processor according to its state at a given moment, it is a unique assignment. If, on the other hand, the tasks can be permanently redistributed according to the state of the system and its evolution, this is called dynamic assignment. Obviously, a load balancing algorithm that requires too much communication in order to reach its decisions runs the risk of slowing down the resolution of the overall problem.

Output:

1) Show Complaints

```
1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
5
Reg ID: 34
Query: There Food Supply Problem
Date and Time: 2020-10-23 14:10:30

Reg ID: 70
Query: I want to say you that there is less supply of food for students of hostel.
Date and Time: 2020-11-06 01:48:38

Reg ID: 456
Query: Water Problem
Date and Time: 2020-10-23 01:56:56
```

2) Provide Solution

```
1: Add Rooms
2: Removing Rooms
3: List of Rooms Allocated
4: List of Rooms Vacant
5: Show Complaints
6: Responding Complaints
7: Exit

Enter the choice:
6
Enter Complaint Id:
70
Enter the Solution regarding Complaint:
We are already looking into the problem and update you soon
```

3) Solution Data in MYSQL Database

```
mysql> select * from sol_data;
+--------------+-----------------------------------------------------------+---------------------+
| Complaint_id | Solution                                                  | date_time           |
+--------------+-----------------------------------------------------------+---------------------+
|           34 | Were are lookin the problem and update you soon           | 2020-10-23 14:12:46 |
|           70 | We are already looking into the problem and update you soon | 2020-11-06 01:51:40 |
|          456 | We're already working on this issue will update you soon  | 2020-10-23 11:48:28 |
+--------------+-----------------------------------------------------------+---------------------+
3 rows in set (0.00 sec)
```

Conclusion: Thus in this experiment we have studied the Load Balancing Algorithm. This algorithm ensures no single server bears too much demand. By spreading the work evenly, load balancing improves application responsiveness. It also increases availability of applications and websites for users.

Aim: : Implementation of Election algorithm

6. Election Algorithm

Distributed Algorithm is an algorithm that runs on a distributed system. Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in distributed systems require a coordinator that performs functions needed by other processes in the system. Election algorithms are designed to choose a coordinator.

Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on another processor. Election algorithm basically determines where a new copy of the coordinator should be restarted.

Election algorithms assume that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has the highest priority number.Then this number is sent to every active process in the distributed system.

The Bully Algorithm –

This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm – Suppose process P sends a message to the coordinator.

1. If coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
2. Now process P sends election message to every process with high priority number.

3. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
    - (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    - (II) If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted.

Output:

```
PS C:\Users\Jayram Nandagiri\Downloads\bully-algorithm-master\src> javac Main.java
PS C:\Users\Jayram Nandagiri\Downloads\bully-algorithm-master\src> java Main
Hey, please enter the amount of processes to start:
5
P3: Coordinator, are you there?
P5: Yes
P3: Coordinator, are you there?
P5: Yes
P4: Coordinator, are you there?
P5: Yes
P2: Coordinator, are you there?
P5: Yes
P1: Coordinator, are you there?
P5: Yes
P4: Coordinator, are you there?
P4: Coordinator is down, I'm  initiating election..
P4: P5 didn't respond
New coordinator: P4
P2: Coordinator, are you there?
P4: Yes
P2: Coordinator, are you there?
P4: Yes
P5: I'm back!
P5: Who is the coordinator?
P4: Me
P5: Resign -> P4
P4: Successfully Resigned
P5: P4, get out of here, I'm the coordinator!
P1: Coordinator, are you there?
P5: Yes
P3: Coordinator, are you there?
P5: Yes
P2: Coordinator, are you there?
P5: Yes
```

Conclusion: Thus we have implemented the Bully algorithm for election algorithm to select the coordinator of a class.

Conclusion

Hostel Room Reservation System is a user-friendly computer-based system for managing hostel facilities in institutions. It has been designed to automate, manage and look after the overall processing of records of students residing in a large hostel. It is capable of managing Enquiry details, Student Details, Payment Details etc. The developed system provides solutions to manual hostel management problems and also provides information such as hostel information, hostel room information, and hostel accounts information. The software offers stability, cost-effectiveness and usability. It provides the most flexible and adaptable standards management system solutions for hostels.