Distributed System Design

COMP 6231 – Winter 2023

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Movie Ticket Booking System Using Java RMI: Assignment-1

By: Mir Pasad

Student ID: 40253287

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sr No. | Topic | Page No. |
| 1 | Introduction | 3 |
| 2 | Overview | 4 |
| 3 | Implementation | 6 |
| 4 | Data Structure | 7 |
| 5 | Test Case | 8 |

1. **INTRODUCTION:**

The full capability of the Java RMI is explained by this system. A technology called Java Remote Method Invocation (Java RMI) enables an object running in one Java Virtual Machine (JVM) to call methods on an object running in another JVM. It makes it easier to share resources and services and call Java object methods remotely.

The key approaches used in this assignment are RMI Application and UDP. Three servers and one client make up this application; the client handles both users and managers at once.

Through RMI, the client connects to three servers, and those three servers communicate with one another via UDP connections. From the user’s name, the client may determine the server. Additionally, based on the admin or customer ID, all user operations are managed from a certain server. Each server's log file contains a record of every action taken on the server side. Additionally, a log file is created for each user by the client's request and reply.

The name "Remote Method Invocation" refers to the fact that when a client calls a method, it actually calls a remote method because the method's logic is located on a server. Only the method's name is known to the client; its location is unknown. An RMI library is necessary in order to allow a client to call a remote procedure. It is the responsibility of the RMI library to establish client connections with the appropriate servers and to send method invocations to those servers, giving the impression to clients that the methods are locally implemented. When a client connection arrives at the recipient servers, the RMI library accepts it, calls the specified methods, and then returns the results. The fact that the invocations were made by remote Clients are completely unknown to the servers when they carry out the procedures. As a result, the RMI library aids in hiding client-server interactions.

1. **OVERVIEW:**

Three separate servers spread over three different stations make up the Distributed Movie Ticket Booking system (DMTBS):

* Atwater (ATW)
* Verdun (VER)
* Outremont (OUT)

There are two categories of clients for this system:

• Admin

• Customer

We must make sure that these clients are using Java RMI to connect to their own servers, and that our three servers are connected using UDP/IP socket programming.

**Admin Specific functions:**

* InsertMovie(): Only the admin server can insert movies.
* RemoveMovie(): admin can only remove movies from their own servers; if a movie was removed, we would need to arrange another movie that was close by for the customers who had previously booked tickets for it. For a server-server connection, UDP is required.
* DisplayAvailability(): This function must be used to collect all movies of a specific type from all three servers. For a server-server connection, UDP is required.
* BookMovie()- This function allows the admin to book movie tickets specified by the customers requirement.
* BookingSchedule(): it allows the admin to obtain the booking schedule for all the movies and work accordingly for further bookings.
* CancelTickets(): The admin can also cancel particular ticket bookings if there was an error.
* Admin Login/Logout(): Allows the admin to login/logout.

**Customer/Admin Specific Functions:**

* PurchaseMovieTickets(): which allows users to reserve events from other servers up to three times per week. For a server-server connection, UDP is required.
* SchedulingReservation(): it shows the customer booking schedule.
* CancelTicket(): Customers can remove or cancel any tickets booked from their own schedules. For a server-server connection, UDP is required.

CustomerID (8 characters): serverID (3 characters) + clientType (C/M) + 4 digit identifier is used to identify clients.

Avatar, Avengers, Titanic, along with their movieType and movieID (10 characters)—serverID (3 characters), movieSlot (M, A, and E), and movieDate—are used to identify events (DDMMYY).

1. **IMPLEMENTATION:**

* RMI is used for client-server communication.

The RMI Registry ports are:

* ATWATER - 1111
* VERDUN - 2222
* OUTREMOUNT - 3333
* UDP/IP Socket programming is used for server-server communication.

The following ports are used:

* ATWATER UDP port: 1322
* VERDUN UDP port: 2213
* We utilised a single server implementation file and a single interface implementation file to reduce code duplication and to make updates and debugging easier.
* The Server and Client each keep their own log files.
* The serverName.txt file can be found in the project directory's src Logs Server directory.
* The project directory's srcLogsClientClientID.txt file is where you can get client logs.
* To achieve the highest level of concurrency, we employed concurrent Hash Maps to store the data.
* The most crucial aspect of the implementation was to keep UDP calls out of infinite loops, particularly in the removeMovie() and DisplayAvailability() methods.
* The removeMovie() method was the hardest to develop when clients were enrolled in the event and some of them came from other servers.

1. **DATA STRUCTURES:**

Each server maintains all the data using the three Map structures depicted in the diagram below.

Diagram

Description automatically generated

Diagram

Description automatically generated

Chart, diagram, box and whisker chart

Description automatically generated

1. **TEST CASE:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr No. | Test | Scenario | Test Case |
| 1 | Login | Username | * Admin ID * Customer ID |
| 2 | Menu | Logout | Menu for logging out |
| 3 | Admin | InsertMovie() | * Invalid MovieID-not added * New MovieID-added * Already existing MovieID-to increase the capacity. * Duplicate Movie-not allowed * MovieID of other servers-not allowed to add |
| 4 | RemoveMovie() | * Not a valid MovieID * Non existing MovieID * Movie with no registration-remove movie * Movie with few registration- remove movie + register to same movieType if possible (UDP if needed) * Other server MovieID-not allowed to add |
| 6 | DisplayAvailability() | * Display all movies of a given type from all the servers. (UDP Needed) |
|  |  | Purchase and Cancel Movie Tickets | * On own server- allowed to purchase and cancel movie ticket after asking for customerID * If movie full-not allowed to purchase or cancel movie ticket. * Other server-Purchase tickets only thrice a week(UDP needed). * Invalid movieID-not allowed to purchase or cancel movie tickets. |
| 7 | Customer | PurchaseMovieTicket() | * On own server-allowed to purchase tickets * Movie capacity is full- not allowed to purchase tickets. * On other server- only three in a week. (UDP needed) * Not a valid MovieID- not allowed to purchase tickets. |
| 8 | SchedulingReservation() | * Display the movie booking schedule of customer * Not a valid customerID- not allowed to login and view anything * Customer does not exits-ok |
| 9 | CancelTicket() | * Cancel tickets on own server-allowed to cancel * Cancel tickets on other servers-allowed to cancel (UDP needed) * Cancel a movie that is not registered - error. * Not a valid MovieID- not a allowed to login or Cancel tickets. |