

PROJECT: SOFTWARE FAILURE TOLERANT DISTRIBUTED MOVIE
TICKET BOOKING SYSTEM USING WEB SERVICES IMPLEMENTATION



SUBMITTED BY:

SHIVAM MISHRA (40204459)

PRIYANG SHAILESHBHAI PATEL (40230020)

PRAGYA TOMAR(40197757)

PROJECT DESIGN SUBMISSION DATE: 20-03-2023

Table of Contents

Overview	2
Background and Project Goals.....	2
Design and Architecture	3
System Specification and Module Design	4
Technologies and Protocols	5
System Data Flow	7
Recovery	10
Testing	11
Test Scenarios	11
Test Results	12
Individual Contribution and Responsibilities	13
Module Design and Integration	13
Conclusion	14
Project Summary	14
References	15

Overview:

Background and Project Goals:

The Distributed Movie Ticket Booking System (DMTBS) is a distributed system that allows users to book and cancel tickets for movies in three different theatre locations (Atwater, Outremont, Verdun). The system has two types of users: admins and customers. Admins are responsible for creating movie slots with specific movie types and booking capacities, while customers can book and cancel tickets for the available movies. The system is designed to identify the user's server by looking at the ID prefix and perform the operation on that server. The system uses WEB-SERVICES between clients and servers and UDP packets for communication between servers.

The goal of this project is to create a **software failure tolerant** or highly available distributed movie ticket booking system. The system should be designed in such a way that it can continue to function even if one or more servers fail or experience other issues.

To achieve this goal, the system will need to have redundancy built into its design. This can be accomplished by replicating data across multiple servers and using load balancing techniques to distribute traffic evenly across the servers. In addition, the system will need to be able to detect and recover from failures quickly, in order to minimize downtime and prevent data loss.

The project also aims to ensure that the system provides a seamless experience for users, with high availability and reliability. This means that customers should be able to book and cancel tickets without experiencing any delays or errors, and administrators should be able to add, delete, and list movie slots without any issues. The system should also maintain accurate records of all transactions and be able to provide detailed logs and reports for auditing purposes.

Overall, the main goal of this project is to create a distributed movie ticket booking system that is highly available, reliable, and capable of withstanding software failures or other issues.

Design and Architecture:

System Specification and Module Design:

The description of the system and design of modules are as follows:

Identifiers: Clients (Customers and Admins) are recognized by unique identifiers called as customerID and adminID , which is constructed from the acronym of their area and a 4-digit number (e.g. ATWA2345 for an admin and ATWC2345 for a customer). When a user performs an operation, the system identifies the server to which the user belongs by looking at the ID prefix and performs the operation on that server.

Locations: The Distributed Movie Ticket Booking System consists of three different servers for three different theatres in three different areas: Atwater (ATW), Verdun (VER) and Outremont (OUT). The system's architecture is distributed, meaning that each server is independent of the other and can function autonomously. The servers communicate with each other through network protocols, allowing the system to perform tasks that would be impossible with a centralized architecture.

Each theatre has admins assigned who can perform a set of operations. Customers can directly access their respective area server. However, if a customer is from another location, they can make a request to their location server, which will then request the server of another location. The system fulfilled the request through the inter-server communication maintaining location and access transparency.

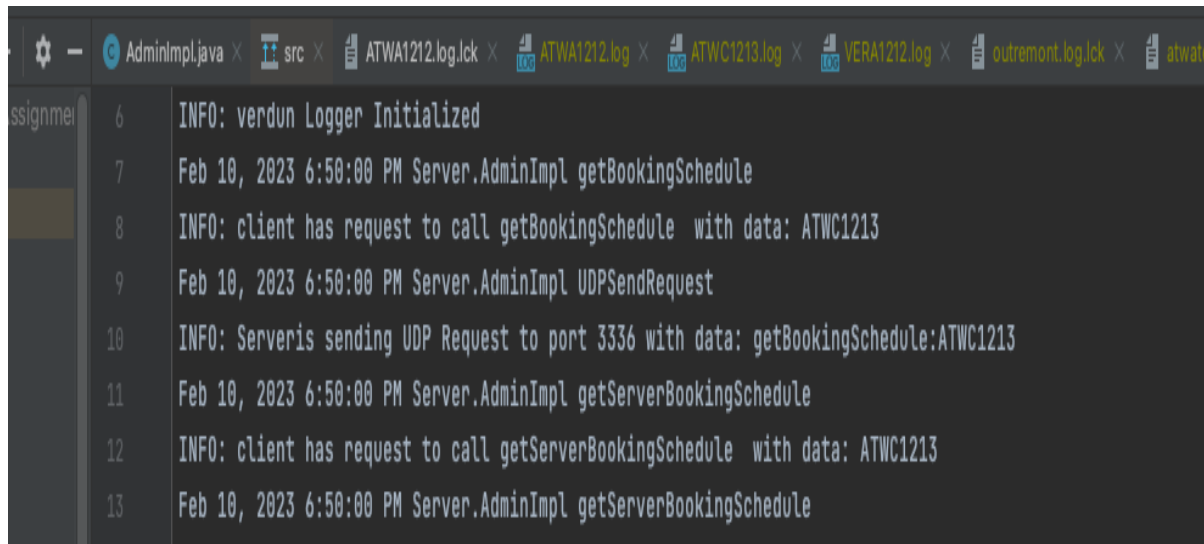
There are three movies for which bookings can be created and there are three movie slots available for each movie on a day (Morning, Afternoon and Evening):

- Avatar
- Avengers
- Titanic

Log Files: The system maintains a log of actions performed by the users and the response from the system when available. Each server maintains a log file containing the history of all the operations performed on that server.

The logs contain information such as date and time the request was sent, request type, request parameters, whether the request was completed successfully or not, and the server's response for the particular request.

Example of logs is shown below.



```

6 INFO: verdun Logger Initialized
7 Feb 10, 2023 6:50:00 PM Server.AdminImpl getBookingSchedule
8 INFO: client has request to call getBookingSchedule with data: ATWC1213
9 Feb 10, 2023 6:50:00 PM Server.AdminImpl UDPSendRequest
10 INFO: Serveris sending UDP Request to port 3336 with data: getBookingSchedule:ATWC1213
11 Feb 10, 2023 6:50:00 PM Server.AdminImpl getServerBookingSchedule
12 INFO: client has request to call getServerBookingSchedule with data: ATWC1213
13 Feb 10, 2023 6:50:00 PM Server.AdminImpl getServerBookingSchedule
  
```

Operations: The system uses UDP/IP messages for inter-server communication for operations such as listMovieShowsAvailability. The system also uses text files for logging operations performed by users and servers. The operations for the customer and admin are given as follows:

Admin Roles:

- addMovieSlots(): Using this method, admin can add movie slots with the information passed to it through the server associated with the admin.
- removeMovieSlots(): Admin can find and delete the indicated movie from their server.
- listMovieShowsAvailability(): Admin can invoke this method to find out the number of tickets available for movie shows in all the servers.

Customer Roles:

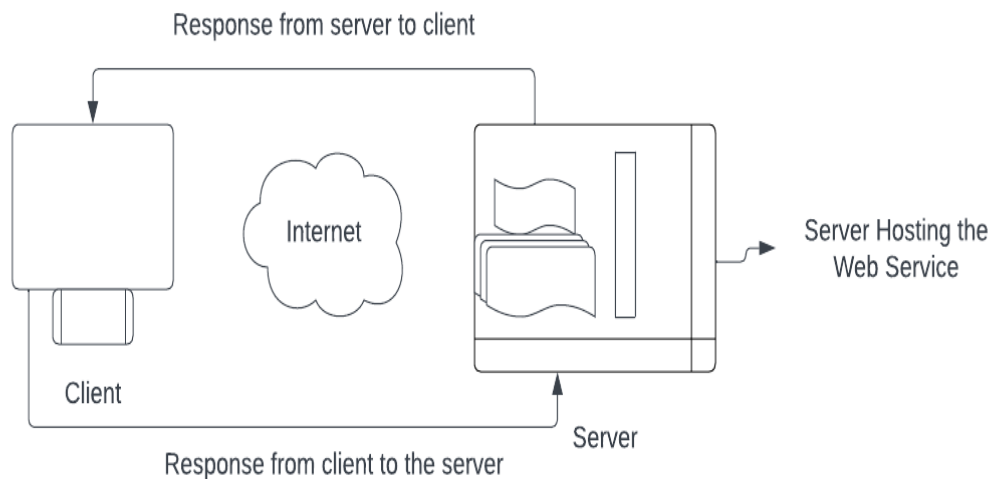
- bookMovieTickets(): Customers can use this method to book the tickets for a movie. Also, customer can only book maximum three movie tickets from other server.
- getBookingSchedule(): Customers can get all the tickets booked for a particular movie through all the theatres.
- cancelMovieTickets(): Customers can invoke this method to cancel the movie tickets for a movie show.
- exchangeTickets(): Customers can swap a booked movie ticket for another movie ticket.

Technologies and Protocols :

The project is implemented using webservice to produce a software failure tolerant Distributed Movie Ticket Booking System.

Web Service Implementation:

A web service is a standardized method for propagating messages between client and server applications. A web service is a software module that is intended to carry out a specific set of functions. It provides a standard protocol or format for communication between applications over the web. The reason behind using this is that it provides platform independent communication. Therefore, two different applications(implementations) can communicate to each other and exchange data or information. Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service.



WSDL(Web Services Description Language):

It is an XML based interface for the web services that is used to describe the functionalities of web services. As it is XML based so it is machine-readable, and tools and the programming language can be used to pass the document and generate the request and response structure.

SOAP (Simple Object Access Protocol):

SOAP is a transport-independent messaging protocol. SOAP is built on sending XML data in the form of SOAP messages. SOAP protocol says that all the exchange of data or information has to be in a format and that format has to be a common format and in case of SOAP it has to be XML. An XML document is attached to each message and information is sent through HTTP which is the standard web protocol. A root element is required in every SOAP document. In an XML document, the root element is the first element. The “envelope” is separated into two halves in which the header comes first which is followed by the body. The routing data is contained in the header and the message is in the body.

UDP (User Datagram Protocol)/ IP(Internet Protocol):

User Datagram Protocol is a communication protocol used primarily for establishing low-latency and loss-tolerating connections between applications on the internet. UDP is standard method to transfer data and information between two computers in a network. UDP is not a connection-oriented protocol as it sends packets directly to a target computer without establishing any connection to the targeted computer. Hence, it is faster but less reliable than TCP (another transport protocol) which establishes a connection via handshake process.

If a UDP datagram is lost in transit, it will not be resent due to which it is able to transfer data much faster than TCP. In our system, UDP is used for inter-server communication. Therefore, the request for a particular operation is send by one server through UDP message to another server. The message body includes all the information required to preform a particular operation.

System Architecture and Data Flow:

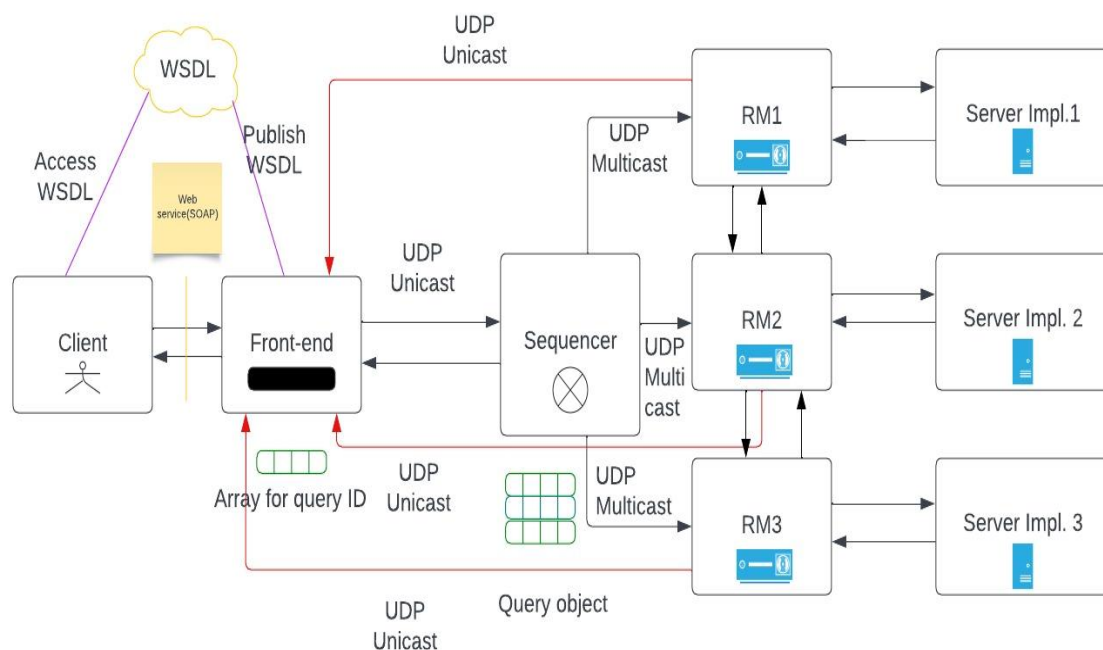
The system architecture describes the major components of the system namely the client, the front end, the sequencer, the replica manager, and the replica implementation.

The **client** communicates with the front-end using web service and all other communications in the system are done through the user datagram protocol. Client sends a **request** to the front-end using web service(SOAP) invocation and a **front-end** is instantiated for the client request. The front end consists of the **publisher** and the **interface** definition. The publisher is used to publish the **WSDL** which can be accessed by the client. The front-end send the request to **sequencer** through the **UDP Unicast** message. The sequencer assign a unique ID to each message request by creating an object which consists of **query ID**, query value and the response (whose default value is null).

Next, the sequencer sends the corresponding messages to the **replica-managers** through **UDP Multicast** message. The query ID is stored by the front-end as well to remember the order in which the request is sent. The corresponding servers will implement the request and send the **response** back to the front end through UDP unicast message. On the basis of result which front-end will get from all the

three replicas it will send the correct result to the client and if the front-end will not receive the result from the replica **within a given time frame** then it will notify about the **software failure** in order to maintain the consistency in the system.

The detailed architecture for the system is shown below with the explanation of each components of the system:



- **Client:**

The client consists of the customer and the admin which can make requests to the front-end using web service invocation method.

- **Front End:**

Front End accept requests through web service (SOAP) invocation method from the client. It consists of the publisher and the interface. The publisher publishes the WSDL and the interface consists of all the methods that needs to be implemented. For each client request, the front-end will send the message to the the sequencer through a UDP unicast message.

The sequencer assigns an ID to each client request which will be stored by front-end as well in order to maintain the order of the requests. The sequencer further sends the multicast request to the replica manager so that the server can implement the required request. In the meantime, front-end will wait for the reply from all the replica managers. Based on the reply it will send the response corresponding to the client request.

- **Sequencer:**

The sequencer is a mediator between the front-end and the replica manager that ensures that client requests are processed correctly and in the correct order. In this architecture, the front-end sends a unicast message to the sequencer with each client request.

The sequencer assigns an ID to the request and sends a multicast message to the replica managers to implement the request. It maintains an object which consists of the query ID, query description and the response whose default value is NULL. The sequencer maintains a sequence number that is incremented for each new client request. This ensures that the request is processed in the correct order, even if there are delays or failures in the network.

- **Replica Manager:**

The replica manager is a component in the system that receives UDP multicast messages from the receiver and sends it to the corresponding server to implement the request in the order of query ID.

Each replica manager maintains a copy of the data in the system and applies the requested operation on its copy of the data. Once the operation is successfully applied, the replica manager sends a reply message to the front-end through UDP unicast message.

The front-end waits for replies from all replica managers and sends a response to the client based on the replies received. Additionally, it maintains the count of number of failures and once the count reaches to

three, it informs the system about the software failure. It ensure that the system remains available and performs correctly in the case of failures.

- **Replica / Server Implementation:**

The server implementation is handled by the replica manager. Each replica is a Java implementation which contains all the methods and the implementation of the functions that the client wants to perform.

A local copy of the data and methods is maintained to implement the methods. The result of the operations will then return to the front-end corresponding to the query ID which can be accessed by the client.

- **Making UDP Reliable:**

Once the **Sequencer** receives the query from Front End, it **creates a Request object** and assigns each query a **unique query ID**, and then passes that Request object to Front End as well as all RMs. Each RM process the query and then attach the response in Request object and return to Front end. FE identifies the response using the Request ID and stores response and return to the client. Every time RM receive a query it **checks it with the last ID received to validate the query order**. In case if any packet is missing, query ID is used to identify the missing packet then it can be obtained from other RMs.

- **Recovery:**

Point of Failure	Handling Technique
Packet lost from Sequencer to RMs in UDP multicast	Every packet will have a request ID and RM will use it to make sure the order of queries. In case of packet loss, RM can request other RM for the query with the query ID.
One Replica generating wrong answers	After 3 consecutive wrong answer from a Replica, FE will send a switch replica message to respective RM, to ensure Replica is replaced with the valid replica. Newly booted replica can process queries from the last query that generated wrong result

	using image of replica till that query. Or it can take a image of current state from other replica.
Packet lost between the FE and Sequencer	Every time Sequencer receives a packet/ query from FE, it assigns the unique ID and returns Request object to FE, so using a waiting time in FE, we can identify if the packet is lost from FE to Seq. or vice versa.

Using Timer with each request sent via UDP, system ensures the response so making it reliable. Also with assigning a query ID to each request made from the client, System ensures the order in which query is processed.

Testing:-

Test Scenarios:

#	Method Name	Scenario	Cases	Status
A		Frontend webservice publishing	WSDL should be accessible on published url	
B		Sequencer udp server startup	Sequencer server must be receiving message	
C		RM server startup	RM server must be receiving message	
D		Front end detecting majority using RMs response	Front end should be detecting error based on all RMs responses	
E		Making sure the order of queries	Storing and identifying the query id to make sure the order of queries	
1		Server Startup after implementing Web Services	3 servers should start	Pass
2		HashMap Initialization	HashMap should be initialized with pre-defined data	Pass
3		Client Connection using Web Services	Client program should run and ask for user ID	Pass
			Client program should be able to determine the type of client and the server to which the user belongs	Pass
			Client should get proper options to perform operation based on its type	Pass

		Log check	Logs generated for server and clients connected in separate files	Pass
5	addMovieSlots(String, String, int, int)	Admin Client	New Movie slot -> Added	Pass
			Fails With Date greater than one week	Pass
			Can Book M, E, A shows	Pass
		Log check	Appropriate logs generated	Pass
6	AdminListMovieShowsAvailability(String) (Admin Client	List all slots available on admin server	Pass
			Fails to get other servers slot details	Pass
		Logs check	Appropriate logs generated	Pass
7	removeMovieSlots(String, String)	Admin Client	remove movie slots	Pass
			remove movie slots and transfer booked customers in that movie slot to next available slot	Pass
		Logs check	Appropriate logs generated	Pass
8	bookMovieTickets(String, String, String, int)	Admin Client/ Customer Client	Book movie for customer by admin	Pass
		Logs check	Book Movie by customer on same server	Pass
			Book Movie tickets by customer on another server but can book only 3 slots in other servers can't more than that	Pass
			Can't Book Movie if already has same slot in different theater	Pass
			Appropriate logs generated	Pass
9	listMovieShowsAvailability(String)	Client customer	List all slots available on 3 servers	Pass
		Logs Check	Appropriate logs generated	Pass
10	cancelMovieTickets(String, String, String, String)	Admin/ Customer	Remove booked tickets for customer by admin	Pass
		Logs check	Remove booked tickets by customer	Pass
			Appropriate logs generated	Pass
11	exchangeTickets(String, String, String, String, String, int)	Admin/Customer	Ticket get Exchanged in Same server and Different Server with all the validation	Pass
			Appropriate Msg when exchange is not possible	Pass

		Logs Check	Appropriate logs Generated	Pass
--	--	------------	----------------------------	------

Individual Contribution and Responsibilities: -

Component	Member
1. Client	1. Shivam
2. Frontend	2. Shivam
3. Sequencer	3. Shivam & Priyang
4. Replica Manager (Query and processing module)	4. Priyang
5. RM (Response and recovery module)	5. Priyang & Shivam
6. Request object and message protocol design	6. All
7. Recovery Design	7. All
8. Testing	8. Pragya
9. Documentation	9. Pragya
10. Individual Replica changes	10. All

Conclusion:

The project successfully accomplished the goals and objectives by enhancing web service implementation of the Distributed Movie Ticket Booking System (DMTBS) and build a software failure tolerance system using replication process. The implementation uses four main components in the system: client, front-end, sequencer, replica manager. The client sends the request to the front-end through web service invocation which further send these requests to the replica manager through a sequencer which assigns a query Id and multicast it to the manager. The replica manager implements the operations and send the response back to the front-end through unicast message from which the client access the corresponding response. It is important to note that all the internal communication are carried out using UDP(User Datagram Protocol) while the client invokes the request to the front-end using web service implementation.

To conclude, the project successfully implemented a reliable and fault-tolerant system for the DMTBS.

References:

1. Prof. R. Jayakumar slides and materials provided.
2. <http://www.java2s.com/example/java/network/>
3. <https://www.geeksforgeeks.org/what-are-web-services/>