DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DISTRIBUTED SYSTEM DESIGN

COMP 6231

SUMMER 2019

Submitted to

SUKHJINDER K. NARULA

Presented By:

Darwin Anirudh Godavari - 40093368

Satish Chanda - 40091187

Venkat Mani Deep Chandana - 40080924

Suthakhar Ponnambalam - 40091123

TABLE OF CONTENTS

OVERVIEW

The Distributed Event management system aims to make sure that the transaction is fully completed or is terminated in whole. This is referred to the property of atomicity. The DEMS has two different systems with their fairly own implementation of the booking system. Each individual system has three different servers namely Montreal, Toronto and Ottawa servers. The DEMS has four different systems operating simultaneously to achieve fault tolerance.

Fault tolerance is a trivial property of a distributed system. In case of a system failure the load from a failed system must be equally distributed between the working components of the system. In order to test this functionality we introduce a software bug that causes the system to crash and visualize how the system responds to a bug. Thus by testing out this property we tend to prove that the system is fault tolerant and highly available.

The DEMS should also be highly available meaning the system should always respond to any user queries without any delay. The load balancing property is essential for a highly available system.

DESCRIPTION OF COMPONENTS

The DEMS has essentially one of the two functionality: Allow manager to access managerial functions and allow users to perform user action. The roles of both manager and user are described below:

The DEMS distinguishes from manager and customer by looking into their login ID. If the ID contains a M as it's fourth letter then it's a manager and if it's a C then it's a customer login. Each server in an individual component will have implementation for the following functionalities.

Furthermore events and their id's have a pattern associated. For instance an event id generated as TORM111219 will be interpreted as below. The TOR denotes that it is an event in Toronto. The M denotes that the event takes place in the morning session and the sequence of numbers denote the date of event in DDMMYY format. Similarly events in Montreal and Ottawa have MTL and OTW as their ids. And events that take place in afternoon have A in them and likewise E for evening.

There are certain constraints in the booking. A customer cannot register for three events that are outside his domain for the same month. But he can book "n" number of events in his city of registration for a given month.

The DEMS incorporates the following functionality for event manager:

1. addEvent (eventId, eventType, bookingCapacity) :-

This method adds an event to the event management system. Once it is up on the system the customer must be able to book for the event.

2. removeEvent(eventId):-

This method is used to remove an event from the event management system. Once an event is removed from the system it must be logged in the logger for future references.

3. listEventAvailability(eventType) :-

This method is used to list the availability for a particular event.

The DEMS also has the functionality for a regular user.

1. bookEvent (customerId, eventId, eventType) :-

This method is used to create an entry for the user to attend an event by using the details obtained in the parameters. The user can book atmost three events outside his city.

2. getBookingSchedule(customerId) :-

This method is used to list the schedule of events currently in schedule for the user based on his id. This should also display events booked outside his city. This involves interserver communication.

3. cancelEvent(customerId, eventId):-

This method is used to cancel an event for a particular user based id.

4. swapEvent() :-

This method allows the customer to swap an event for another. In such a case the booking for a particular event is cancelled and a new booking is made for another event.

ARCHITECTURE DIAGRAM:

The DEMS is designed using the CORBA standards and UDP communication services. Each individual system has three servers for Montreal, Toronto and Ottawa. All servers communicate via the standard UDP protocol. When one of the system fails the replica manager kicks in and performs load balancing between the running components. The user communicates with the Front Ends (FE's). The system sends an output to the user based on the mentioned calculus. When two systems produce the same output and one system produces a different output then the voting goes to the systems having similar outputs.

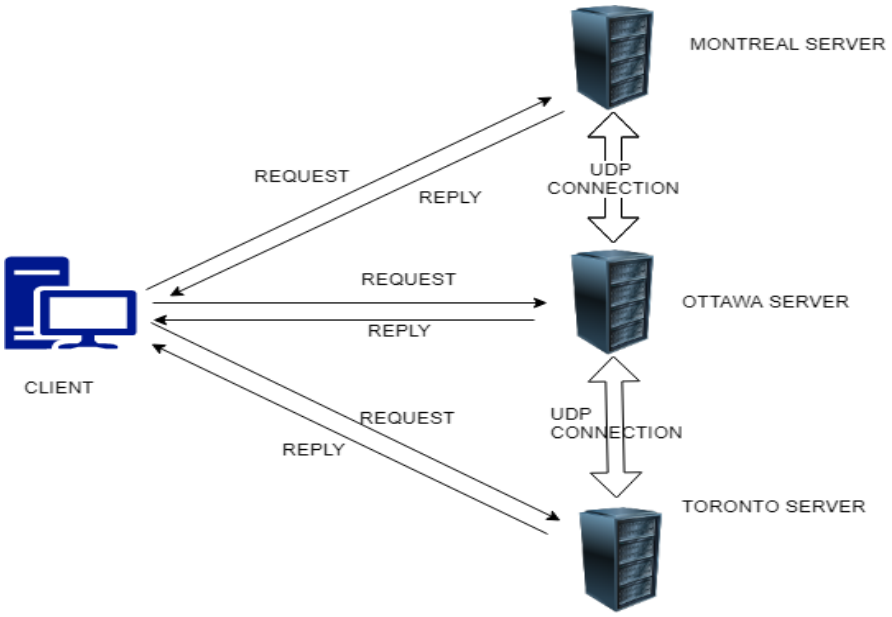

Fig 1a: A simple client server interaction.

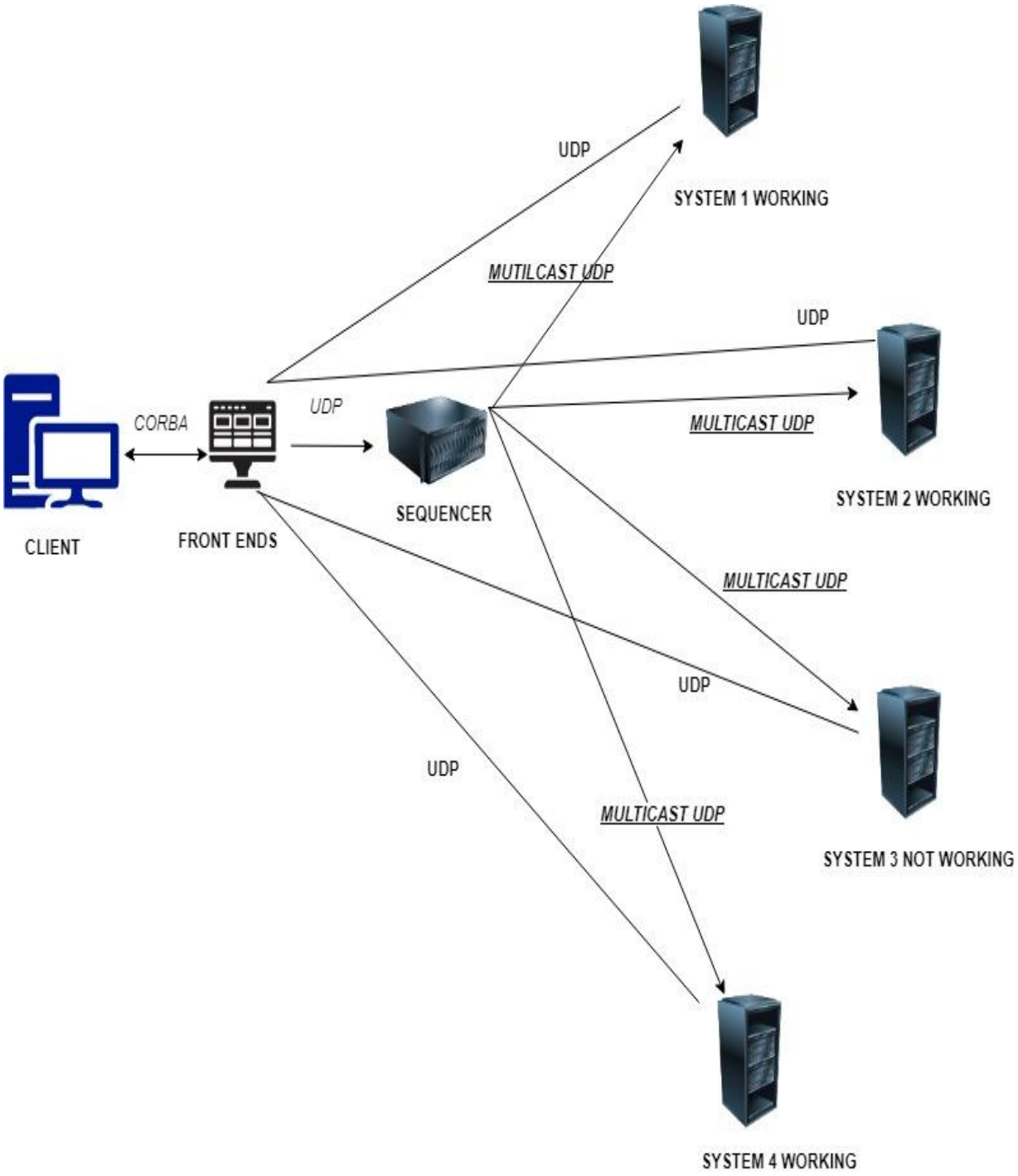

Fig 1b: Architecture diagram for DEMS

Test Scenarios:

TEST CASE	SUMMARY	EXPECTED RESULT	OBSERVATION
T1	The user (manager/customer) is able to log in and his/her functionalities are displayed.	The user logs in and menu items are displayed.	The client made successful login and menu was displayed correctly.
T2	If the client is a Manager all . the six functions are accessible. This is identified in user id using "**M**".	The manager will have managerial and customer functions,	The manager on successful logging had all the six functions available to him/her.
T3	If the client is a customer then only three primitive functions are accessible.	The customer will have options to book, cancel and get schedule.	The customer had only three basic functionalities thus validating the code
T4	The customer is able to book an event with his user id, unique id and event type.	The customer is able to book an event.	The customer booked an event successfully
T5	The customer cancels a booked event.	The event record will be removed and the total capacity will be incremented.	The customer cancelled a booking successfully.
T6	The customer displays her booking schedule using get booking schedule function.	The list of events booked for a customer will be displayed.	The schedule of events booked was displayed.
T7	The customer cannot book more than 3 events outside his city.	The customer will not be allowed to book a fourth event outside his zone.	The customer couldn't perform a fourth booking operation since it violated the booking policy.
T8	The manager adds an event to the DEMS and it is updated in the hash map.	The event is added with its id, type and capacity in the hash map.	The event was added successfully to the DEMS
T9	The manager can remove an event.	The manager removes an event and it is reflective in the hash map.	The output was reflective in hash map thus validating the operation.
T10	The manager finds availability for all events.	The availability in all the servers for all the	The availability pertaining to servers to all events was reflective.

		events will be displayed.	
T11	The server response is logged in the logger for every request.	The logger file will be appended with all the status messages after each server response.	The logger was updated successfully
T12	The DEMS should not allow the manager to remove an event that does not exist in the system.	The DEMS will probe the user that an event does not exist and hence cannot proceed with cancellation.	The event could not be cancelled as it does not exist
T13	The DEMS should not allow the manager to add an event that already exists.	The system did should not create a duplicate event.	The system did not create a duplicate entry for the same event.
T14	The DEMS will only display available events based on event types.	The system will not display events until the event type parameter is provided.	The event type parameter was provided and the availabilities were displayed.
T15	The DEMS will allow for a swap event that is already registered in the system.	The system is checked if the event exists for a swap and if so then performs the swap.	The system performed a successful swap
T16	The system will not swap an event that is not listed.	The system checks for event if present.	The event is not listed and so the swap is not swapped.
T17	The system will not cancel an event for another event when the event to be cancelled is not available.	The system checks for event if present.	The event is not listed and so the swap is not swapped
T18	The DEMS system when performing a swap checks if the user is not booking three events outside his city.	The system checks for max booking when performing a swap.	The swap was not performed when the max booking capacity was reached.

CHALLENGES FACED:

The following challenges were encountered during the development of DEMS.

1. Inter server Communication

The communication establishment among servers was very difficult to code as the return statements needed to be re adjusted to every single letter and the flow of the program should be similar.

2. Multicast Communication:

Spent a lot of time testing for fault tolerance using multicast UDP communication , assigning the right sever IP address and creating a group for multi cast.

3. Concurrency issues

Implementing multithreading was a tedious task and were faced with many errors while coding.

REFERENCES:

- https://www.techopedia.com/definition/1293/common-object-request-broker-architecture-corba
- https://www.corba.org/
- https://www.javatpoint.com/q/4780/corba
- https://www.geeksforgeeks.org/client-server-software-development-introduction-to-common-object-request-broker-architecture-corba/
- https://www.tutorialspoint.com/software_architecture_design/distributed_architecture.htm

Contributions: -

Replica Manager 1 and Replica Manager 2	Mani Deep and Satish
Replica Manager 2 and Replica Manager 3	Darwin and Suthakar
Sequencer	"As a team"
Front End	"Team as a whole"
Modifications of server 1 & 2 re - doing	Mani Deep and Satish
Modifications of server 3 & 4 re - doing	Darwin and Suthakar
Test case Design and implementations	"As a team"