Report on

"Real-Time Weather Event Notification System using Pub/Sub"

SANTA CLARA UNIVERSITY

Submitted by

Prathmesh Bhondave (07700006006) - pbhondave@scu.edu

Mrunank Pawar (07700005998) - mpawar@scu.edu

Suyash Salvi (W1653460) - ssalvi@scu.edu

Guided By:

Prof.Ramin Moazzeni

Table of Contents

***Abstract*—This paper presents the Real-Time Weather Event Notification System which uses the publisher subscriber approach to ensure that it alerts users of weather events in real-time and precisely. Thus, this project is aimed at solving the main issues that affect distributed systems: scalability, reliability, fault tolerance, consistency, concurrency, and security. With the help of these technologies and distributed systems algorithms such as leader election, mutual exclusion, heartbeat, the system provides reliable and efficient messaging concerning critical data. The application architecture is inherently highly available and easily scalable, with an AWS EC2 environment. Besides providing the practical implementation for the real-time weather update notification service, this project provides a good opportunity for practical learning of the distributed systems concepts and their application in practice. The efficiency and effectiveness of the system is examined through specific parameters in order to give a proof of improved system functionality. These are plans for future enhancements of the system by adding edge computing, a dynamic load balancer, and better monitoring of the consumers and the whole system.**

I. Project Goals and Motivation

In today's fast-paced world, staying informed about immediate weather changes and potential hazards is increasingly challenging due to busy schedules and frequent distractions. The idea behind the development of the Real-Time Weather Event Notification System stems from the recognized requirement to provide users with proper, individual weather notifications that will allow them to organize their day and protect themselves from potential threats. The existing weather alerting systems do not possess the required speed and relevance for them to be efficient. It is common for users to receive messages that are not localized and hence, do not meet the needs of the particular community or region for which the information is being conveyed hence making communication of important information inefficient. Thus, the employment of technologies and distributed systems principles in this project will help to overcome this, and make sure that the users get updated and appropriate information on weather. With the increasing frequency of extreme weather events, there is a growing need for reliable systems that can quickly and accurately notify users of impending weather conditions. This system aims to meet that need by providing real-time updates and personalized alerts, thereby improving user's ability to respond to weather events effectively.

Furthermore, the project serves as a platform for learning and innovation. It offers an opportunity to explore and implement distributed systems concepts, providing practical insights and experience that can be applied to future projects. The motivation is not only to create a functional system but also to push the boundaries of current weather notification systems and explore new possibilities in distributed computing.

- **Real-time Weather Alerts:** The main purpose is to offer the user current weather information and to make sure they receive information about immediate changes and dangers. This is very important in planning for the daily activities and protecting oneself especially in cases of natural disasters.
- **Distributed System Architecture:** Another notable objective is to ensure that there is a proper architecture of the distributed system. This architecture is aimed at making the system more reliable and effective to render high volumes of data and several requests from the users. With the help of modern technologies and architectural solutions, high availability and tolerance to failures can be achieved.
- **User-Centric Personalization:** One of the major objectives of the work is the creation of user-oriented personalization. The system is expected to use state of the art geographic and preference based filtering technologies to provide an individual weather alert. This makes certain that the users get the right information suitable for their needs and geographical locations, making the system more useful to the users.

The following are the project goals for the Real-Time Weather Event Notification System which seeks to solve this problem of inadequate personalized weather updates. While developing this system, some objectives were set in order to make the system efficient and effective as follows. The project includes the provision of timely weather reports, usage of a strong distributed system infrastructure, and the incorporation of specific user's preferences.

II. Design Choices and Distributed Systems Challenges Addressed

The Real-Time Weather Event Notification using Pub-Sub Distributed System project raises several issues that are characteristic of distributed systems, and which seek to improve the effectiveness and timeliness of weather alerts in a rapidly changing information environment.

- **Message Ordering:** It is very important to manage the lifecycle of messages to avoid sending out old weather alerts that were sent to the clients. This is done by putting in place measures that will enable the setting of Time-to-Live (TTL) on messages to make them timely. The messages are given the TTL values which determine the temporal validity of the information. The backend services take advantage of TTL fields in the database schema to self-prune old messages, while periodically cleanup jobs that are used to prune the system of old messages, which makes the delivered information to the users relevant and accurate.
- **Message Scheduling:** Another major issue in the implementation of the application is the optimization of the delivery of the weather updates depending on the

user's preferences and activity. To tackle this, there is a mechanism of delayed messaging and users can choose when they want to receive weather updates. This approach makes it possible to notify the users at the right time when the information is most beneficial to them, thus improving the usability of the application.

- **Leader Election:** Leader election is used for the synchronization of activities and to maintain consistency across the distributed components. This challenge is solved by the use of leader election algorithms to elect the leader node that directs decision-making. The elected leader node is responsible for some of the most important coordination and makes sure that the different parts of the system are coherent. The 'ManagerService' class is to coordinate the election of the leader among the backend nodes and to provide the system with the necessary stability even when failures or other problems occur.
- **Reliability and Fault Tolerance:** Availability of services even in case of failures or network problems is one of the key non-functional requirements. This is done through data and service duplication on various nodes so that the services can be available even if some of the nodes are down. A failure is detected and traffic is automatically rerouted to a healthy node, and nodes are continually checked for their health. The backend services are developed using Java which offers excellent fault tolerance and can be easily replicated, and the database replication provides data availability and data integrity, which also increases the reliability of the entire system.

To make sure that the Real-Time Weather Event Notification System can be easily expanded and is stable, several design decisions were made. AWS EC2 instances offer the needed flexibility because the system can easily increase resources for accommodating more users and data as the load increases, without slowing down the system. For the purposes of reliability and fault tolerance, prioritized and reliable message storage and delivery are provided to guarantee that critical alerts are processed and delivered as soon as possible. Using several AWS EC2 instances is useful in preventing the creation of a single point of failure, thus making the system more robust. Also, the failover mechanism is implemented to provide uninterrupted services in case of failures, redirecting traffic to the functional nodes. These design choices collectively contribute to the system's robust performance, scalability, and reliability.

III. Prior Work

When designing the Real-Time Weather Event Notification System, we analyzed the existing systems and technologies that can be used to develop this system and to follow the best practices. The knowledge of the pub/sub brokers, leader election algorithms, middleware for real-time notifications, and multicast programming helped to consider the peculiarities of distributed systems' implementation.

The D-MQTT system is one of the earliest works in the design of pub/sub brokers for distributed settings, proposed by Staglianò, Longo, and Redondi [1]. Based on the findings, it is evident that the "D-MQTT" system is uniquely developed to improve data management and capacity in a distributed environment. This system also stresses on message brokering, which is very central to our project since it determines the speed and reliability of delivering the weather notifications. The main aspect of the architecture of the D-MQTT system is the reduction of latency and high throughput, which is crucial for applications such as the notification of a weather event. By leveraging advanced data distribution techniques and ensuring robust message delivery, the D-MQTT system provides a reliable framework that significantly influences our design choices for the Real-Time Weather Event Notification System.

Leader election is one of the core issues in distributed systems, which is important for keeping the consistency and availability of the whole system when some nodes are failed or the network is partitioned. The paper by Yadav, Lamba, and Shukla [2] and the work by Garcia-Molina [3] give a detailed overview of leader election algorithms. Their findings reveal several algorithms and protocols that guarantee the availability of a leader node in the nodes coordination. These leader election mechanisms are the foundation of the system's ability to be fault tolerant and recover. Through the use of good leader election algorithms, there is the ability to maintain consistency and functionality of the system in spite of partial failures, thus upholding the functionality and reliability of the weather notification service.

Another area of related work is the employment of messaging broker middleware in real-time notification systems. A look at using middleware in the monitoring of public transportation systems, and how it can help in understanding how to maintain proper communication between the components of the system. From their findings, they show how middleware can manage large amounts of data and deliver notifications on time, which is relevant to our project. Thus, we plan to employ similar strategies to provide efficient message distribution and low latency communication to the users to immediately deliver weather alerts.

Multicast programming techniques are crucial for the delivery of messages to many users in a large network. The work by Eugster, Boichat, Guerraoui, and Sventek [5] gives the guidelines for multicast programming in large scale systems. Their approaches affect our multicast implementation, especially for disseminating weather alert to the multiple subscribers. Through the use of multicast, it is possible to disseminate weather notifications to many users at once thus avoiding overloading the network while at the same time ensuring that the notifications get to the intended users as soon as possible. This approach improves the modularity of our system so that it can accommodate more users in the future without much difficulty.

IV. Project Design

A. Key Design Goals in Designing Distributed System

The following objectives have been considered while designing the Real-Time Weather Event Notification System to meet the expected level of performance, dependability, and

satisfaction level of the user for a weather notification service. The following sections elaborate on each of these design goals in detail:

- **Heterogeneity:** The Real-Time Weather Event Notification System is built with an architecture that makes it portable across different hardware and software platforms. This heterogeneity is possible through making the system's components independent of the platform they are used in.
- **Openness:** Another important non-functional requirement is openness which means that the system should be compatible with other platforms and systems. This is achieved by publishing RESTful APIs through which other systems and clients can access the weather notification service.
- **Security:** Security is an important factor that is considered in the development of the Real-Time Weather Event Notification System. For security, the communication and data are safeguarded from other unauthorized persons and attacks through the use of encryption and authentication.
- **Failure Handling:** To make sure that the service is always available, the system has mechanisms of fault tolerance and recovery. Such mechanisms include data and services replication to different nodes, such that in case some nodes are down, the system is still running.
- **Concurrency:** One of the most important issues in the distributed systems is the concurrent access and update of data. The Real-Time Weather Event Notification System solves this problem through the use of concurrency control such as locks and semaphores. These mechanisms help to synchronize the access to a certain piece of code or data, so that only one process can use the critical section at a time.
- **Quality of Service:** The delivery of the weather alerts is another factor that should be well coordinated so that the message gets to the target users on time and most importantly delivered as intended. There are QoS mechanisms to support the important messages and control the flow of the data. It has priority queues that help in sorting high priority weather alerts to be delivered to the users separately.
- **Scalability:** Scalability is a key non-functional requirement that makes it possible for the system to accommodate many users and data without degrading performance. AWS EC2 instances are used to ensure that the system can easily and efficiently manage the resources needed depending on the current situation.
- **Transparency:** The operations of the systems should be transparent in order to support a good user experience. The concept of the Real-Time Weather Event Notification System is intended to be transparent to the users; thus, the system should not interfere with the users' activities in any way. This is done by using several forms of automated processes and interfaces that shield the users from the actual distributed system environment.

B. Key Components and Algorithms

The components and algorithms of the Real-Time Weather Event Notification System are as follows: This section gives a comprehensive description of these crucial elements and the algorithms that are involved and their roles in the general framework.

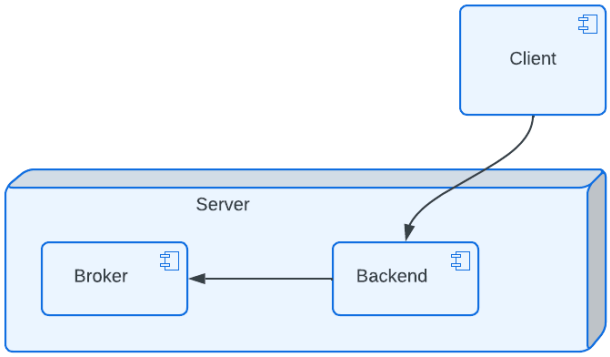
- **Weather Data Publishers:** Weather Data Publishers are part of the Real-Time Weather Event Notification System. These components are used for gathering the actual weather information from different sources. After being gathered, this data is then transformed and normalized to fit the system's specifications. The publishers then forward this weather data to the system for more processing and dissemination.
- **Message Broker:** The Message Broker is also very central in handling the delivery of the weather alert to the subscriber. While such purposes are typically served by a message broker with similar features might be used in our system. The message broker gets the weather data from the publishers, filters and forwards it to the subscribers according to the topics and channels set.
- **Subscribers:** They are the final consumers of the system who receive customized weather notification. A subscriber can define his or her preferences and location and the system will only send out notifications of weather in those areas of interest. The system guarantees that all the subscribers get the right information at the right time and in the right place making the weather notification service more useful and valuable.
- **Leader Election Algorithm:** The Leader Election Algorithm is critical in ensuring that the different nodes in the network are well coordinated and in harmony. A very important factor in a distributed system is to have a node that is responsible for the important tasks and decision making. The leader election algorithm guarantees that just one leader node is elected out of a number of nodes even in the event of failures.
- **Heartbeat Protocol:** The Heartbeat Protocol is used for checking the health of nodes in the system and is similar to the two protocols described above. Every node transmits an acknowledgment signal after a fixed time to show that it is still active. If a node does not send a heartbeat signal within a given time frame, it is assumed that this node is down or faulty. This can then prompt the system to perform corrective actions such as issuing failover to healthy nodes for the tasks. This protocol helps to guarantee that the failure of nodes can be identified as soon as possible to maintain high availability and reliability of the system.
- **Broadcast or Multicast:** The Real-Time Weather Event Notification System employs both Broadcast and Multicast techniques to convey the weather alerts. In a broadcast approach, all nodes in the network are informed, and each user gets the notification even if they are not subscribed to the sender's list. This is particularly helpful in disseminating mass messages for instance alerts such as natural disasters. On the other hand, multicast

directs messages to a particular set of subscribers who have expressed interest in receiving the messages, thereby minimizing the load on the network and making sure that only those who want to receive the notifications get them. It is cost-effective in the utilization of network resources and delivers the alerts in the most efficient and customized manner.

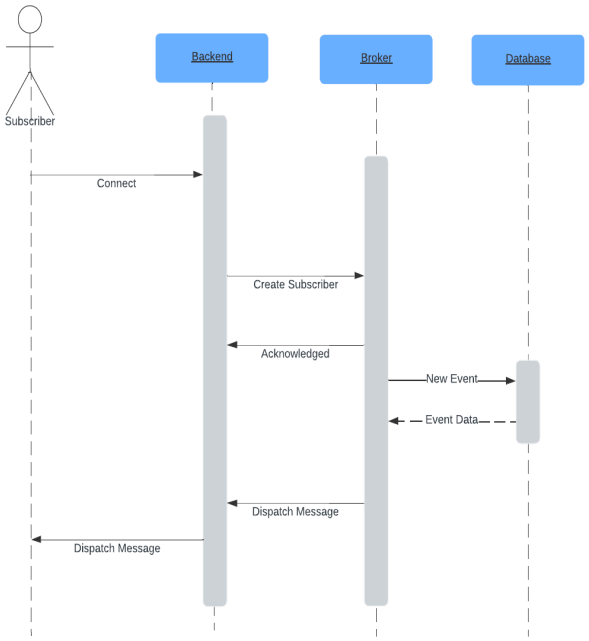
V. Architecture

The system architecture consists of three primary parts. The user establishes a socket to the server. There is a concept of a middleware that is between the user and the server which processes the messages between the two. The middleware is also involved in the election of the leader as one of its significant responsibilities. The architecture can be broadly categorized into client-server architecture with the client sending requests and the server processing these requests.

A. Component Diagram



B. Sequence Diagram



VI. Evaluation

The chosen approach for Real-Time Weather Notification Pub-Sub Distributed System is based on the algorithms used for communication, subscription, and event notification. The algorithms are intentionally and logically developed, which makes them correct and fast.

A. Evaluating algorithms

- **Leader Election Algorithm:**
  - *Description:* The leader election algorithm is used to elect a single leader node out of the distributed nodes to coordinate some important tasks. It is a simple and reliable protocol for a distributed environment. The leader node is responsible for maintaining the state of the system, scheduling updates and receiving and processing requests from the clients.
  - *Correctness:* The leader election process is made fault-tolerant, that is, it is possible to elect a new leader even if some nodes or network partitions are unavailable. The algorithm ensures that there is only one leader and all nodes have to agree with the decision made by the leader thus ensuring system coherence and integrity.
  - *Complexity:* The process of leader election is complex and it takes O(n) communication steps where n is the number of nodes in the system. In the election, each node sends request messages to a majority of other nodes to get votes and

hence the process is efficient and can support a large number of nodes.

- **Heartbeat Protocol:**
  - *Description:* The heartbeat protocol is used to check the health status of the nodes in the system. Every node sends a heartbeat message to other nodes to show that it is working properly. If a node does not send a heartbeat within a given time, then it is regarded as a failed node.
  - *Correctness:* The heartbeat protocol is efficient in identifying failed nodes by the lack of heartbeat signals in the network. This makes it possible for the system to quickly start the failover processes hence enabling it to run at all times with little interruption.
  - *Complexity:* The time required to send and receive heartbeat signals is constant and depends on the number of nodes, thus it is O(1). The overall system overhead is proportional to the number of nodes, but each individual heartbeat operation is simple and efficient to reduce the overhead of health monitoring.

- **Notification Mechanism:**
  - *Description:* Notification mechanism is used to inform subscribers of new events depending on the subscriptions and uses flags and events dictionaries.
  - *Correctness:* The notification mechanism enables the subscribers to receive only the type of weather notifications they want. It is effective in identifying the subscribed users and provides the right updates to the clients, thereby preserving the credibility of the notification system.
  - *Complexity:* The time required to complete the notification mechanism is also O(n), where n is the number of subscribers that will receive the notifications. This makes it possible to have efficient handling of the notifications as the subscribers of the system increases.

B. Addressing Key Issues

- **Fault Tolerance:** Availability is another major characteristic of the Real-Time Weather Event Notification System since it is important for the system to continue running despite failures. This is done through several methods like Data and Service Replication in which the system copies data and services on different nodes. This redundancy means that in case one of the nodes is down, there is another one that can perform all the tasks of the former node without any interruption or loss of information. Also, the leader election algorithm that is implemented guarantees that there is always a leading node in charge of the critical functions. If the current leader cannot perform the task, the algorithm immediately selects a new leader to ensure the continuity of system organization and order. The system is able to check the health of nodes through a heartbeat protocol in order to detect any problem. To check the status of each node, it sends a heartbeat signal at a given interval of time. If a node does not send a heartbeat within a certain time period it is considered to be down.

- **Performance:** It is important to achieve the best results as far as providing timely weather reports are concerned. Java used for the backend is a very stable and efficient platform to work with that can easily process high loads of data. The backend takes the weather data and applies the business rules to it in a very short time. The system incorporates fast algorithms for data processing so that it can process data quickly and in large quantities. Data is cached where necessary to enhance the rate of access and use of data as well as to minimize the time taken in the process.

- **Scalability:** The ability for growth is important to handle more users and more data. AWS EC2 instances are chosen as the infrastructure as it is flexible and can be easily scaled. It is possible to add or remove instances depending on the current load, which helps to implement the concept of scalability. A load balancer helps in distributing the traffic that is received in an organization across several instances of EC2. This helps in avoiding any given instance to be a point of contention and hence makes efficient use of resources. The concept of modularity in the system enables the different parts of the system to be sized differently. This means that specific portions of the system, for instance, the data processing or messaging part, can be scaled according to the load that is expected to be met. This aspect of scalability is beneficial because it allows the system to expand and accommodate more users without compromising the system's efficiency. AWS EC2 instances allow to meet varying loads and load balancers help to distribute traffic evenly. The modularity of the system enables the efficient management of growth and resources while keeping the system highly efficient.

- **Consistency:** It is necessary to achieve consistency among the nodes distributed in the system to guarantee the correctness of data and the state of the system. The leader node is in charge of synchronizing update and making sure that all nodes have the same data and state. Leader election algorithms guarantee that a new leader is chosen in the shortest time possible if the current one is out of service, so the system remains coordinated. This is achieved through the use of distributed transactions that guarantee the application of updates on all the nodes. The transactions are made to be atomic, it means that either all the nodes that are involved in the transaction will apply the update or none of them will. These consistency mechanisms are useful because they avoid data

inconsistencies and guarantee that all components of the system are harmonized.

- **Concurrency:** Concurrency control is a crucial aspect of a system that has multiple users accessing the information at the same time to avoid data corruption. Locks and semaphores guarantee that only one process can be in the critical section of code or data at any one time. This avoids problems and guarantees that the information is correct. Isolation property of ACID means that concurrent transactions should not affect each other. Update transactions are not mixed to avoid the corruption of data, and updates are only made if they can be done uniformly.

- **Security:** Privacy is very important in preventing unauthorized access to information and allowing only the permitted users to access the system. Data that is stored and data that is in the process of being transmitted is protected using common encryption methods. This ensures that only the intended recipient receives the message and no one else intercepts the message. All interactions between the client and server are done through HTTPS, which makes the data going through the network to be encrypted. These security measures are effective because they safeguard the system from many risks such as leakage of data and unauthorized access.

VII. Implementation Details

A. *Architectural Style*

In terms of architecture, our Real-Time Weather Event Notification System mainly involves the client-server and distributed system style, which can effectively guarantee the system scalability, reliability, and real-time performance. This architecture breaks down the system into layers of modularity, where unique components of the system are individually responsible for accomplishing specific functions ranging from data gathering and analysis to notification distribution. To achieve this, we incorporate a leader election mechanism and the store and forward scheme as sinks to maintain the anonymity of the publishers and the subscribers. This design enables us, therefore, to manage large volumes of data and the users' request without the system slowing down in delivering customized and timely weather alerts to the users.

Key Points of the Architectural Design

1. **Client-Server Model:** Chosen for its simplicity and effectiveness in ensuring modularity, the client-server model allows the seamless addition of new subscribers without impacting the core system, making it easy to scale.

2. **Leader Election Model:** Designed for improved co-coordination and robust failure handling, the leader electioning exercises help achieve reliability to the system as it self-organizes in the event of failure by the current leader.

3. **Layered Architecture:** The problem is that this approach isolates issues associated with broker performing leader election and messaging from server logic that deals with subscriptions. Sharing of data in this design also improves modularity as well as the flexibility of the system, thereby making the extension of the system easier.

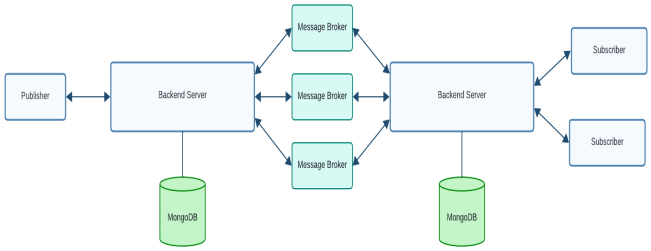
4. **Gossip Protocol:** Used in health monitoring, and failure notification, the gossip protocol is effectively used in making sure that the system can detect nodes that have failed and efficiently handle the situation without causing large disruptions to the entire system.

5. **Broker Rebalancing:** To ensure continuous service, broker rebalancing redistributes the load among remaining brokers if one fails, maintaining service availability and system reliability.

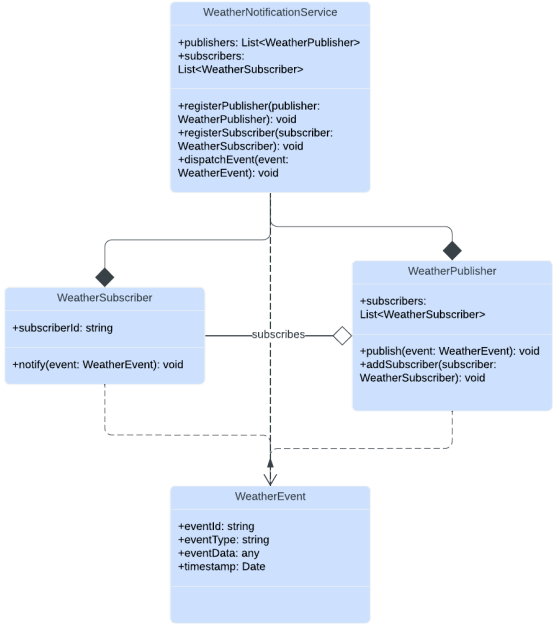
6. **Publisher and Subscriber Login/Sign-Up:** User registration: enable users to sign up on the specified platform and also makes it easy for the user to verify their identity hence offer them personalized access to the system thus improving the experience.

7. **Topic-Specific Data Management:** Helps to present rich options for communicating since the publishers can post messages to the selected topics while the subscribers can receive messages for only the topics they subscribed to, thus providing timely and relevant information.

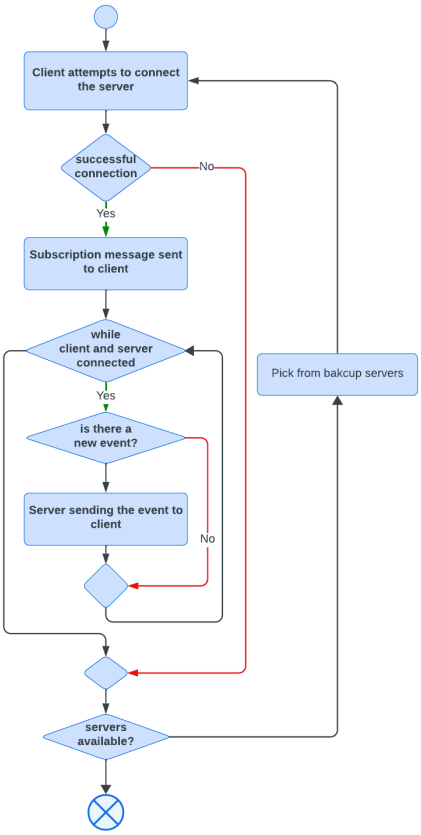
B. *Architecture Diagram*



C. *Class Diagram*



D. Interaction Diagram



E. Software Components and Services

- **HTTP Communication:**
  - *Description:* The communication in the system is mainly carried out through HTTP-based interactions. Publishers, broker, and subscribers use http requests for communication. The broker receives messages from the publishers through HTTP calls and the broker has the responsibility of handling the system and directing the messages to the subscribers.

- **Leader Election:**
  - *Description:* The leader election mechanism is also a part of our system and is implemented to be highly available and reliable. In the event that the current leader node is incapacitated or becomes unreachable, the system will automatically trigger an election for a new leader, thus ensuring continuous functionality.

- **Backend Services:**
  - *Description:* It involves managing the publisher and subscriber, routing the APIs, and reading messages starting from a certain offset. It handles incoming requests, performs validation checks on data, and communicates with the message broker to direct messages to the correct destinations. It is also responsible for user authorization, user and subscriptions data storage, and business logic to ensure the integrity of the system.

- **Message Broker:**
  - *Description:* It is responsible for the coordination of the communication and the management of the leader election process as well as the proper delivery of messages between the publishers and the subscribers. Relays messages from the publishers to subscribers and guarantees that the messages are delivered to the right subscribers based on their preferences.

These are the main software components and their interactions, which make up the Real-Time Weather Event Notification System. Through communication, leader election processes, availability of strong backend services, and effective message routing, the system achieves a coherent and reliable distributed system design. This arrangement ensures that weather information is delivered to subscribers at the appropriate time and in a way that is relevant to them.

VIII. Demonstration

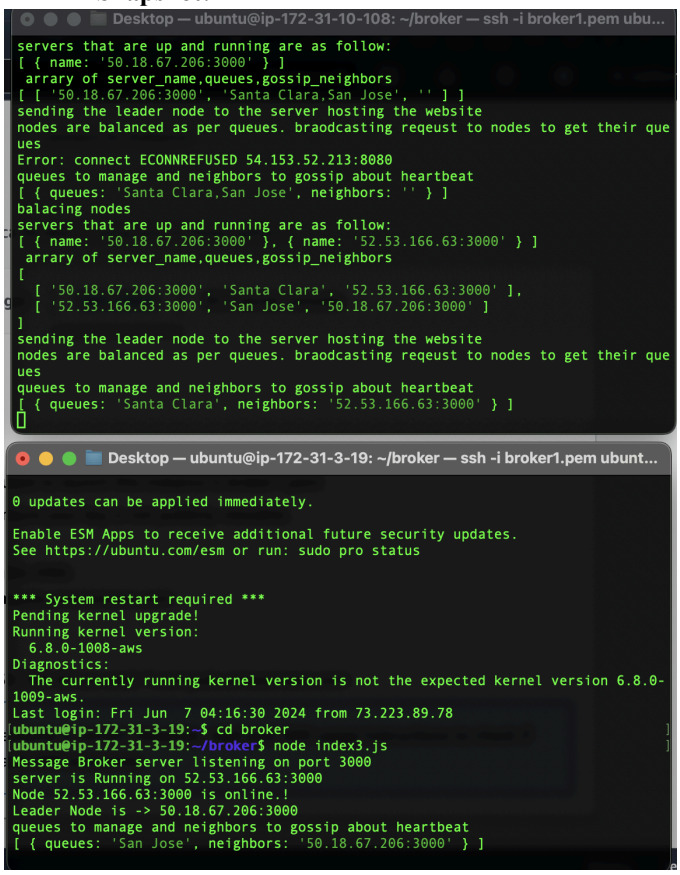
A. Successful Run

- **Description:** The system needs to perform the process of leader election, notify the neighboring nodes, connect the subscribers, and facilitate in consuming message with offset from the broker in a successful run.

- **Steps:**

1. The system successfully elects a leader.
2. The list of neighboring nodes is updated.
3. Subscribers connect to the system without issues.
4. Subscribers read messages from the broker with a specified offset.

- **Snapshot:**



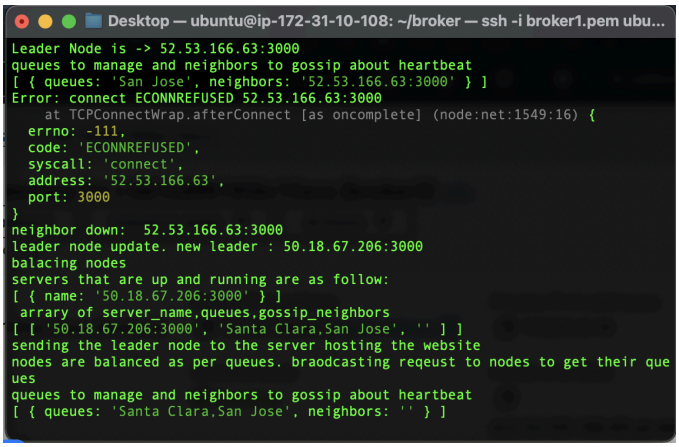
B. Failure – Leader Fails

- **Description:** When the current leader node fails, the leader election algorithm initiates, resulting in the election of a new leader and notification to all nodes.

- **Steps:**

1. **Leader Node Failure:** The system detects the failure of the current leader node.
2. **Leader Election Initiation:** The leader election algorithm is triggered.
3. **New Leader Election:** A new leader is elected.
4. **Neighbor Update:** Neighbor nodes update their leader information.
5. **Balancing Nodes:** The system balances nodes and updates their status.

- **Snapshot:**



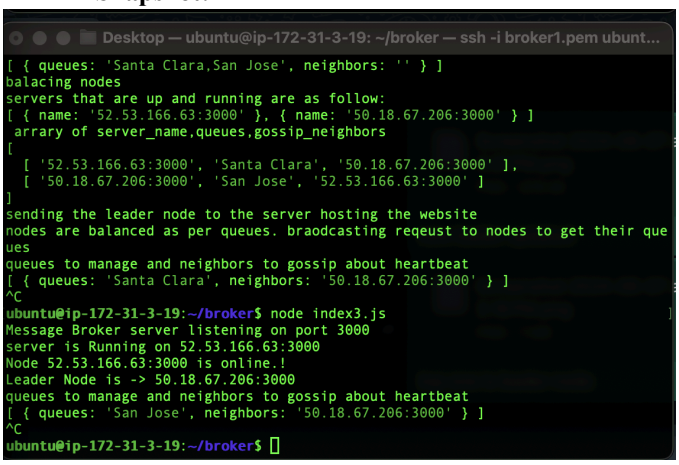
C. Failure – Non-leader Fails

- **Description:** If a node which is not the leader node is faulty, the system removes it from the list of nodes and the list of neighbors and ensures that the system is connected to the other node.

- **Steps:**

1. **Node Failure Detection**: The system detects the failure of a non-leader node.
2. **Node Removal**: The failed node is removed from the active nodes list.
3. **Neighbor List Update**: The neighboring nodes list is updated to exclude the failed node.
4. **Reconnection**: The system ensures connections are made to other available nodes.

- **Snapshot:**



These are examples of how the system is likely to respond and remain steady in case of failure in the above possibilities. The Weather Notification Pub-Sub Distributed System has provisions for various situations such as disconnection, failure of a leader node, and various other errors; thus, the Pub-Sub system is stable and can be considered safe to use.

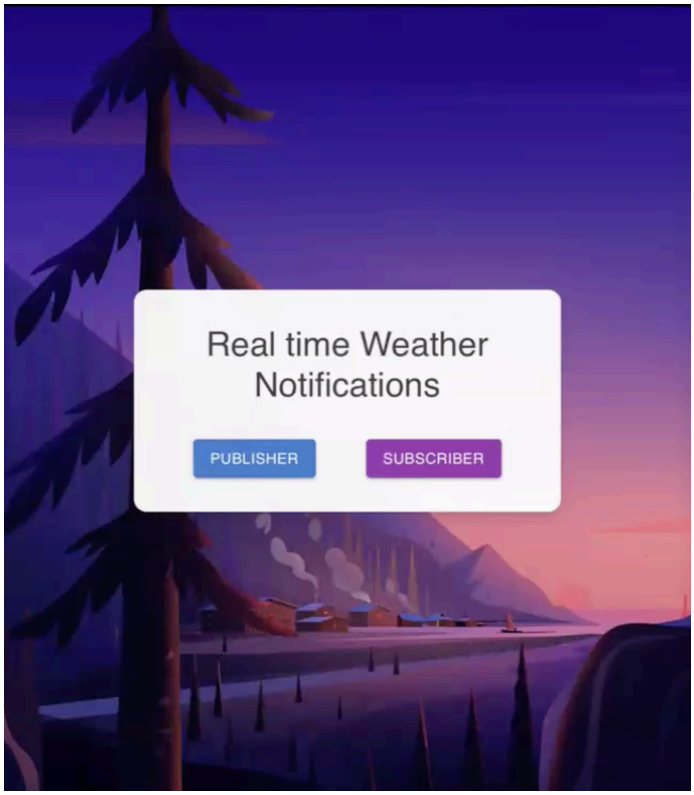
D. Working of Weather Notification PubSub System

The weather notification service application comprises multiple components that collaborate to deliver a seamless messaging experience to the user.

1. Landing Page Description

The image displays the interface of the Real-Time Weather Notifications system where it provides the landing page. Here, there is a user option for selecting the user's role as either the Publisher as well as the Subscriber. They can then proceed to choose which role to be assigned to them in the system where they can then login to access the system functionalities.

- Publisher: Users in this role can publish weather notifications.
- Subscriber: Users in this role can subscribe to weather notifications and receive real-time updates.

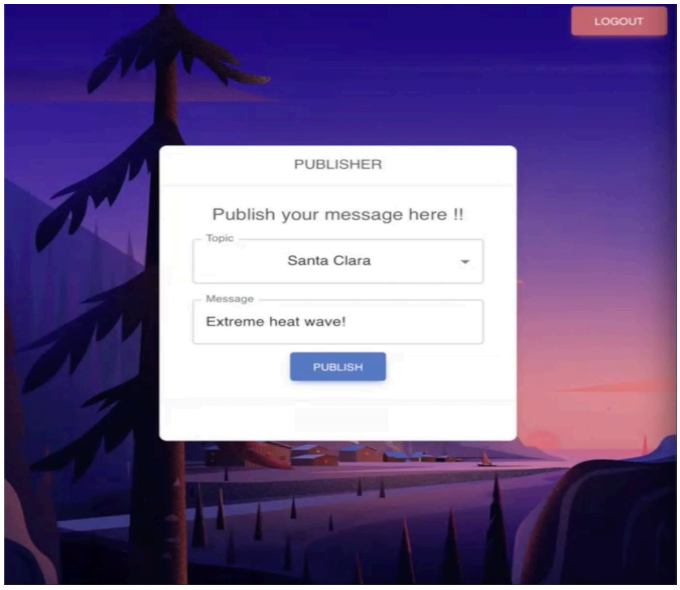


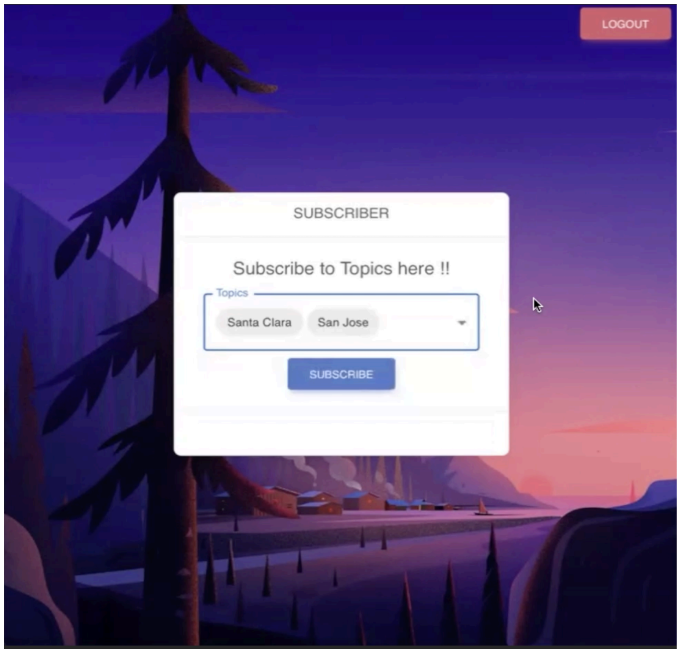
2. Publisher and Subscriber Dashboards

The images illustrate the dashboards for publishers and subscribers within the Real-Time Weather Notifications system.

Publisher Functionality: Enables the publishers to post messages relative to weather if it impacts city topics.

Subscriber Functionality:Allows its users to subscribe to topics of the city of their choice and receive messages related to the said topic.

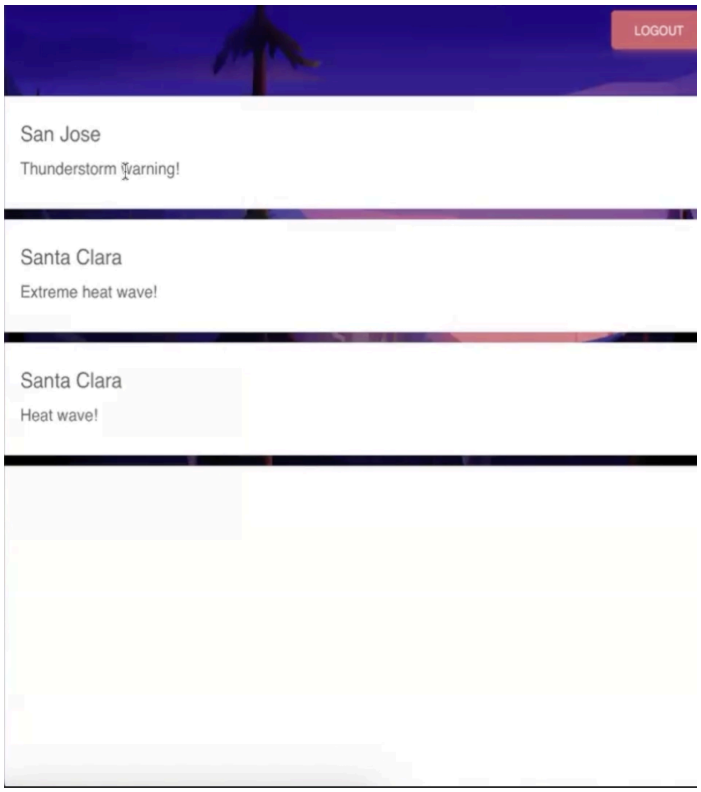




3. Subscriber Message View

Subscriber's view of messages for the topics they have subscribed to.

Functionality: Subscribers can see weather messages for the topics they have subscribed to.



IX. Analysis Of Expected Performance

A. Overall Results

The project has thus addressed the objectives set and has ensured that by the end of its implementation, it develops a Real-Time Weather Event Notification System that ensures that the subscribers are informed immediately weather information on them is available. Some of the novelties incorporated into the architecture of the solution include employing communication; the leader election process as well as introducing dynamic subscription management. The effectiveness of the design strategy could also be highlighted based on the following: “The quality of the system is high in terms of handling interactions with the subscribers, working with the leader changes, and generating events.” This also affirms the efficiency of the system and a confirmation that the functionalities of the system are properly undertaken as it ensures that subscribers occasionally recover proper weather notifications.

B. Measurement Metrics

- *Throughput:* Response rate according to the number of subscribed people served with the weather updates in the given timeframe is throughput. In the evaluation it determines the system competency in the management and distribution of notifications that are relayed. In our simulation analysis, this became apparent in the terms of the number of updates per second we were able to push through.
- *Latency:* Latency is the time between when a weather update is made and the time the subscribers are notified. It determines how much is the system is anticipating. In our simulation analysis, the latency is presented in the form of average time elapsed between updates, in seconds.

C. Performance in Simulation

The simulation was conducted in a basic environment with a single server and one subscriber. The server was supposed to send events to the subscriber and was initially provided with a limited number of events, N. The first event was sent and before all the (N) events were given to the client, the time was recorded. They are fired on the fly and although this may slow the service the concept is more realistic because the system has to send each event as a single entity rather than in a batch. This is in contrast to traditional queueing models that offer ideal throughput and latency estimates. These measurements are only of the time required by the server to process the new message and do not include time for passing the message to the client. They are the averages of the runs, (1,000,000/N). The messages included were all labeled as “test”. This simulation was performed on the MacBook Pro M2 Pro with 16 GB of RAM..

TABLE I. Simulation Results for Server

Events (N)	Results for Server		
	Time (sec)	Throughput (events/sec)	Latency (sec/event)
10	0.0000837	114,000	0.00000837
100	0.000823	128,000	0.00000823
1,000	0.0976	104,000	0.00000976
10,000	0.0964	100,000	0.00000964
100,000	0.843	117,000	0.00000843
1,000,000	7.61	125,000	0.00000761

As seen from the simulation, the following insights are clear. When it comes to the event processing capacity, the server seems to be capable of processing almost 100,000 events per second. Such constant performance proves the server’s ability to handle a large number of events at once effectively. The variability in the handling of events might be attributed to random variation of the host computer. However, as the number of events increases, the time that is taken to process each increases linearly and this is in consonance with our analysis based on algorithmic complexity. In summary, these simulation results show that the proposed service is efficient and can be easily scaled to handle thousands of events.

X. Testing Results

A. Test Cases

- **Register Publisher and Subscriber:**
  - *Test Description:* Ensure that a new publisher and Subscriber can be registered by using API command of the POST request containing the username and password information.
  - *Expected Result:* The response body should contain a message that would state that the publisher was successfully registered.

- **Publish Message:**
  - *Test Description:* Ensure that a message can be published successfully by publishing a message through a post request with a valid managerId, queue and message.
  - *Expected Result:* . The response body should contain a response indicating that the message was published. The database should contain fields for recording the details of the message.

- **Leader Election and management:**
  - *Test Description:* Check whether the system is able to incorporate the leader election when the current leader node is unresponsive.
  - *Expected Result:* It should identify when the current leader is not available, choose a new leader, and provide the new leader's information in the database.

- **Request Rate Limiting :**
  - *Test Description:* Consider the use of the rate limiting when it comes to managing the request rates.
  - *Expected Result:* If the rate at which the request is made is beyond the specified limit, then the system should return 429 status code along with an error message.

- **Node Health Monitoring:**
  - *Test Description:* Evaluate the effectiveness of the system in diagnosing node failures and the corresponding response mechanisms.
  - *Expected Result:* Upon detecting a node failure, the system should update its state to reflect the failed node's removal. If node is leader, then the leader election algorithm should trigger.

B. Test Results and Discussion

- **Register Publisher and Subscriber:**
  - *Result:* Publisher/Subscriber able to register in system properly. The publisher/Subscriber details were found in the MongoDB database.
  - *Discussion:* The test case confirms that the application correctly registers a new publisher/Subscriber and stores their details in the database.

- **Publish Message:**
  - *Result:* All tests were successful, which are the following: The message was sent, the confirmation of the success was received, and the message was stored into the database..
  - *Discussion:* This test verified that the system can publish messages and store them in the database correctly, as evidenced by the appropriate response and database entries.

- **Leader Election and management:**
  - *Result:* All tests run passed and were completed without any issue. The system detected that the leader is not available, selected a new leader, and stored the leaders' information into the database.
  - *Discussion:* This test demonstrated the system's robustness in managing leader node failures by ensuring a new leader is elected promptly to maintain continuous operation.

- **Request Rate Limiting:**
  - *Result:* All the test cases tested positive. If the API received more than the allowed rate limit, the node returned the status code 429 along with an appropriate message.
  - *Discussion:* The test confirmed that the rate limiting effectively prevented excessive requests, safeguarding the system from potential overload and ensuring equitable usage.

- **Node Health Monitoring:**
  - *Result:* All scenarios passed. As expected, the system was capable of detecting node failures and then also capable of removing the failed nodes from the state of the system. In a leader failure case a new leader is also elected to take charge of the organization or the group.
  - *Discussion:* The test proved the reliability of the protocols for monitoring the node's health and the compliance between the system state and the operational nodes.

In this way, performing comprehensive testing of the PubSub application verified that the system is working without any issues and is completely reliable. All of the test cases are passed which proves that the system is able to handle various situations on the nodes. These results prove the effectiveness and reliability of the system to support the continuous and efficient operations of the organization despite the presence of disturbances.

XI. Conclusion

A. Lessons Learned

The development of our Real-Time Weather Event Notification System highlighted several key lessons: the need to have a sound and well-prepared architecture based on the client-server as well as distributed-systems models; the significance of the leader-election mechanisms for further fault tolerance and high availability; the significance of real-time data processing and latency management; the effectiveness of the gossip protocol for health monitoring and failure detection; and the significance of user-centered design for interaction. These findings will be useful in the future to refine and develop further improvements and initiatives in constructing dependable, comprehensible, and efficient systems.

B. Possible Improvements

As for the areas that we have considered for future enhancement in a bid to enhance the effectiveness, capacity and resilience of our Real-Time Weather Event Notification System, there are several. First of all, the latency is going to be significantly decreased with the help of edge computing, which makes the improvements and responses more real-time. This should help to decentralize the servers and make the overall use of the application more efficient as the data processing and responses are almost real-time.

Secondly, practical support for innovative monitoring and analysis will assist in the real-time monitoring of the processes in the project, better logging, and detailed analytics in the form of visual dashboards. These features will help in supervising the functioning of the system and show the use statistics, which will be valuable for further improvements. Last but not the least, there will be the dynamic load balancing with the capability of resource control that plays an important role in increasing the scalability of the system and the performance of the container orchestration. This enhancement will assist in ensuring that the workload distribution to the available resource is spread to avoid congestion of these resources hence improving the general utilization of infrastructure. These improvements will be implemented to ensure that we can vouch for the continued functionality of our system while also ensuring that it will be able to meet the user's needs in the future.

C. Conclusion

This personalized Weather Alert is developed based on the distributed system to enhance the Real-Time Weather Event Notification System. I believe that this project has provided a good practical exposure and insight into how distributed systems principles can be effectively used in practice to design a system that is highly scalable and fault-tolerant. It also shows how distributed architectures can significantly enhance the availability and speed of the real-time notification services. By working on this project, I was able to understand how theoretical concepts may be applied in practice and we have also seen the advantages of using distributed systems in building dependable and powerful services.

References

1] L. Staglianò, E. Longo and A. E. C. Redondi, "D-MQTT: design and implementation of a pub/sub broker for distributed environments,"

[2] D. K. Yadav, C. S. Lamba and S. Shukla, "A new approach of leader election in distributed system,"

[3] Garcia-Molina, "Elections in a Distributed Computing System," in IEEE Transactions on Computers,

[4] H. P. Pratama, A. S. Prihatmanto and A. Sukoco, "Implementation Messaging Broker Middleware for Architecture of Public Transportation Monitoring System,"

[5] Eugster, P.T., Boichat, R., Guerraoui, R. and Sventek, J. (2001), Effective multicast programming in large scale distributed system