Middleware Final Report

Team1 Yuyang Chen, Yanqi Gu, Fuyao Li

Public Safety Alert System --Message Queue Based Middleware Implementation

Overview

Safety alert is a very important issue nowadays. It can send users warning messages when emergency like flooding, fire or gunshot happen. It is very necessary to make this alert system efficient and reasonable. So the overall objective of this project is to implement a Public Safety Alert System based on a Pub/Sub Middleware Environment.

In a Pub/Sub system, there are subscribers, publishers and a service broker. Obviously, the publisher would publish information to the broker and then the broker would send out the message to the subscriber according to some rules. According to the structure of a Pub/Sub system, in the safety alert system, the users are subscribers and the government office may be the publisher who wants to warning people to watch out the danger.

To make our system more efficient and reasonable, we mainly implement a priority based message queue in the broker service component. The priority depends on the geological information of subscribers, because we may want to warn the users close to the position first when emergency happens in a prioritized way. This means the user who has the higher priority may get the notification more quickly than others with lower priority.

The challenge here is about how to get the real-time locations of mobile users. It may involve the problem of privacy and accuracy. We ask the subscriber to submit their local address, it can be a community name or just zip code. If we can't get the real-time location of a user, then we can just use the submitted address to determine the priority.

In the meantime, we observe that different safety events may have different characteristics which need different procedure to take care. When it comes to some natural disasters, like flooding and earthquake, we want to inform all the people living in that area not only the people actually be there at that time. However when something like robbery and sex assault happens, we mainly warn people around that place at that specific time. So the point is that we can classify different events and send different types of events in different manners.

In regard to the implementation of our system, we finished following components:

1. Build a message Queue based on messaging middleware system to issue alert messages when emergency happens.
2. Send message to certain clients based on the their location and the emergency type.
3. Send message with priority. Clients closer to the emergency site will be prioritized and receive the message in shorter time.
4. Realize all the functionalities with three popular messaging middleware, RabbitMQ, ActiveMQ and Kafka. Compare the performance metrics.
5. Integrate all functionalities with Springboot (Spring based backend application framework). Implemented a UI test interface with thymeleaf.

Related Work

To better understand the variants of the paradigm of publish/subscribe scheme, and help pick the proper tool for developing out system, it's necessary for us to first do a survey on different kinds of pub/sub systems, with detailed analysis on performance of communicating entities in time, space and synchronization[1].

The basic system model for publish/ subscribe interaction relies on an event notification service providing storage and management for subscriptions and efficient delivery of events. The decoupling that the event service provides between publishers and subscribers can be decomposed along the three dimensions: space decoupling, time decoupling, synchronization decoupling. There are five common communication paradigms:(1)message passing(2)RPC(3)notifications(4)shared spaces(5)message

queue. Depends on different ways of specifying the events of interest, there are three variations for Pub/Sub systems, Topic-Based, Content-Based and Type-Based Publish/Subscribe systems.
For reference, we also look at two kinds of specific Pub/Sub systems which greatly inspire our work. The first one is [2]. When we design our application, we considered to build a web application. So we try to find some related application built upon a web application. In this paper, it tells us that web applications based on B/S mode are much more convenient than applications based on C/S mode, because they don't need any clients, they need web browsers instead. As a result, applications based on B/S mode are taking place of applications based on C/S mode in a lot of aeas rapidly. So it is very necessary to design a system of messaging by using B/S mode. The system architecture consists of three parts:
(1)The monitoring application for capturing instant messages (such as windows service application).
(2)The web application for displaying instant messages to users.
(3)The message queue for maintaining the communication between the monitoring application and the web application.
The most important design of this system is called Comet which is a new server-push technique which can push instant messages to web browser from web server. The work process of Comet is very similar to a pub/sub system. When a user login the web page, it will send a handshake request, a connect request and a subscribe request. When the subscribe request is received at the web application side, the user would be added to the user set. Once there are any messages in the message queue which are captured by monitoring application, the user in user set would receive the message.
The second one is location-based[3][9][10].In our system, we are going to build a safety alert system to send messages to users based on geographical locations, while the location aware pub/sub system was proposed by researchers. In this kind of system, users are notified instantly when there is a matching event nearly. In Elaps proposed in [3], the authors developed a concept called impact region that allows center to identify whether a safe region is affected by newly arrived events. Safe region are widely used to reduce communication cost for continuous moving query processing. Like this work, our system is initially supposed to be a location-aware pub/sub system, continuous spatial queries and location-aware news feed system. A region is safe if its minimum distance to any matching event is larger than the user’s notification radius. Researchers tried to use Voronoi-based method and grid-based method to define the impacted area. There’re surveys mention the incremental direction-aware GM. However, those are way far beyond our concern for the project and will not be discussed in the survey.
In general, these two special cases give us some insights on how to define boundaries of a impacted region and how to realize the location-aware pub/sub system based the cost model.
When looking for functional frameworks to implement our message queue based system, we also referred some papers talking about popular frameworks. [4] [5][6] compare common functions and different mechanisms of Kafka and RabbitMQ with experiments, they also give suggestions on proper use cases of these frameworks. [7][8] discuss about advantages of ActiveMQ and compare it with RabbitMQ regarding to messaging mechanism.

Structure of the system

There are several parameters we need to define before we introduce the structure of our system.
Client: Client have a current location and a home location.
Emergency Type: We have six emergency type choices (Flooding, Robbery, Earthquake, Sexual Assault, Gun shot, Fire). Different types will use different client locations to send messages.
Message: Messages will be send to client based on how far they are away from the emergency issue. We have three alert levels. The higher level messages will be prioritized during sending.
Broker: The terminal to distribute messages to users.
The workflow of our system is as follows:

First step is data preparation. As Figure 1 shows, the client user sends its information to database, including client id, client current location and client home location. These data are stored in database built on mySQL.

Figure 1. Data Preparation

After that, the system is ready to work. In figure 2 we can see the structure of workflows. Once the center wants to commit safety alert, it decides the messaging platform, and then issues an emergency event with the type of disaster. If the emergency is natural disasters like flooding or earthquake, the messages are sent based on client home location, since the users might want to move from their house, it's reasonable to send messages based on home location. If the emergency type is more likely to be caused by human, like robbery or shot, the messages will be sent based on client current location. After that, prioritized messages will be scheduled and sent to message brokers based on framework architecture, and users can get alerts and detailed analysis results from message queues.

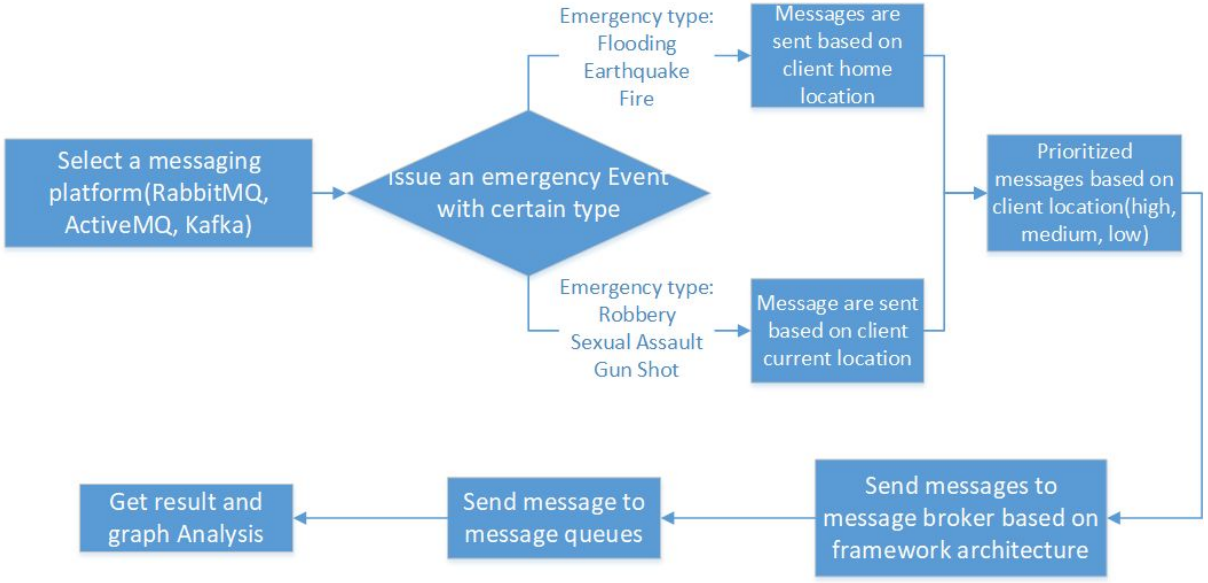

Figure 2. Workflow

The prioritization of messages is based on location distances. We calculate the Euclidean distance between the client location and the emergency issue location. If the distance is less than 10 miles, the client will be set up in high priority group, if the distance is between 10 ~ 20 miles, the client will be in medium group. Otherwise, the client will be in low priority group. Then we add call back functions to get the elapsed time for further analysis. Finally we customize the approach to realize prioritization based on different framework mechanism (RabbitMQ, ActiveMQ, Kafka).

Frameworks

We choose RabbitMQ, ActiveMQ, Kafka as three frameworks to implement the message queue and do the comparison. The implementation details and basic introductions are as follows.

RabbitMQ

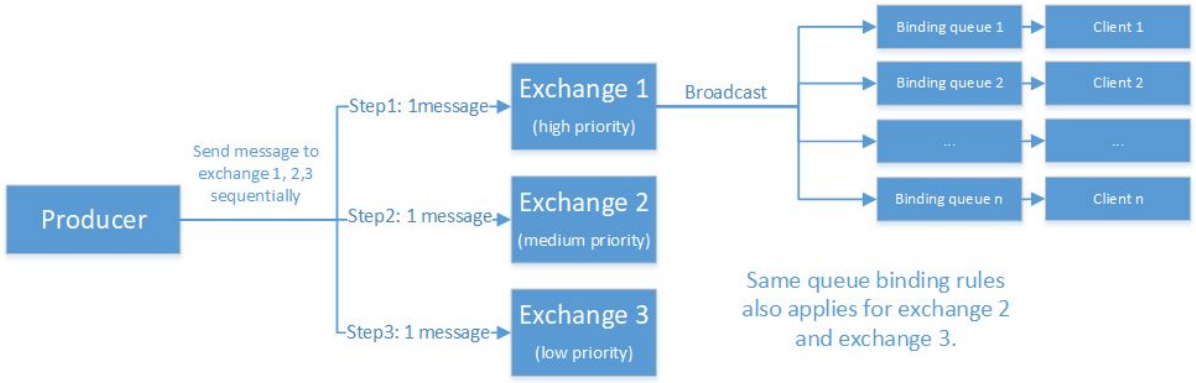

Figure 3. RabbitMQ Message Queue Architecture

1.RabbitMQ is an AMQP based reliable message broker written in Erlang.
2.In our system, the RabbitMQ producer will produce 1 message for one exchange. The exchange will broadcast this message to the binding queues, which corresponds to a certain client belonging to this group.
3.We send messages to exchange in a sequentially order based on priority. The high priority clients will receive message relatively earlier, but with broadcast the strict priority of receiving is not guaranteed. We can see this in the experiment part.
4.Prioritization: Produce messages sequentially and push into exchange based on priority, then broadcast messages to clients with corresponding queues.

ActiveMQ

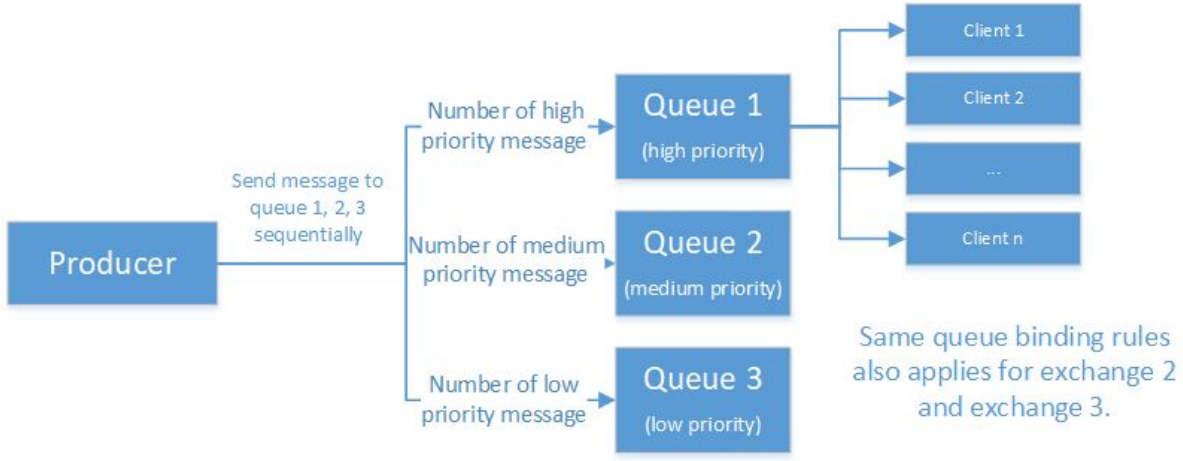

Figure 4. ActiveMQ Message Queue Architecture

1.Support many Cross Language Clients and Protocols, easy to use, fully support JMS and J2EE.
2.There is no exchange in ActiveMQ, we need to produce as many messages as the client's number.
3.We ensure the priority by sending messages to queue 1, 2, 3 sequentially. This strictly guarantee all the messages with higher priority are received before lower priority message.
4.Prioritization: Produce message to corresponding queues sequentially based on priority.

Kafka

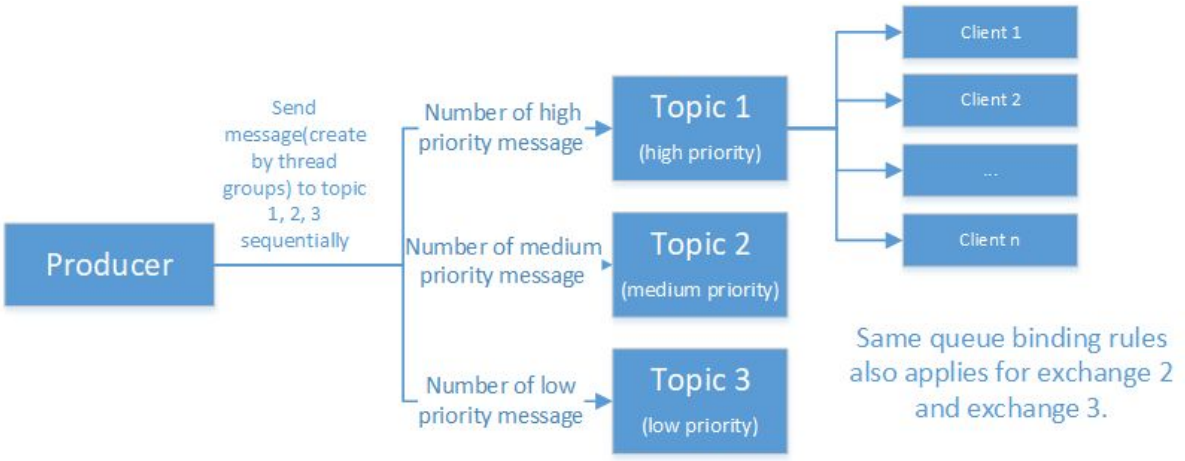

Figure 5. Kafka Message Queue Architecture

1.Kafka is a high-throughput distributed messaging system distributed, partitioned, replicated commit log service.
2.In our implementation, we use multi-threaded programming to start multiple thread groups sequentially based on priority groups. And we use similar mechanism of ActiveMQ to implement listening clients. This generally guarantee the priority sequence.
3.Prioritization: Create thread groups sequentially based on priority to create messages and publish message to corresponding topics.

Experiment

We simulate our system based on Ubuntu18.04.1 LTS with i5-8259U CPU and 8G memory.
The location simulation is shown in Figure 6. The red node represents the emergency location. Users in the inner circle are with high priority, since they are closest to where emergency happens.Between boundary 1 and boundary 2 are users with medium priority. Outside boundary 2 are users with low priority.

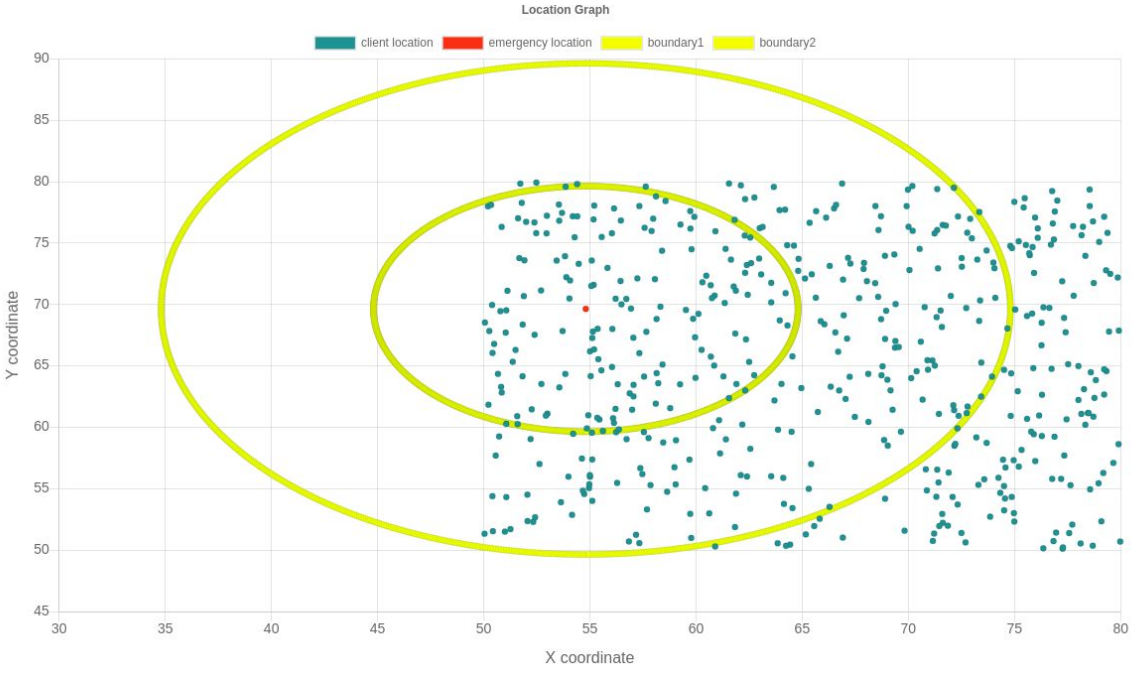

Figure 6. User Location Simulation

Below are the performance analysis results we get. We test each framework with 100,200,300,400,500 clients and get the average performance, we output the average time cost and message sending time for clients with different priorities.
For RabbitMQ:

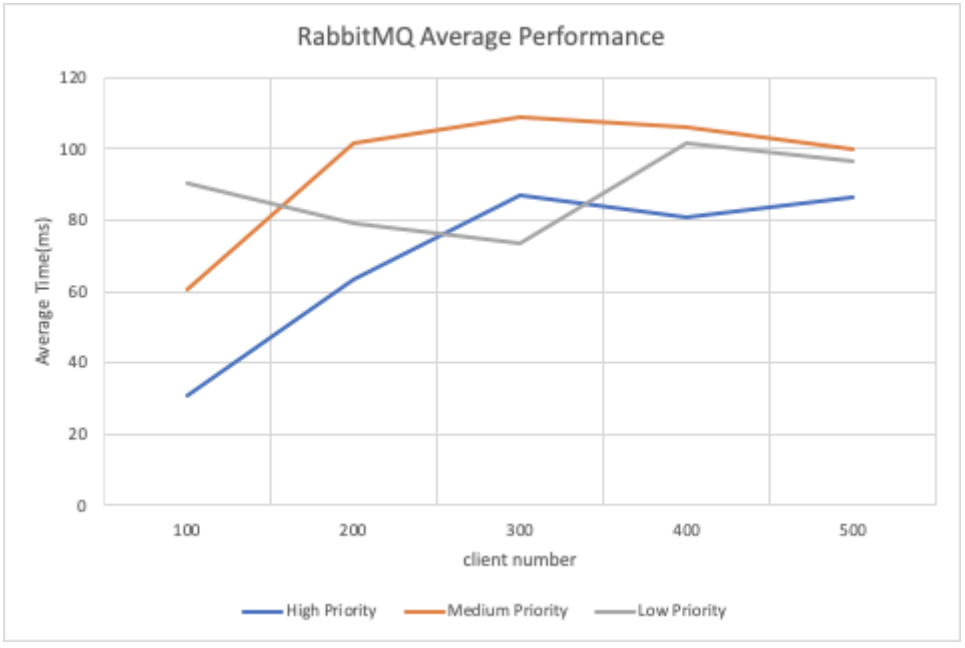

Figure 7. RabbitMQ Average Performance

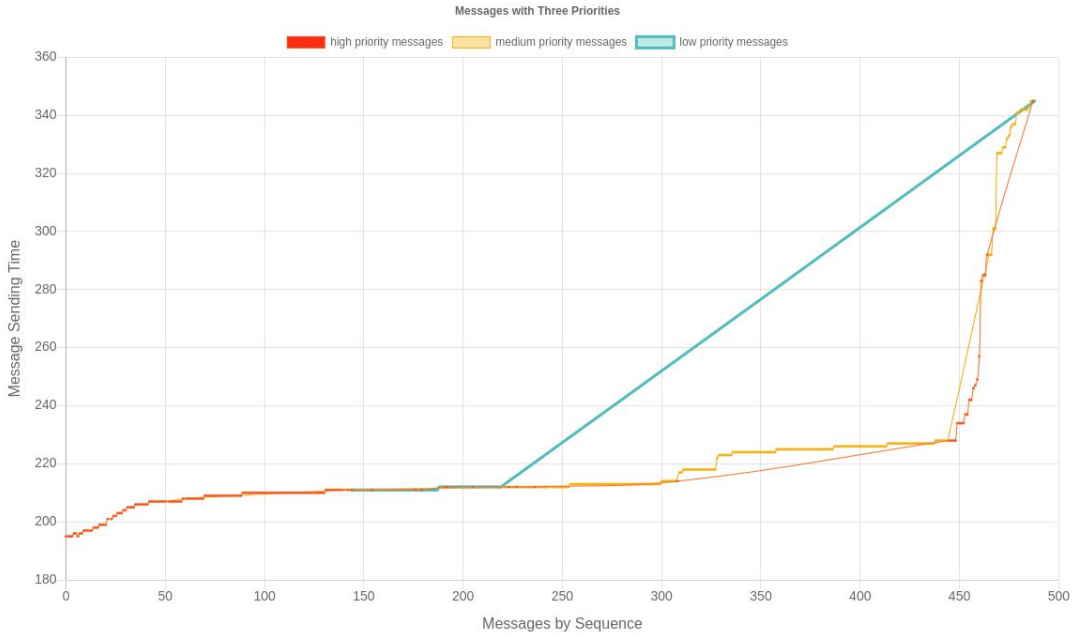

Figure 8. RabbitMQ Message Sending Time with 500 Users

Analysis: From Figure 7 we can see that result of RabbitMQ is unpredictable. Priority works not so well, because we just send three messages in the priority order which can make the three exchanges broadcasting the message nearly in the same time. So when receiving, some clients in high priority may receive the message later than someone in lower priority. In Figure 8 messages with different priorities are not obviously separated, the reason we get this ladder-like result is that in RabbitMQ, messages are produced and sent to exchanges for broadcast, and when the

number of clients is high, the broadcast operation can not send all the message to all the clients at the same time. Users with high priority can receive messages relatively earlier, but the broadcast latency will perturbate this strict priority.

For ActiveMQ:

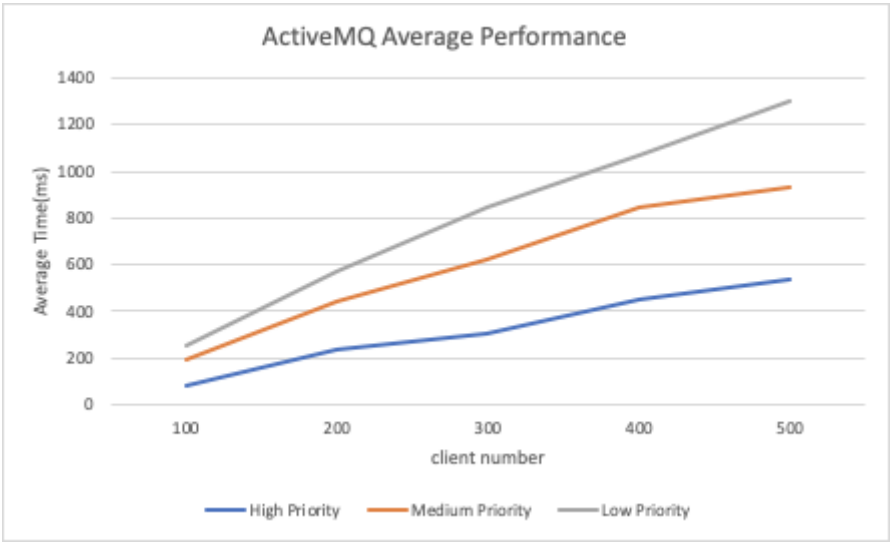

Figure 9. ActiveMQ Average Performance

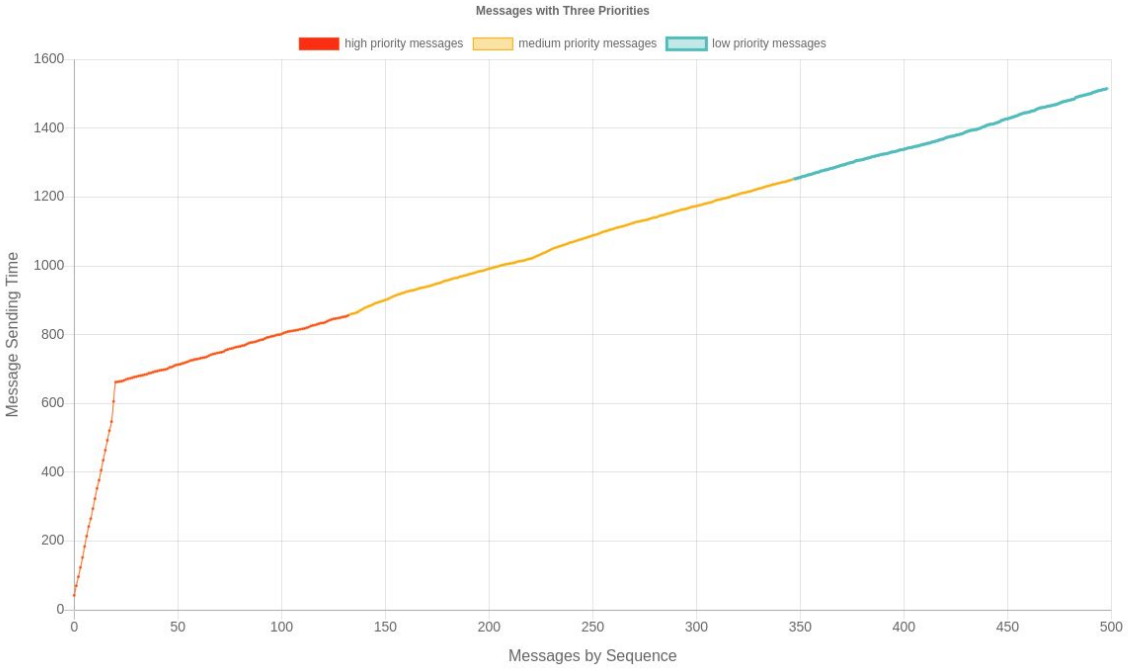

Figure 10. ActiveMQ Message Sending Time with 500 Users

Analysis: We can see that in Figure 9 the performance of ActiveMQ is not very good compared to other two, but it's the most stable one. Also in Figure 10 users with different priorities are obviously separated with message sending time. This is also because of the implemented mechanism of ActiveMQ in our system. Because we send every message sequentially, one group of clients can finish listening before lower priority group starting receiving messages. So the receiving time of higher priority would be shorter.

For Kafka:

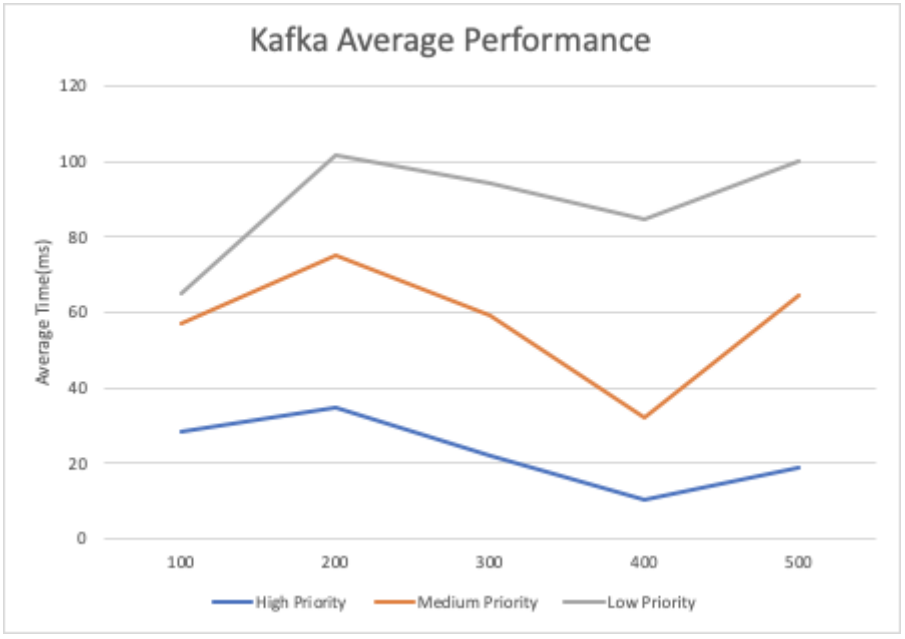

Figure 11. Kafka Average Performance

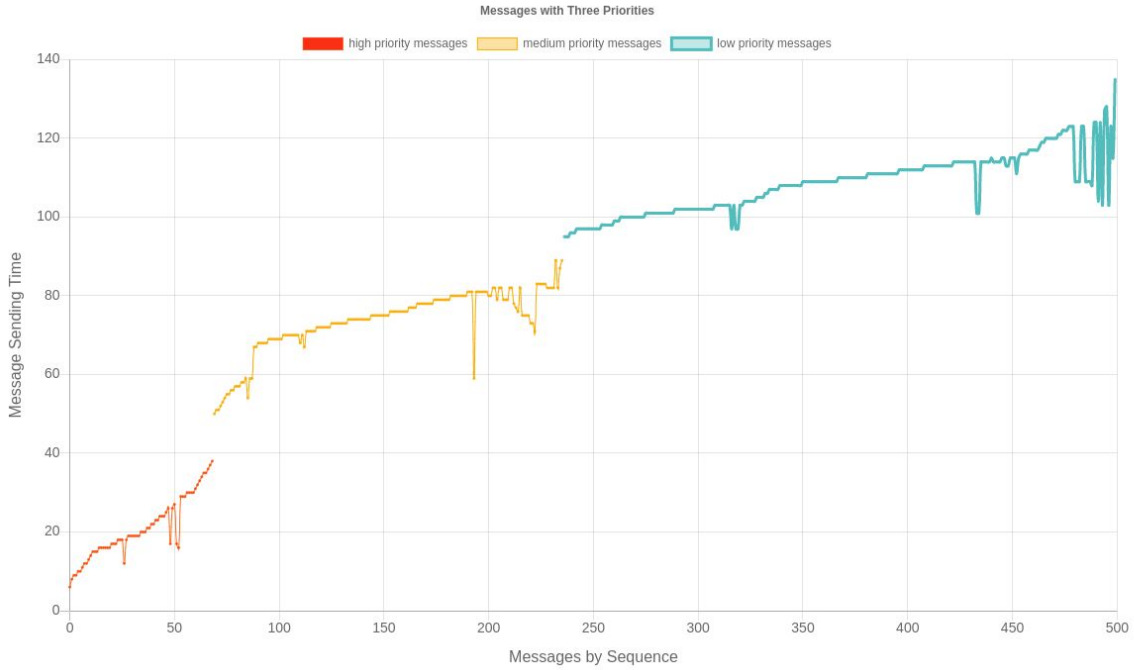

Figure 12. Kafka Message Sending Time with 500 Users

Analysis: In Figure 11 we can see that the performance of Kafka may vary. We think one of the reason may be that the scale of our system is not that large to see the increasing of receiving time. Here is just some normal fluctuation. In Figure 12 the sending time of users with different priorities are separated, but the lines in the figure are “shaking” and unstable. These are all due to the multi-threading scheme of Kafka. Kafka creates clients in group with same priority, and time cost of

creating client groups leads to the gap in Figure 12. Also the inner thread scheduling makes the performance relatively unstable.

Summary and future work

In this project, we implement a public safety alert system based on message queue. We design the structure of our system. To build our system, we choose three different message queue tools: RabbitMQ,ActiveMQ,Kafka. , We also simulate the users for testing. After that we compare performance of three methods of message queues in our simulated environment, and analyze the results, which agrees with what we expect to see. We made a module for prioritization, and we can do more about this in the future, for example implement actual location for users and design a more accurate and smart algorithm to calculate the priority.

Reference:

[1]Patrick Th.Eugster, Pascal A.Felber, Rachid Guerraoui and Anne-Marie Kermarrec, “The Many Faces of Publish/Subscribe”, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.
[2]Gao Ying and Du Zhenxing, "A research of the instant messaging system architecture based on Comet and Message Queue," 2010 2nd International Conference on Education Technology and Computer, Shanghai, 2010, pp. V4-390-V4-394.
[3]Guo, Long, et al. "Location-aware pub/sub system: When continuous moving queries meet dynamic event streams." Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015.
[4]Dobbelaere, Philippe, and Kyumars Sheykh Esmaili. "Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper." Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. ACM, 2017.
[5]John, Vineet, and Xia Liu. "A survey of distributed message broker queues." arXiv preprint arXiv:1704.00411 (2017).
[6]Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.
[7]Bruce Snyder, Dejan Bosanac, and Rob Davies, “Introduction to Apache ActiveMQ”, ActiveMQ in Action, 2010.
[8]V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), Craiova, 2015, pp. 132-137.
[9]M. Hasan, M. A. Cheema, X. Lin, and Y. Zhang, “Efficient construction of safe regions for moving knn queries over dynamic datasets,” in SSTD, 2009, pp. 373–379.
[10]B. Bamba, L. Liu, A. Iyengar, and P. S. Yu, “Safe region techniques for fast spatial alarm evaluation,” Georgia Institute of Technology, 2008.