# Public Safety Alert System

Team 1. Yuyang Chen, Fuyao Li, Yanqi Gu

Message Queue Based Middleware Implementation

# Project Overview

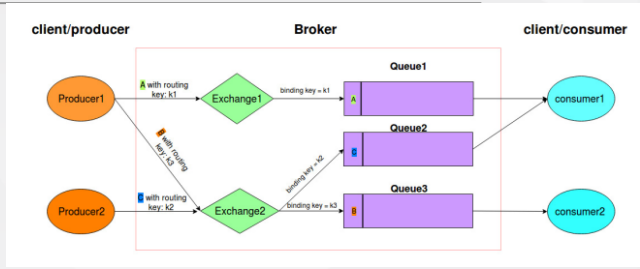
1. **Build a message Queue based on messaging middleware system** to issue alert messages when emergency happens.
2. Send message to certain clients **based on the their location and the emergency type**.
3. **Send message with priority.** Clients closer to the emergency site will be prioritized and 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

- *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.*

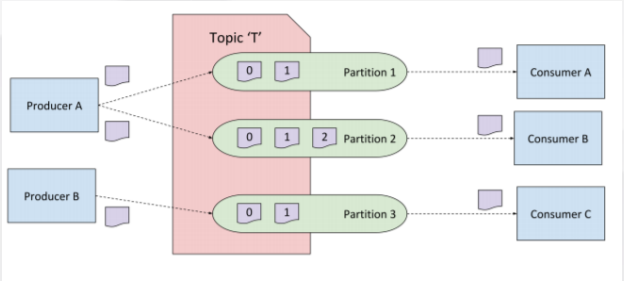
**Related work:** This paper compares the common functionalities and different mechanisms in RabbitMQ and Kafka and give suggestion on **best use cases for these two frameworks**. In architecture side, RabbitMQ, based on AMQP architecture, uses a routing key to a network of exchanges to publish message. Kafka publish message to disk based on appending log that is topic specific. **Delivery Guarantees**(failure rate, can be reduced by replication), **ordering guarantees, availability are also discussed(replication).** This paper helps us to learn the tradeoffs in different MQ architectures.




AMQP architecture

- *John, Vineet, and Xia Liu. "A survey of distributed message broker queues." arXiv preprint arXiv:1704.00411 (2017).*

**Related Work:** This paper compared the Kafka Architecture and AMQP(RabbitMQ) architecture by doing experiment. Finally, they reached the conclusion that, **Kafka Architecture** is better for cases which **have higher requirement in throughput**. **AMQP architecture is better for cases need high reliability**(Encryption enabled, multiple broker work mode: p2p, publisher/consumer, broadcasting). This paper help us to interpret the testing results.




Kafka architecture

- *Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.*

**Related Work:** This paper gives us **a overview of Kafka architecture** and performance metrics compared with RabbitMQ. This paper introduces the topic based **architecture of Kafka, efficient transfer, stateless broker, distributed coordination**. We use these concepts to design our code for Kafka demo.

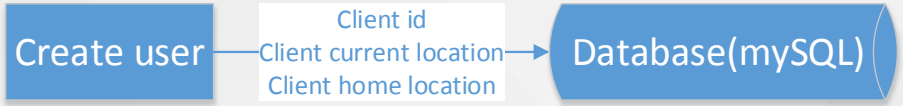
# Workflow

## Keywords

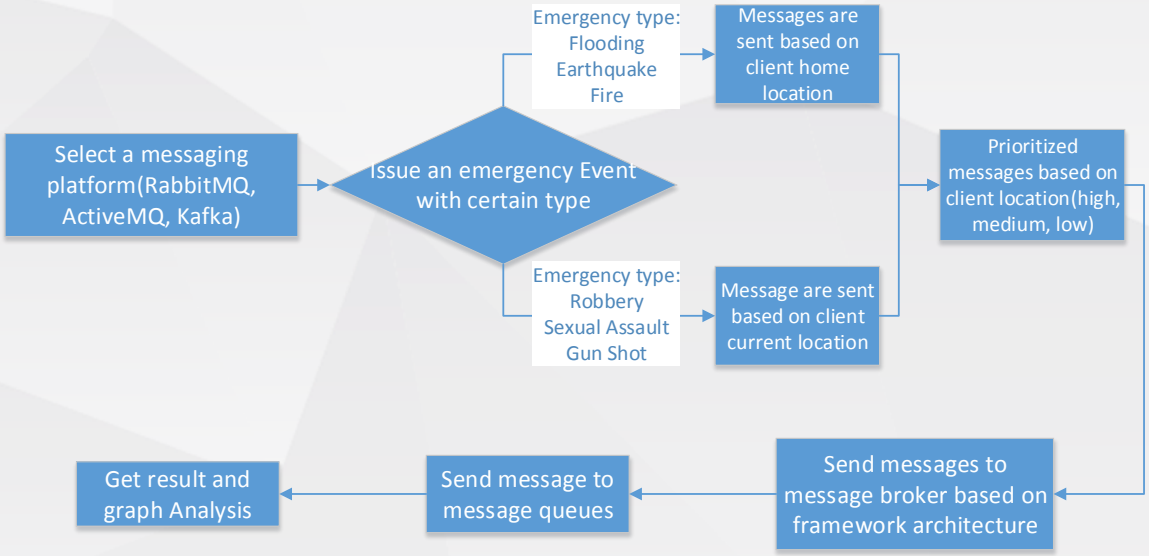
- **Client:** Client have a current location and a home location.
- **Emergency Type:** We have six emergency type choices(Flooding, Robbery, Earthquake, Sexual Assault, Gun shot, Fire). Different types will use different client location to send messages.
- **Message:** Messages will be send to client 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.

## Work flow diagram

- **Data Preparation**

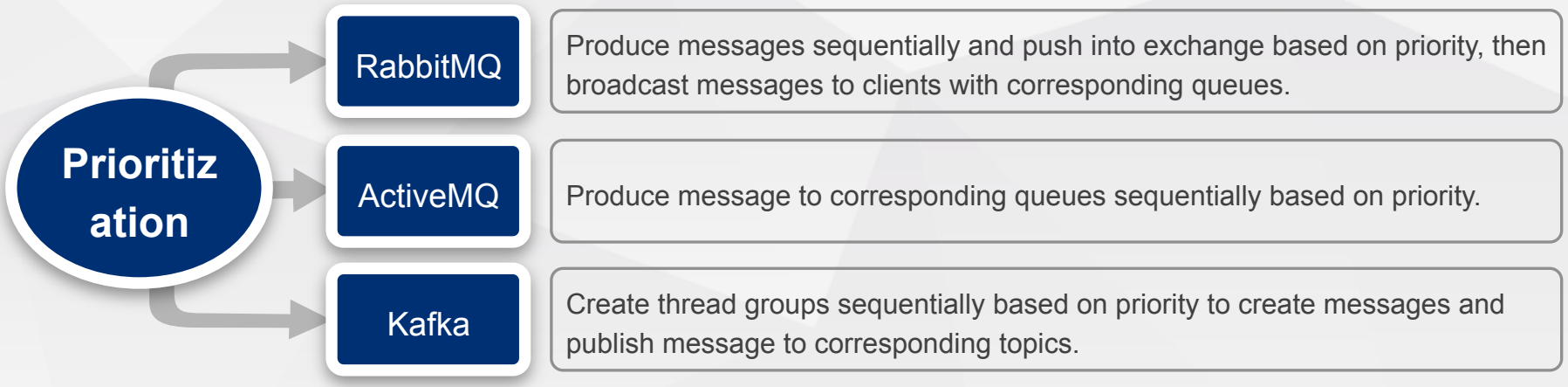



- **Demo Process**



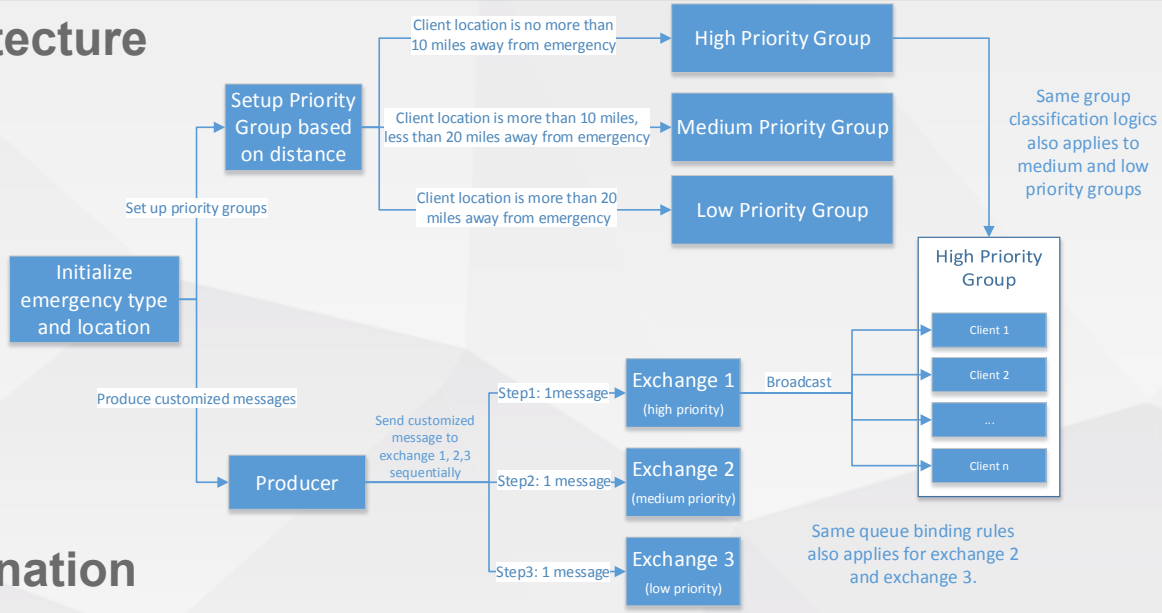
# Prioritization for messages

1. **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 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.
2. **Add call back functions to get the elapse time** for further analysis.
3. **Customize the approach to realize prioritization** based on different framework mechanism (RabbitMQ, ActiveMQ, Kafka).



# RabbitMQ Implementation

- **Architecture**

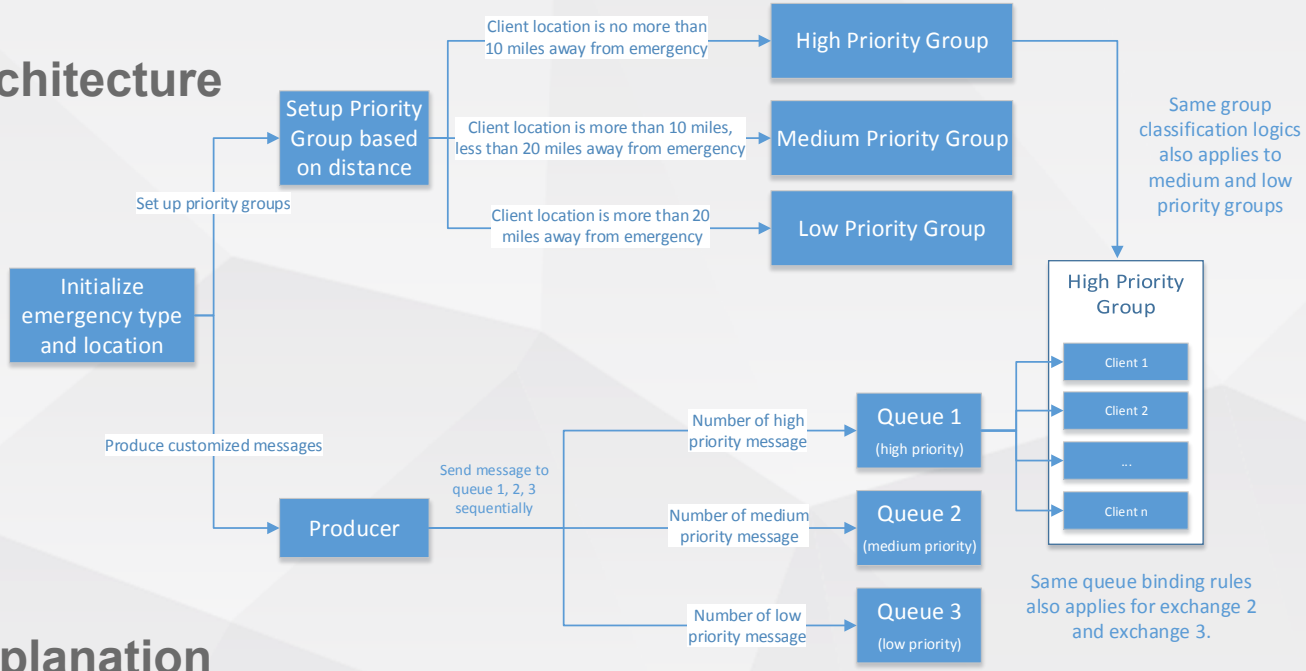



- **Explanation**

1. We implement the exchange as a queue for a group of clients and let our prioritization module to do the matching job.
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 latency, the strict priority is not guaranteed.

# ActiveMQ Implementation

- Architecture

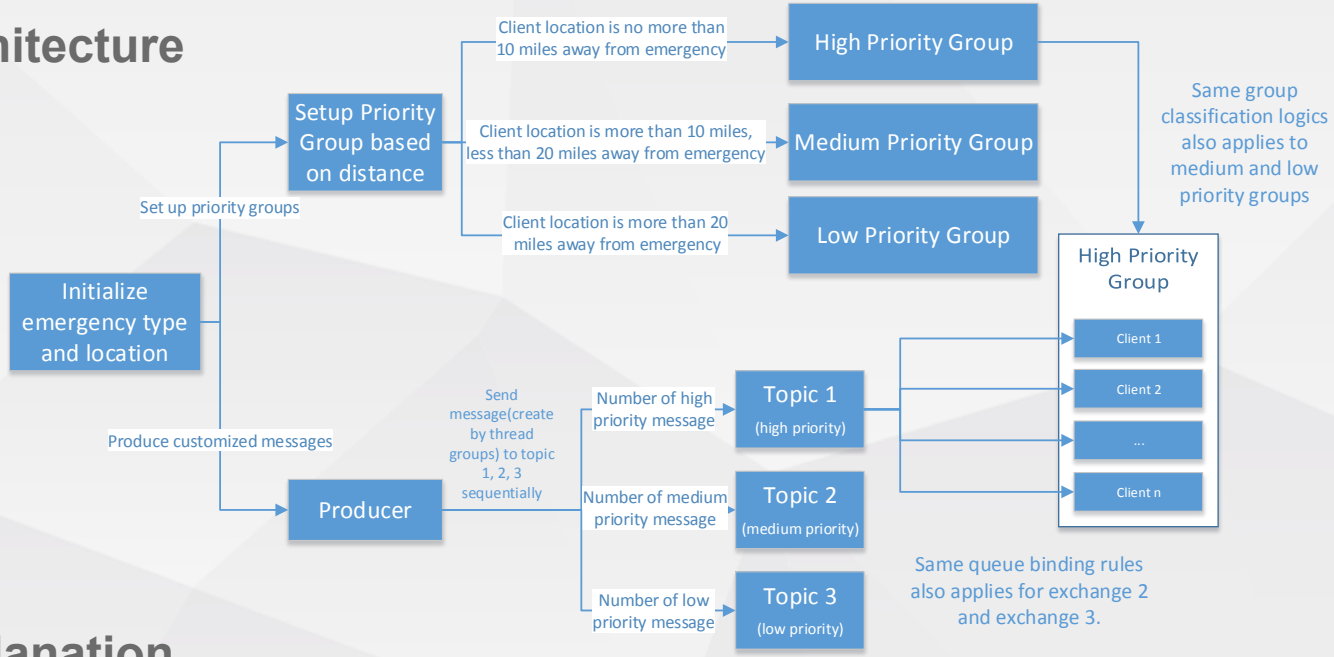



- Explanation

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.

# Kafka Implementation

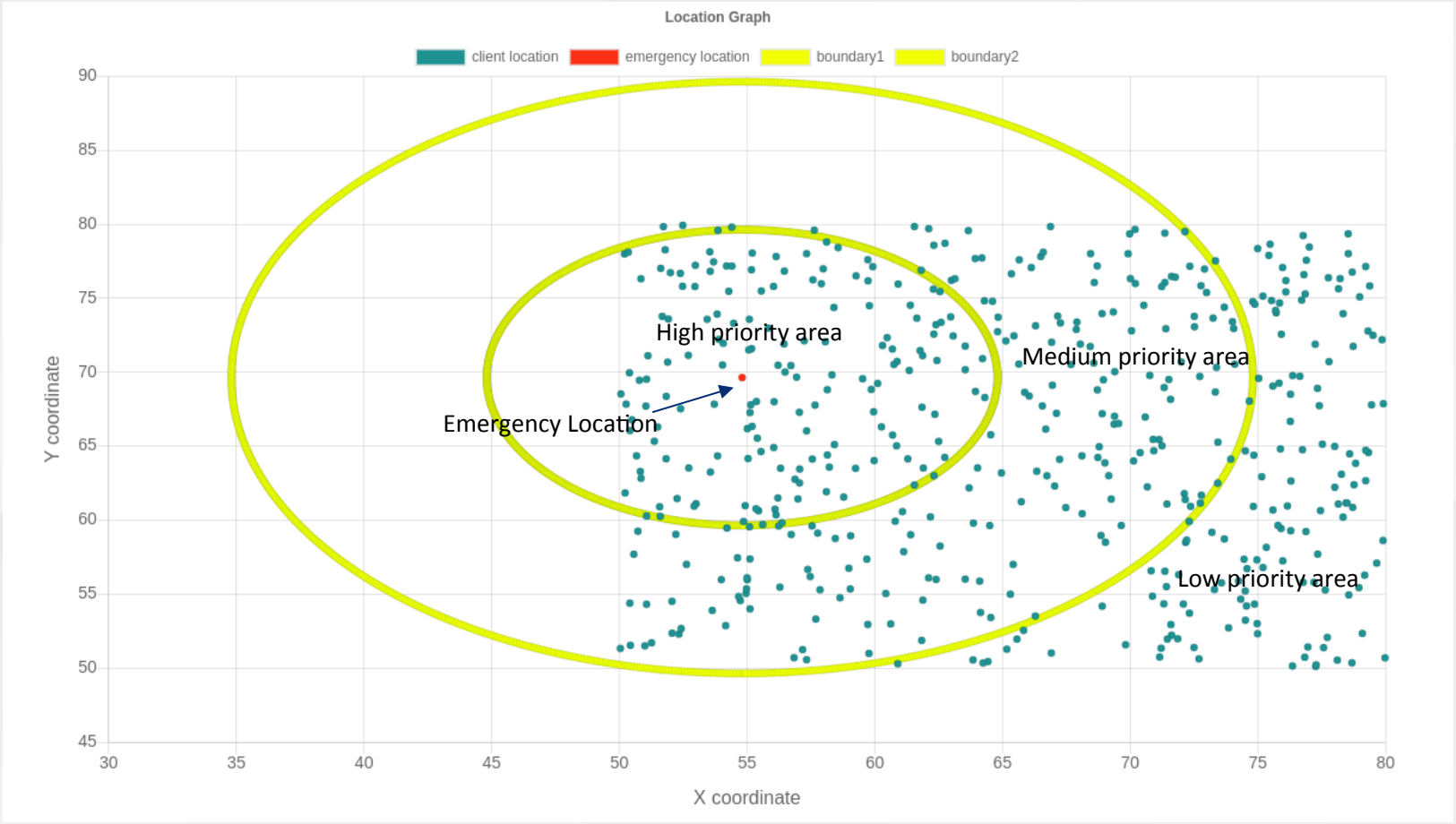
- Architecture



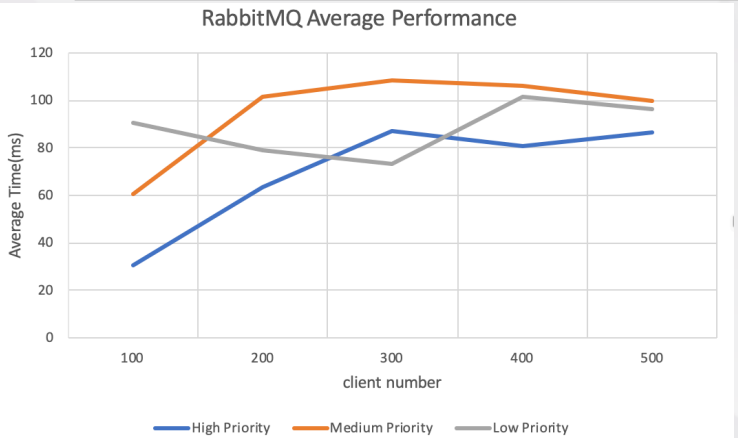

- Explanation

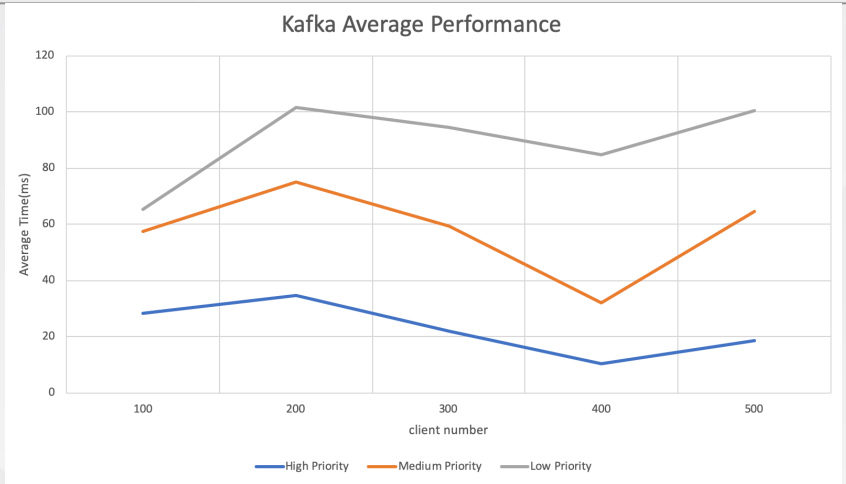
1. Kafka is a high-throughput distributed messaging system distributed, partitioned, replicated commit log service.
2. In our implementation, we use multi-threaded programming to start multiple thread groups sequentially based on priority groups. This generally guarantee the priority sequence.

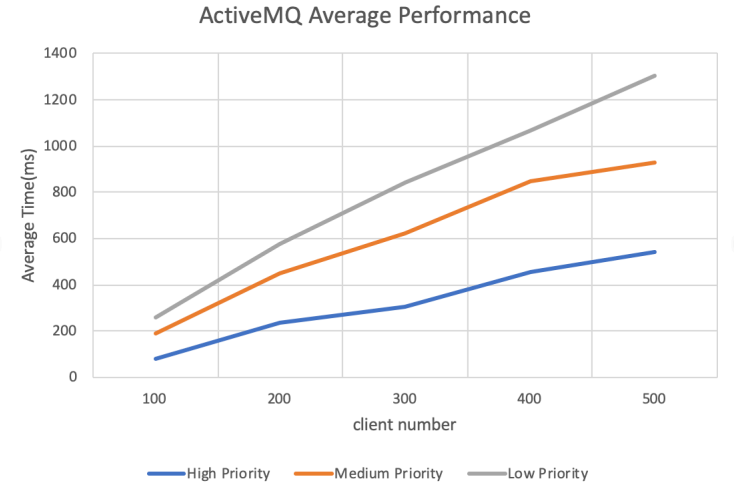
# Simulation Visualization



# Performance Analysis(Average Result)



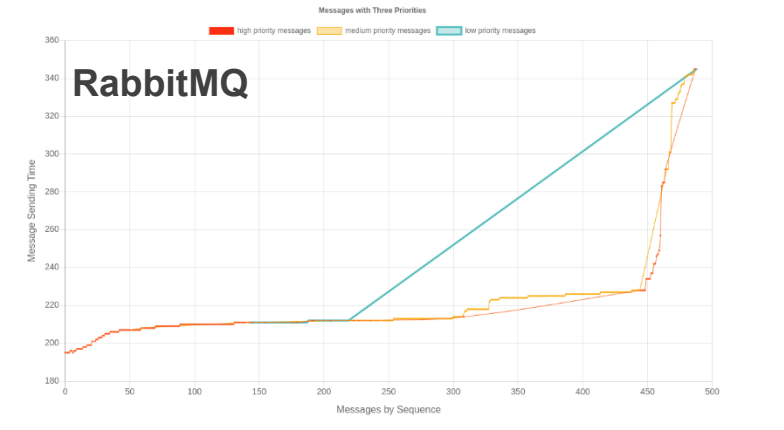



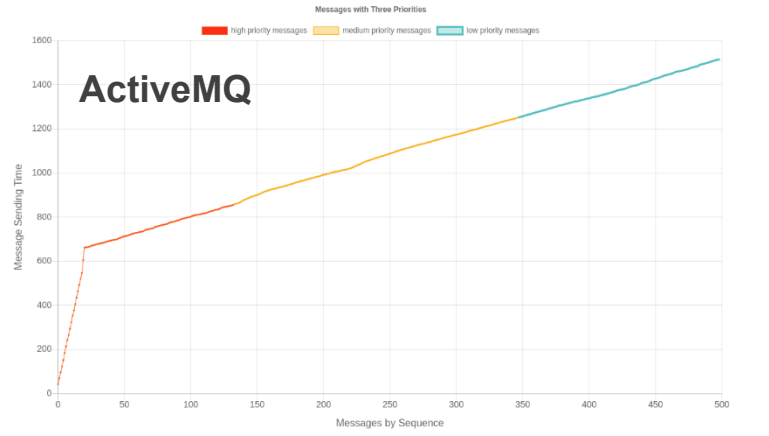

Result of RabbitMQ is unpredictable. Priority works not so well

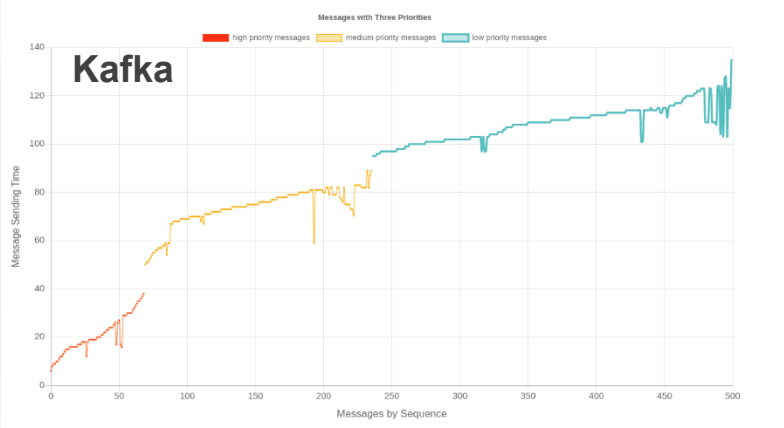
ActiveMQ may take more time, but is more stable.

No significant difference in Kafka.(Multi-Threaded) Kafka can handle very high throughput.

# RabbitMQ/ActiveMQ/Kafka Sample Experiment



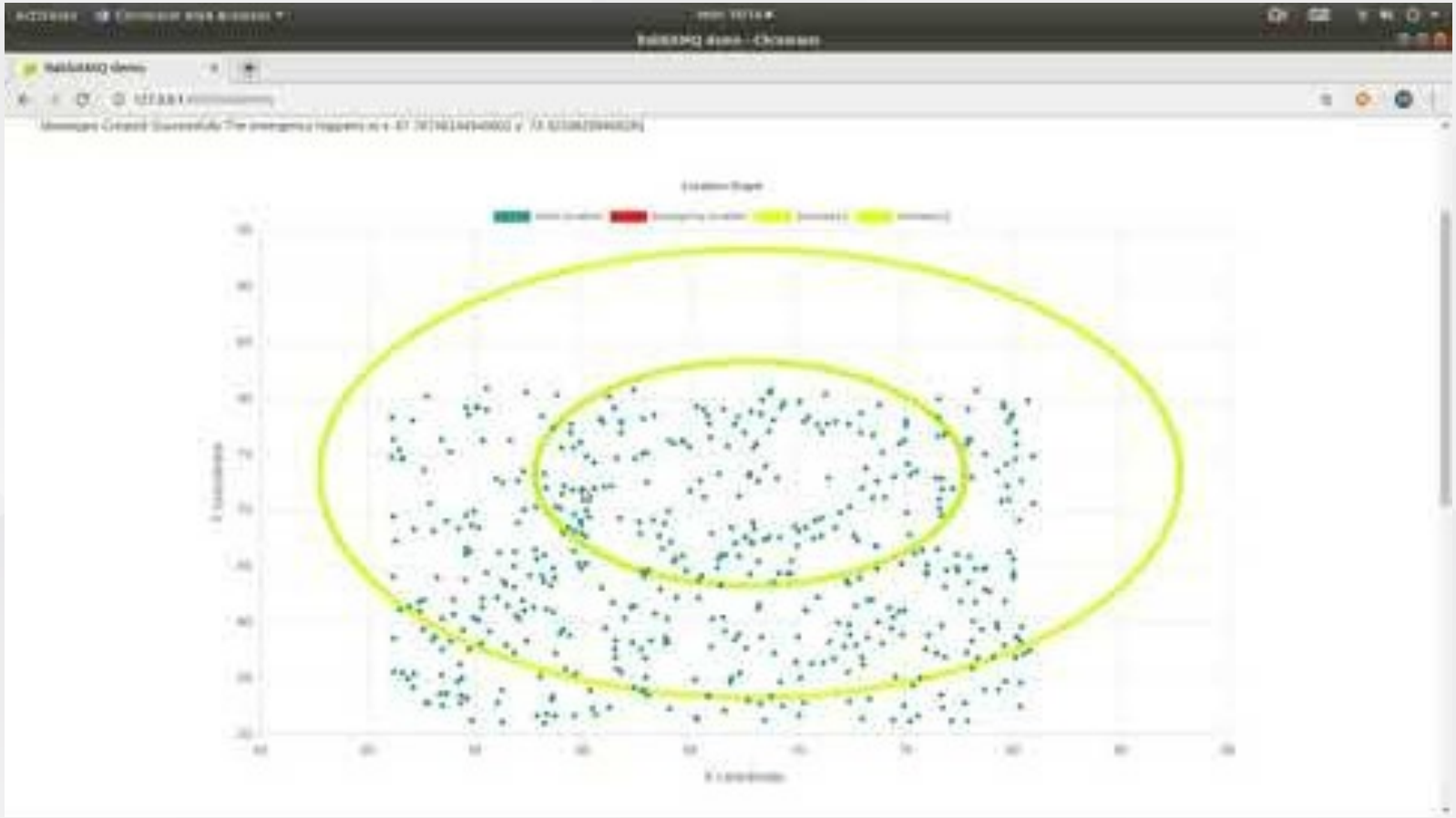




**Red Line:** **High priority message**
**Yellow Line:** **Medium priority message**
**Blue Line:** **Low priority message**

**X coordinate: the sequential order in all the received message**
**Y coordinate: the message receiving time.**

# Demo Video



# Thanks

## Q & A session

Thanks for listening!