# c_uparog : ActiveMQ vs RabbitMQ vs Kafka

Arshdeep Singh Syal, Jubeen Shah, Rayan Dasoriya, Sujal
*Computer Science Department*
*North Carolina State University*
Raleigh, NC, USA
{asyal, jnshah2, rdasori, ssujal}@ncsu.edu

***Abstract*— Message brokers is an arbitrator which enables communication between applications which use different messaging protocols. Publish/Subscribe pattern is a distributed interaction paradigm which allows the message brokers to have a scalable and reliable system. ActiveMQ, RabbitMQ, and Kafka are the three most popular open-source message broker software that are used by thousands of companies for message transfer. Given the popularity of the three, a subtle yet important question arises: Which broker is better, and which one should I use? In this report, we summarize both, the qualitative, and quantitative aspects of the message brokers in terms of performance and bug finding using tools like JMeter, Gatling, FindBugs, and SonarLint. In addition, we also highlight community statistics collected from different sources such as GitHub, Reddit, Stack Overflow, Google Trends, and StackShare.**

***Keywords—Apache ActiveMQ, RabbitMQ, Apache Kafka, Message Broker, bug finding, stress testing, performance testing.***

## I. Overview

The world today generates data at an unprecedented rate. The streaming giants like YouTube, Netflix, iTunes, Amazon Video, and Hulu together account for more than 56.48% of the global internet traffic [1]. With emerging technologies like the Internet of Things becoming more and more popular with the everyday consumer, internet traffic is expected to be burdened with these publish-subscribe systems. For these two applications alone, an efficient, performant and scalable publish-subscribe system is necessary for daily transactions with data. For these reasons, message brokers like ActiveMQ, RabbitMQ or Kafka are very popular not only within the developer community but also from an industrial perspective.

This paper is a comparative analysis of the message queuing broker tools like Apache ActiveMQ [2], Apache Kafka [3], and RabbitMQ [4] on the basis of performance testing tools like JMeter [5], and Gatling [6], and error/bug finding like SonarLint [7], FindBugs [8] along with some interesting community stats. The testing results using these tools are available on our GitHub page[1].

The rest of the paper is divided into the following sections. Section II talks about the various approaches we have taken for performance testing, finding the errors and bugs, and the challenges faced while using Gatling. In Section III, we outline the results of JMeter performance testing, FindBugs and SonarLint, followed by Section IV in which we discuss the community aspects of the project analyzing the GitHub repositories of each of the projects, Reddit Community, Stack Overflow and StackShare statistics. Additionally, we also discuss the industrial usage of the brokers and try to reason out the trend that is developing for each of the message brokers. Finally, we give a conclusion referring to which particular use-case would benefit the most out of a specific type of broker.

## II. Different Approaches

### A. *JMeter for Performance Testing*

After installing JMeter, for each of the message brokers, there are different plugins that we needed to install to start testing the brokers. ActiveMQ did not need any special configuration, it worked out of the box with JMeter, because it makes use of JMS publisher/Subscriber node which is built into JMeter. RabbitMQ needed an AMQP [9] plugin for consuming RabbitMQ messages. Kafka has two plugins available for testing with JMeter: Pepper-box [10] and Kafkameter [11]. We used pepper-box for our project because it has a more convenient interface to work with. The implementation for each of the message brokers is described on the GitHub page.

### B. *Challenges faced with Gatling*

While working with Gatling, we faced several challenges during implementation. A detailed report is given on our GitHub page. However, the main issue that hindered our testing was the incompatibility between the extensions of Gatling for Kafka[12] and RabbitMQ[13] with their respective underlying setup (including Scala, SBT, Gatling and server releases). According to our understanding, Gatling v2.2 was unable to find a class named *"io/gatling/commons/util/ClockSingleton"* which would have been used to benchmark the clock timings to measure various parameters of the protocol.

Since we were unable to execute Kafka v0.10 earlier, we tried to implement it with latest Kafka version (2.12) and couldn't get through this issue. We found a workaround for this issue in which we needed to implement a Rest API which would be exposed to the broker's server, thus enabling message sending and receiving, but due to time constraints, we couldn't implement the solution.

### C. *SonarLint and FindBugs for Error and Bug Report*

SonarLint is an IDE extension that helps to detect and fix the issues in a code before committing the changes. SonarLint provides us with the flexibility to analyze the code at various levels within the file structure. After analyzing the code, a report will be generated. We can check the report by selecting the reports tab in the IDE.

FindBugs is an open-source static code analyzer for detecting the possible bugs in a Java Program. It categorizes the

[1] https://github.com/rayandasoriya/Message_Broker_Analysis

bugs as scariest, scary, troubling and concern. We installed the FindBugs plugin for IntelliJ IDEA. Then, we analyzed the project code to inspect the results. We can also use the command window to categorize the defects into categories of bug, class, package, and bug rank. RabbitMQ was not supported by either of them. Hence, we were unable to run the static code analyzer on it.

## III. Results

### A. Performance Testing

We ran the tests on MacBook Pro, running macOS Mojave Version 10.14.2 Beta with a 2.7 GHz Intel Core i7 processor with 16 GB 2133 MHz LPDDR4 memory. The results that we give in our paper would differ based on the machine the test is being run on; however, the results should be similar. The test was repeated for each broker for 10K, 100K, and 1M producers for a message size of 10 Bytes. The average rate of messages per second for 1 Million achieved for ActiveMQ (5.15.2) was 39K messages/sec (2.968 Mbps), RabbitMQ (3.7.9) it was much higher at 160K messages/sec (12.206 Mbps) whereas it as staggeringly high for Kafka (2.1.0) at 790K messages/sec (60.24 Mbps). The results for ActiveMQ, RabbitMQ and Kafka are shown in Fig. 1, Fig. 2, and Fig. 3 respectively.

The results for mean latency that was observed for each of the brokers are shown in Fig. 4. These results help us understand that the ActiveMQ has the least throughput out of the three and the highest latency. The reason, we think, for this is described in Section IV when we discuss community statistics.

### B. Error and Bug Report

*1) SonarLint:* We were able to analyze the code and find the bugs in Kafka and ActiveMQ using SonarLint. It categorized the bugs as critical, minor, major, info, and blocker as seen in Fig. 5. Kafka has a smaller number of bugs right now maybe because ActiveMQ is an old message broker software.

*2) FindBugs:* With FindBugs, we were able to find the bugs in ActiveMQ, but we were unable to find any bug in Kafka. We were able to find 175 bugs in ActiveMQ Broker and 659 bugs in ActiveMQ core. However, no errors or missing classes was reported in either of them.

## IV. Community Statistics

We studied different statistics about the message broker software and identified some interesting facts about the usage of these software in the industry.

### A. Industrial Usage

RabbitMQ is the most popular in the industry, despite Kafka having better performance. With over 788 companies using RabbitMQ, as compared to 370 using Kafka, and only 29 for ActiveMQ. The reason we think is, Kafka was late to the market and lacks Enterprise Support [14], and by then RabbitMQ had already taken over the market share from ActiveMQ, because of its high complexity and low switching cost.

### B. Popularity

Using Google Trends, we observed that in the past year, Apache Kafka has been the most popular message queueing service. This fact is also supported by the number of stars on GitHub which can be due to a high preference among developers. Also, we observed that during the span of three months, we studied the project, the industrial usage of Kafka rose by 4.225% in comparison to RabbitMQ and ActiveMQ which rose by 3.007% and 0% respectively.

### C. Community Insights

Fig. 6 shows the activeness of the ActiveMQ community has drastically decreased from 2013 when RabbitMQ and Kafka came into the market. RabbitMQ was very popular in 2010 but later on, as the popularity of Kafka increased, the commits per year increased. It is interesting to know that the number of commits per year for Kafka increased at an average rate of *61%* from 2011 to 2017. For the same period, RabbitMQ had a growth rate *-12%*, with ActiveMQ having a growth rate of *-6%*. This trend can be associated with how developers are favoring Kafka more over either RabbitMQ and ActiveMQ.

Talking about other aspects of the community, the retention of contributors very low for RabbitMQ and Kafka where only three and four top 10 contributors respectively still play an important role in the community. In contrast, Kafka has eight of the top 10 contributors still working on the project. It could mean that the community is really helpful in case of Kafka and that developers are willing to work more on Kafka than other message brokers. Also, the number of contributors for each of the brokers is a clear indicator of how well received the Kafka community is.

Kafka community is making sure to include as many developers as possible to grow the community by accepting the pull requests. Relatively speaking, Kafka has much better support on Stack Overflow and Reddit than RabbitMQ and ActiveMQ. This can be inferred from Fig. 7, leading to a much more welcoming environment for new developers.

## Conclusion

RabbitMQ is currently the most favored amongst the industry, but there is a shift in affinity towards Kafka, both from the perspective of developers and industrial adoption. The rate at which Kafka is growing is much higher than RabbitMQ, which in contrast seems to be slowly declining its growth rate. ActiveMQ is the least favored given the low industry adoption and developers retention rate. So, if a new developer wishes to contribute to a community, we would recommend contributing to the Kafka community, because of its high rate of activity, retention, support and overall clarity in the documentation. If however, a developer wants to start learning about message brokers, Message Oriented Middleware, and its implementation we personally found ActiveMQ to be a good starting point and then transitioning towards Kafka. RabbitMQ would require a higher learning curve if the developer is unfamiliar with Erlang.

Talking about the performance, Apache Kafka gave the best performance with a very high throughput and a low latency rate. ActiveMQ is preferred over Kafka when traditional enterprise messaging is taken into consideration. However, RabbitMQ does a better job at throughput, latency and overall community support than ActiveMQ. Kafka, because of its low latency, and very high throughput, fault-tolerance, and its highly distributed architecture is most useful in stream processing, event sourcing, commit log and log aggregation, and traditional messaging. RabbitMQ would be more useful in pub-sub messaging, request-response messaging, and also act as an underlay layers for IoT applications. Hence, depending on the specific use-case, you can choose either RabbitMQ or Kafka.

## References

[1] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. Youtube traffic characterization: a view from the edge. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07). ACM, New York, NY, USA, 15-28. DOI: https://doi.org/10.1145/1298306.1298310

[2] "Apache ActiveMQ " [Online]. Available: https://activemq.apache.org [Accessed: 26-Nov-2018]

[3] "Apache Kafka." [Online]. Available: https://kafka.apache.org/ [Accessed: 26-Nov-2018]

[4] "Rabbit MQ" [Online]. Available: http://www.rabbitmq.com [Accessed: 26-Nov-2018]

[5] "JMeter" [Online]. Available: https://jmeter.apache.org [Accessed: 26-Nov-2018]

[6] "Gatling" [Online]. Available: https://gatling.io [Accessed: 26-Nov-2018]

[7] "SonarLint" [Online]. Availale: https://www.sonarlint.org [Accessed: 26-Nov-2018]

[8] "FindBugs" [Online]. Available: http://findbugs.sourceforge.net [Accessed: 26-Nov-2018]

[9] "AMQP plugin for RabbitMQ to use with JMeter " [Online]. Available: https://github.com/jlavallee/JMeter-Rabbit-AMQP [Accessed: 26-Nov-2018]

[10] "Pepper-box Plugin for kafka to use with Jmeter" [Online]. Available: https://github.com/GSLabDev/pepper-box [Accessed: 26-Nov-2018]

[11] "Kafkameter plugin for kafka to use with Jmeter" [Online]. Available: https://github.com/BrightTag/kafkameter [Accessed: 26-Nov-2018]

[12] "Kafka Plugin for Gatling" [Online]. Available: https://github.com/mnogu/gatling-kafka [Accessed: 26-Nov-2018]

[13] "RabbitMQ Plugin for Gatling" [Online]. Available: https://github.com/fhalim/gatling-rabbitmq [Accessed: 26-Nov-2018]

[14] N. Nannoni, "Message-oriented Middleware for Scalable Data Analytics Architectures." KTH, Skolan for informations-och kommunikationsteknik (ICT), 01-Jan-2015 [Online]. Available: https://www.openaire.eu/search/publication?articleId=od______260::426cdfd2d497eeac93862aef4960f8

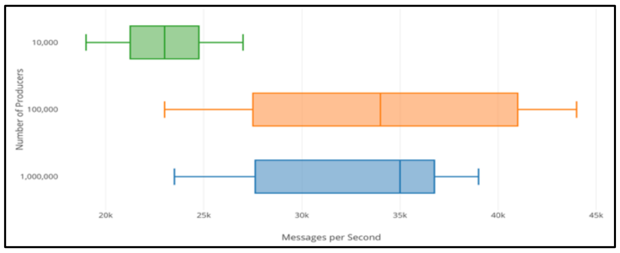

Fig. 1. JMeter results for Apache ActiveMQ

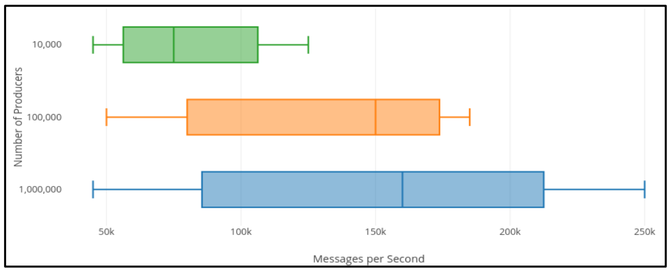

Fig. 2. JMeter results for RabbitMQ

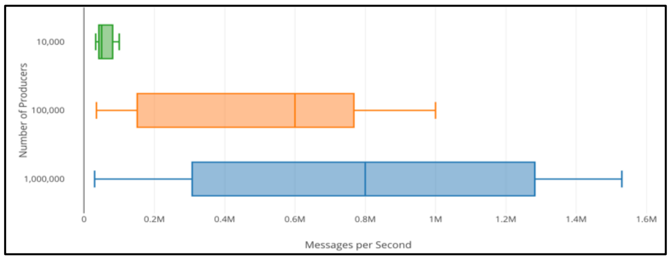

Fig. 3. JMeter results for Apache Kafka

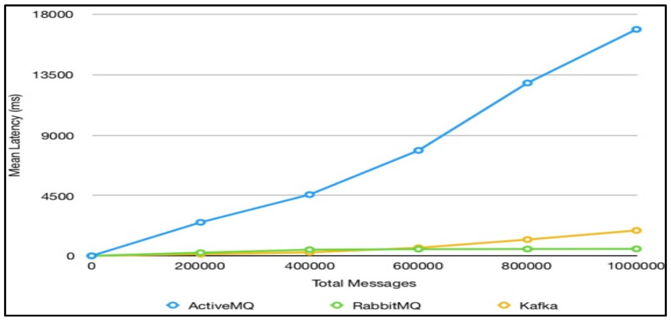

Fig. 4. Average Latency (ms) for message brokers

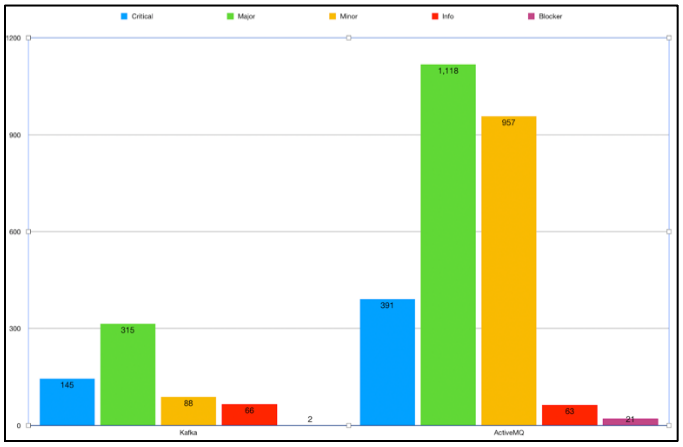

Fig. 5. Bug count analysis using SonarLint

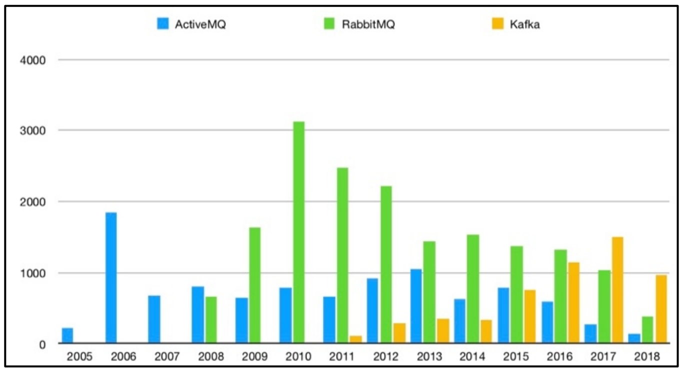

Fig. 6. Commits per year

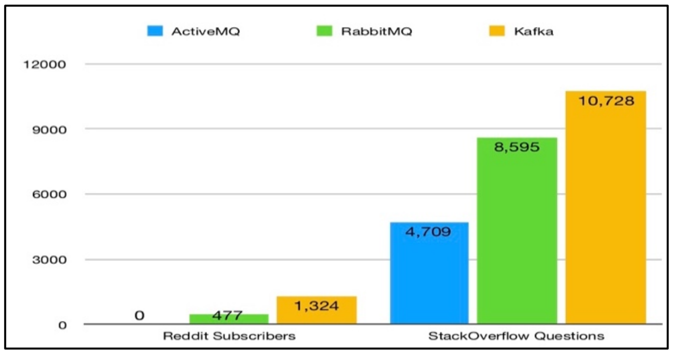

Fig. 7. Reddit and Stack Overflow