ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

Performance analysis of RabbitMQ as a Message Bus

Sneha Shailesh[1], Kiran Joshi[2], Kaustubh Purandare[3]

P.G. Student, Department of Computer Engineering,V.J.T.I, Mumbai, Maharashtra, India[1]

Assistant Professor, Department of Computer Engineering,V.J.T.I, Mumbai, Maharashtra, India[2]

Senior SW Manager,Nvidia Corporation, Pune, Maharashtra, India[3]

ABSTRACT: Message bus plays a key role in transporting messages among various applications that are dependent on each other. It provides a set of mutually agreed-upon message schemas which are independent of the application altogether. Message Bus can be used in a variety of enterprise applications such as Smart Garage and Smart Cities. This paper provides the performance analysis of one of the most popular Message Brokers, which is RabbitMQ, when it is used as a Message Bus. Evaluation is done using the RabbitMQ management interface. A continuous stream of messages is passed to a single consumer, or receiver, from multiple producers, or senders to evaluate the performance.

KEYWORDS: Performance Evaluation, Message Bus, Message Broker, AMQP, RabbitMQ

I. INTRODUCTION

Message Bus and its need

In enterprise solutions consisting of various applications provided by various vendors, there arises a need for these applications to communicate with each other via messages. These applications might increase or decrease in number at any given instance, thus the enterprise solution must be able to accommodate this number easily. This can be troublesome if the applications are highly coupled. Also, each application may have its own platform and interface. Considering this, a requirement for a logical component to connect these heterogeneous applications comes up. In such a situation, a Message Bus can be used. A Message Bus must allow all applications running on different platforms to easily communicate with each other without causing a lot of dependencies. It should be able to handle a large number of messages being generated and consumed. Also, routing of messages play a key role here [8].

RabbitMQ – A Popular Message Broker

RabbitMQ is a popular implementation of the AMQP, or Advanced Message Queuing Protocol. Hence, it implements a broker architecture, which means that messages are stored in a queue on a central node or RabbitMQ server before being sent to clients. This means that RabbitMQ is easy to use and deploy, as features such as routing, load balancing or persistence in message queuing can be implemented in fewer lines of code. However, it is also less scalable and “slower” as the central node adds latency and message envelopes are quite big [6].

FEATURES OF RABBITMQ:

i. **Asynchronous messaging -** RabbitMQ can support multiple messaging protocols such as AMQP, MQTT or Message Queue Telemetry Transport, etc. It also supports message queuing, delivery acknowledgement, flexible routing to queues, multiple exchange type [6].

ii. **Developer experience -** RabbitMQ can be deployed with many virtualization and life cycle management softwares such as BOSH, Chef, Docker and Puppet. It can be used to develop cross-language messaging with popular languages such as: Java, .NET, PHP, Python, JavaScript, Ruby, Go, and many others [6].

iii. **Distributed deployment -** RabbitMQ can be used to deploy as clusters for high availability and throughput; federate across multiple availability zones and regions [6].

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

iv. **Enterprise and Cloud ready** - RabbitMQ provides features such as pluggable authentication, authorization, supports TLS, i.e., Transport Layer Security, and LDAP, i.e., Lightweight Directory Access Protocol. RabbitMQ is also Lightweight and easy to deploy in public and private clouds [6].

v. **Tools and Plugins** - RabbitMQ comes with a diverse array of tools and plugins supporting continuous integration, operational metrics, and integration to other enterprise systems. It also has a flexible plug-in approach for extending RabbitMQ functionality [6].

vi. **Management and Monitoring** - RabbitMQ provides HTTP-API, command line tool, and UI for managing and monitoring [6].

KEY TERMS USED IN RABBITMQ:

i. **Producing** basically means sending. A **Producer** is a program that sends messages [6].

ii. A **queue** stores all the messages that flows in between the applications. It is only bound by the host's memory & disk limits, it's essentially a large message buffer. Many producers can send messages that go to one queue, and many consumers can try to receive data from one queue [6].

iii. **Consuming** means receiving the messages from the queue. A **consumer** is a program that mostly waits to receive messages [6].

iv. In RabbitMQ, the producer does not send any messages directly to a queue, it can only send the messages to an **exchange**. An exchange receives messages from producers and pushes them to queues depending on the exchange type [6].

v. The relationship between exchange and a queue is called a **binding**. It can also be said that the queue is interested in messages from this exchange [6].

TYPES OF EXCHANGES SUPPORTED BY RABBITMQ:

i. **Direct exchange** - A message goes to the queue or queues with a binding key that is exactly same as the routing key of the message [7].

ii. **Default exchange** - When the default exchange is used, the message will be delivered to the queue with a name equal to the routing key of the message. Every queue is automatically bound to the default exchange with a routing key which is the same as the queue name [7].

iii. **Topic exchanges** route messages to queues based on wildcard matches between the routing key and the routing pattern specified by the queue binding. Messages are routed to one or many queues based on a matching between a message routing key and routing pattern [7].

iv. The **fanout** copies and routes a received message to all queues that are bound to it regardless of routing keys or pattern matching as with direct and topic exchanges [7].

II. RELATED WORK

In [1], the arguments have been framed in a holistic approach by establishing a common comparison framework based on the core functionalities of pub/sub systems. Using this framework, a qualitative and quantitative (i.e. empirical) comparison of the common features of the two systems is done. Additionally, the distinct features that each of these systems has been highlighted. After enumerating a set of use cases that are best suited for RabbitMQ or Kafka, this paper guides the reader through a determination table to choose the best architecture given his/her particular set of requirements.RabbitMQ is, in essence, an open-source implementation of AMQP, a standard protocol with a highly-scrutinized design. As such, it enjoys a higher level of interoperability and can easily work with (and even be replaced by) other AMQP-compliant implementations [1].In addition to AMQP, RabbitMQ supports a few other industry standard protocols for publishing and consuming messages, most notably MQTT (a very popular choice in the IoT community) and STOMP. Hence, in settings with mixed use of protocols, RabbitMQ can be a valuable asset [1].RabbitMQ, in addition to clustering, also supports federated exchanges which is a good match for Wide-area deployment with less-reliable network connections. Compared to Clustering, it has a lower degree of coupling. A very useful feature of the federated exchanges is their on-demand forwarding. Furthermore, through its Shovel mechanism, RabbitMQ provides another convenient and easy way to chain brokers/clusters together [1].RabbitMQ ships with an easy-to-use management UI that allows user to monitor and control every aspect of the message broker, including: (i)

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

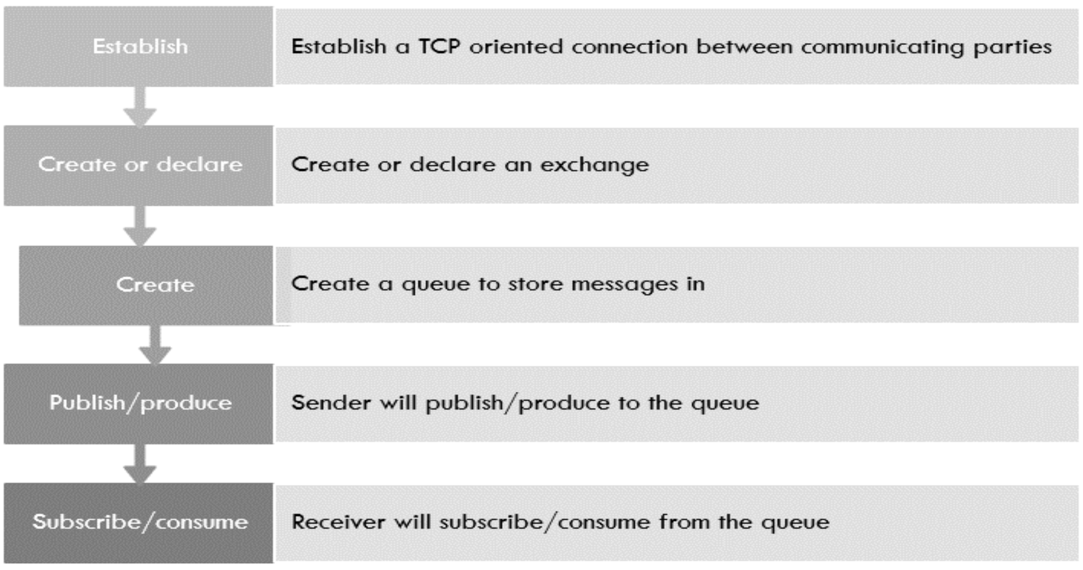
connections, (ii) queues, (iii) exchanges, (iv) clustering, federation and shoveling, (v) packet tracing, (vi) resource consumption. Together, these offer excellent visibility on internal metrics and allow for easy test and debug cycles [1].RabbitMQ implements the notation of Virtual Hosts which is defined by AMQP to make it possible for a single broker to host multiple isolated environments [1].At queue level, it keeps state, and knows exactly what consumers have consumed what messages at any time [1].RabbitMQ does not require disk space to route packets, if persistence is not a requirement. This makes it a good choice for embedded applications and restricted environments [1].RabbitMQ can stop publishers from overwhelming the broker in extreme situations. This can be used in a flow control scenario when deletion of messages is not acceptable [1].A queue can be limited in size. This mechanism can help in a flow control scenario when deletion of messages is acceptable [1].A message can be given a “Time To Live”. If it stays beyond that time in any queue, it will not be delivered to the consumer. This makes a lot of sense for realtime data that becomes irrelevant after a specific time. The TTL can be attached to a queue at creation time, or to individual messages at the time of publishing [1].

In [2], the RabbitMQ and ActiveMQ brokers are compared and the results that show the performance difference for both sending and receiving messages are presented. After testing the broker performance, it can be seen that ActiveMQ is faster on message reception, i.e., when the client sends messages to the broker, but the same tests show that RabbitMQ is faster on producing messages, i.e., when the client receives messages from the broker, as compared to ActiveMQ.In GPS applications for a taxi company that displays the location of its automobiles, MOM services are designed to transmit coordinates to a database, and then back to the administrator when required. In this case, RabbitMQ is used as many messages are sent, but the server is queried less [2].

In [3], a performance evaluation of message broker system, RabbitMQ in high availability - enabling and redundant configurations is presented. The scalability and high availability design issues are discussed. Since HA and performance scalability requirements are in conflict, scenarios for using clustered RabbitMQ nodes and mirrored queues are presented. The results of performance measurements are reported.

III. DESIGN

A single server was listening continuously, which was the subscriber. To analyze efficiency, 3 publishers were implemented which were continuously sending messages at a rapid pace. The RabbitMQ Message bus was implemented in Python using the following steps.


Establish
Establish a TCP oriented connection between communicating parties
Create or declare
Create or declare an exchange
Create
Create a queue to store messages in
Publish/produce
Sender will publish/produce to the queue
Subscribe/consume
Receiver will subscribe/consume from the queue

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

The **RabbitMQ server** was deployed on a machine with the following specifications:
HARDWARE SPECIFICATIONS:

Processor: 8x Intel® Core™ i7-6700K CPU @ 4GHz
Memory(RAM): 16 GB
Hard disk: 1 TB

SOFTWARE SPECIFICATIONS:

Operating System: Ubuntu 16.04.2 LTS
Programming Language: Python (pika package required for RabbitMQ APIs)
Softwares to be installed: RabbitMQ server

The **publisher and the subscribers** were deployed on a machine with the following specifications:
HARDWARE SPECIFICATIONS:

Processor: Intel® Core™ i7-6600U CPU @ 2.60GHz 2.81 Ghz
Memory(RAM): 8 GB
Hard disk: 256 GB

SOFTWARE SPECIFICATIONS:

Operating System: Windows 10
Programming Language: Python
Softwares to be installed: None

IV. IMPLEMENTATION

The Subscriber was implemented using the following steps:

1. Establish TCP connection with RabbitMQ server. The Publisher will be constantly listening for incoming connections.
2. Create/Declare a queue for communication purpose.
3. Create a callback function and subscribe to the queue. The name of the queue to which it should subscribe must be mentioned here.
4. Wait for the messages to be consumed from the queue

The Producers were implemented using the following steps:

1. Establish TCP connection with RabbitMQ server.
2. Create/Declare a queue for communication purpose.
3. Create/Declare an exchange to decide which messages go to which queue. In this case, since there is only one Subscriber and it has to process messages from all producers in the same manner, only one queue is created, and the default exchange is used.
4. Publish the messages onto the exchange. The exchange will take care of routing them onto the proper queue.

V. SIMULATION RESULTS

i. **Using single publisher**:

The graph below depicts the number of messages queued in the RabbitMQ message queue, alongwith the message rates, i.e., the number of messages received by the RabbitMQ server per second. From the graph we can observe that at messages received at less than 50000/s, RabbitMQ need not store the messages onto the queue. The RabbitMQ server is able to easily process the messages at this rate.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

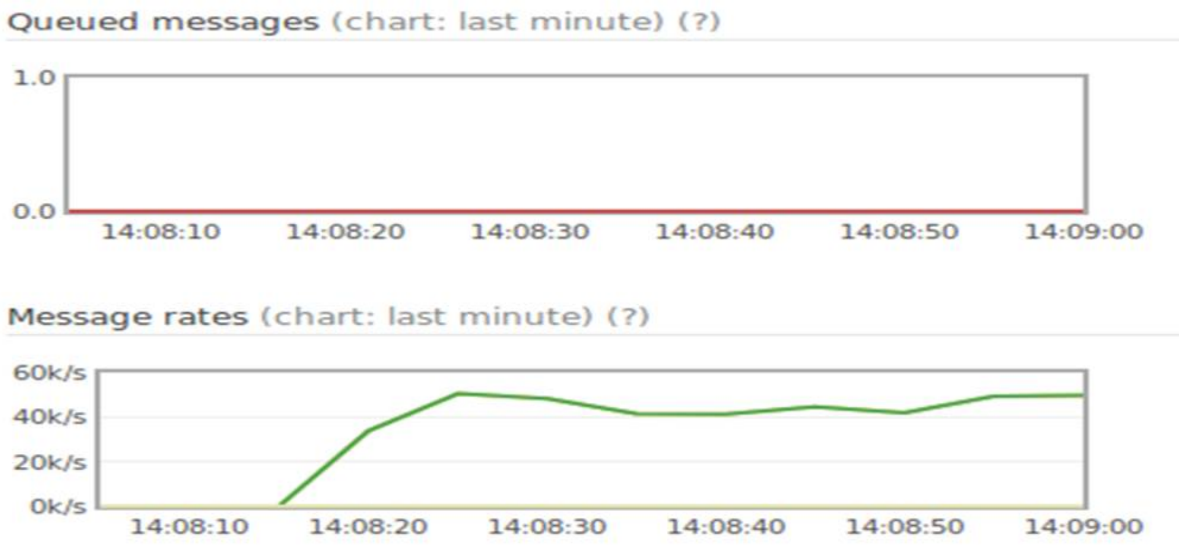


Fig.1. Graph when a single publisher is sending the messages

ii. **Using two publishers**:

As the number of messages to be processed increases, the requirement for the message broker to store the messages onto the queue arises. As seen from the graph, the messages start getting accumulated in the queue, waiting for the server to process it. However, the server is able to process each and every message, at a slower rate than before.

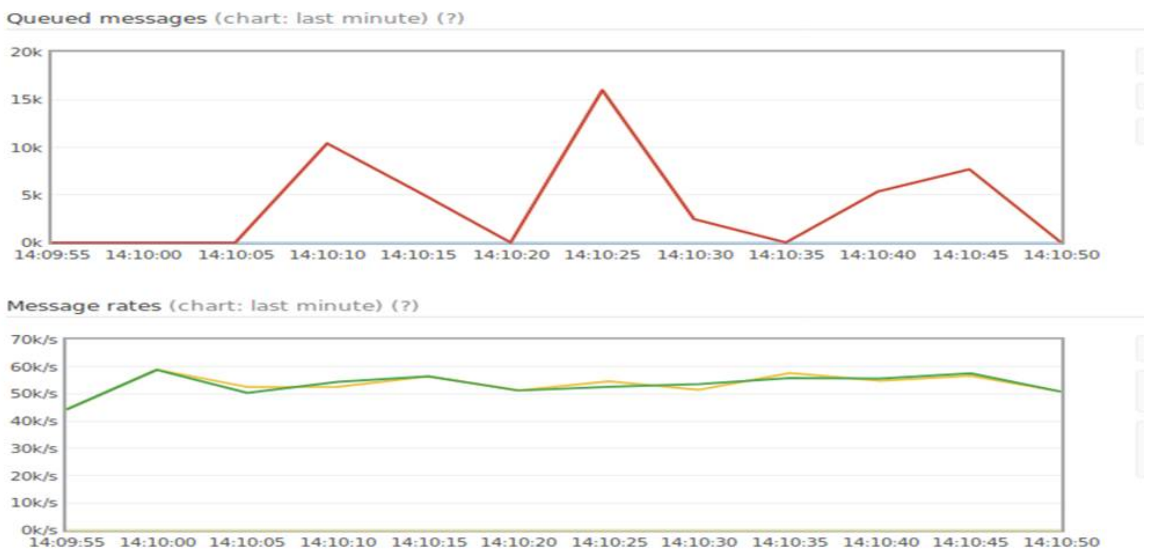


Fig.2. Graph when two publishers are sending the messages

iii. **Using three publishers**:

As seen from the graph, when three publishers simultaneously send messages to a single server, the number of messages stored in the queue increases. However, at a particular point, the number of queued messages remain constant, despite the publishers still sending messages. Which means the messages that are sent beyond this

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 1, January 2018

point are lost. In the machine with the given specifications, this number is found to be around 9 Million. Once the publishers stop sending messages, the number of queued messages begin to come down.

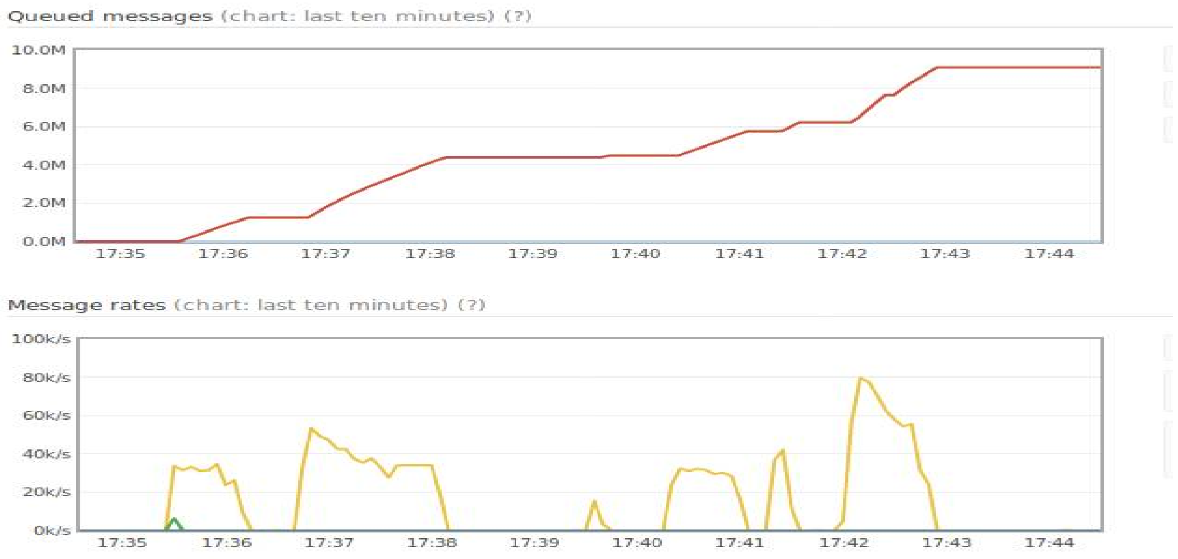


Fig.2. Graph when three publishers are sending the messages

VI. Conclusion

On the machine with the specifications given above, RabbitMQ can store and process upto 10 Million messages. Beyond that, the messages are lost. RabbitMQ Message broker can be used to easily implement a message bus for various distributed systems. RabbitMQ APIs are available in various popular languages such as Java, Python, Objective C, etc. which provides developers with a high level of flexibility. RabbitMQ also provides easy routing capabilities, which is an essential requirement of a message bus. Alongwith this, RabbitMQ also allows asynchronous messaging, which is very useful to implement a Message Bus for systems with high complexity. Also, it provides a good level of security as there are authentication and authorization functionalities within the broker APIs. Overall, RabbitMQ can be used to create quite an efficient Message Bus for enterprise applications which do not transfer messages at an extremely high rate.

References

1. Dobbelaere, Philippe, and KyumarsSheykhEsmaili. "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.
2. Ionescu, Valeriu Manuel. "The analysis of the performance of RabbitMQ and ActiveMQ." RoEduNet International Conference-Networking in Education and Research (RoEduNet NER), 2015 14th. IEEE, 2015.
3. Rostanski, Maciej, Krzysztof Grochla, and Aleksander Seman. "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ." Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014.
4. Rong, Chen. "Preliminary exploration on enterprise application integration based on message-oriented middleware." Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on. IEEE, 2011.
5. Peng, Zhen, Zhao Jingling, and Liao Qing. "Message oriented middleware data processing model in Internet of things." Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on. IEEE, 2012.
6. https://www.rabbitmq.com/
7. https://www.cloudamqp.com/
8. https://msdn.microsoft.com/en-us/library/ff647328.aspx
9. https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html
10. https://en.wikipedia.org/wiki/RabbitMQ