EXAMENSARBETE INOM DATATEKNIK,
GRUNDNIVÅ, 15 HP
STOCKHOLM, SVERIGE 2016

Comparing message-oriented middleware for financial assets trading

Jämförelse av meddelandeorienterade mellanprogramvaror för värdepappershandel

JOHN ERIKSSON

KTH
SKOLAN FÖR TEKNIK OCH HÄLSA

Comparing message-oriented middleware for financial assets trading

Jämförelse av meddelandeorienterade mellanprogramvaror för värdepappershandel

JOHN ERIKSSON

Degree project in Computer Engineering,
First Cycle, 15 hp
Supervisor at KTH: Anders Cajander
Examiner: Ibrahim Orhan
TRITA-STH 2016:63

KTH
School of Technology and Health
136 40 Handen, Sweden

Abstract

Many different message-oriented middlewares (MOM) exist today on the market. With different underlying designs, purposes and features, it is not easy to make the right choice of MOM for your system. The IT company Nordicstation is in such a situation where they need to make a choice of MOM. They currently own a system called Sharelock which generates reports about violations of financial assets trading. They wish to make the process of generating a report more asynchronous by using a MOM and they have a couple of requirements.

This thesis was carried out with the purpose of finding the most suitable products for the system in mind, compare their features, performance, licenses and ease-of-use. This was accomplished by analysing their installation process, monitoring interfaces, documentation on their websites and performance in a simple throughput test.

The results showed that RabbitMQ was the strongest candidate. It had good performance, an attractive web interface for monitoring, an easy installation and it also offered commercial support. Apache Artemis was also found to be an attractive choice but it did not have a web interface which made it hard to manage the system.

Keywords
Message-oriented middleware, MOM, RabbitMQ, ActiveMQ, Artemis, throughput, message, documentation, installation

Sammanfattning

Idag finns det många meddelandebaserade mellanprogramvaror (MOM) på marknaden. Dessa har olika grundläggande designer, syften och kännetecken vilket kan göra det svårt att göra ett bra val av MOM för ett visst system. IT-företaget Nordicstation befinner sig i en sådan situation där de behöver göra ett val av MOM. De har ett system som kallas Sharelock som genererar rapporter om regelbrott inom värdepappershandeln. De vill göra processen mer asynkron genom att använda en MOM och de har ett antal krav på produkten.

Det här examensarbetet utfördes med syftet att hitta de mest passande produkterna för det tänka systemet, jämföra deras särdrag, prestanda, licenser och användarvänlighet. Detta gjordes genom att analysera deras installationsprocess, övervakningsgränssnitt, dokumentation på deras hemsida och prestanda i ett enkelt prestandatest.

Resultaten visade att RabbitMQ var den starkaste kandidaten. Den hade bra prestanda, ett attraktivt webbgränssnitt, en enkel installation och den erbjöd också kommersiell support åt kunder. Apache Artemis var också ett attraktivt val men den hade inget webbgränssnitt vilket gjorde det svårt att övervaka och hantera systemet.

Nyckelord
Meddelandebaserade mellanprogram, MOM, RabbitMQ, ActiveMQ, Artemis, prestanda, meddelande, dokumentation, installation

Preamble

I would like to thank Mårten Palm and Johan Mannerqvist at Nordicstation. As my supervisors they offered great help and pointed me in the right direction. It was valuable to hear their opinion on different matters such as how the tests were to be structured.

I would also like to thank my supervisor at KTH, Anders Cajander. He had much helpful advice for me when writing my report and made the process a whole lot easier.

Table of contents

1 Introduction

There exists many different message-oriented middlewares (MOM) that help with the communication in bigger software systems. Some of them are designed to simply get a message from one point to another while some are more advanced and offer various features that aim to improve speed, stability or ease of use. With a plethora of choices, it is easy to choose a product that is not properly designed for one's application and be forced to tweak the product in order for it to be effective.

The IT company Nordicstation currently owns a system called Sharelock in which bankers can enter their stock possessions. Since there are many rules that must be followed when dealing with stocks, Sharelock can generate reports that show any violations to these rules. These reports require heavy computations on large amounts of data when fetched. To reduce the time spent waiting for reports, these heavy computations could run asynchronously as soon as data is added to the system. This can be accomplished by using a message-oriented middleware (MOM) on which messages are added when a banker adds or modifies data. A consumer can then poll messages off of the MOM and process them without making the banker have to wait.

For this reason, it is of interest to investigate which MOMs that Nordicstation can choose between and what their advantages and disadvantages are. There are a couple of requirements that the MOMs will need to meet such as being able to run in a Windows environment and cooperate with .NET and Microsoft SQL Server products. It is important that the product does not cost much when installing a new instance, which means that a free or open source product is preferable.

1.1 Expected outcome

In short, this thesis aims to compare different MOMs to see which ones are suited for this purpose with the previously mentioned requirements.

This is the expected outcome in a more detailed list:

- Investigation of available MOMs to see which ones are at all compatible with the environment in mind.
- Investigation of their strengths and weaknesses to see which ones are worth testing.
- A performance testing environment and performance tests on the MOMs that were found to be the most fitting.
- Evaluation of their licensing models.

1.2 Delimitations

- The project was carried out over 10 weeks between the end of March 2016 and the end of May 2016.
- A maximum of 3 MOMs were to be thoroughly investigated, compared and tested for performance.

2 Background and theory

This chapter explains the key concepts of a MOM and describes some of the earlier works that are relevant to this thesis.

2.1 Messages

A message in this context is not a traditional e-mail or such with sentences or words, but instead a set of data regardless of type. The sender of a message is often referred to as the producer while a receiver of a message is often referred to as a consumer. Examples of what a message could contain are notifications of finished tasks or, like in the case of the Sharelock application, notifications of new transactions in the system.

2.2 Message-oriented middleware (MOM)

Middleware is often described as the glue between different software components. In short, middleware allows developers to focus less on the communication and more on the actual purpose of their software. A message-oriented middleware (MOM) allows different components in a system to communicate with each other, despite their differences, using messages. Chapter 2.3 explains some of the most common types of MOMs.

2.3 MOM types

This section gives an introduction to three different types of MOM: message queue, message broker and service bus.

2.3.1 Message queue

At a fundamental level, a message queue provides asynchronous communication between parties. This is accomplished by letting one party (producers) put messages on the queue and another party (consumers) read messages from the queue. However, a modern message queue provides a little more than that. Some of the things they can provide are routing (publish-subscribe, point-to-point, fan-out and more), durability through saving messages to disk and certain policies such as message time-to-live.

2.3.2 Message broker

A message broker is very similar to a message queue, but there are differences. A big difference is that a message broker can transform messages. They can transform messages from one format to another, split a message into multiple messages or combine multiple messages into one message.

2.3.3 Service bus

A service bus software is often a large framework which provides resilience and some smart functions to messaging systems. They often include a set of tools and functions such as monitoring and scheduling software or load balancing. A key difference between a service bus and a message queue or message broker is that a service bus provides accountability by only allowing a single publisher for a certain event type. However, a lot of software without this constraint are marketed as service busses. [1]

2.4 Persistence

Messages can either be persistent or nonpersistent. A persistent message is guaranteed to not be lost in the case of a server crash. This is handled differently by different MOMs, but they all somehow save messages to the disk until they are certain that a message has been delivered. Persistent messages are crucial for important data that must not be lost in transit. A MOM will naturally process messages slower when dealing with persistent messages, partly because there needs to be some form of confirmation between the MOM and the producer or consumer. [2]

2.5 Messaging protocols

Today there exists several application messaging protocols. Some of them function similarly but their purposes are different. Knowing which protocols were suitable for the system in mind was crucial in order make a decision about which MOMs to consider. This section briefly summarises the key characteristics for the three most popular open protocols: AMQP, STOMP and MQTT.

2.5.1 AMQP

AMQP (Advanced Message Queuing Protocol) is a binary protocol that provides reliability and interoperability. It has many features and fit well in larger enterprise systems. Many large companies use AMQP, such as Google, JP Morgan and NASA [3]. AMQP has been an OASIS standard since 2012 and an ISO standard since 2014.

2.5.2 STOMP

STOMP (Simple (or Streaming) Text Orientated Messaging Protocol), as the name implies, is a text-based protocol. It is intended to be interoperable and easily implemented. Even something as simple as a telnet client can communicate with a STOMP broker. [4]

2.5.3 MQTT

MQTT (Message Queue Telemetry Transport) is a binary protocol that is extremely lightweight in terms of memory and network usage. This makes it very useful for smaller sensors and mobile devices [5]. Despite it’s name, MQTT does not support queues, it only provides publish-and-subscribe messaging [3]. Since lightweightness is one of the goals of MQTT, it does not support certain advanced functions such as message persistence.

2.6 Related work

In “Message-oriented Middleware for Scalable Data Analytics Architectures” [6], Nicolas Nannoni analyses the capabilities, benefits and drawbacks of using a MOM. He also compares Apache Kafka and RabbitMQ by looking at their features and performing some benchmarks. His conclusions are that while Kafka has better performance when dealing with persistent messages, RabbitMQ is overall more attractive. It is more attractive because it is easier to setup and manage, it is self sufficient and is very mature. According to Nicolas, managing a Kafka instance was not a simple thing since the software is not conveniently packaged. He describes Kafka as a nearly experimental product compared to RabbitMQ.

This information helped when deciding which products to compare later on. One of the requirements from Nordicstation was that the product needed to be fairly easy to get up and running on a new system, something that can be difficult with a product that is not self sufficient.

3 Methods and results

This chapter first explains the reasoning behind the choices of methodology used for this thesis. It also explains why some messaging protocols and products were chosen to investigate further or not.

Section 3.2 explains which MOM types that were chosen and section 3.3 explains which protocols were considered. Section 3.4 and 3.5 shows the list of MOMs that were available and also explains which method was used to find them. Section 3.7 explains how the performance tests were conducted and section 3.8 shows their results.

3.1 Methodology

In this thesis, literature studies and performance tests were used to determine which MOM was most appropriate.

Literature studies were used since many of these products have well-structured websites that are easy to extract information from. This was more appropriate than interviews since interviews have a tendency to get varying biased results depending on who is interviewed, especially when asking questions about products from different companies or organisations.

Performance tests were used since earlier works and benchmarks did not have scenarios that were similar enough. The benchmarks found tested small messages (1024 bytes or smaller) and very large messages (several megabytes) but not messages around 500 kilobytes, which was the expected size of the messages in Sharelock. Additionally, since performance was an important factor, it was important to have accurate and realistic performance benchmarks performed on a similar system that the MOM would eventually run on. There was also a large amount of data available that had been collected in the past that could be used as test data for the performance tests.

3.2 Choice of MOM type

Since there were multiple data sources (batch loading, real time, manual insertion), one of the demands on the messaging system was support for multiple producers. This meant that a service bus would not be appropriate and that a message queue or message broker would need to be used instead.

3.3 Choice of protocol

Since all messages in Sharelock contain important data, persistent messages were essential and therefore a MOM without support for persistent messages was not an option. This of course meant that MQTT was not an option. Also, MQTTs strength is lightweightness which was not a big concern. Some quick benchmarks [7] shows that while AMQP is faster in most cases, brokers using STOMP can have similar speeds. Because of this, both AMQP and STOMP implementations will be considered. Some MOMs use their own protocols and these will also be considered. This is because it was difficult to find benchmarks for less popular messaging protocols.

3.4 Finding available MOMs

Different search engines were used to find which MOMs were available. Google, Google Scholar, KTHs database of academic works “DiVA” and IEEE Xplore were searched using search terms such as “message-oriented middleware”, “messaging systems”, “message queue”, “message broker” and “service bus”. The products that appeared most frequently in those search results were chosen to investigate.

3.5 Available MOMs

This section lists a series of message queues and message brokers that were found. It also explains why some products were chosen to further investigate and why some where not. Table 3.5 below shows which of the requirements that were met for each product. The products in green had the required features and were inspected further. The products that are not green either did not have the requirements (red) or were deprecated/inactive (yellow).

Table 3.5: A table listing the found messaging products. The items in green have all the required features.

Product	Open source	Persistence	.NET support
RabbitMQ	Yes	Yes	Yes
Apache ActiveMQ	Yes	Yes	Yes
NATS	Yes	No	Yes
Apache Kafka	Yes	Yes	Yes
Kestrel	Yes	Yes	Unknown
NSQ	Yes	Yes	Inactive / Old
WebSphere MQ	No	Yes	Yes
Apache Apollo	Yes	Yes	Platform neutral
Apache Artemis	Yes	Yes	Platform neutral

3.5.1 Inappropriate products

NATS is a cloud-native messaging built with lightweightness and flexibility in mind. However, it is also designed to be a fire-and-forget system, meaning it only sends a message at most once [8]. This also means that message persistence is not available which made NATS an inappropriate choice.

Kestrel was a distributed message queue system built by Twitter that is now deprecated. It was difficult to find any .NET client implementations however, so it was probably not the best choice anyway.

NSQ is a realtime distributed messaging platform. There were a few .NET and Java implementations but they all seemed inactive and old. Since ease of installation and maintenance was one of the requirements, it did not appear as a suitable option.

WebSphere MQ (or IBM MQ) is a paid family of software products for messaging. The fact that it is a very beefy enterprise product and also not free did not make it interesting enough for this system.

3.5.2 Further investigation

Upon further investigation, it was found that Apache Apollo and Apache Artemis are not completely dissimilar messaging systems. They are both based on ActiveMQ and have similar objectives but use different approaches.

Apollo uses a different threading and message dispatching architecture compared to ActiveMQ[9]. It is written in Java and Scala but is no longer actively developed. Since ease of installation and maintenance was an important factor, Apollo did not seem like a good option since there is not much support available for an inactive or deprecated product.

Artemis is the codename for a codebase that was handed to the Apache Foundation from HornetQ. Since it was difficult to find comprehensive benchmarks that could give an idea of how it performed and since Artemis will possibly be the successor to ActiveMQ 5, it was a product that was very interesting to investigate further. [10]

Apache Kafka is not self sufficient, it requires an installation of Apache ZooKeeper and requires a whole lot more configuration to set up and maintain [6] compared to other products, such as RabbitMQ. Again, ease of installation and maintenance was an important factor and Kafka just did not seem to fit that description well enough to be tested further.

RabbitMQ was found in several comparisons and papers about MOMs where it received positive reviews. It seemed to be a very popular and ambitious project that worked well right out of the box. This made RabbitMQ an interesting product to compare against other strong competitors.

ActiveMQ was also found in several comparisons and papers about MOMs. Since Apache claims that ActiveMQ is the most popular and powerful messaging server, along with the fact that is is strongly related to Artemis, made ActiveMQ an interesting choice.

3.5.3 Appropriate products

Based on the information in previous chapters, it was chosen that the 3 MOMs that would be compared and tested were:

- RabbitMQ, by Pivotal Software, Inc.
- ActiveMQ, by the Apache Software Foundation.
- The possible successor of ActiveMQ, Artemis.

3.6 Characteristics and features

This chapter shows the results of the detailed investigation for the three candidates: RabbitMQ, Apache ActiveMQ and Apache Artemis. Each section describes their characteristics, core purposes and architectures. An analysis about how they compare to each other can be found in chapter 4.

3.6.1 RabbitMQ

RabbitMQ is an open source MOM written in Erlang. It was first released in February 2007 and is developed by Rabbit Technologies Ltd which is indirectly owned by VMware. It offers commercial support and a big range of clients for many platforms including Java, .NET, Ruby, JavaScript and Python [11]. It runs on both UNIX and Windows systems and even offers a Windows installer. While AMQP is the "core" protocol supported by RabbitMQ, it also supports STOMP and MQTT.

3.6.1.1 Documentation

The documentation that RabbitMQ provides is very extensive and consists of detailed guides and API references. For the server, all of the configurable parameters are listed and described in tables. The documentation also contain information about how certain functionalities are implemented and what developers should consider to avoid problems.

The client documentation consists of API guides and references for the Java, .NET and Erlang clients. The API guide demonstrates how to connect to a server, send and receive messages using different patterns and handle errors. The API reference, on the other hand, is more similar to a typical Javadoc page with all the classes, functions and their attributes and parameters listed.

3.6.1.2 Installation & Management

As mentioned earlier, RabbitMQ can be installed on Windows using an ordinary ".exe" installer file which is downloaded directly from their website. An installation of Erlang is required for RabbitMQ to work and the installation instructions on the RabbitMQ website provides a link to the Erlang installation file for Windows. Like with most installers, both the Erlang and RabbitMQ installers only requires a few clicks on the "Next" button and the installations are complete.

For basic commands such as starting, stopping and restarting the RabbitMQ service, the installation provides Start Menu buttons. A RabbitMQ Command Prompt can be used for more advanced commands. The service is started by using the Start Menu button or by using the "rabbitmq-server" command. The user can choose to enable a management plugin which when enabled activates a web interface. Through this web interface, the messaging system can be monitored and managed. The web interface shows information about connections, channels, exchanges, queues, messages and users. This information is by default updated every 5 seconds but the time interval is adjustable. It also offers the ability to add, modify or delete exchanges, queues and messages.

One of the web interface's pages is particularly interesting for this thesis. It shows valuable information about the RabbitMQ nodes running on the system. For example, the number of Erlang processes, amount of memory in use and I/O statistics (number of messages passing through the system per second) can easily be read from diagrams.

3.6.1.3 License
RabbitMQ is released under the Mozilla Public License 1.1 (MPL-1.1). MPL-1.1 allows you to use the software freely for both personal and commercial use. When redistributing modified MPL-1.1 software however, the original files must be made available for anyone to see.

3.6.2 Apache ActiveMQ
According to its website, Apache ActiveMQ is the most popular open source messaging server. It is written in Java and is a very mature product with its 1.0 version released all the way back in 2004. Since ActiveMQ is developed by a community of volunteers (unlike RabbitMQ which is developed by a company) there is no commercial support provided by Apache. However, they provide a list of companies that specialize in ActiveMQ that can provide that kind of support. There are client implementations for many platforms including Java, .NET, Ruby, JavaScript and Python.

Like RabbitMQ, ActiveMQ supports the three big messaging protocols AMQP, MQTT and STOMP. However, its default protocol is called OpenWire. OpenWire is designed for performance and size while sacrificing ease of implementation [12].

3.6.2.1 Documentation
ActiveMQ's documentation consists of Javadocs, a "Getting Started" document and a FAQ section. The "Getting Started" document explains how to install and setup ActiveMQ on Unix and Windows systems, how to test and monitor the messaging system and how to configure it. There are instructions for installing using the binaries as well as the source for those who are interested in modifying the software.

The FAQ page has many sections with many questions. The sections include "General", "JMS" (Java Message Service, which is an API that ActiveMQ is an implementation of), "Using Apache ActiveMQ", "Configuration", "Persistence", "Errors" and "Developing ActiveMQ". With each section having about 20 questions each, it is very likely that a user will find an answer to their question there.

3.6.2.2 Installation & Management
ActiveMQ requires an installation of the Java Runtime Environment (JRE). This is easily installed by downloading and running an installation file from Java's website. When the JRE is installed, ActiveMQ is installed by downloading a .zip file and extracting the contents to a desired directory. No installation file needs to be run, everything ActiveMQ needs (except for the Java installation) is inside the extracted folder.

To start the ActiveMQ system, a batch file is executed with the ".\activemq start" argument. Once ActiveMQ is running, it can be monitored and managed through a web interface that looks very similar to RabbitMQ's web interface. This interface offer similar abilities and information. A slight difference is that most information is displayed in percentages, compared to the "raw" values that RabbitMQ provides.

When starting the system for the first time, two errors were encountered. The first error stated that the “JAVA_HOME” environment variable was not set. This was solved by adding an environment variable to the Windows system and pointing it to the Java installation. The second error occured due to lack of permission for the user and was solved by running the command terminal as the administrator.

3.6.2.3 License

Apache ActiveMQ is released under the Apache 2.0 license (APL-2.0). APL-2.0 allows you to use the software freely for both personal and commercial use. When redistributing software with the APL-2.0 license, proper attributions must be given.

3.6.3 Apache Artemis

Apache Artemis is a messaging system that originates from HornetQ and ActiveMQ. The HornetQ code base was donated to the Apache ActiveMQ community and together they created a messaging system that is supposed to have better performance and stability than its predecessors [13]. Like ActiveMQ, Artemis supports AMQP, MQTT, STOMP and OpenWire. Since it is a fairly new product, there are not many client implementations available. However, clients that work with ActiveMQ 5 are also supported by Artemis.

3.6.3.1 Documentation

Artemis’ documentation consists of a user manual. There is much information packed into the manual but it is not very structured. Information about how Artemis works under the hood, how a user should use Artemis and what variables are available is all in the same document (over 240 pages long in the PDF version) which can be a little confusing.

The Artemis project provides plenty of examples. While this might not be classified as documentation, examples are often a good way to get started. Over 90 examples come with the distribution and they are meant to demonstrate Artemis’ features.

3.6.3.2 Installation & Management

Like ActiveMQ, Artemis is installed by downloading a .zip file and extracting its content. There is however one more step that needs to be performed: creating a broker instance. A broker instance is a directory with configuration and runtime data and is created by running a simple command: “.\artemis create <broker_name>”. After a broker instance has been created, it can be installed as a Windows service. If it is installed as a Windows service, it can easily be started, stopped and restarted through Windows’ Task Manager.

Artemis does not have a web interface that comes with the installation. There are third-party tools that can be used to monitor the system. A list of third-party tools [14] is provided but none of them are easy to install or connect to the broker. Many of them are old and not actively developed or require multiple XML files to be configured.

Just like ActiveMQ, Artemis requires the “JAVA_HOME” environment variable to be set and must be run as an administrator. However, an additional problem was encountered when setting up Artemis. When running the basic examples, Artemis quickly ran out of memory and crashed. This was solved by changing a parameter in the Artemis startup script which gave the Java Virtual Machine more memory.

3.6.3.3 License

Apache Artemis is released under the Apache 2.0 license (Apache-2.0), just like ActiveMQ.

3.7 Performance testing

This section first explains how the performance tests were conducted and how the results were obtained. It then explains certain things that were taken into consideration to get more accurate results and the reasoning behind the choice of message sizes. Finally, an overview of the hardware and software used is provided as well as an explanation of the workflow of the test program.

3.7.1 Way of testing

To examine the performance of the MOMs, 6 different test cases were created for each MOM. Both producing and consuming messages to the MOM was tested using 3 different message sizes. The reason why different message sizes were tested despite knowing what a message would contain in Sharelock was that it could be subject to change in the future. All messages that were sent and received were persistent messages, which meant that the MOM not only stored each message in the RAM but also on disk.

To test the throughput for either sending or receiving, a simple program was run where messages were sent to or received from the MOM. The result was then obtained by calculating throughput using a simple formula.

$$T = x / t \tag{1}$$

Formula 1 demonstrates how throughput is calculated. Throughput, denoted as T, is obtained by dividing the number of units of information (messages in this case), denoted as x, by the time elapsed, denoted as t.

3.7.2 Improving accuracy

To get a more accurate result, 10 runs were performed for each test case. Their mean result as well as a confidence interval for each test case was calculated. To decrease the risk of random factors having an impact on the results, a couple of things were considered. Some test runs were performed before the actual tests to “warm up” the environment. This was done to prevent just-in-time compilation from being performed in the middle of the tests. Additionally, the garbage collector was run just before the tests so that it would be less likely that it would run in the middle of the tests. The computer’s airplane mode was also used so that process time would not be spent on any sort of communication.

3.7.3 Message size and quantity

The message sizes used were 400, 500 and 600 kilobytes. 500 was chosen because it was the expected size of the largest messages in the future Sharelock implementation. 400 and 600 were chosen as a 20% decrease and increase from 500 to give an idea of how the performance would change if the message content were to change in the future. In each test, 10000 messages were sent or received to the MOM. A larger amount of messages could have been used to get a more accurate result but there was not enough time to run each test case with a larger message count.

3.7.4 Testing environment and versions

The tests were run on a Lenovo ThinkPad with an Intel Core i7-4700MQ processor with 4 cores at 2.4 GHz. It had 8GB of DDR3 1600 MHz RAM and a 256GB SSD. The computer was running an installation of Windows 10 Enterprise version 1511 (build:10586.318).

The following versions of the products and their dependencies were used:

- RabbitMQ version 3.6.2 with Erlang/OTP version 18.3
- ActiveMQ version 5.13.3 with Java JDK version 8 update 91.
- Artemis version 1.2.0 with Java JDK version 8 update 91.

The test program that was built was a simple C# console application. The program took 4 parameters:

1. Name of a MOM (rabbitmq, activemq, artemis)
2. Action to perform (send, receive)
3. Number of messages to send or receive
4. Message size in bytes (only used with the “send” action)

The classes for the different products all implemented an interface called “MessageQueue” which had 4 methods:

- Setup: this method was called first and prepared the MOM by setting up a message queue.
- Send: this method was called if the chosen action was “send” and it simply produced a message to the MOM.
- Receive: this method was called if the chosen action was “receive” and it simply consumed a message from the MOM.
- Teardown: this method was called after the chosen action was done and properly closed the queues and connections that had been established.

The workflow of the program looked something like this:

1. Setup the message queue.
2. Warm up by producing 1000 messages to the queue and then consuming them.
3. Run the garbage collector.
4. Start a timer.
5. Perform the chosen action (send or receive messages).
6. Stop the timer and output the result.
7. Run the teardown method.

The source code for the test program can be found at:
https://github.com/johneriksson/MQBenchmarking

3.8 Test results

This chapter shows and explains the results of the performance tests with some charts and tables. Results from the sending tests are demonstrated in section 3.8.1 and results from the receiving tests are demonstrated in section 3.8.2. An analysis of the results can be found in chapter 4. The raw results, standard deviations and confidence intervals can be found in a spreadsheet at:

https://github.com/johneriksson/MQBenchmarking/blob/master/Results.xlsx

3.8.1 Sending

The results from the sending tests can be seen in figure 3.1. RabbitMQ had the highest throughput when sending messages to the queue regardless of message size. With a throughput of 232 messages per second using messages of 500 kB, it was around 4 times faster than ActiveMQ and Artemis. Artemis performed better than ActiveMQ, but only about 8 messages per second faster.

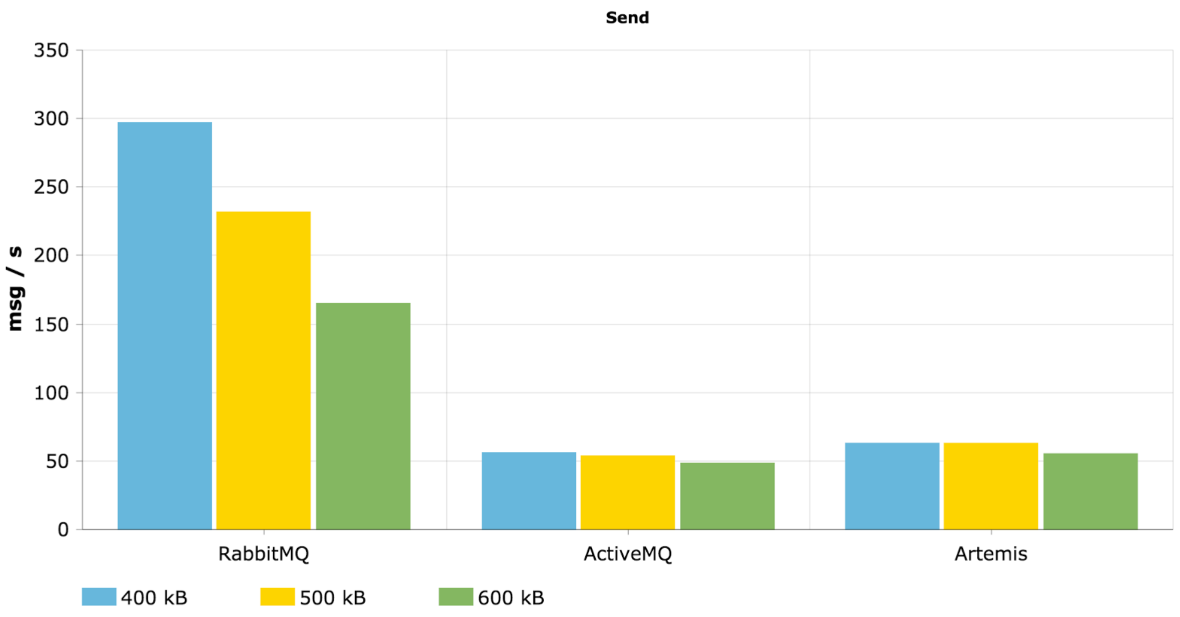

Figure 3.1: The results from the sending tests. Shows the throughput (msg / s) for the 3 MOMs using different message z

3.8.2 Receiving

The results from the receiving tests can be seen in figure 3.2 below. When receiving, RabbitMQ turned out to be the slowest of them all, regardless of message size. Artemis was the fastest with all message sizes and was more than twice as fast as RabbitMQ when using messages of 500 kB. The difference between Artemis and ActiveMQ was bigger when receiving. Artemis was around 60% faster than ActiveMQ when using messages of 500 kB.

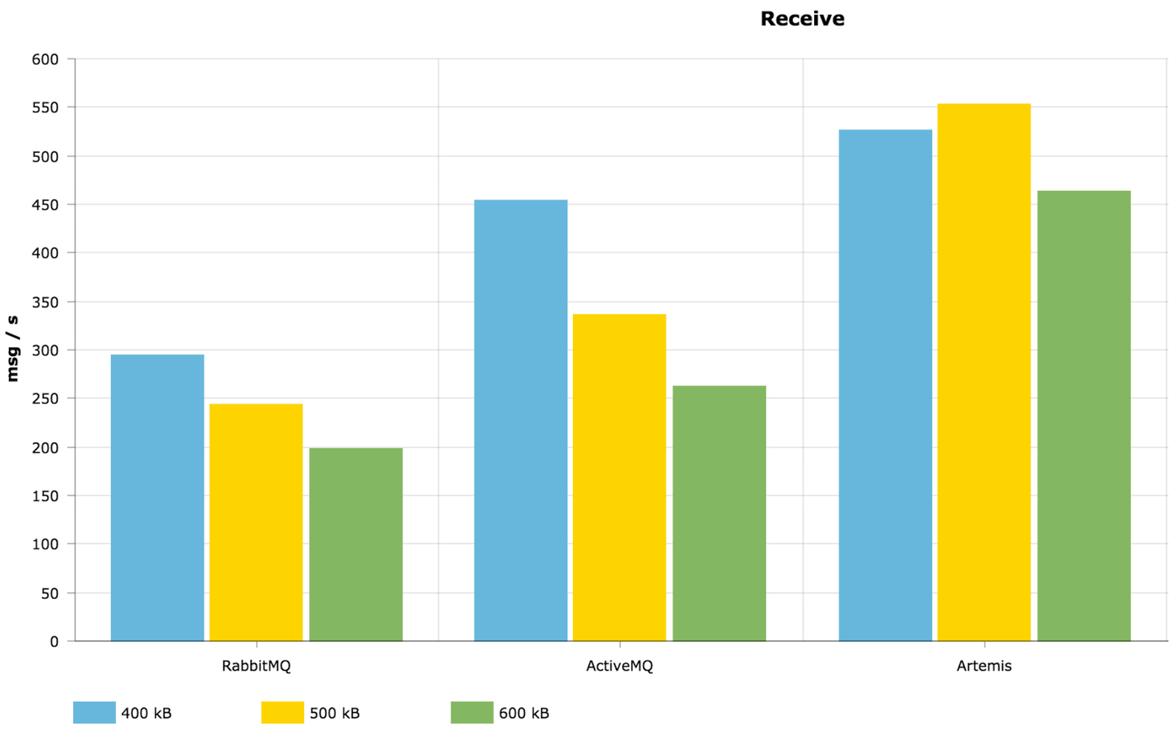

Figure 3.2: The results from the receiving tests. Shows the throughput (msg / s) for the 3 MOMs using different message sizes.

4 Analysis and discussion

This chapter contains an analysis of the results from chapter 3 and discusses some observations that were made. It aims to explain why the results looked the way they did and how the results could have been improved. A conclusion of the results and analysis can be found in chapter 5.

4.1 Documentation

RabbitMQ’s documentation was not needed when setting up the test program, the code from their examples worked right away and were easy to comprehend. The documentation was however very detailed with specific instructions for .NET users. The only negative thing about RabbitMQ’s documentation was that it was not easy to navigate. Some pages were very long and should have been split up into several pages.

ActiveMQ’s documentation had all the necessary information required to get started. It was a little difficult to navigate to the correct page to find the information however. Some parts of the documentation was found at the top of the webpage while some parts were found by scrolling down to the bottom. The FAQ section helped greatly when trying to solve the problems that were encountered, such as not being able to start ActiveMQ because of permission issues.

Artemis’ user manual also had all the necessary information required to get started. The examples were in Java but still helped greatly when setting up the test program in .NET since many classes and methods looked the same. A FAQ section would have been of great help when dealing with memory problems, but a solution was found fairly easily by taking a look at the configuration reference. The user manual had a nice feature that the other candidates did not offer: a search function for the documentation. This search function was not a simple function that found occurrences of a word but instead suggested appropriate pages related to the search term.

ActiveMQ and RabbitMQ’s websites were too hard to navigate when looking for documentation compared to Artemis’ website. Artemis’ documentation seemed a little bit messy at first with that much information in one place, but proved to be very effective and was the most pleasant one to browse. Artemis’ examples also greatly complimented any possible lack of information in the user manual.

4.2 Installation

RabbitMQ was by far the easiest product to install with an easy-to-use Windows installer. It worked well right out of the box and did not need any configuration at all. It was also easy to start with the Start Menu buttons that came with the installation. ActiveMQ almost worked right out of the box, only a minor problem regarding the path to JAVA_HOME needed configuration. Artemis, on the other hand, was more difficult to install. Since the broker instance that was created resided in a new directory, configuration files resided in both the Artemis directory and the broker instance directory. This made it confusing to setup the system, especially with no previous experience of using a MOM.

RabbitMQ's website says that it is "messaging that just works", and in this case it was true. RabbitMQ's installation and setup process was easily the best of the 3.

4.3 Management / Monitoring

The web interfaces that come with RabbitMQ and ActiveMQ were very pleasant to use. They both offered the ability to browse, add and delete queues as well as send messages to them. RabbitMQ's web interface did offer a bit more though, such as detailed information about the current throughput, history displayed in graphs and the number of processes that the broker was using. Artemis did not have a web interface at all and important information was only accessible through a REST API which was not convenient.

RabbitMQ's interface was slightly richer altogether and felt better for that reason.

4.4 License

The two licenses that were encountered, Mozilla Public License and Apache License are fairly similar. The Apache License is a tiny bit more permissive but since no modifications are intended to be made to the software in the future, the licenses did not make a difference for the Sharelock system. For anyone that plans to make modifications to a piece of software that uses the Mozilla Public License, they should thoroughly read the license so they know if they need to take certain actions.

4.5 Performance

RabbitMQ had the best performance when looking at sending and receiving together. It had a good and consistent performance with a throughput of around 250 messages per second for both sending and receiving. It did however seem to struggle a little when sending larger messages. The performance when sending messages of 600 kB was around half as good as when sending messages of 400 kB. This was however not going to be a problem since the message sizes used in the tests were slightly overdimensioned.

ActiveMQ had a decent performance while receiving but was not fast at all when sending these large messages. Since the system worked almost right away with no configuration needed, not much time was spent looking at the different parameters that ActiveMQ has. The bad performance while sending could possibly have been improved greatly by using a more optimal configuration.

Artemis' performance was interesting. It performed only slightly better than ActiveMQ when sending with a not so impressive 63 messages per second using messages of 500 kB. When consuming messages however, RabbitMQ performed much better than any of the other two. It is possible that the throughput when sending messages could have been higher if more time was spent on configuring the many parameters that Artemis had.

4.6 Economic, social and environmental aspects

The system will play a vital role in todays fast moving society by allowing stock trading processes to flow without unnecessary delays. Since RabbitMQ had an overall better throughput and not a significantly higher processor power or memory usage, a system using RabbitMQ would be using less time and energy on producing and consuming messages compared to an equivalent system using ActiveMQ or Artemis. This is of course good for a company's economy and their environmental impact since less electricity is used and computer parts are subject to less wear and tear. Users of the system would also experience shorter waiting times.

5 Conclusions

The study found that among the most popular MOMs RabbitMQ and Apache ActiveMQ were the most suitable products for the system in mind. They were investigated, compared and tested along with Apache Artemis, the possible successor of ActiveMQ.

The installation process was fairly simple for all products but RabbitMQ's installation stood out with their convenient Windows installer program. ActiveMQ and Artemis both had simple command line tools but were of course not as easy to get started with as RabbitMQ's Start Menu buttons.

On the other hand, the documentation was not RabbitMQ's best side. The documentation on the website was detailed but not well structured and it was easy to get lost in there. The same went for ActiveMQ's documentation. The HTML version of Artemis' documentation was much better structured, offered a useful search tool and a simple but powerful configuration reference.

Both RabbitMQ and ActiveMQ had easy-to-use web interfaces that came with the installation. RabbitMQ's interface was only slightly better because it offered a little bit more of information about the running nodes such as current amount of processes running. Artemis did not have a web interface at all and had to be monitored via a REST API which was very inconvenient.

When it came to performance, RabbitMQ had the best performance overall. It had a decent throughput when consuming messages but was blazing fast when producing messages. ActiveMQ was slow in both situations while Artemis had a slow producing throughput but the fastest consuming throughput.

Altogether, RabbitMQ offered the best product for the system in mind but also in general. The software came nicely packaged with an easy installation, good performance and an easy-to-use web interface. These qualities along with offering commercial support made RabbitMQ the recommended choice for most systems, especially those who run in a Windows and .NET environment. It did of course have room for improvements such as a better structured documentation. Artemis was also an attractive product and could very well have been the recommended choice for many systems if it had a web interface like RabbitMQ and ActiveMQ.

5.1 Suggestion for future work

In this project, bigger paid products were not compared and tested. It would be interesting to see how they perform compared to the open source products. This is however probably difficult to try unless a company is already using some other component of one of these bigger paid products like WebSphere MQ.

References

[1] Udi Dahan (Expert on software architecture and design).
“Bus and Broker Pub/Sub Differences”. Published 2011-03-24.
http://udidahan.com/2011/03/24/bus-and-broker-pubsub-differences/

[2] Oracle. “Delivery Mode (Persistent/Nonpersistent Messages) (Sun GlassFish Message Queue 4.4 Administration Guide)”. Accessed 2016-04-13.
https://docs.oracle.com/cd/E19879-01/821-0027/aeojo/index.html

[3] Andy Piper. “Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP - VMware vFabric Blog”. Published 2013-02-19.
http://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html

[4] “STOMP”. Accessed 2016-04-14. https://stomp.github.io/

[5] “MQTT”. Accessed 2016-04-14. http://mqtt.org/

[6] Nicolas Nannoni. “Message-oriented Middleware for Scalable Data Analytics Architectures”. Published 2015-02-13.
http://kth.diva-portal.org/smash/get/diva2:813137/FULLTEXT01.pdf

[7] Muriel Salvan. “A quick message queue benchmark”. Published 2013-04-10.
http://blog.x-aeon.com/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/

[8] Apcera. “NATS - Cloud Native, Open Source, High Performance Messaging”. Accessed 2016-04-13. http://nats.io/

[9] The Apache Software Foundation. “Apollo”. Accessed 2016-04-27.
https://activemq.apache.org/apollo/

[10] The Apache Software Foundation. “Apache ActiveMQ ™ -- How does ActiveMQ compare to Artemis” Accessed 2016-04-27.
http://activemq.apache.org/how-does-activemq-compare-to-artemis.html

[11] Pivotal Software. “RabbitMQ - Clients & Developer Tools”. Accessed 2016-04-27.
https://www.rabbitmq.com/devtools.html

[12] The Apache Software Foundation. “Apache ActiveMQ ™ -- How does OpenWire compare to Stomp”. Accessed 2016-04-27.
http://activemq.apache.org/how-does-openwire-compare-to-stomp.html

[13] The HornetQ Team Blog. “The HornetQ Team Blog: HornetQ Apache donation and Apache Artemis 1.0.0 release” Published 2015-06-01.
http://hornetq.blogspot.se/2015/06/hornetq-apache-donation-and-apache.html

[14] The Apache Software Foundation. “Apache ActiveMQ ™ -- How can I monitor ActiveMQ” Accessed 2016-04-27.
http://activemq.apache.org/how-can-i-monitor-activemq.html

TRITA 2016:63