Available online at sjuoz.uoz.edu.krd

Science Journal of University of Zakho

Vol. 5, No. 4, pp. 307 –312, Dec.-2017

p-ISSN: 2410-7549
e-ISSN: 2414-6943

JAVA MESSAGE SERVICE BASED PERFORMANCE COMPARISON OF APACHE ACTIVEMQ AND APACHE APOLLO BROKERS

Qusay I. Sarhan [a,*] and Idrees S. Gawdan [b]

[a] Dept. of Computer Science, College of Science, University of Duhok, Kurdistan Region, Iraq – (qusay.sarhan@uod.ac).
[b] Dept. of Refrigeration & Air-Conditioning, Technical College of Engineering, Duhok Polytechnic University, Duhok, Kurdistan Region-Iraq – (husseinidrees@yahoo.com)

Received: Aug. 2017 / Accepted: Dec., 2017 / Published: Dec., 2017 *https://doi.org/10.25271/2017.5.4.376*

ABSTRACT:
Software integration is a crucial aspect of collaborative software applications and systems. It enables a number of different software applications, created by different developers, using different programming languages, and even located at different places to work with each other collaboratively to achieve common goals. Nowadays, a number of techniques are available to enable software integration. Messaging is the most prominent technique in this respect. In this paper, two leading open-source messaging brokers, Apache ActiveMQ and Apache Apollo, have been experimentally compared with each other with regard to their messaging capabilities (message sending and receiving throughputs). Both brokers support exchanging messages between heterogeneous and distributed software applications using several messaging mechanisms including Java Message Service (henceforth JMS). A number of experimental test scenarios have been conducted to obtain the comparison results that indicate the one-to-one JMS messaging performance of each broker. Overall performance evaluation and analysis showed that Apache Apollo outperformed Apache ActiveMQ in all test scenarios regarding message sending throughputs. Whereas, Apache ActiveMQ outperformed Apache Apollo in most test scenarios regarding message receiving throughputs. Moreover, the evaluation methodology (test conditions, test scenarios, and test metrics) proposed in this paper has been carefully chosen to be adopted by software developers to evaluate other messaging brokers to determine the acceptable level of messaging capabilities in distributed environments of heterogeneous software applications.

KEYWORDS: Performance comparison, Messaging broker, Java Message Service (JMS), Messaging throughputs, Test methodology.

1. INTRODUCTION

Software applications that cover many aspects of our daily life are heterogeneous. They are created by different developers using different methods and tools. To provide a collaborative environment that enables a number of heterogeneous software applications to communicate with each other and perform various day-to-day activities, software integration is required. Currently, there are a dozen of techniques that can be used to achieve software integration (Hohpe & Woolf, 2003). Out of the available integration techniques, messaging is the most prominent technique. It allows two or more independent and different software applications to collaborate with each other by sending and receiving different types of messages via a messaging broker/server (He & Xu, 2014). This process is performed by using messaging frameworks, libraries, or services that provide Application Programming Interfaces (APIs). Such APIs can be utilized by different programming languages under different names to enable software integration. In the Java world for example, JMS is developed to provide an ultimate flexible service for exchanging different types of messages between collaborative Java-based software applications (Richards *et al.*, 2009). Heterogeneous Java-based software applications for instance can create a message exchange channel, create a queue as a message or data repository, send messages to the created queue, retrieve messages from the created queue, and many other operations all via invoking a set of JMS APIs orderly (Hsiao *el al.*, 2003). Presently, there are a number of brokers that support messaging using JMS to allow heterogeneous Java-based software applications to interact with each other. Each broker has its own set of features and specifications. However, performance is considered to be the key factor for collaborative software applications. Performance is very critical for software applications that rely on the speed of the software integration or collaboration process. Within this context, this paper experimentally compared two well-known and open-source messaging brokers that implement JMS specification: Apache ActiveMQ (Apache ActiveMQ, 2017) and Apache Apollo (Apache Apollo, 2017). Generally, there are many comparison factors that can be considered when comparing different messaging brokers. Throughput in terms of messaging capabilities (message sending and receiving) is the most common factor for comparing messaging brokers (Menth *et al.*, 2006). Thus, this study considered the throughput performance factor alongside many directions. However, the results of this study should help developers to select the broker that can meet the performance needs of collaborative software applications in a messaging environment.

The remainder of this paper is organized in many sections as follows. Section 2 presents the most related works available in literature. In Section 3, a set of test conditions, test scenarios, test metrics, and experimental setups used in this study have been provided. Section 4 presents the results of comparing the brokers with each other according to various test scenarios. Finally, some conclusions including a comparison summary table of both brokers and future works have been given in Section 5.

* Corresponding author

2. RELATED WORKS

This section briefly presents the most relevant studies and works that evaluate the performance of different messaging brokers via JMS. The authors in (Tran *et al.*, 2002) have evaluated the messaging capabilities of IBM MQSerios v5.2 messaging broker using different evaluation metrics. However, the study did not compare the broker with other messaging brokers. In a technical report by (Crimson Consulting Group, 2003), the brokers Sun Java Message System Queue v3.5 and IBM Websphere MQ v5.3 have been overviewed distinctly. Besides, they have been evaluated and compared with each other using different performance metrics and testing variables. In (Greenfield, 2004), the brokers IBM MQ Series and TIBCO's Rendezvous have been evaluated via different test scenarios. Also, the study shows the impact of using different Quality of Service (QoS) attributes on the overall performance of each broker. In (Ahuja & Mupparaju, 2014), the brokers Open MQ v4.1, Apache Active MQ v4.1, and Mantaray MQ v2.0.1 have been benchmarked, functionally compared, and qualitatively studied. In (Klein *et al.*, 2015), a comparative analysis of the brokers Apache ActiveMQ v5.10.0 and OpenMQ v4.5.2 has been conducted experimentally to determine the performance of each broker.

It is worth mentioning that the authors of this paper have faced a number of issues while studying the literature related to the topic of this study. For example, some papers in the literature have not presented distinctly how they conducted their test methodologies. No enough details were provided on how they measure the performance of a messaging broker or on how they compared experimentally two or more messaging brokers with each other. Many others have not provided the specifications of their software and hardware testing environments. Therefore, such papers have not been included in this section.

Nevertheless, all the studies included in this paper were useful in providing outstanding explanation of messaging systems, brokers' architectures, and Message Oriented Middleware (MOM) based applications. Besides, they were valuable in providing a general evaluation metrics for this study.

To the best of our knowledge, no previous study in the literature compared experimentally between Apache ActiveMQ and Apache Apollo open-source brokers in terms of messaging capabilities. Thus, this was the rationale behind this study to be conducted.

3. TEST METHODOLOGY

In this paper, test methodology represents the conditions, scenarios, metrics, and testbed setup that have been applied and used to compare the performance of the brokers in terms of messaging capabilities.

3.1 Test Conditions

All tests have been performed under the following conditions:

- Both brokers have been tested with their default configurations and settings.
- Every test scenario has been applied on both brokers with the same scenario related parameters (e.g. message size).
- Test applications (to send/receive messages to/from brokers) have been developed and executed on the same computer to ensure using the same software and hardware specifications.
- Before starting the test process and measurements, all user applications (excluding test applications) have been closed.
- The used computer has not been connected to the Internet during the test process and measurements.
- No message processing has been performed by the test applications. Thus, allowing sending and receiving messages as fast as possible in order to reach the highest level of messaging capabilities.
- Multithreading technique has been used to depict simultaneous test applications. Thus, each thread represents a single test application.
- Every test scenario has been repeated 5 times (10 minutes for each) and measurements have been averaged and rounded to ensure accuracy.
- All test results have been recorded after establishing client-broker connections, after creating messaging sessions, after creating messaging queues, after creating message sending and receiving objects, etc.
- The message size in each test scenario is the total length of header, properties, and body (payload). Since the size of the first two parts of each message is the same with all test scenarios, only the length of each message body was variable. Thus, three different message sizes have been used in each test scenario: 1 char (1 byte), 1000 char (1000 byte), and 10000 char (10000 byte).
- After finishing each test scenario (using a specific message size), the created queue (or queues) is deleted with its all messages and then the broker is restarted to ensure accuracy.

3.2 Test Scenarios

The messaging performance of both brokers is compared experimentally via six different test scenarios, as follows:

- **Scenario 1:** In this scenario, one test application continuously sends persistent text messages to a queue in the broker for a period of 10 minutes. Figure 1 depicts this scenario.

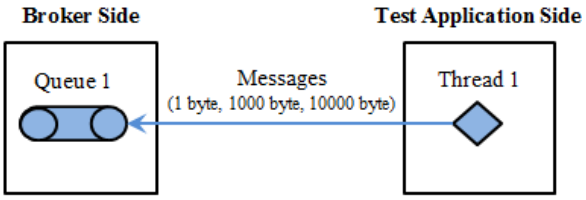

Figure 1. Representation of scenario 1

- **Scenario 2:** In this scenario, ten test applications continuously and simultaneously send persistent text messages to a queue in the broker for a period of 10 minutes. Figure 2 depicts this scenario.

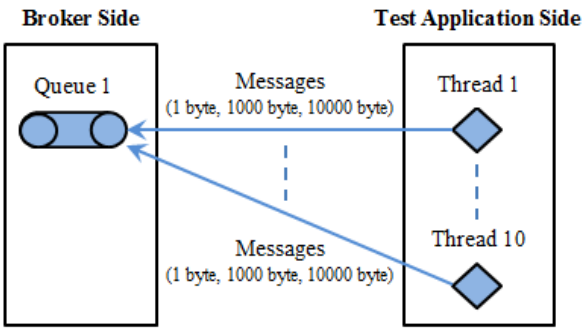

Figure 2. Representation of scenario 2

- **Scenario 3:** In this scenario, ten test applications continuously and simultaneously send persistent text

messages to ten queues in the broker for a period of 10 minutes. Figure 3 depicts this scenario.

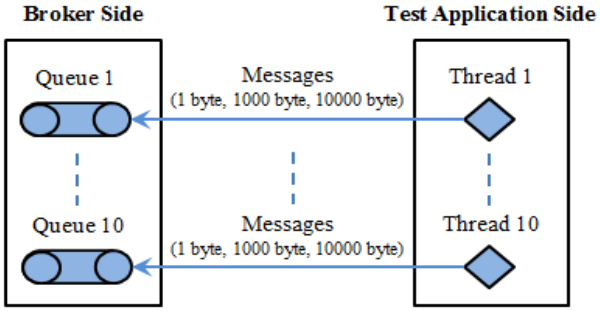

Figure 3. Representation of scenario 3

- **Scenario 4:** In this scenario, one test application continuously reads persistent text messages from a queue in the broker for a period of 10 minutes. Figure 4 depicts this scenario.

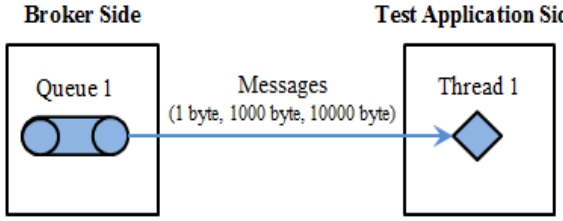

Figure 4. Representation of scenario 4

- **Scenario 5:** In this scenario, ten test applications continuously and simultaneously read persistent text messages from a queue in the broker for a period of 10 minutes. Figure 5 depicts this scenario.

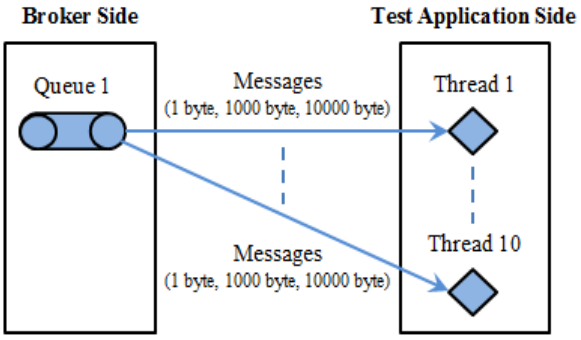

Figure 5. Representation of scenario 5

- **Scenario 6:** In this scenario, ten test applications continuously and simultaneously read persistent text messages from ten queues in the broker for a period of 10 minutes. Figure 6 depicts this scenario.

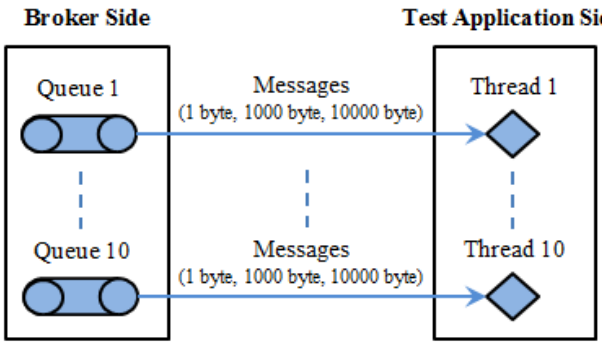

Figure 6. Representation of scenario 6

It is worth mentioning that the aforementioned test scenarios have been carefully chosen to cover different aspects of each broker's overall performance.

3.3 Test Metrics

Two dominant metrics have been used to evaluate and compare the performance of each broker, as follows:

- **Message sending (storing) throughput:** It represents the average number of messages that can be sent (stored) in a specific period of time. This metric has been measured in scenarios 1, 2 and 3 respectively.
- **Message receiving (retrieving) throughput:** It represents the average number of messages that can be retrieved in a specific period of time. This metric has been measured in scenarios 4, 5, and 6 respectively.

3.4 Testbed Setup

The test environment of this study has been setup with software and hardware which their specifications are presented in Table 1 and 2 respectively.

Table 1. Software specifications

	Software	Version
Test applications	Java JDK	1.8.0_91
	NetBeans IDE	8.2
JMS Brokers	Apache ActiveMQ	5.13.1
	Apache Apollo	1.7.1
JMS specification	JMS API	1.1
Web Browser	Mozilla Firefox	50.1.0
Operating System	Microsoft Windows	7 Home Basic (64-bit)

Table 2. Hardware specifications

	Hardware	Detail
Computer System	Laptop Model	ASUS K34S Series
	CPU Type	Intel Core i5-2450M
	CPU Speed	2.5 GHz
	CPU Cores	4
	RAM	6 GB
	Rating (Windows Experience Index)	4.5

4. EXPERIMENTAL RESULTS AND DISCUSSIONS

This section presents the obtained results of the experimental messaging performance analysis of the brokers via six test scenarios as shown in Figures 7-15.

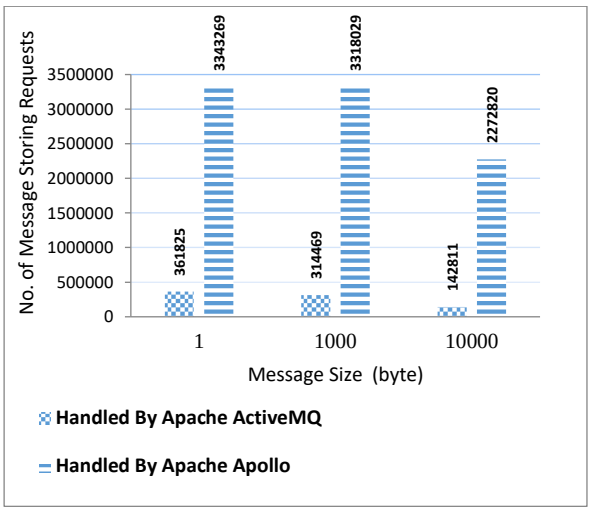

Figure 7. Scenario 1 results chart

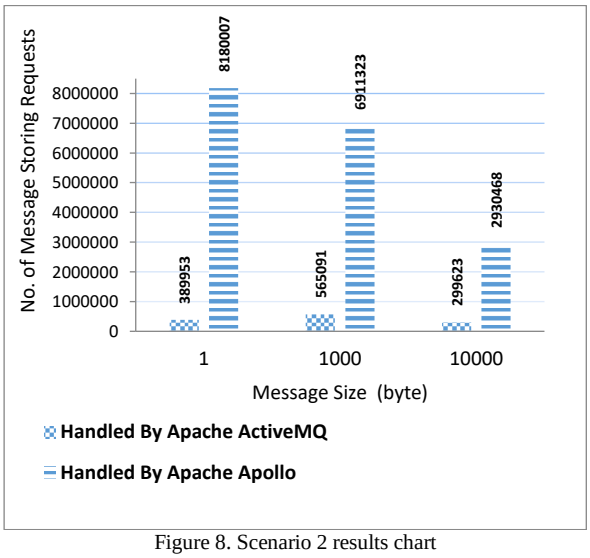

Figure 8. Scenario 2 results chart

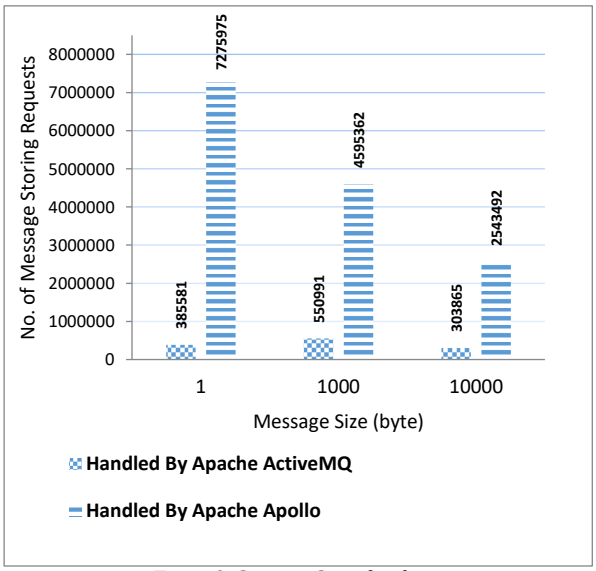

Figure 9. Scenario 3 results chart

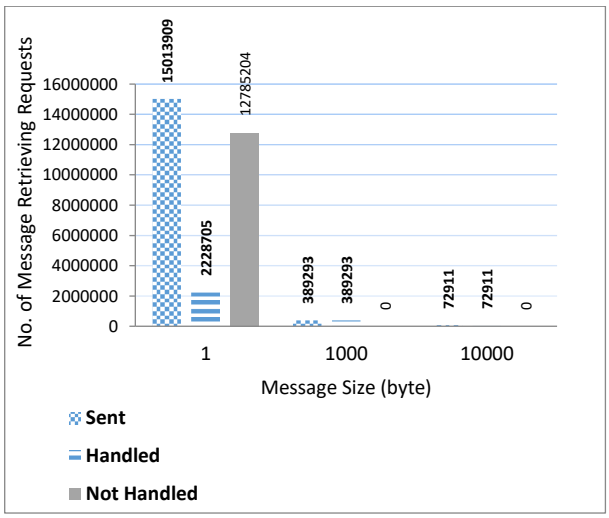

Figure 10. Scenario 4 (Apache ActiveMQ) results chart

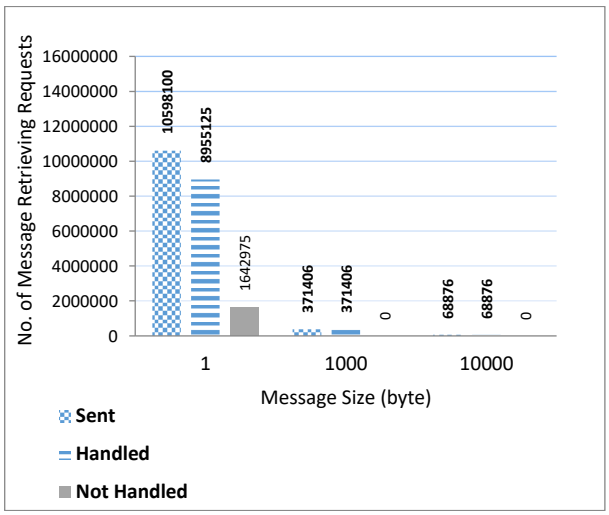

Figure 11. Scenario 4 (Apache Apollo) results chart

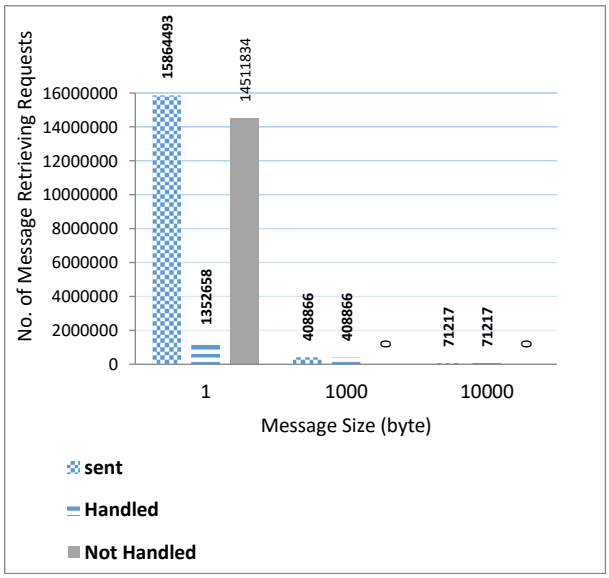

Figure 12. Scenario 5 (Apache ActiveMQ) results chart

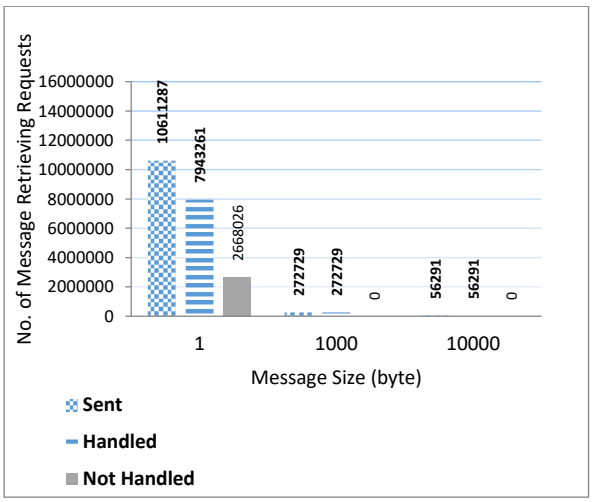

Figure 13. Scenario 5 (Apache Apollo) results chart

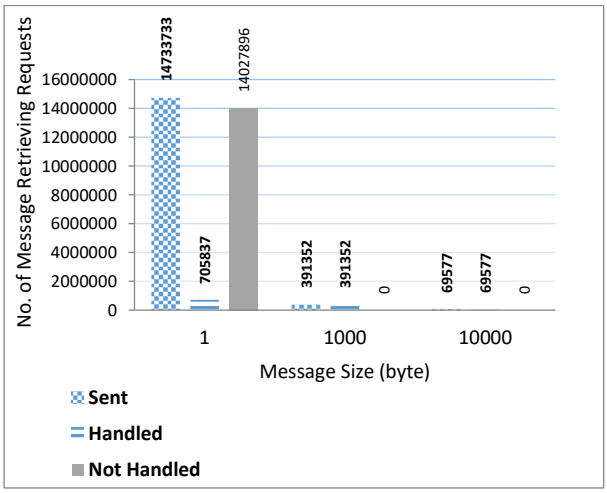

Figure 14. Scenario 6 (Apache ActiveMQ) results chart

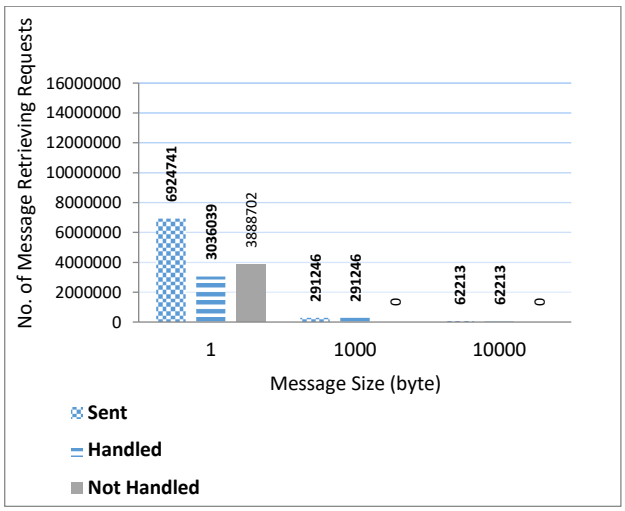

Figure 15. Scenario 6 (Apache Apollo) results chart

Figures 7-9 show that in the scenarios 1, 2, and 3; each broker handled all message sending (storing) requests sent to it across the entire message sizes. Thus, handling rate was 100% of each broker. But within the duration of each scenario, Apache Apollo handled more requests in all the aforementioned scenarios compared to Apache ActiveMQ.

Figures 10-15 show that the results of scenarios 4, 5, and 6 are classified into three different facets, as follows:

a- The total number of messages retrieving requests sent to each broker in order to be handled.
b- The number of messages retrieving requests that each broker handled successfully.
c- The number of messages retrieving requests that have not been handled by each broker, this number is simply calculated by subtracting (b) from (a).

Analyzing the results of scenarios 4, 5, and 6 shows that Apache Apollo handles small-size messages (1 byte) retrieving requests better than ActiveMQ. On the other hand, Apache ActiveMQ handles large-size messages (1000 byte and 10000 byte) retrieving requests better than Apache Apollo. Tables 3 and 4 summarize the results of all test scenarios used in this study.

Table 3. Apache ActiveMQ vs. Apache Apollo: Message Sending (Storing) Requests

Scenario	Message Size (byte)	Handling Message Sending (Storing) Requests		Superior Broker	Superior Rate
		Apache ActiveMQ	Apache Apollo		
Scenario 1	1	100%	100%	Apache Apollo	89%.
	1000	100%	100%	Apache Apollo	91%.
	10000	100%	100%	Apache Apollo	94%.
Scenario 2	1	100%	100%	Apache Apollo	95%.
	1000	100%	100%	Apache Apollo	92%.
	10000	100%	100%	Apache Apollo	90%.
Scenario 3	1	100%	100%	Apache Apollo	95%.
	1000	100%	100%	Apache Apollo	88%.
	10000	100%	100%	Apache Apollo	88%.

Table 4. Apache ActiveMQ vs. Apache Apollo: Message Retrieving Requests

Scenario	Message Size (byte)	Handling Message Sending (Storing) Requests		Superior Broker	Superior Rate
		Apache ActiveMQ	Apache Apollo		
Scenario 1	1	100%	100%	Apache Apollo	89%.
	1000	100%	100%	Apache Apollo	91%.
	10000	100%	100%	Apache Apollo	94%.
Scenario 2	1	100%	100%	Apache Apollo	95%.
	1000	100%	100%	Apache Apollo	92%.
	10000	100%	100%	Apache Apollo	90%.
Scenario 3	1	100%	100%	Apache Apollo	95%.
	1000	100%	100%	Apache Apollo	88%.
	10000	100%	100%	Apache Apollo	88%.

5. CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

This paper presented an experimental approach to evaluate the performance of two leading open-source JMS messaging brokers, namely Apache ActiveMQ and Apache Apollo in terms of one-to-one messaging capabilities. Six experimental test scenarios have been conducted to achieve the aforementioned goal. Overall performance evaluation and analysis showed that Apache Apollo outperformed Apache ActiveMQ in test scenarios 1, 2, and 3 across the entire different message sizes. Also, Apache Apollo outperformed Apache ActiveMQ in test scenarios 4, 5, and 6 with message size of 1 byte. On the other hand, Apache ActiveMQ outperformed Apache Apollo only in test scenarios 4, 5, and 6 with message sizes of 1000 and 10000 byte. Table 3 and 4 present the summary of this study. Moreover, in this paper a well-defined test methodology has been proposed to be used to measure the performance of other messaging brokers. This is crucial to help developers to select a broker with an acceptable level of messaging capabilities in a distributed environment.

5.2 Future Works

Some possible future works are listed below:

- Applying the evaluation approach and test scenarios used in this paper to evaluate the same brokers with each other but via publish-subscribe messaging capabilities.
- Measuring the impact of changing message properties (e.g. QoS, filters, etc.) along message body on the overall performance of each broker.
- Sending and receiving messages using different techniques and protocols (e.g. MQTT, AMQP, STOMP, etc.) would be a good choice for further performance evaluation of each broker.

REFERENCES

Ahuja, S. P., & Mupparaju, N. (2014). Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware. *Computer and Information Science*, 7(4), 9-20.

Crimson Consulting Group (2003). High-Performance JMS Messaging: A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ. Los Altos, CA.

Greenfield, P. (2004). QoS evaluation of JMS: an empirical approach. *Proceedings of the 37th Annual Hawaii International Conference System Sciences*, 1-10.

He, W. & Xu, L. D. (2014). Integration of Distributed Enterprise Applications: A Survey. *IEEE Transactions on Industrial Informatics*, 10(1), 35-42.

Hohpe, G., & Woolf, B. (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. *Addison-Wesley Professional*, 1st edition.

Hsiao, T.-Y., Cheng, M.-C., Chiao, H.-T., & Yuan, S.-M. (2003). FJM: a High Performance Java Message Library. *IEEE International Conference on Cluster Computing (ICCC)*, 460-463.

Klein, A. F., ŞtefĂnescu, M., Saied, A., & Swakhoven, K. (2015). An experimental comparison of ActiveMQ and OpenMQ brokers in asynchronous cloud environment. *the 5th International Conference on Digital Information Processing and Communications (ICDIPC)*, 24-30.

Menth, M., Henjes, R., Gehrsitz, S., & Zepfel, C. (2006).Throughput Comparison of Professional JMS Servers. University of Wurzburg, Institute of Computer Science, Research Report Series, Report No. 380, 1-23.

Richards, M., Monson-Haefel, R., & Chappell, D. A. (2009). Java Message Service. O'Reilly, 2nd edition.

The Apache Software Foundation, Apache ActiveMQ web link: http://activemq.apache.org/, accessed 10/05/2017.

The Apache Software Foundation, Apache Apollo web link: http://activemq.apache.org/apollo/, accessed 10/05/2017.

Tran, P., Greenfield, P., & Gorton, I. (2002). Behavior and performance of message-oriented middleware systems. *Proceedings of the International Conference on Distributed Computing Systems*, 645-650.