Performance Optimization Methods of MQ systems

Vishal Anand, Vinay M. Rao, Satheesh Babu K.

Abstract— MQ systems are messaging and queuing middleware, with point-to-point, publish/subscribe, and file transfer modes of operation. Applications publishes the messages to many subscribers over queues to be consumed by other system component.MQ messaging system performance issue would have a cascading effect into all aspects of an enterprise application. It will impact the turnaround time for transactions which goes through the MQ systems, consequently degrading the response time or throughput for the end users. It is critical to optimize the MQ systems to scale up the application or enhancing the response time.

Experiment-based performance optimizing approaches have been introduced as an alternative to prediction-based performance optimizing techniques. A representative system (testing environment) like the production environment allows a MQ system administrator to accurately sense the performance gain/loss for a tuning task. In this paper all the important parameters of the MQ systems were explained and designates a novel approach to experiment-based performance optimization techniques.

A proof-of-concept implementation was shown to achieve the optimal results implying the configuration changes one by one and creating the baseline for each performance changes that achieved the optimal results. It is advised to test another configuration change on the baseline achieved after first set of configuration change.

It is very important to monitor the hardware resources for every test to find a trade-off between increased hardware usage to the performance enhancement of the application. Depending on the system resource utilizations and results achieved after the tuning efforts, combined advantageous parameter of the MQ system would be preferred over other as shown in the tests.

Index Terms- Acknowledgement Mode, Benchmarking, Baselining, Performance Enhancement, Resource utilization, Tradeoff

1 INTRODUCTION

MQ system is one of the main components of any enterprise application. Performance problems in the MQ systems will have ripple effect in every aspect of an application.MQ system vendors always provide the standard guidelines and configuration settings to have the good performance, but it is the responsibility of the MQ administrator and Performance Engineer to fine tune the system to get the better performance with any given application [1]. A fined tuned MQ system would help to achieve the service level agreements (SLA) of an application. It's always good to have testing environment which should be close replica of the production environment because tuning work carried out on the system could be invalidated if testing and production environment are not same.

There are other factors that pose much more subtle performance problems which include upgrades for the MQ system software, patches for the change in configuration parameters (i.e. depreciated
configuration settings or change in a default value set at installation), change in workloads and business requirements. It is wise to check the available upgrades and patching for the MQ systems first before proceeding to the tuning effort.

All the important parameters of the MQ systems were discussed in detail to understand and use it in tuning effort. Every tuning change should be implemented singularly to gauge the impact of the optimization of the MQ system and all the load test results along with the resource utilization of the system should be carefully analyzed to arrive on the optimal results.

An adaptive resource allocation method is proposed to trade-off between resource utilization and performance (response time and throughput) of the application. Performance metrics are compared with the baseline and results obtained after configuration changes. A trade-off is decided between increment in hardware usage to the enhanced performance of the application.

2 THE PERFORMANCE TUNING PROCESS:

Performance of a messaging queues (MQ) application depends on the interaction between the application and the MQ message service. Therefore, maximizing performance requires the combined efforts of both the performance Engineer and MQ administrative skills. The process of optimizing performance begins with application design and tuning of the message service configurations. Performance optimization process includes the following stages [2]:

1. Defining the performance requirements of an application.
2. Designing the application after considering the factors that affect performance (especially trade-offs between reliability and performance)
3. Establishing baseline performance measures
4. Tuning or reconfiguring the MQ message service to optimize performance.

2.1 Aspects of Performance Engineering

Performance is a measure of the speed and efficiency with which a message service delivers messages from one end (Generator) to another end (consumer). There are several different aspects of performance that could be important for optimization depending on the requirement [2,3]. There are always trade-offs between reliability and performance of an application, so these different aspects of performance are interrelated. If message throughput of application is high which means that messages are less likely to be backlogged in the message server hence latency should be low (a single message can be delivered very quickly). However, latency depends on many factors, the speed of communication links, message server processing speed, and client processing speed.

2.2 Benchmarking and Baselining Usage Patterns:

Benchmarking is the primary method of measuring the performance of an application. Benchmarking refers to running a set of representative tests on different configurations and types of hardware machine and measuring the results (message throughput, Latency and Stability). Benchmark results are used to evaluate the performance of a given system on a well-defined workload. Benchmarks are developed to pinpoint performance problems of new systems.

Test environment is created with the Generating and Consuming clients which will have same number of connections, sessions, and message producers, sending the persistent or non-persistent messages of a standard size to the number of queues that will consume the messages in the test environment's destinations (Depending on messaging application design) at specified rate which will measure the time taken between generation and consumption of messages or the average message throughput rate, and monitor the system to observe connection thread usage, message storage data, message flow data, and other relevant metrics [4]. Rate of message generation are ramped up until performance is negatively impacted. The maximum throughput that can be achieved becomes baseline for the messaging service application.

Few characteristics of test environment can be modified using the benchmark and note all the changes to analyses the results. For example, note the impact on performance after increasing the number of connections or the size of messages five-fold or ten-fold, changing the broker configuration, change connection properties, thread pool properties, JVM memory limits, limit behaviors, built-in versus plugged-in persistence. Benchmarks should be run in a controlled test environment for some time until message service gets stabilized (Performance is negatively impacted at startup by the Just-In-Time compilation that turns Java code into machine code).

It is important to establish baseline usage patterns once a messaging application is deployed and to establish base-line usage pattern message server need to be monitored over an extended period. Build the average and peak values looking at production data such as number of connections, number of messages stored in the broker, message flows into and out of a broker, numbers of active consumers.

Baseline metrics should be checked against design expectations like checking that connections are not left open or consumed messages are not being left unacknowledged. These coding errors consume message server resources and could significantly affect performance. Baseline usage pattern helps in tuning the MQ system for the optimal performance.

2.3 Factors That Impact Performance of MQ Systems:

Message latency and message throughput, two of the main performance indicators, generally depend on the time it takes a typical message to complete various steps in the message delivery process [5]. These steps are shown below for the case of a persistent, reliably delivered message. The steps are described as given below.

1. The message is delivered from generation point to message server
2. The message server reads in the message delivered by Generator.
3. The message is placed in persistent storage.
4. The message server confirms receipt of the message.
5. The message server determines the routing for the message
6. The message server writes out the message
7. The message is delivered from message server to consuming client
8. The consuming client acknowledges receipt of the message.
9. The message server processes client acknowledgement.
10. The message server confirms that client acknowledgement has been processed.

All these steps are sequential and each of them could be potential bottleneck in the delivery of messages from generating clients to consuming clients [2,6]. These steps depend upon physical properties of the messaging system such as network bandwidth, computer processing speeds, message server architecture.

The following subsections discuss the impact of both application design factors and messaging system factors on performance. While application design and messaging system factors closely interact in the delivery of messages therefore each category is considered separately.

2.4 Application Design Factors that Impact Performance:

Application design decisions can have a significant effect on overall messaging performance. The most important factors affecting performance are those that impact the reliability of message delivery and these are the following factors:

1. Delivery Mode (Persistent/Non-persistent Messages)
2. Use of Transactions
3. Acknowledgement Mode
4. Durable vs. Non-Durable Subscriptions
5. Use of Selectors (Message Filtering)
6. Message Size

7. Message Body Type

There is always trade-off between performance and reliability and factors that increase reliability tend to decrease performance. The following table shows how the various application design factors generally affect messaging performance. The table shows two scenarios—a high reliability, low performance scenario and a high performance, low reliability scenario—and the choice of application design factors that characterizes each. Between these extremes, there are many choices and trade-offs that affect both reliability and performance.

2.5 *Delivery Mode (Persistent/Non-persistent Messages)*

Persistent messages guarantee message delivery in case of message server failure. The broker stores the message in a persistent store until all intended consumers acknowledge they have consumed the message. Broker processing of persistent messages is slower than for non-persistent messages for the following reasons [7]:

1. A broker must reliably store a persistent message so that it will not be lost should the broker fail.
2. The broker must confirm receipt of each persistent message it receives. Delivery to the broker is guaranteed once the method producing the message returns without an exception.
3. Depending on the client acknowledgment mode, the broker might need to confirm a consuming client's acknowledgement of a persistent message.

2.6 Use of Transactions:

A transaction is a guarantee that all messages produced in a transacted session and all messages consumed in a transacted session will be either processed or not processed (rolled back) as a unit [8]. A message produced or acknowledged in a transacted session is slower than in a non-transacted session for the following reasons:

Acknowledgement Mode:

There is mechanism for ensuring the reliability of JMS message delivery is for a client to acknowledge consumption of messages delivered to it by the MQ message server. If a session is closed without the client acknowledging the message or if the MQ message server fails before the acknowledgment is processed, the broker redelivers that message, setting a JMSRedelivered flag and client can choose one of three acknowledgement modes [2,9].

1. AUTO_ACKNOWLEDGE. The system automatically acknowledges a message once the consumer has processed it. This mode guarantees at most one redelivered message after a provider failure.
2. CLIENT_ACKNOWLEDGE. The application controls the point at which messages are acknowledged. All messages processed in that session since the previous acknowledgement are acknowledged. If the message server fails while processing a set of acknowledgments, one or more messages in that group might be redelivered.
3. DUPS_OK_ACKNOWLEDGE. This mode instructs the system to acknowledge messages in a lazy manner. Multiple messages can be redelivered after a provider failure.

2.7 Durable vs. Non-Durable Subscriptions:

Subscribers to a topic destination fall into two categories, those with durable and non-durable subscriptions and Durable subscriptions provide increased reliability at the cost of slower throughput for the following reasons:

- The MQ message server must persistently store the list of messages assigned to each durable subscription so that should a message server fail, the list is available after recovery.
- Persistent messages for durable subscriptions are stored persistently, so that should a message server fail, the messages can still be delivered after recovery, when the corresponding consumer becomes active [10]. By contrast, persistent messages for non-durable subscriptions are not stored persistently (should a message server fail, the corresponding consumer connection is lost, and the message would never be delivered).

Use of Selectors

Specific consumers are selected to target set of messages for the optimal performance. Group of messages are targeted to unique destination or choosing single destination. Selectors are having the unique string of character that matches to the string configured in the consumer e.g. (selector NumberOfOrders > 1 delivers only messages with a NumberOfOrders property value of 2 or more) [2,11]. Registering consumers with selectors lowers performance (as compared to using multiple destinations) because additional processing is required to handle each message.

Message Size

Message size affects performance of MQ channel accordance with the amount of data being parsed from generating client to broker and from broker to consuming client. However, by batching smaller messages into a single message, the routing and processing of individual messages can be minimized, providing an overall performance gain [2].

Hardware

CPU processing speed and available memory are primary determinants of message service performance. Software shortcoming can be limited by increasing hardware resources however, it is generally expensive to overcome bottlenecks by upgrading the hardware [2].

Java Virtual Machine (JVM)

The MQ message server is a Java process that runs on the host

JVM thus JVM processing is an important determinant of how fast and efficiently a message server can route and deliver messages [2].

Connections

The number and speed of connections between client and broker can affect the number of messages that a message server can handle as well as the speed of message delivery.

Message Server Connection Limits

All access to the message server is by way of connections and any limit on the number of concurrent connections can affect the number of generating or consuming clients that can concurrently use the message server. The number of connections to a message server is generally limited by the number of threads available. MQ uses a thread pool manager which can be configured to support either dedicated or shared thread model [7]. The dedicated thread model is fast because each connection has dedicated threads, however the number of threads limits the number of connections available (one input thread and one output thread for each connection). The shared thread model places no limit on the number of connections, however there is significant overhead [2,8].

3 Performance Enhancement of MQ Systems:

There is no universal rule to optimize the application which will be applicable on every system. Experiment-based performance optimizing approaches is the solution where series of load test to be executed with the different configuration set up and evaluate the result to arrive on the optimal performance of the systems. The best strategy for the optimization is to tweak one configuration at a time (e.g. Number of connection in our case) keeping all other setting constant to observe the measurable improvement or degradation of performance. In this case study, JMS application (WebSphere MQ V5.0) over WebSphere Application Server V6.0 was used and hereafter JMS application (WebSphere MQ V5.0) will be referred as MQ system.

Number of load tests were executed to benchmark the performance of the MQ systems with the mutually exclusive configuration. There is always been trade-off between the performance and resources utilization (CPU, Memory and Disk) of the system therefore approach is to increase the throughput of the system until resource utilization of the systems are under limit [9].

In this case study, MQ systems was having the dedicated thread pool modal therefore maximum number of connections is always maintained half of the maximum number of threads in the thread pool. Optimal results (in terms of message throughput, Response time and Pass/fail count) were produced at 90 concurrent clients threads for the JMS application (WebSphere MQ V5.0) over WebSphere Application Server V6.0.

load test with one-hour steady state were executed with the five message sizes (50 KB, 100 KB,250 KB, 500 KB, 1MB and 2 MB). Transaction per hour of the messages request were achieved as per requirement of the application with the initial configuration of 30 connections. Messaging application had processed all the message with in the SLA of response time however test was to identify the best result with the tuned parameters without severely impacting the resource utilization.

There was huge increment of 42.05% in the message servicing throughput when number of connection has increased from 30 to 60. It had incremented to 12.46% and 7.86% when the connection has increased to 75 and 90 respectively. Resource utilization were also monitored for the test period as shown in the graph. Average CPU utilizations of MQ system were 80% and 88% when the number of connection was 75 and 90 respectively. There was no breakdown of the system and throughput of messaging service when the connections were increased at 90 connections, but CPU utilization was maxed to 88% therefore we have considered the safe and best throughput of MQ systems when the number of connection was 75 since CPU utilization (88%) on MQ system is breaching the SLA for an application.

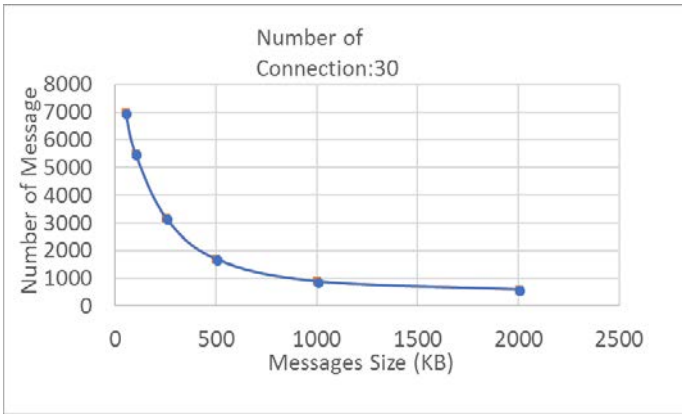
Now this configuration (number of connection: 75) became baseline for the further tuning of the application and same strategy implemented to get another optimal result with the next set of tuning which will be new baseline for the next set of tuning. This tuning cycle goes on until application achieves the desired results.

04 load tests were conducted with the varying number of connections, thread pool connections and mixed sizes of the message body (which was same across all the tests).

3.1 Experiment 01 (*Load Test 01):*

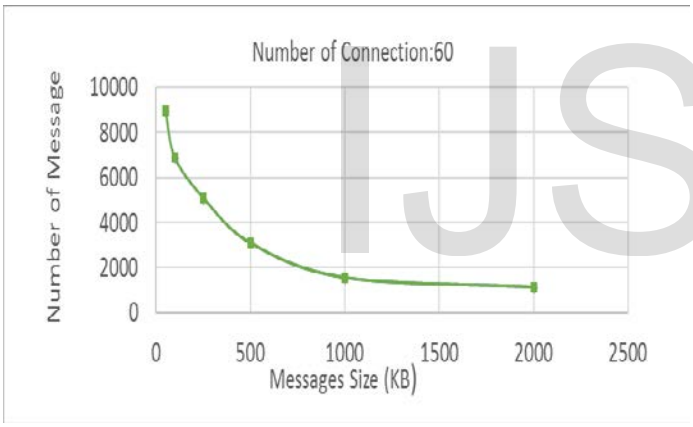
6 Message body sizes (50, 100, 250, 500, 1000 and 2000 KB) were used in combinations with 30 number of connections and 60 number of thread pool size. Response time were under target and there were no failures observed during the test. Number of processed message is given below in summarized sheet for all the load tests.

Message Size (KB)	Number of Connections =30	Number of Connections= 60	Number of Connections=75	Number of Connections =90
	Processed Messages/hour	Processed Messages/ Hour	Processed Messages/ Hour	Processed Messages /Hour
50	7000	9000	9613	10145
100	5500	6900	7243	7800
250	3200	5120	6213	6700
500	1700	3111	3980	4312
1000	900	1567	1980	2312
2000	600	1145	1165	1245



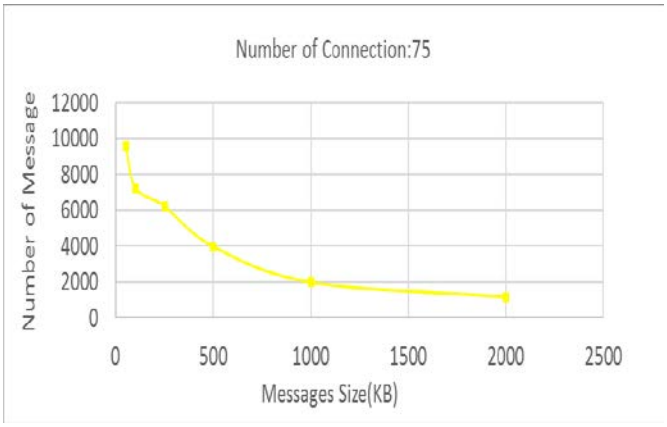
3.2 Experiment 02 (Load Test 02):

6 Message sizes (50, 100, 250, 500, 1000 and 2000 KB) were used with 60 number of connections and 120 number of thread pool size. Response time were under target and there were no failures observed during the test.



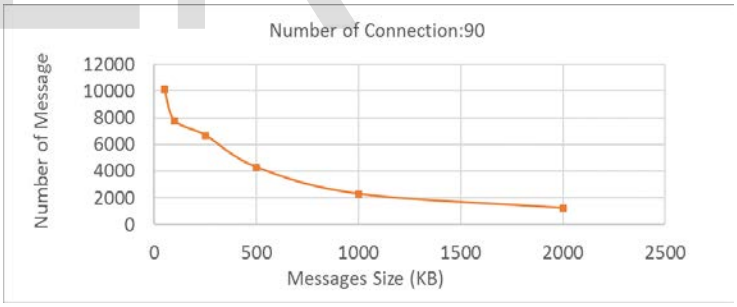
3.3 Experiment 03 (Load Test 03):

6 Message sizes (50,100,250,500,1000 and 2000 KB) were used with 75 number of connections and 150 number of thread pool size. Response time were under target and there were no failures observed during the test.



3.4 Experiment 04 (Load Test 04):

6 Message sizes (50,100,250,500,1000 and 2000 KB) were used with 90 number of connections and 180 maximum number of thread pool size. Response time were under target and no failures were observed during the test. Maximum message throughput was achieved with the settings with the given settings.

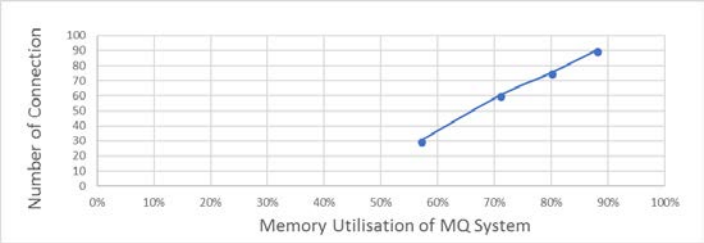


4.0 Resource Utilization:

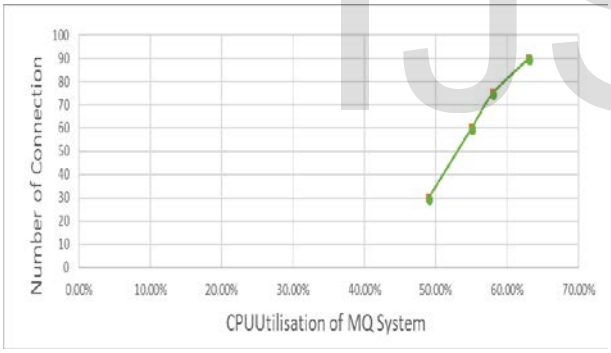
Resource utilization of the system had increased with every increment in the number of connections. CPU utilization has reached to 88% when number of connections incremented to the 90 which was SLA breach for the application therefore we have considered the number of connection (75) appropriate for the system. Other performance metrics (Memory and Disk) were not concern even at 90 as shown in the below graph.

Number of Connection	CPU Utilization	Memory Utilization
30	57%	49.00%
60	71%	55.00%
75	80%	58.00%
90	88%	63.00%

4.1 MEMORY UTILISATION:



4.2 CPU UTILISATION:



5.0 *CONCLUSION:*

Message queues are the way to connect different systems of the applications. It is robust way of sending messages back and forth. MQ systems are immune to failures and provides the guaranteed reliability, error reporting and security.MQ systems are comparatively slow to its peer technology. It is always recommended to optimize the MQ systems to enhance the end user experience and saving the huge resources of the company. Various components of the MQ system and their role in the performance of the application was explained above and detailed approach was outlined to tune the application. Paper has described the tuning steps which recommends of doing one configuration update at a time.

Tuning effort has increased the system throughput to 72% (by changing the Connection and Thread pool setting) However resource Utilization of the system were also increased while changing the configurations. Resource Utilization should be carefully monitored to decide the tradeoff. It is suggested to execute the load tests until it crosses the SLA (Response Time, Pass/Fail Count or Resources Utilization of the systems) of the application. Later decide the best configuration settings among tested settings. Best results were achieved with the following configurations (Connection 75 and Thread Pool Size 150) after establishing the tradeoff.

REFERENCE:

1. Brian S Paskin et al, Performance Monitoring and Best Practices for WebSphere on z/OS.
2. https://docs.oracle.com/cd/E19909-01/817-3727/tuning.html
3. https://www-01.ibm.com/software/integration/wmq/wmqlibrary/
4. http://webcache.googleusercontent.com/search?q=cache:vQfE1yjZoSwJ:www.theedison.com/pdf/Edison_IBM_WebSphere_MQ_vs_ActiveMQ_White_Paper
5. https://labs.mwrinfosecurity.com/assets/141/original/mwri_websphere-mq-security-white-paper-part1_2008-05-06.pdf
6. https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-ibm-mq-security-presentation-defcon15-2007-08-03.pdf
7. https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pro.doc/q002620_.htm
8. Ebay (June 21, 2013). Ebay. http://ebay.com/.
9. Ebay Tech Blog (June 21, 2013). Cloud Bursting for Fun and Profit. http://ebaytechblog.com/2011/03/28/cloud-bursting-for-fun-and-profit/.
10. https://webspherecompetition.wordpress.com/2014/04/02/websphere-mq-7-5-and-apache-activemq-5-9-performance-comparison-results-part-3/
11. https://docs.oracle.com/cd/E14004_01/books/PerformTun/PerformTunEAI4.html