Behavior and Performance of Message-Oriented Middleware Systems

Phong Tran, Paul Greenfield
CSIRO Mathematical and Information Sciences,
North Ryde, Sydney, Australia
{phong.tran, paul.greenfield}@csiro.au

Ian Gorton
School of Information Technologies, University of Sydney
Sydney Australia

Abstract

The middleware technology used as the foundation of Internet-enabled enterprise systems is becoming increasingly complex. In addition, the various technologies offer a number of standard architectures that can be used by designers as templates to build applications. However, there is little concrete understanding in the software industry on the strengths and weaknesses of competing technologies, and the different trade-offs that various component architectures impose. The SACT Group at CSIRO has qualitatively and quantitatively evaluated a number of commercially available Message-Oriented Middleware (MOM) systems. This paper focuses on the results obtained from the performance evaluation of the IBM's MQSeries V5.2. It presents an overview of the technology, and discusses the metric used in this study for performance measurement The test results related to the sustainable performance of the system using various test configurations are described and their implications discussed.

Keywords: *Distributed system, message-oriented middleware, MQSeries, performance evaluation*

1. Introduction

There are a number of strongly competing products in the Message-Oriented Middleware (MOM) technology market place. They all appear at first sight to offer much the same set of core features. In reality, these products are actually quite different, aiming at meeting the diverse needs of different core markets. Not surprisingly, they have their own strengths and weaknesses. These become crucial when an IT department is deciding which product will best suit its particular needs. In particular, MOM technology performance is often a key differentiator during the acquisition process.

A literature survey was carried out to determine the current status of research on the performance of the most recent releases of various commercially available MOM products and how the system performance was measured. The survey result shows that the only papers related to the performance measurement of MOM systems are from the products' vendors. Two performance studies have been published recently comparing MQSeries and MSMQ, one by IBM [1] and the other commissioned by Microsoft [2]. The studies have different test and measurement methods and the results are essentially conflicting.

In response to these issues, the SACT research group in CSIRO, an independent research organization in Australia, initiated the Middleware Technology Evaluation (MTE) project. The major aim of this project was to rigorously and independently evaluate competing middleware products and disseminate the results to industry through publications and seminars. Previous work in this project includes studies on CORBA and EJB-based technologies [3][4][5]. The results of the work are reported in a series of comprehensive reports that are available via the Cutter Consortium[1] and CSIRO publishing[2].

As part of the MTE project, a research program on the performance evaluation of MOM systems has been undertaken. Three of the most widely used MOM products, namely IBM's MQSeries v5.2, Microsoft's MSMQ v2.0 and TIBCO Rendezvous v6.6 were selected. While the comprehensive report on MOM technology [6] presents a complete picture and the performance results of each of the MOM products evaluated, this paper limits itself to a study on the behavior of the MQSeries V5.2.

2. Issues on performance measurement

A key aspect of the performance measurement of a MOM system is the establishment of a scenario that best reflects "real world" MOM-based applications. Based on

[1] www.cutter.com
[2] www.cmis.csiro.au/adsat/mte_reports.htm

Proceedings of the 22 nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02)
0-7695-1588-6/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

this scenario, a metric for performance comparison can be established.

The studies by IBM [1] and Microsoft [2] are based on different usage scenarios and reach contradictory conclusions. Apart from presenting the results, they reveal very little about the behaviors of the systems. The study by IBM is based on a closed-loop test scenario. In this scenario, a sender and a receiver are synchronized with the MQSeries transmission agent, such that the processing of a message between these processes is serialized. In other words, the sender puts a message on the queue and waits for it to be returned. The sender only sends the next message when it receives a reply to the previously sent message.

This scenario is not ideal for most MOM applications where message senders and receivers are asynchronous. In many practical scenarios, messages are sent and forgotten. In an asynchronous scenario, senders, receivers and queue managers are concurrently active and can push the application beyond its capacity, resulting in unsustainable performance.

In this study, a different scenario has been used. The end result of this study is the establishment of a new metric that is then used for performance measurement.

3. Overview of MQSeries

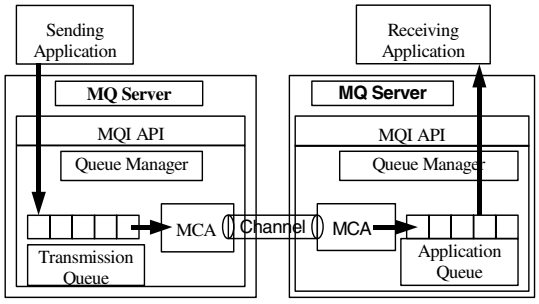

Figure 1: MQSeries architecture

IBM's MQSeries [7] deals with queued messages. Figure 1 depicts the architecture of the system. Applications write messages to queues, where they can be sent to other queues and read by their receiving applications. MQSeries comes in both client and server forms. An MQSeries server contains all MQSeries components, including MQSeries Explorer, MQSeries Services and Queue Managers (QMs). It can provide messaging services to local applications and remote MQSeries clients, as well as exchanging messages with other MQSeries servers. An MQSeries client does not have its own queue managers. It relays messaging calls from applications to MQSeries servers.

MQSeries servers communicate with each other over message channels. These consist of a transmission queue and two Message Control Agents (MCAs) connected by a network link. The MCAs are responsible for transmitting messages between queue managers.

MQSeries provides direct access to a large number of parameters that an administrator can use to tune its behavior and optimize performance for a particular environment and workload. The default configuration settings include standard binding for non-persistent messages and a 64KB log buffer with three 1024KB log files for persistent messages. Alternative to the standard binding is *FastPath* binding for non-persistent messages. The *FastPath* binding links the queue manager into the application, reducing overheads and improving performance. The maximum log buffer is 2 MB.

4. Test scenario

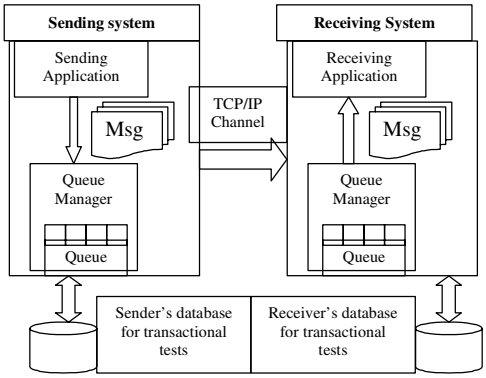

Figure 2: Open-loop test scenario

Figure 2 depicts a scenario involving a multi-threaded sending application (sender), a multi-threaded receiving application (receiver) and two queue managers. As these components operate asynchronously, the sender and receiver are not synchronized. As depicted in Figure 2, the sender puts messages into a local queue and lets its queue manager transmit them to a remote queue. The receiver extracts them without replying to the sender.

This is regarded as an open-loop scenario. In this scenario, these components are independent processes and as such they can handle messages at different rates. The difference between the transmission rate and the sending rate determines the queue depth at the sender's queue. Similarly the difference between the transmission rate and the receiving rate determines the queue depth at the receiver's queue.

5. Test platform

The sender and its supporting middleware are run on a single Dell PC, connected over a 100Mb/s switched LAN

IEEE COMPUTER SOCIETY

to another equivalent Dell PC running the receiver. The configuration of these test machines is shown in Table 1.

Computer	Sender	Receiver
CPU	1x 733MHz Pentium III	2x 450MHz Pentium II
RAM	0.5 GB	0.5 GB
Disk Type	1x IBM Deskstar 75GXP disk 7200 rpm, 8.5ms avg seek UltraATA/100	1x Seagate ST118202LW disk 10,000 rpm, 6.2ms avg seek Ultra2 Wide SCSI
Operating System	Microsoft Windows 2000 Server SP2	Microsoft Windows 2000 Server SP2
Database	SQL Server V7.0	SQL Server V7.0

Table 1: Test platform

6. Behavior of the system

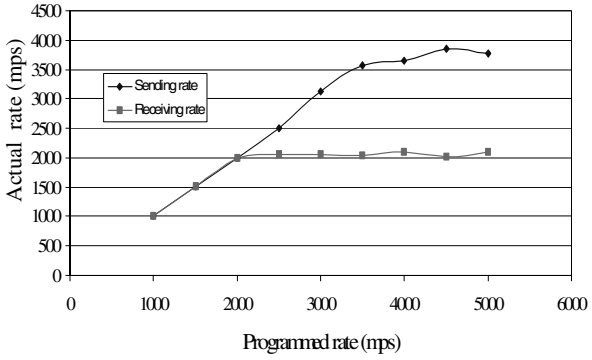

Figure 3: Recorded rates vs. programmed rate

An initial experiment with the MQSeries was carried out to study the behavior of the system. Figure 3 shows a typical behavior of the system configured with its default configuration parameters. In this test, when the sender was sending non-persistent 64-byte messages at a rate below 2000 messages per second (mps), the receiver's receiving rate was almost the same as the sender's sending rate. While the sending rate continued to increase beyond 2000 mps, the receiver saturated at about 2000 mps, causing the receiving rate to level off. We call this rate the saturation point. Messages are now being queued in the sender's transmission queue while the queue at the receiver remained empty. This means the QM's transmission rate has reached its maximum and is the same as the receiver's receiving rate.

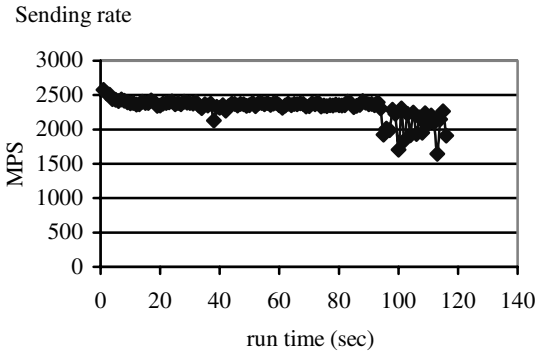

Figure 4: Sending rate vs. run time

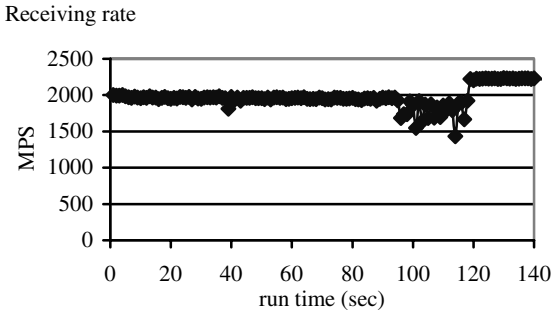

Figure 5: Receiving rate vs. run time

Figures 4 and 5 show another behavior of the system using the same message properties (non-persistent 64-byte length). In this case the sending rate was set beyond the saturation point at 2400 mps. When the sender was sending messages at this rate, the receiver was initially receiving messages at the rate of 2000 mps. These rates remained stable for the first 100 seconds of run time. During this time the queue depth at the sender's queue increased continually at a relatively linear rate with unsent messages while the queue depth at the receiver's queue was almost zero.

After the 100-second mark, the sending rate fluctuated sharply and at the same time dropped (Fig. 4), causing a ripple effect on the receiving rate (Fig. 5). The system at this stage was in the unstable state. At this time, the queue contained about 40000 messages. When the sender process stopped at the 120-second mark (Fig. 4), no more new messages were put into the queue. At this time the sender's queue manager and the receiver continued to process the unsent messages, and the receiving rate increased sharply, becoming steady at 2200 mps (Fig. 5).

This behavior shows that the sender's queue manager cannot sustain the same level of performance when its queue is allowed to grow beyond a certain queue depth. The behavior also shows that the queue manager's transmission capacity is constrained by the message queuing processing at the queue. In other words, when the

IEEE COMPUTER SOCIETY

sender and queue manager are concurrently active, the transmission rate is lower than when only the queue manager is active. The explanation for this behavior is as follows:

When the queue is empty, there is no disk I/O activity. When the queue starts to grow, its queue manager starts to fill up the queue's buffer. When the buffer is full, the queue manager flushes those in the buffer to a disk file. This condition persists while the sender is running.

As the messages have the same delivery priority, under this condition, the queue manager now has to read and send the old messages from the file first while continuing to enqueue new messages and flush those in the buffer to the same disk file. This causes a high level of disk I/O activity and a high level of CPU usage. As a result, when the queue depth reaches beyond a certain level, the queue manager starts to thrash as the CPU usage level reaches 100%.

When the sender has stopped and no new messages are sent to the queue, the queue manager only needs to de-queue the unsent messages, including those in the file. It was observed that the disk I/O level dropped considerably and the CPU usage level also dropped. This means the queue manager has more resources with less disk I/O and as a result it can send the unsent messages at a higher transmission rate than the initial rate. This condition is equivalent to sending pre-loaded messages.

7. Metric for performance measurement

The system's behavior as described above leads to a conclusion that the system operates in two different states – sustainable and unsustainable. A sustainable state is the state in which the system can maintain the same level of performance for an unlimited run time. The unsustainable state causes erratic and reduced performance.

As a result, the tests quantified the system's performance by measuring the maximum sustainable throughput (MST). This is the throughput at the saturation point. This throughput level does not cause the queue depth to increase at either the sender's or the receiver's queue. Also the performance measurement is conducted with queues initially empty, and with both the sender and receiver running.

There are a number of factors that can influence the performance of the system. These include:

- the quality of service that defines the level of message delivery guarantee
- the system's configuration parameters
- the message length,
- the receiver's number of threads.

Therefore it is important to explore the extent of influence of these factors on the performance.

7.1. Quality of Service (QoS) variable

MOM technologies are about delivering messages reliably. Products normally offer three levels of QoS. They are *non-persistent, persistent* and *transactional*. In non-persistent delivery**,** the MOM will do its best to deliver the message. Undelivered messages are only kept in memory and can be lost if a system or network fails before a message is delivered. With persistent delivery**, t**he MOM guarantees to deliver messages despite system and network failures by logging them to disk as well as keeping them in memory. This means they can be recovered and subsequently delivered after a system failure.

With the transactional delivery**, t**he MOM tightly integrates persistent messaging operations with application code, not allowing transactional messages to be sent or received until the application commits their enclosing transaction. Transactional messaging also allows message sending and receiving to be coordinated with other transactional operations, such as database updates. Transactional messaging normally involves a transaction manager (TM).

Combining these levels of QoS results in 3 different alternatives for performance measurement. They are non-persistent non-transactional (NPNT), persistent non-transactional (PNT) and persistent transactional (PT).

There are two variants of PT. They are Persistent Local Transaction (PLT) and Persistent Global Transaction (PGT). In PLT messaging, there is no integration with external transactional operations such as database updates and the MOM product itself manages the transaction with single-phase commits. PGT messaging involves external transactional operations such as database updates and the TM performs XA-compliant 2-phase commits. In these performance tests, MQSeries is used as the TM.

7.2. Other variables

The performance tests are carried out with various configuration parameter values (default or tuned values recommended by the vendor). The configuration parameters include method of binding (*standard* or *FastPath*), log buffer size and number of log files as described in Section 3.

With the message length, the performance measurement is limited to 64-byte, 1024-byte and 4096-byte messages. The selection of these lengths may be arbitrary but should be sufficient to indicate how the system's performance is affected.

The performance tests are also carried out using two different types of receiver. The first test uses a single-threaded receiver and the second test uses a multi-threaded receiver. This is to show how throughput changes when the number of threads in the receiver is

IEEE COMPUTER SOCIETY

increased, and to give an indication of how well the products will scale in a more realistic test case with many receiving applications.

8. Performance results and analysis

Table 2 shows the maximum sustainable throughput (MST) obtained for 64-byte NPNT messages with varying number of receiving threads, message lengths and different types of bindings. It shows that the system performs best with a single-threaded receiver and the *FastPath* binding. *FastPath* binding improves the MST by about 40%. The throughput declines when multiple receiving threads are used, possibly due to lock contention somewhere inside the MQSeries code. It was observed that the CPU usage level remained stable at about 60% on both the sending and receiving machines for all of these tests.

Receiving threads	Mgs length (bytes)	FastPath binding	Standard binding
1	64	3000	2150
	1024	2700	-
	4096	1650	-
5	64	2750	2050
15	64	2650	2050
20	64	2600	2050

Table 2: MST for NPNT messages

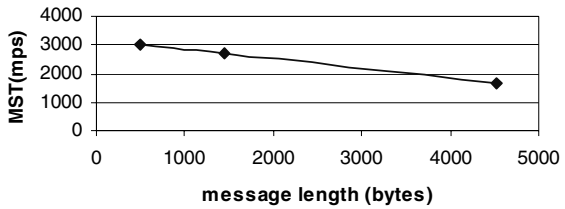

Figure 6: MST vs. message length for NPNT messages

Since MQSeries precedes the message with a header of 430 bytes when sending a message, the actual message length is the sum of the length of the user's message and the length of the header. Figure 6 shows the relationship between the MST and the actual message length. It shows that the MST decreases almost linearly with the increase of the message length.

Table 3 shows the MST obtained from the PNT messaging tests. It shows that the impact of varying the number of receiving threads is negligible, and increasing the message length from 64 bytes to 1024 bytes reduces throughput by less than 1%. Tuning the log configuration to a 2MB buffer and twelve 4096KB log files, has no effect.

Receiving threads	64-byte message		1024-byte message
	Default log	Tuned log	Tuned log
1	120	120	110
15	130	130	120
20	Not tested	130	120

Table 3: MST for PNT messages

The overall performance for the PNT messages is significantly lower than that for the NPNT messages. This is to be expected given that messages are now have to be logged to disk for recovery purposes. The disk I/O operations for message logging largely limit the throughputs shown in these tests.

Table 3 also shows that the throughput is unchanged with increasing number of receiver threads. This is because when queuing PNT messages, MQSeries locks the queue while updating the log to ensure that updates to the queue are serialized. This means there can only be one operation in progress at the queue at any one time.

Receiving threads	64–byte message		1024-byte message	
	Default log	Tuned log	Default log	Tuned log
1	90	100	80	90
15	520	650	380	570
20	510	-	370	-
30	480	-		-

Table 4: MST for PLT messages

Table 4 shows the MST obtained from the PLT messaging tests using different message lengths, log configurations and number of receiving threads. It shows that the MST increases with the increasing number of the receiving threads, reaching 520 messages per second with 15 receiving threads using the default log. The tuned log further improves the MST by 25% for 64-byte messages, and by 50% for 1024-byte messages. Increasing the number of receiving threads more than 15 only decreases the performance. This is because the machine is disk I/O and CPU bound, causing contention.

With the tuned log and 15 receiving threads, increasing the message length from 64 bytes to 1024 bytes causes the throughput to fall by 14%, from 650 to 570. Note that increasing the number of sending threads makes no difference to the performance.

For PLT messaging, the log management and queue management are performed in parallel and asynchronously. Message logging takes place at the same

IEEE COMPUTER SOCIETY

time as messages are being written to queues, allowing multiple applications to run simultaneously. This behavior is quite different to the serialized behavior for PNT messaging as explained earlier. As a result, compared with the MST for the PNT messages, the MST for PLT messages is five times higher.

Logging becomes more efficient as the size of the log buffer and number of log files are increased. The use of larger buffer and more log files allows for more efficient and less frequent disk operations, with fewer checkpoints. This reduces disk I/O substantially and leads to the throughput improvements.

Table 5 shows the MST obtained for PGT messages with different message lengths and varying number of receiving threads using the tuned log configuration. The transaction in each thread includes an update of a SQL Server V7.0 database table.

Receiving threads	Message length		
	64-byte	1024-byte	4096-byte
1	30	30	30
15	60	60	60
20	60	60	60

Table 5: MST for PGT messages

Compared with the MST for the PLT messages, the MST for PGT messages is significantly lower. There are a number of factors that affect the MST in this case. The overhead of managing global 2-phase commit transactions, database updates, and interfacing with external resource managers are some of those that contribute to the drop in the MST. Because of these factors, increasing the message length has no observable impact on the performance. The highest MST is 60 mps with 15 receiving threads.

9. Conclusion

This paper presents the result of a study on the behavior of the MQSeries V5.2 using an open-loop test scenario with asynchronous senders and receivers. The result indicates that the system operates in two different states – sustainable and unsustainable. The system can be in one of these two states depending on the message sending or receiving rate. It remains in the sustainable state if the sending rate does not cause the queue to grow beyond a certain stable queue depth. Otherwise if the queue is allowed to continuously grow, the application enters an unsustainable state. The result of the study enables a clear metric to be established for the performance measurement of MOM technologies. The system's performance is quantified by measuring the maximum sustainable throughput (MST).

This paper presents the performance results using four types of QoS - non-persistent non-transactional, persistent non-transactional, persistent local transactional and persistent global transactional. The results show that the impact of various factors – message length, tuning parameters, QoS and the receiver's number of threads - on the performance is significant in many aspects. The MST decreases almost linearly with the increase of the message length for NPNT messages. Logging for PNT messages reduces the peak MST from 3000 mps to 130 mps. While the system does not scale for PNT messages, it scales well for both PLT and PGT messages, as increasing the receiver's number of threads increases the MST for PLT messages by more than six times, from 100 mps to 650 mps. The MST for PGT messages is constrained by the local database updates and the overhead of the transaction manager. It does not show the same level of scalability as PLT messaging

References

[1] A. Rindos, M. Loeb, S. Woolet, "A Performance Comparison of IBM MQseries 5.2 and Microsoft Message Queue 2.0 on Windows 2000", IBM SWG Competitive Technical Assessment, Research Triangle Park, 3/2001.

[2] http://www.microsoft.com/msmq/whitepapers.htm, "Performance Evaluation of Microsoft Messaging Queue, IBM MQSeries and MQBridge".

[3] I.Gorton, A.Liu, P. Tran, "The Devil is in the Detail, A Comparison of CORBA Object Transaction Services", the 6th International Conference on Object-Oriented Information Systems, pages 211-221, 18-20 December 2000, London.

[4] S.Ran, P. Brebner, I.Gorton, "The Rigorous Evaluation of Enterprise Java Bean Technology", in 15th International Conference on Information Networking (ICOIN-15), Beppu City, Japan, Feb 2001, IEEE

[5] P. Tran, I Gorton, "Analyzing the Scalability of Transactional CORBA Applications", TOOLS Europe 2001, Zurich, Switzerland, 12-14/3/01, IEEE proceedings TOOLS 38, pp 102-110.

[6] P. Tran, P. GreenField, I. Gorton, H. Tran, "Performance Evaluation of Message-Oriented Middleware Technology - Report", CSIRO Mathematical and Information Sciences, Australia, 8/2001.

[7] IBM, "MQSeries– Planning Guide", 9th Edition, 3/2000.

IEEE COMPUTER SOCIETY