SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2003; **13**:1–12 (DOI: 10.1002/stvr.279)

Evaluating the sustained performance of COTS-based messaging systems‡

Phong Tran[1], Jeffrey Gosper[1] and Ian Gorton[2,*,†]

[1] *CSIRO Mathematical and Information Sciences, North Ryde, NSW 2113, Australia*
[2] *Information Sciences and Engineering, Pacific Northwest National Laboratory, Richland, WA 99352, U.S.A.*

SUMMARY

Messaging systems, which include message brokers built on top of message-oriented middleware, have been used as middleware components in many enterprise application integration projects. There are many COTS-based messaging systems on the market, but there is little concrete understanding in the software industry on the performance of these different technologies. The authors have carried out a scenario-based evaluation of three leading messaging systems to provide insight into performance issues. The evaluation process includes a study of the sustained performance of the system under load. The result of this study is used to derive a generic metric for quantifying a messaging system's performance. The paper describes a synthetic transactional scenario, which is used for load tests and performance measurement. The results from executing this test scenario with three messaging systems are then presented and explained. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: message-oriented middleware; message broker; performance evaluation; messaging system; system validation; test and validation

1. INTRODUCTION

Messaging systems, which include message brokers built on top of message-oriented middleware (MOM), have become essential components in the world of enterprise application integration (EAI). They enable two-way, synchronous and asynchronous communication between different business applications within a single enterprise. They channel, route and transform messages from one source or form to another. The MOM component handles the messaging and the message broker component handles the routing, transformation and other functions.

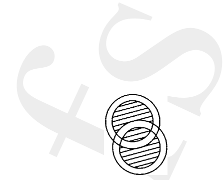
*Correspondence to: Ian Gorton, Information Sciences and Engineering, Pacific Northwest National Laboratory, Richland, WA 99352, U.S.A.
†Email: ian.gorton@pnl.gov

‡An earlier version of this paper was originally presented at the *First International Workshop on Verification and Validation of Enterprise Information Systems*, held in Angers, France on 22 April 2003. It is reproduced here in modified form with the permission of the Workshop organizers and the publishers of the Proceedings, ICEIS.

Copyright © 2003 John Wiley & Sons, Ltd.

Received 8 May 2003
Accepted 31 July 2003

Uncorrected proofs



Uncorrected Proofs

There are a number of competing products in the messaging technology market place. They all appear at first sight to offer much the same set of core features. In reality, they are actually quite different in their underlying architecture and implementation, with each aimed at meeting the diverse needs of different core markets. Not surprisingly, they all have their own strengths and weaknesses.

A messaging system is like a pipe linking two water tanks; one is constantly filled while the other is drained. The messaging system's throughput is analogous to the amount of water flowing through the pipe, with the width of the pipe representing the bandwidth of the network between the sender and the receiver. If the tank is filled faster than the water inflow to the pipe, or the water outflow from the pipe to the other tank is faster than the tank is drained, either or both tanks will eventually overflow. Consequently, a messaging system plays an important role and its performance is critical in MOM-based EAI.

For the research community studying middleware component architectures, the performance of messaging systems is an important topic of interest. Performance is used as a measure to compare the efficiency and quality of the architectures implemented in those systems. Different architectures will inevitably result in different performance and message throughput.

For IT professionals, performance is an important criterion that is often used as a key differentiator during the acquisition process of a messaging technology. It is also a key element used in computing resource planning for deployed applications.

A survey of the literature on performance measurement of commercial messaging systems shows that there are only a handful of papers, all written by the products' vendors [1–3]. Rindos *et al.* [1], as well as Microsoft and NSTL [2] present results on performance of IBM's WebSphere MQ and Microsoft Message Queue (MSMQ), respectively; Dun and Branagan [3] present results on performance of individual elements used in message broker applications to highlight the cost of using each element rather than the cost of running a message broker application. Clearly then, there are a limited number of publications in this field, particularly from vendor independent sources. As such, insufficient validated and dependable information is available to help industry select a suitable product.

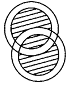
In response to these issues, the authors have initiated a project to evaluate the performance of commercially available messaging systems. The aim of the project is to generate credible and reliable evaluations and performance measurements and disseminate these to the industry. The authors' previous work includes evaluations of CORBA, EJB and J2EE technologies [4–8]. For this project, three of the most popular messaging systems, namely IBM WebSphere MQ/MQIntegrator (WMQI), TIBCO Rendezvous/MessageBroker V4.0 and Mercator Integration Manager V6.0, were selected for evaluation and testing.

This paper discusses generic issues related to the performance of messaging systems. It then describes the derivation of a metric to measure the sustained throughput of a messaging system, and presents the results of empirical comparisons of the IBM WMQI, Mercator and TIBCO messaging systems.

2. MEASURING MESSAGING THROUGHPUT

Messaging systems are designed to process messages placed in queues. The rate at which a receiving application takes messages from a queue and processes them determines how many messages build up in the queue, and consequently how long each message takes to be processed.

Copyright © 2003 John Wiley & Sons, Ltd.



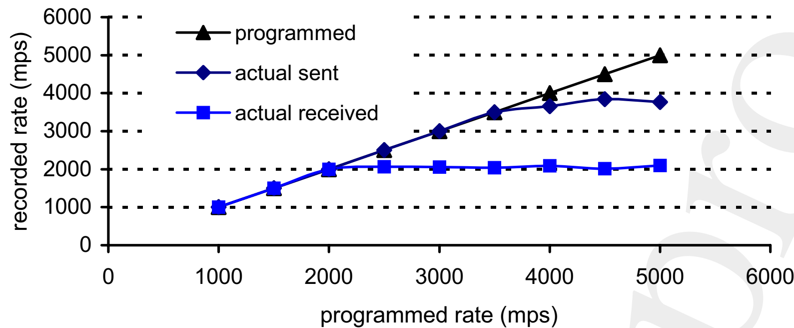


Figure 1. Recorded rates versus programmed rates.

When the message arrival rate is consistently greater than the ability of the application to process messages, the queue continually grows. The inevitable result is that messages take longer to be processed, system performance degrades and eventually some application, middleware or system resource will be exhausted, causing the application to fail. Under this kind of overload condition, testing reveals that messaging systems tend to operate predictably at certain times and unpredictably at others. This makes performance measurement problematic, as if unstable conditions prevail during a measurement period, results can be inconsistent and misleading. In response to these problems, it was decided to investigate a more meaningful measure of throughput for a messaging technology.

2.1. Load test investigations

Some initial experiments with WMQI were carried out to study its behaviour. These experiments involved a sender process configured to send 64-byte messages at a fixed rate to a queue, a receiver process retrieving the messages as quickly as possible and a process recording the actual sending and receiving rates.

In the first experiment, the message sending rate was gradually increased from 1000 to 5000 messages per second (mps).

The result of this experiment, shown in Figure 1, clearly shows the difference between the actual sending and receiving rates. The diagram also shows that when the sending rate is less than approximately 2000 mps, the receiving rate is the same as the sending rate. However, as the sending rate increases beyond 2000 mps, the receiving rate stagnates. It was observed that messages were now being queued in the sender's queue while the queue at the receiver remained empty.

In the second experiment, the sending rate was fixed at 2400 mps and the sender was stopped after executing for 120 s, while the receiver remained running until all messages were received from the queue. The result of this experiment is shown in Figure 2, which depicts the sending and receiving rates plotted against the execution time.

Figure 2 shows that the sending and receiving rates are stable at approximately 2400 mps and 2000 mps, respectively, for approximately the first 100 s of the test period. During this time the queue

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

Uncorrected Proofs

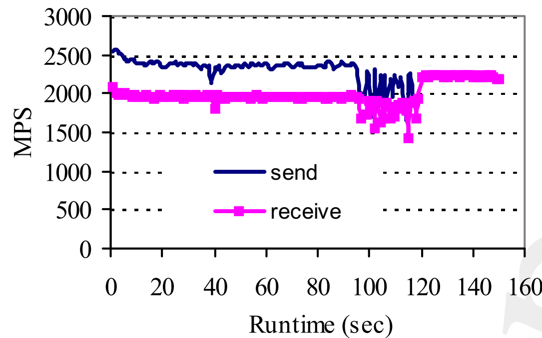


Figure 2. Sending rate versus runtime.

depth at the sender's queue increased continually at a linear rate with unsent messages while the queue depth at the receiver's queue was almost zero.

After 100 s, the sending rate fluctuates sharply and on average degrades, causing a ripple effect on the receiving rate. At this stage, it was observed that the CPU usage level and the level of disk I/O at the sender machine also fluctuated unpredictably, and the queue contained over 40 000 messages.

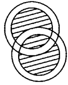
After 120 s, the sender process stopped and no more messages were put into the queue. However, the sender's queue manager and the receiver continued to process the unsent messages, with the receiving rate rising to approximately 2200 mps until all messages were received.

2.2. Load test analysis

The results of these experiments indicate 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. This is likely due to the queue manager exceeding the buffer space it has to store undelivered messages, which consequently must be flushed to disk. This causes a high level of disk 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 now only needs to de-queue and deliver the unsent messages. It was observed that the disk activity and the CPU usage level also dropped considerably at this stage. This means the queue manager has sufficient resources and as a result it can send the unsent messages at the highest rate the queuing software and network can support.

Copyright © 2003 John Wiley & Sons, Ltd.



3. SUSTAINED PERFORMANCE METRIC

Similar load tests were also carried out with messaging technologies from other product vendors and it was found that they behaved in a similar manner. Consequently, it was concluded that in general a messaging system operates in two states, namely sustainable and unsustainable. A sustainable state is one in which the system can maintain the same level of message throughput for an unlimited execution time, with no steadily increasing build-up of messages internally within the message queues. The unsustainable state, caused by consistently sending messages to the queue faster than they can be delivered, causes erratic and reduced performance.

Consequently, for any given messaging system configuration, there is a maximum sending rate that can be maintained while the messaging system remains in a stable state. This is defined as the *maximum sustainable throughput* (MST). If messages are consistently sent at a rate higher than a given configuration's MST, eventually the system will become unstable due to resource exhaustion. Hence the MST gives a meaningful metric with which to compare the performance of messaging systems.

4. PERFORMANCE MEASUREMENT

The achievable MST for any messaging technology is dependent upon the message arrival rate, the processing required and the hardware platform that the tests are executed on, These are all explained in the following subsections.

4.1. Test scenario

In order to measure the MST, a test application scenario was defined that reflected various aspects of 'real world' applications. Essentially, the scenario defines various business transactions that require messages to be sent to a MOM queue. A message broker takes messages from the queue, performs various data transformations, calculations and database operations, and finally delivers a result message to an output queue.

The scenario comprises three simulated purchasing transactions, which require access to a relational database. These are as follows.

Create transaction. The business logic for this transaction requires a new record to be inserted in a relational database table. Data from the input message are extracted, a new key value is generated for the record and the new record is inserted into the table. Finally, an output message is generated containing the input data plus the generated new key value for the inserted record.

Update transaction. This transaction has relatively complex business logic. Essentially data from the input message are validated against values read from the database. If a threshold is exceeded, an exception is thrown and an error message is sent to the output queue. If the input data are valid, data in the database are updated and a new record inserted into a table. The output message contains the input data and a flag indicating the success of the transaction.

Read transaction. This transaction uses the input message data to form a single database query. The query result can contain up to 20 rows of data, which are inserted into the output message.

The transactions are designed to place different loads on the message broker and underlying messaging system. The *Read* transaction is regarded as a lightweight transaction as it involves a simple

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected Proofs

Table I. Specifications of the test machines.

	Broker	Sender	Receiver	Oracle
No. of CPUs	4	1	4	4
CPU MHz	750	1000	750	750
RAM (Gb)	4	0.5	4	4
Disk type	SCSI	Std	SCSI	SCSI
Disk rpm	15 000	7200	15 000	15 000
OS	W2K Server			
CPU type	Pentium III			

database read operation. The *Create* transaction is medium-weight, involving a two-step database read and write operation. The *Update* transaction is heavyweight, involving multiple database operations, error checking, message transformations and simple calculations.

4.2. Test platform and architecture

The test scenario comprises up to five different machines that host the messaging technology under test, the message sender process, the message receiver process and an Oracle database server. As handling of non-persistent messages does not require disk I/O, the sender and receiver share the same machine with no discernable side-effects on performance measurement. However, for persistent messages, the sender and receiver processes must execute on separate machines to avoid disk contention, as handling persistent messages creates high levels of disk activity. In addition, for persistent messaging, two machines are used to host two sender processes. This is because the one of the sender machines has only a single CPU, and testing revealed that using only one sender machine was insufficient to fully load the message broker, which was hosted on a much faster 4-CPU machine. All these machines are networked on a 100 MB isolated local area network (LAN). The specifications of these machines are shown in Table I and the structure of the test system is shown in Figure 3.

4.3. Input messages

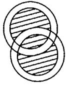
All input messages are in XML and are sent at a fixed ratio of one *Create*, three *Updates* and one *Read* message in a batch. Example input messages for the *Create, Update and Read* transactions respectively are shown below:

```
<?xml version=\"1.0\>
<Stockonline>
  <TransType>NewAccount</TransType>
  <Customer>
    <Name>Phong Tran</Name>
    <Address>MacquarieUni</Address>
    <Credit>1000000</Credit>
  </Customer>
</Stockonline>
```

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

September 26, 2003 Marked proof Ref: STVR275/26006ae Sheet number 7



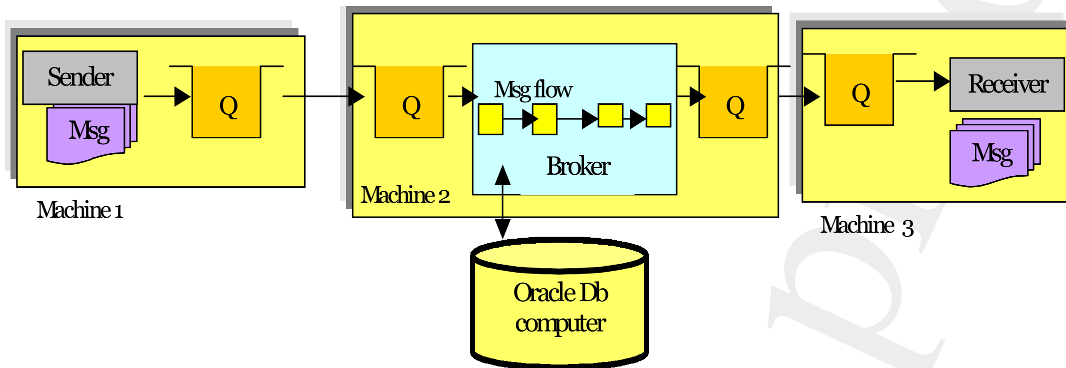


Figure 3. Test system architecture.

```
<?xml version=\"1.0\"?>
<Stockonline>
    <TransType>BuyStock</TransType>
    <AccountNo>25000</AccountNo>
    <Qty>5</Qty>
    <StockId>4020</StockId>
</Stockonline>

<?xml version=\"1.0\"?>
<Stockonline>
  <TransType>GetHolding</TransType>
  <AccountNo>20303</AccountNo>
  <StartStockIdNo>0</StartStockIdNo>
  <HoldingList></HoldingList>
</Stockonline>
```

Note, the input message for the *Read* transaction initially contains an empty holding list which is filled with up to 20 result entries after the transaction is completed.

4.4. Message flow

Figure 4 shows a snapshot of a WMQI tool that depicts the message flow that implements the test scenario described in Section 4.1. The message flow takes a series of XML input messages and routes each to one of the three alternate execution paths based on the value of the field <TransType> field in the message.

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

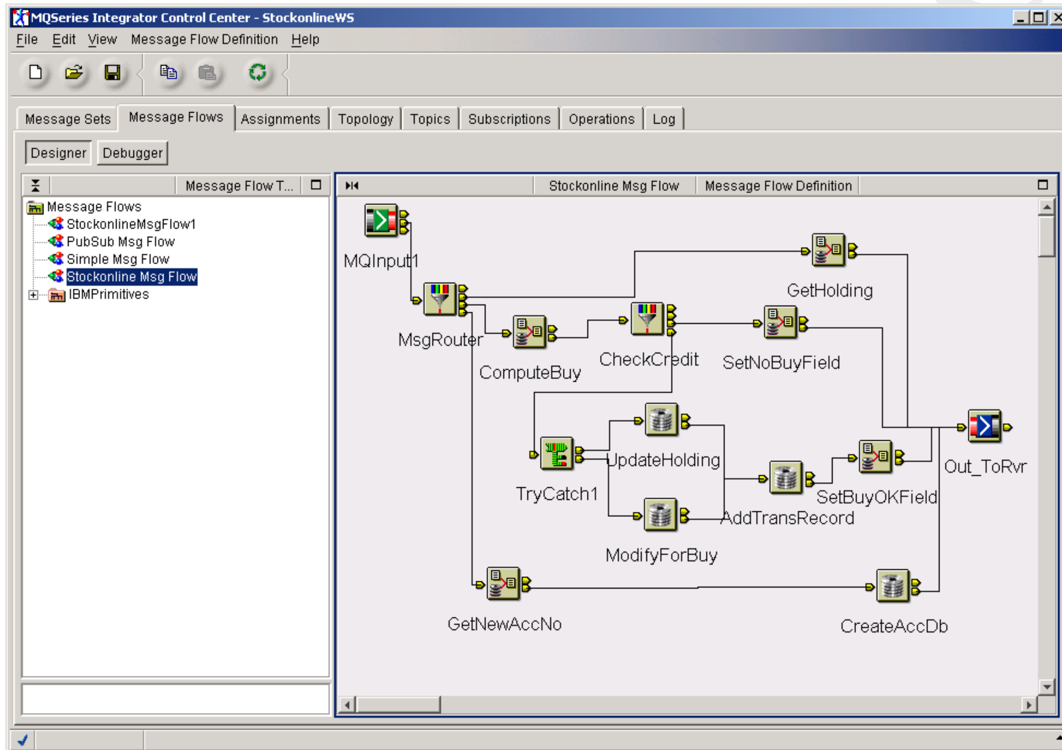


Figure 4. Example message flow for transactions in WMQI.

4.5. Test variables

In the performance tests, two variables are used to explore the range of features offered by the technologies.

First, the different qualities-of-service (QoS) for message delivery were explored. Messaging systems normally offer three levels of QoS for message delivery. These are non-persistent, persistent and transactional.

With non-persistent delivery, messages are delivered as fast as possible. Undelivered messages are only kept on queues in memory and can be lost if a system fails before they are delivered.

With persistent delivery, messages are delivered in the face of system and network failures. Persistent messages are logged to disk as well as kept in memory. This means they can be recovered and subsequently delivered after a system failure.

Transactional delivery is intended for persistent messages within a transactional context. Persistent transactional messages are only delivered after the sender or receiver commits their enclosing

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected Proofs

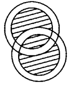


Table II. MST for non-persistent messages.

		No. of threads				
		1	3	5	10	15
1 EG	CPU (%)	10	30	50	65	70
	MST	80	170	230	260	240
2 EGs	CPU (%)	30	—	95	—	—
	MST	170	—	385	—	—
3 EGs	CPU (%)	40	97	100	—	—
	MST	250	390	400	—	—

transaction. If the commit fails, all messages are rolled back to their state prior to the 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 component that belongs to the messaging system.

Second, most messaging systems are designed to be scalable to improve performance. WMQI uses a process known as an execution group (EG) to host multi-threaded message flows. An execution group is an independent process running in its own address space with its own resources. WMQI allows many execution groups to replicate the same message flow instances. The number of threads used to service a message flow and the number of replicated execution groups are configurable scale factors, which can be used to tune the performance of the system.

5. PERFORMANCE RESULTS AND ANALYSIS

Tables II and III show the results of the performance tests for non-persistent and persistent messages respectively, with different numbers of threads and execution groups configured for different tests.

Table II shows that with one execution group, the MST of one thread is 80 mps. It increases with increasing numbers of threads and reaches the maximum of 260 mps with only 65% CPU usage. Figure 5 shows the relationship between the throughput and the number of threads for one execution group. It shows that no additional performance is gained by increasing the number of threads to 15, and in fact the MST starts to degrade with an optimal number of threads being approximately 10. This is due to the additional contention placed on the database by the increasing number of concurrent connections.

While the server is in the stable state, there is no disk activity for non-persistent messages. It was observed that the database machine's CPU utilization was around 20% and the disk activity was low, indicating that the database was not the bottleneck.

Table II also shows that increasing the number of execution groups to three, each with one thread, increases the MST from 80 to 250 mps with only 40% CPU usage. This indicates that multiple execution groups are more efficient than multiple threads, because each execution group has access

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

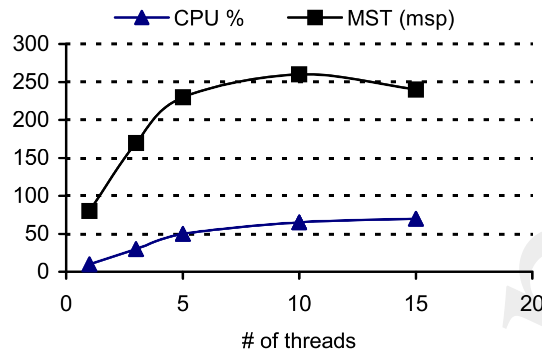


Figure 5. MST and CPU% versus number of threads for non-persistent messages with one execution group.

Table III. Throughput for persistent messages.

		No. of threads				
		1	3	5	10	15
1 EG	CPU (%)	10	30	45	65	70
	MST	50	100	140	200	200
2 EGs	CPU (%)	20	—	50	60	—
	MST	90	—	200	200	—
3 EGs	CPU (%)	50	60	60	—	—
	MST	130	200	200	—	—

to its own resources. As seen in Table II, three execution groups, each with one thread, sustain 250 mps compared with 170 mps produced from three threads in one execution group.

With two execution groups, increasing the number of threads to five increases the throughput to 385 mps with 95% CPU usage. With three execution groups, the throughput is 400 mps (again with five threads) and the machine becomes CPU bound (100%). This means that increasing the number of execution groups or threads further has no effect on performance, as the server hardware becomes the bottleneck.

Table III shows that the MST for persistent messages with one execution group and one thread is 50 mps. The MST increases with an increasing number of threads and reaches a maximum of 200 mps with only 67% CPU usage. It was observed that when this maximum MST was reached, the number of disk writes per second for message logging was 210.

Table III also shows that the MST increases with increasing numbers of execution groups and also with the number of threads. With three execution groups, and utilizing a single thread, the MST is

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

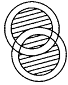


Table IV. Highest MST (mps) of WMQL, Mercator IM and TIBCO MB.

QoS	IBM WMQI	Mercator IM	TIBCO MB
Non-persistent	400	280	120
Persistent	200	200	160

130 mps. In this case it was observed that the disk activity was 180 writes/s. However, with multiple threads, the MST reaches 200 mps with only 60% CPU utilization, and the disk activity remained at 210 writes/s. This means disk accesses limit the performance and increasing the number of execution groups or threads further produces no effect.

5

6. COMPARISON BETWEEN IBM WMQI, MERCATOR IM AND TIBCO MB

Table IV shows the highest MST achieved in any configuration in the test environment for IBM WMQI, Mercator IM and TIBCO MB, for both persistent and non-persistent messaging. Table IV shows that IBM WMQI outperforms TIBCO MB in both non-persistent and persistent messaging, essentially due to its better scaling ability through the execution group mechanism. IBM WMQI also performs better
10 than Mercator IM for non-persistent messages. However, for persistent messages, the two messaging technologies achieve the same performance level, which is basically the highest possible on the test hardware due to disk speed limitations.

7. CONCLUSION

This paper presents the results of a study on the behaviour of messaging systems. It shows that
15 different messaging technologies have similar general behaviour. They operate in two different states—sustainable and unsustainable.

The result of the study enables a concrete metric to be established for the performance measurement of messaging systems. The performance is quantified as the maximum sustainable throughput (MST), which defines the intersection point between the stable and unstable states.

20 This paper also presents the detailed results of the performance evaluation of WMQI V2.0.2 under various QoS options for message delivery and with different scaling configurations. The results show that the impact on performance of message delivery QoS and scaling configurations are significant. Also, multiple execution groups are more efficient than multiple threads, due to the extra resources associated with execution groups. For persistent messages, the performance is limited by the disk
25 speed rather than by the CPU. Utilizing the hardware configuration in the test platform, the MST of WMQI reaches 200 mps with only 60% CPU usage while the disk bandwidth is saturated.

Copyright © 2003 John Wiley & Sons, Ltd.

Uncorrected proofs

The paper also compares the best performance of IBM WQMI, Mercator IM and TIBCO MB. IBM WMQI outperforms TIBCO MB, as it scales better than TIBCO MB. IBM WMQI also performs better than Mercator IM for non-persistent messages but has the same performance as Mercator IM for persistent messages.

REFERENCES

1. Rindos A, Loeb M, Woolet S. 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, NC, March 2001.
2. Microsoft, NSTL. Performance evaluation of Microsoft messaging queue. *IBM MQSeries and MQBridge*. http://www.microsoft.com/msmq/MSMQ_Final_report.doc.
3. Dun T, Branagan R. Websphere MQ Integrator for Windows NT and Windows 2000 V2.1. *Performance Report*, IBM U.K. Hursley Park Laboratories, March 2002.
4. Gorton I, Liu A, Tran P. The devil is in the detail, A comparison of CORBA object transaction services. *Proceedings of the 6th International Conference on Object-Oriented Information Systems*, London, December 2000; 211–221.
5. Ran S, Brebner P, Gorton I. The rigorous evaluation of enterprise Java bean technology. *Proceedings 15th International Conference on Information Networking (ICOIN-15)*, Beppu City, Japan, February 2001. IEEE Computer Society: Los Alamitos, CA, 2001; 93–100.
6. Ran S, Brebner P, Tran P, Gosper J, Chen S, Hu L, Gorton I, Palmer D. J2EE technology performance evaluation methodology. *Proceedings of Distributed Objects and Applications*, University of California, November 2002; 13–16.
7. Brebner P, Ran S. Entity bean A, B, C's: Enterprise Java beans commit options and caching. *Proceedings of the International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.
8. Tran P, Gorton I. Analysing the Scalability of Transactional CORBA Applications. *Proceedings of TOOLS Europe 2001*, Zurich, Switzerland, March 2001. IEEE Computer Society, Los Alamitos, CA, 2001; 102–110.

Uncorrected proofs

Copyright © 2003 John Wiley & Sons, Ltd.

Annotations from stvr279.pdf

Page 12

Annotation 1
Author: Please provide publisher and location details for [4]

Annotation 2
Author: Please provide publisher and location details for [6]