A PERFORMANCE COMPARISON OF IBM MQSERIES 5.1 AND MICROSOFT MESSAGE QUEUE (MSMQ) 2.0 ON WINDOWS 2000

EXPRESS DELIVERY MODE

IBM Software Group Competitive Technical Assessment

March 2000

ABSTRACT

In this paper, we present a performance comparison of the two major message queuing software products for Windows operating systems on the market today: ***IBM MQSeries 5.1*** *and* ***Microsoft Message Queue (MSMQ) 2.0****. The messages per second and throughputs for both running on the* ***Windows 2000*** *operating system, using express delivery mode, were compared under a variety of conditions (different message sizes, number of clients and sessions per client, number of preloaded messages in the server response queue). Using a test setup and procedures that stressed the entire client-server response path (and were therefore representative of an actual customer environment), MQSeries significantly outperformed MSMQ for every combination of conditions examined.*

ITIRC KEY WORDS

Performance

Benchmarking

MQSeries

Message Queuing

MSMQ

Express Delivery Mode

Windows 2000

CONTENTS

ABSTRACT iii

ITIRC KEYWORDS iii

INTRODUCTION 1

MESSAGE QUEUING PERFORMANCE 5

TEST CONFIGURATION AND PROCEDURES 5

TEST RESULTS 8

Effects of Varying the Number of Clients and Threads . . . 8

Effects of Preloading the Server Response Queue 17

Effects of Varying Message Size 20

Previously Published MSMQ Performance Results 25

CONCLUSIONS 28

REFERENCES 29

NOTES 30

INTRODUCTION

Message queuing performance is essential to any e-business solution

Message queuing is a strategic component of any e-business solution. By providing an encapsulated, asynchronous means by which different programs can communicate, it provides the distributed benefits of remote procedure calls and traditional transaction processing, without many of their inherent problems [1]. Robustness is introduced by encapsulating remote procedure calls and other requests into asynchronous, independent messages that are then handled by a designated queue manager. The queue manager oversees the course of those messages as they travel between different machines across a network, or between different programs within the same machine. The client application and/or machine originating the message can move on to processing other tasks as it awaits the message's response. Meanwhile, the message queuing application can manage and respond to a wide variety of networking or other problems that might otherwise hang up or crash the original requesting program.

Message queuing therefore allows an e-business application designer to (a) increase program efficiency and system capacity by utilizing multiple servers to execute e-business applications, and (b) create e-business solutions that require a number of (otherwise) incompatible programs. In the latter case, through their use of messages and message brokers, message queuing solutions allow an e-business application programmer to write separate translation or other bridging programs (that might run on separate servers) between two otherwise incompatible programs. This allows the programmer to utilize the broadest possible array of advanced application software in his/her final application, in a timely and easily manageable manner.

To provide this capability, message queuing clients and servers must be able to process a large number of messages per unit time, for the full range of possible message sizes. A relatively large single client/server capacity to process messages implies an efficient processor execution path, which in turn implies less processor utilization consumed by the message queuing application software. This translates to better overall performance of simultaneously executing applications, which are presumably the important end user applications (the "ends") that a message queuing solution should be facilitating (as the "means"). Likewise, a higher server capacity implies greater system scalability, i.e., the ability to support increasingly larger numbers of clients without an exorbitantly expensive increase in supporting resources.

Furthermore, any message queuing solution should provide robust performance in the face of server congestion, especially due to the accumulation of unretrieved messages. In a real customer environment, clients may need to share request/response queues, so that messages will naturally accumulate in a server's message queues over time. Some of these messages may remain unretrieved for some extended period of time. Any inefficiencies in the ability of a client to locate its own message(s) within its response queue is potentially detrimental to a solution's ability to scale. If the performance impact is severe, the affected message queuing solution should be deemed unacceptable for use in any e-business solution.

Therefore, in order to provide an e-business solution designer with meaningful consumer data so that he or she can make an informed choice, in this paper, we compared the performance of the two major message queuing software products for the **Windows[1] 2000** operating system on the market today, **IBM[2] MQSeries[2]** and **Microsoft[1] Message Queue (MSMQ[1])**. The two were compared under a variety of conditions and system parameters, including different message sizes, number of clients, number of threads per client and number of preloaded messages (in the server response queue). The express delivery mode was examined, since it provided the best performance for both applications. (It avoided highly variable disk access delays.) The TCP/IP (Transmission Control Protocol / Internet Protocol) was used throughout, since it is the prevalent communications protocol used within most customer networks, as well as over the Internet.

For both applications, great care was taken to individually tune the systems in accordance with performance recommendations made by IBM [2], [3] and Microsoft [4], [5]. However, in this particular study, we examined the performance of a system using a test scenario that stressed the entire client-server response path. Such a test scenario is therefore more indicative of message queuing performance in an actual customer environment (in contrast to test scenarios specifically manufactured to generate unrealistically large benchmark statistics; see, e.g., [4]). The test setup and procedures are carefully described in the section entitled **Test Configuration and Procedures**.

Executive Summary of the Results

As the data will show, for express delivery mode:

(1) IBM MQSeries client and server significantly outperformed those of MSMQ for every combination of system parameters examined (i.e., for every combination of number of clients, number of threads per client, message size and initial response queue size).

As Table 1 below shows, MQSeries single client and overall peak server throughputs were several times those of MSMQ. For example, for the common message size of 1K bytes, MQSeries yielded over 16 times the single client peak throughput of that of MSMQ. Likewise, MQSeries overall peak server throughput was more than 4 times greater than that of MSMQ.

Table 1. Peak Throughput vs. Message Size

Message Size (bytes)	Single Client Peak Throughput (kilobytes per second)		Overall Peak Server Throughput (kilobytes per second)	
	IBM MQSeries	MSMQ	IBM MQSeries	MSMQ
100	51	3	51	12
512	270	15.5	270	60.7
1,024	510.6	31.3	510.6	125.3
5,120	1,575.1	155	1,587.8	594.8
10,240	2,802.9	303.3	2,802.9	1,129.9
20,480	4,363.1	591	4,363.1	2,182.5
30,720	5,548.2	855.6	5,548.2	3,040.6
40,960	6,234.9	1,104.3	6,542.1	3,938.2
65,536	6,372.1	1,655.2	7,135.6	5,617.8

Table 1. MQSeries and MSMQ single client peak performance and overall peak server capacity (measured as the maximum throughput achieved, in kilobytes per second) for a variety of message sizes.

MQSeries performance substantially exceeded that of MSMQ for every situation examined. Single client peak throughput was the maximum throughput achieved over all threads examined (i.e., over 1 through 15 threads) running on a single client, for a given message size. Overall peak server throughput was the maximum throughput achieved over all combinations of number of clients and threads per client examined (i.e., over 1 through 8 clients, running 1 through 15 threads per client), for a given message size. It represented the overall server capacity to process messages. For both peak throughputs (single client and overall server), the maximum was always achieved over the range of the number of clients and threads per client that were examined. Note: For IBM MQSeries, both peak throughputs (single client and overall server) were often the same, since a single client was already capable of saturating this particular server (given the much greater efficiency of the MQSeries client code relative to that of MSMQ).

(2) IBM MQSeries performance was extremely robust as the server request queue was preloaded with an increasing number of initial messages. In sharp contrast, MSMQ performance collapsed as the initial request queue size increased.

Preloading the server request queue mimicked the expected conditions that would arise within a real customer environment, as this queue would grow over time with increasing customer requests. Performance results yielded by this test therefore determined the ability of a message queuing solution to scale up with an increasing number of users.

As Table 2 below shows, MQSeries performance remained relatively unchanged as the initial request queue size (Q_0) was increased to several hundred messages. In contrast, MSMQ performance was devastated by even a small number of preloaded messages. For Q_0 = 50, MSMQ performance dropped from its peak performance of 122.3 messages per second (mps) to 71.3 mps (a reduction of nearly 42%). At Q_0 = 1000 messages, MSMQ performance completely collapsed to 7.4 mps. In contrast, the corresponding MQSeries performance was 376.5 mps.

Table 2. Messages/sec vs. Initial Request Queue Size

Initial Request Queue Size Q_0 (Messages)	Messages per second (mps)	
	IBM MQSeries	MSMQ
0	498.6	122.3
50	489.9	71.3
100	484.7	49.4
150	482.2	37.6
200	475.1	30.2
300	471.2	21.6
500	441.3	14.1
1,000	376.5	7.4

Table 2. MQSeries and MSMQ performance (measured in messages per second) as the server response queue was preloaded with a fixed number of messages.

This test scenario mimicked expected conditions within an actual message queuing customer environment. MQSeries' performance advantage over MSMQ increased as the number of preloaded messages increased (tracking the collapse of MSMQ performance). 1024 byte messages were used throughout, with 6 clients running a single thread each. This was the parameter combination that maximized MSMQ overall peak server throughput for that message size (see Table 1).

It should once again be emphasized that MQSeries and MSMQ were tested using the identical setup (given in Figure 1), using the same test procedures. This setup was tuned in accordance with the recommendations given by Microsoft in [4] and [5]. A series of tests using Microsoft's recommended hard drive architecture did not yield significantly different performance results. These results will be released in a subsequent report.

Previously published benchmark results avoid MSMQ's performance Achilles' heels

Microsoft has previously published unrealistically inflated MSMQ performance numbers [4], using a test procedure and setup specifically designed to avoid the performance "Achilles' heels" demonstrated by Tables 1 and 2 above. In those reported tests, messages were allowed to queue up within a single (independent) client, and were then blasted over to the server. The messages were subsequently discarded. The client was never required to search the server response queue to find a given return message. In the real world, such a return message would typically contain processed data or information requested by the client in the corresponding original message it initially sent to the server. Microsoft's test procedures therefore minimized the processing costs associated with both the client and server. As the data above shows (which will be amplified by the data presented below), the MSMQ queue searching algorithm performed extremely poorly. This document will therefore demonstrate that when a test scenario is used that realistically stresses the entire client-server response path, the introduction of these very real costs yields quite a different performance story from the one Microsoft would like to sell to consumers. **These performance shortfalls seriously compromise MSMQ's ability to scale for large customer environments. Furthermore, the associated costs to processor efficiency mean poor performance of the other (e-business) applications that must simultaneously run on an MSMQ-enabled client or server.**

Throughout this paper, sufficient details regarding the test setup, measurement process and system parameters will be provided, so that the reader may easily replicate these tests.

This paper is intended for the IBM marketing community. It may also be distributed to customers, at the discretion of an IBM sales representative.

Note: The comparative information published in this document reflects laboratory tests undertaken at IBM's facility in Research Triangle Park, NC. Performance in individual cases may vary depending on customer environment, workload, and any unique characteristics of the products in the customer location. Applications examined were the very latest available from a given vendor as of March 24, 2000.

MESSAGE QUEUING PERFORMANCE

TEST CONFIGURATION AND PROCEDURES

Figure 1 shows the configuration that was used to measure the performance of the IBM and Microsoft message queuing products. The test setup consisted of up to 8 IBM PC300GL client workstations (300 MHz Intel[3] Pentium[3] II processor; 32-bit, 33 MHz PCI bus), each with an IBM Etherjet[2] PCI (Peripheral Component Interconnect) adapter (100 Mbps Ethernet, full-duplex mode). The server was an IBM Netfinity[2] 7000 M10 (4 x 400 MHz Xeon[3] Pentium II processors; 32- and 64-bit, 33 MHz PCI buses; 1 GB RAM; 18.2 GB SCSI hard drive), with an IBM Netfinity Gigabit Ethernet SX adapter (1 Gbps, full-duplex, 64-bit). The operating system on all clients was Windows 2000 Professional (General Availability- or GA-level), while that on the server was Windows 2000 Advanced Server (GA-level). The clients and server were interconnected via an Intel Express[3] 510T Ethernet switch (using multimode optical fiber to the server, and twisted pair copper cabling to the clients).

The message queuing applications examined were MQSeries version 5.1 and MSMQ version 2.0 (the latter provided with the Windows 2000 operating system). Message queuing clients were configured as dependent. (Messages were not prestored on the client.) When multiple clients were examined, they shared a single request and response queue, managed by a single queue manager. The performance of this scenario was deemed extremely important in assessing the scalability of the two applications. Only express delivery mode performance is reported in this study, in order to avoid disk access delays which would otherwise reduce performance. (In express delivery mode, messages are only stored in memory until successful retrieval, and are not written to disk. They are therefore unrecoverable should the system crash before they are retrieved.) The performance of persistent (or recoverable, in MSMQ terms) and transaction delivery modes, as well as that of multiple queues and queue managers, were also examined, and will be reported in a subsequent paper.

The communication protocol throughout was TCP/IP (Microsoft's version), using the default maximum frame and TCP window sizes.

All power management features (screen saver, etc.) were disabled, to preclude interruptions to the clients'/server's processors during testing. A 16-way Black Box[4] ServSwitch[4] was used to allow all client and server machines to be controlled using a single monitor, keyboard and mouse. Given the length of the tests, the ServSwitch was set to monitor a machine not involved in the testing during data collection, so that no test machine could be inadvertently interrupted through the mouse or keyboard.

IBM MQSeries performance was tuned in accordance with the recommendations given in [2] and [3]. As prescribed, MQIBindType was set to FASTPATH using the MQSeries Services interface. In addition, the MaxChannels and MaxActiveChannels were set to a value greater than the product of the largest number of clients (here, 8) times the largest number of threads per client examined (here, 15), again using MQSeries Services. For this study, we used MaxChannels = MaxActiveChannels = 950 (well above the maximum number of clients times threads per client). Applications performance optimization was enabled, since it yielded the best MQSeries performance numbers.

MSMQ performance was tuned in accordance with the recommendations given in [4] and [5]. As prescribed, auditing was deactivated. In addition, applications performance optimization was disabled and the size of the paging file was increased to the recommended size.

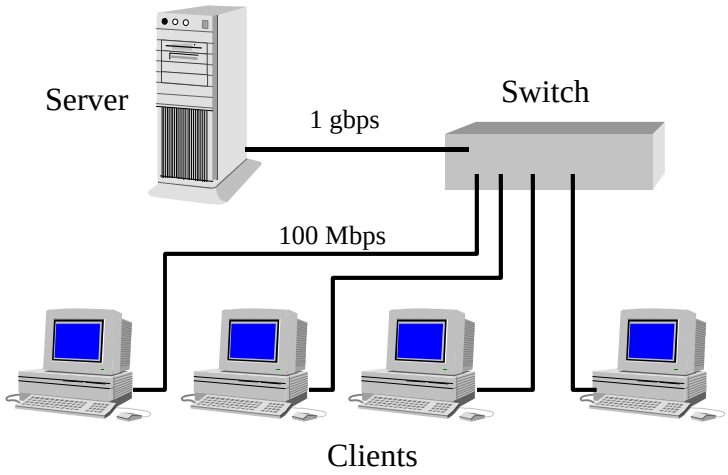

FIGURE 1. Test Setup

Up to 8 clients (300 MHz Pentium II; 100 Mbps full-duplex Ethernet; Windows 2000 Professional) connected to a server (4 x 400 MHz Xeon; 1 Gbps full-duplex Ethernet; Windows 2000 Advanced Server) via a Gigabit Ethernet switch, using TCP/IP.

Machines with only a single SCSI (Small Computer System Interface) hard drive were used in this particular study, being representative of the lower cost systems more typically found in a small to medium sized business. In a forthcoming study, we will compare the performance of MQSeries and MSMQ running on more expensive systems, with the multiple striped hard drive configuration used and recommended in [4]. However, as that study will show, the differences in hard drive configurations were not the performance gating factors responsible for the differences in MSMQ performance reported here and in [4]. The differences arose from this study's use of a test procedure (described in the next paragraph) that required a given transmitting client thread to await the reply from the message queuing server before transmitting its next message. This scenario stressed the entire client-server response path of each message queuing application. It was therefore deemed more representative of how these applications would perform in real customer environments. When a client thread was allowed to continuously transmit messages without receiving any server replies acknowledging message receipt, MSMQ (and MQSeries) performance numbers were even higher than those reported in [4] (presumably due to this test setup's greater server processor capacity).

A specialized test application was written to send a message of a fixed size to the queue manager on the server, which would then transmit the message back to the client. For each thread running on the client, another message would not be sent to the server until the previously sent message had been successfully received. Therefore, in order to maximize the number of messages transmitted per unit time by a given client, the benchmark application could define multiple (i.e., ***n***)

threads to be run on each client (allowing for up to ***n*** simultaneously outstanding messages per client to exist at any given moment). The benchmark application would then measure the number of successfully transmitted messages for a predefined sampling period, across all threads. In order to allow server queues to reach equilibrium, as well as to guarantee that all threads had started transmitting messages, a waiting period between the start of message transmission and actual measurement collection could be defined.

When multiple clients were used, all were programmed to start transmitting messages at the exact same time. Prior to each test, all client clocks were synchronized to the server's clock. In addition (unless the response queue was deliberately preloaded with a fixed number of messages), the request and response queues within the server were cleared before every test. (If the response queue was not cleared prior to each test, it was observed that MSMQ performance would deteriorate over time; discussed later.)

Reported response times were calculated as the reciprocal of the number of messages per second measured for a single client running a single thread. Because the transmission rate was gated by the round trip response time (as described above), this represented an accurate measure of the response time from a minimally loaded server, with the following caveats:

(1) A single MQSeries client was capable of saturating the server, so that the server load was not necessarily light, but was as low as the test procedures and setup allowed.

(2) For MSMQ, the amount of processing time required of the server for any given message increased with server congestion (due to its extremely inefficient queue searching algorithm; demonstrated and discussed later). The rapidly increasing search times that would contribute to the final measured response time were therefore minimized when only a single client running a single thread was used.

In preliminary runs (prior to collection of the data presented below), the Performance/System Monitor application (part of Windows 2000's Administrative Tools) was used to verify that all 4 server Xeon processors were being utilized (and for example, saturated when MSMQ overall peak server performance numbers were obtained). It was also used to verify that system memory was available and more than adequate for peak capacity testing.

In addition, numerous tuning tests suggested that a sampling period of 5 minutes was optimal in reducing variations in measurements between runs under identical circumstances. They also suggested a 3 minute waiting interval prior to data collection. These values were used for all tests whose results are summarized below.

TEST RESULTS

Effects of Varying the Number of Clients and Threads: Messages per Second

In this section, we analyze in detail the effects of varying both the number of clients and the number of threads per client on the overall number of messages per second that can be processed by the client and server, using either IBM MQSeries 5.1 or MSMQ (Microsoft Message Queue) 2.0. Since performance also varied between the two message queuing solutions with message size, we present the same detailed performance data for 1024 and 65,536 byte messages. 1024 bytes is a frequently occurring (small) message size, while 65,536 bytes is a relatively large message size that typically maximizes network throughput and minimizes client/server processor utilizations in standard network benchmark tests.

The effects on performance of a wide variety of message sizes are examined in the later section **Effects of Varying Message Size: Throughput**.

1024 byte Message Size

Figure 2 compares the performance of IBM's MQSeries 5.1 vs. Microsoft's Message Queue (MSMQ) 2.0 server, using overall server messages per second (mps) as a function of the number of simultaneous threads per client, for 1 through 4 clients. Figure 3 represents the same comparison using 5 through 8 simultaneously transmitting clients. All data is for a message size of 1024 bytes, using express delivery mode.

As Figure 2 demonstrates, at a message length of 1024 bytes, a single IBM MQSeries client running a single thread was so efficient that it was capable of driving the MQSeries server at its peak capacity, which for this system was 498.6 mps. The corresponding response time was 2.0 msec. In contrast, MSMQ server peak performance for this system was only 122.3 mps (i.e., less than 1/4 that of IBM MQSeries). This was achieved using 6 clients running a single thread (see Figure 3). MSMQ single client/single thread response time was 35.9 msec.

This superiority was not limited to peak performance, however. **IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads per client examined.**

Effects of increasing the number of clients: Additional clients yielded no increase in MQSeries performance, since a single client was already capable of saturating this particular server (given the much greater efficiency of the MQSeries client code relative to that of MSMQ).

MSMQ single client peak performance was quite low, i.e., 30.5 mps, which was only 6% of the 498.6 mps achieved by a single IBM MQSeries client. Since this was approximately 1/4 of the capacity of the MSMQ server, overall messages per second increased linearly with added clients, up to about 4 clients. Above 4 clients, slight marginal increases in peak performance were observed. However peak performance actually declined for 8 or more clients.

Effects of increasing the number of threads per client: As was the case for additional clients, additional threads per client yielded no increase in MQSeries performance. As stated above, given the much greater efficiency of the MQSeries client code (relative to MSMQ client code), a single client running a single thread was capable of saturating this particular server. Additional threads per client only added more overhead time for switching between threads, with no possible benefit to overall performance, thereby reducing single client efficiency and subsequent mps and throughput (see the discussion below).

For 1 through 4 clients, MSMQ performance increased slightly and then plateaued as more threads per client were added. This was because a single thread's performance was gated by its round trip response time (i.e., it had to wait for the return of a transmitted message before sending the next one). Therefore, the only way to increase performance was to add more threads. The point at which performance plateaued represented the client's capacity to execute MSMQ client code (a function of the client processor capacity and the code pathlengths).

However, a very sudden large percent decrease in MSMQ performance was observed for 4 clients, when the number of threads per client exceeded 12 (see Figure 2). Likewise, for 5 or more clients, MSMQ performance decreased as additional threads per client were added (see Figure 3). IBM MQSeries performance also declined with additional threads per client, but remained significantly higher than that of MSMQ throughout.

For example, as stated above, overall MSMQ peak performance was 122.3 mps, occurring at 6 clients and 1 thread. At 15 threads per client, however, the performance of 6 simultaneously transmitting clients had declined to only 46.2 mps, or a little over 1/3 of peak performance. In contrast, MQSeries performance at 6 clients and 15 thread was 352.5 mps, over 7 times that of MSMQ. This sharp decline in MSMQ performance was due to the inefficient way in which its clients must search through the response queue to find their messages. The effect of preloading the response queue prior to testing was therefore devastating to MSMQ performance (presented in the next section, **Effects of Preloading the Server Response Queue**). The efficient search algorithm used by MQSeries mitigated any deleterious performance effects due to a large server response queue size.

In general, for both MQSeries and MSMQ, once peak performance had been achieved, overall server messages per second decreased as additional clients and/or threads per client were added. This was due to (a) queuing delays at the server (expected as the load was increased), which depressed the request rate of a given thread by increasing the time between the messages it could send, and (b) additional overhead time associated with switching between threads on a given client (when multiple threads were running).

As stated previously, these MSMQ performance numbers were significantly less than those reported in [4]. However, comparable (in fact higher) numbers were obtained, using the same test setup of Figure 1, by permitting a given client thread to continuously transmit messages without waiting for a return message from the server (as was done in [4]). MQSeries performance numbers were similarly inflated when this test scenario was used. The numbers were higher than those reported in [4] presumably because of the more powerful server used in this study. (The server in this study contained four 400 MHz Xeon processors, as opposed to that used in [4], which contained only a single 200 MHz Pentium Pro processor.) The use of test machines with single SCSI hard drives was not a significant performance gating factor. (A more complex hardware-striped multiple disk architecture was used in [4].) Microsoft's published benchmark data, along with their test setup and procedures, are discussed later in greater detail in the section **Previously Published MSMQ Performance Results**.

(It should be noted that the exact combination of number of clients and threads per client at which peak throughputs are obtained may vary slightly between different setups, and even between tests run on the same setup at different times. Likewise, there can be small experimental variations in the actual values measured between test runs, as would be expected.)

In summary, for a 1024 byte message size, MQSeries provided more than 16 times the single client performance of that of MSMQ. Similarly, MQSeries provided 4 times the overall server capacity of that provided by MSMQ. Furthermore, MQSeries performance was far more robust in the face of server congestion than MSMQ.

Most importantly, the IBM MQSeries solution provided far more messages per second for a given unit of client or server processor utilization than did MSMQ. For the same performance, the IBM solution therefore utilized far fewer cycles of a client or server processor, as compared with the Microsoft solution. The end result to the customer will be far better performance of all the other important applications that are running simultaneously with message queuing.

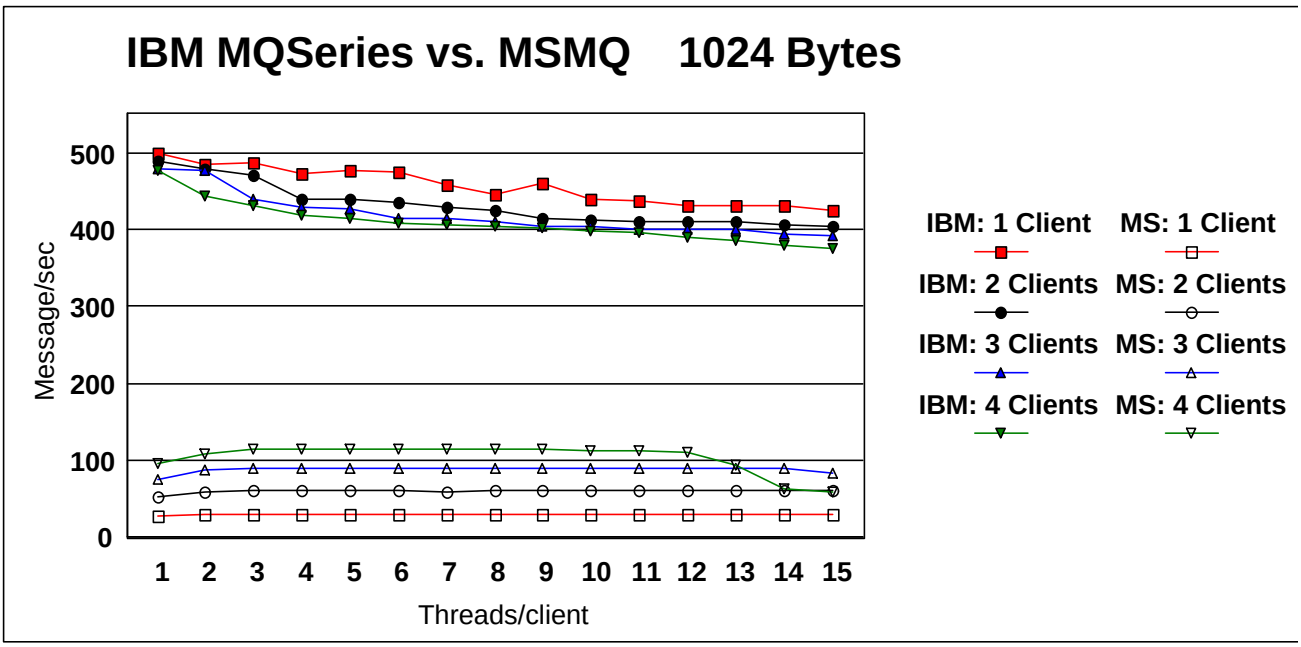

FIGURE 2 IBM MQSERIES vs. MSMQ
MESSAGES/SECOND vs. THREADS/CLIENT
1, 2, 3 and 4 CLIENTS
EXPRESS DELIVERY MODE
1024 BYTE MESSAGE SIZE

IBM MQSeries and MSMQ messages/second (mps) vs. number of threads/client, for one, two, three and four clients communicating to a server, using a 1024 byte message size and express delivery mode.

In the legend above: IBM = IBM MQSeries; MS = Microsoft Message Queue (MSMQ)

MQSeries peak performance = 498.6 mps
MSMQ peak performance = 122.3 mps (See Figure 3)

MQSeries single client/single thread response time = 2.0 msec
MSMQ single client/single thread response time = 35.9 msec

IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads/client examined.

At a message length of 1024 bytes, a single IBM MQSeries client running a single thread was so efficient that it was capable of driving the MQSeries server at its peak capacity.

A sudden large percent decrease in MSMQ performance was observed for 4 clients, as the number of threads/client increased above 12. Because MSMQ single client performance on these machines was so much less than the capacity of the server, overall mps increased linearly with added clients (for less than 12 threads/client).

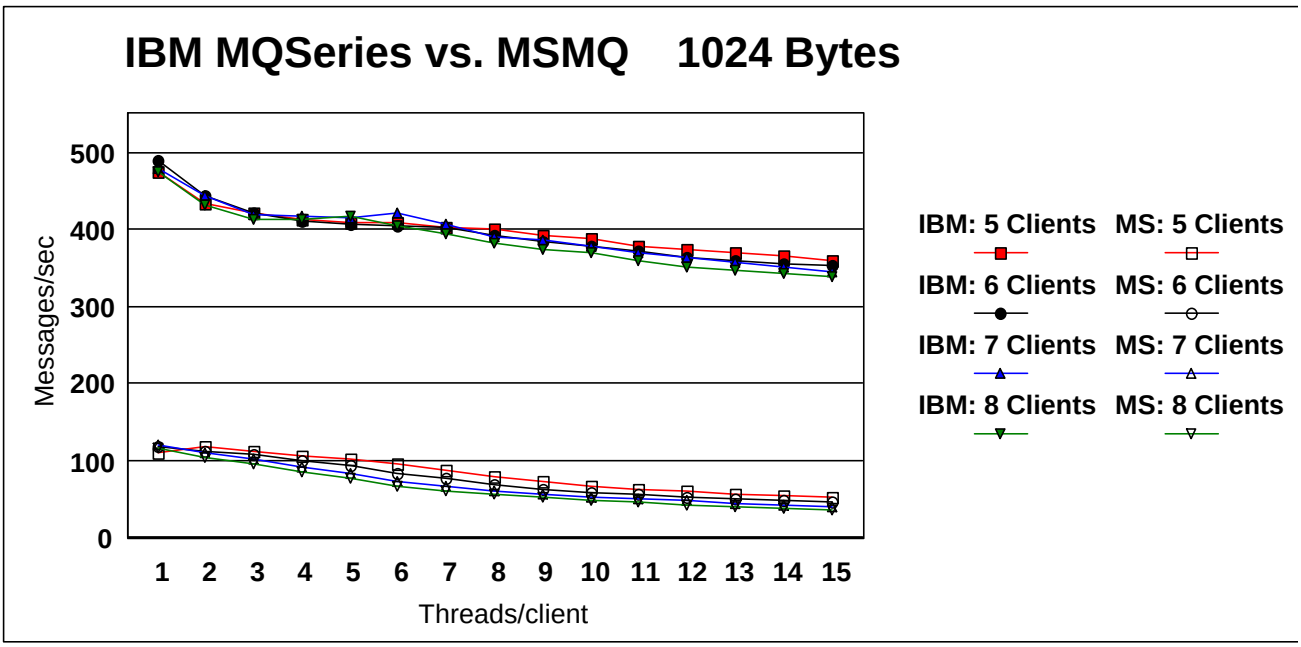

FIGURE 3 IBM MQSERIES vs. MSMQ
MESSAGES/SECOND vs. THREADS/CLIENT
5, 6, 7 and 8 CLIENTS
EXPRESS DELIVERY MODE
1024 BYTE MESSAGE SIZE

IBM MQSeries and MSMQ messages/second (mps) vs. number of threads/client, for five, six, seven and eight clients communicating to a server, using a 1024 byte message size and express delivery mode.

In the legend above: IBM = IBM MQSeries; MS = Microsoft Message Queue (MSMQ)

MQSeries peak performance = 498.6 mps (See Figure 2)
MSMQ peak performance = 122.3 mps

IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads/client examined.

In general (as discussed in the text), overall server mps decreased as additional clients and threads/client were added, due to (a) queuing delays at the server (expected as the load was increased), and (b) additional overhead time associated with switching between threads.

65,536 byte Message Size

Figure 4 compares the performance of IBM's MQSeries vs. MSMQ server, using overall server messages per second (mps) as a function of the number of simultaneous threads per client, for 1 through 4 clients. Figure 5 represents the same comparison using 5 through 8 simultaneously transmitting clients. All data in this section is for a message size of 65,536 bytes, using express delivery mode.

As Figure 4 demonstrates, for a message length of 65,536 bytes, IBM MQSeries achieved its peak performance of 108.9 mps at 2 clients running 2 threads. MQSeries single client/single thread response time was 19.2 msec. In contrast, MSMQ server peak performance for this system was 85.7 mps, achieved using 6 clients running a single thread. MSMQ single client/single thread response time was 45.4 msec. As before, this superiority was not limited to peak performance, since **IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads per client examined.**

For both IBM's and Microsoft's message queuing solutions, the performance as measured in mps (for any given number of clients and threads per client) achieved for 65,536 bytes was always less than that achieved for 1024 bytes. This is as expected, since the larger message size would be expected to require more processing and network transmission time. What Figure 4 does not show, and which will be shown in the later section, **Effects of Varying Message Size: Throughput**, is that system throughputs (obtained by multiplying mps by the message size) increased with larger message sizes. This phenomenon has been commonly observed for a wide variety of performance benchmark tests. Network transmission, client/server processing, etc. times for a given unit of data (e.g., file size, physical frame size, protocol window size, etc.) tend to be dominated by a fixed overhead time, which is more efficiently amortized with increasing data unit size. The result is typically an increase in throughput with data unit size, up to a point, with a simultaneous decrease in packets, messages, transactions, etc. per second.

Once again, since MSMQ single client capacity (indicated by single client peak mps) was much less than that of its server (i.e., approximately 29% of server capacity), overall messages per second increased linearly with 2 and 3 added clients, with a substantial (sublinear) increase when a fourth client was added. Slight marginal increases in peak performance were observed for 5 and 6 additional clients. However, peak performance actually declined for 7 or more clients. While adding a second client when only a few threads were running yielded a slight increase in MQSeries performance, single client performance was already very close to peak server capacity.

For 1 through 3 clients, MSMQ performance increased slightly, plateaued and then decreased slightly as more threads per client were added. For 4 or more clients, MSMQ performance decreased precipitously as threads per client increased. This was similar to the behavior observed above for 1024 byte messages when the product of the number clients and threads per client became large. In contrast, following a very small decrease, MQSeries performance was extremely stable as the number of threads per client (and/or product of the number of clients and threads per client) increased. As stated previously, observed performance degradation with an increase in the number of clients and/or threads per client was due to increasing server congestion (and subsequent growth of the server response queue length) and/or cumulative thread switching overhead. Because of its inefficient response queue searching algorithm, MSMQ was extremely sensitive to growth in the size of the server response queue.

In summary, for a 65,536 byte message size, MQSeries provided almost 4 times the single client performance of that of MSMQ. Similarly, MQSeries provided 27% more overall server capacity when compared to MSMQ. Furthermore, as was the case for a 1024 byte message size, MQSeries performance was far more robust in the face of server congestion than MSMQ. MSMQ performance declined dramatically as the server load increased with a large number of simultaneous clients and/or threads per client. Once again, the overall benefit of MQSeries was far greater performance, at less processor cost, which will provide customers with better system response times and better overall performance of all their other simultaneously running applications.

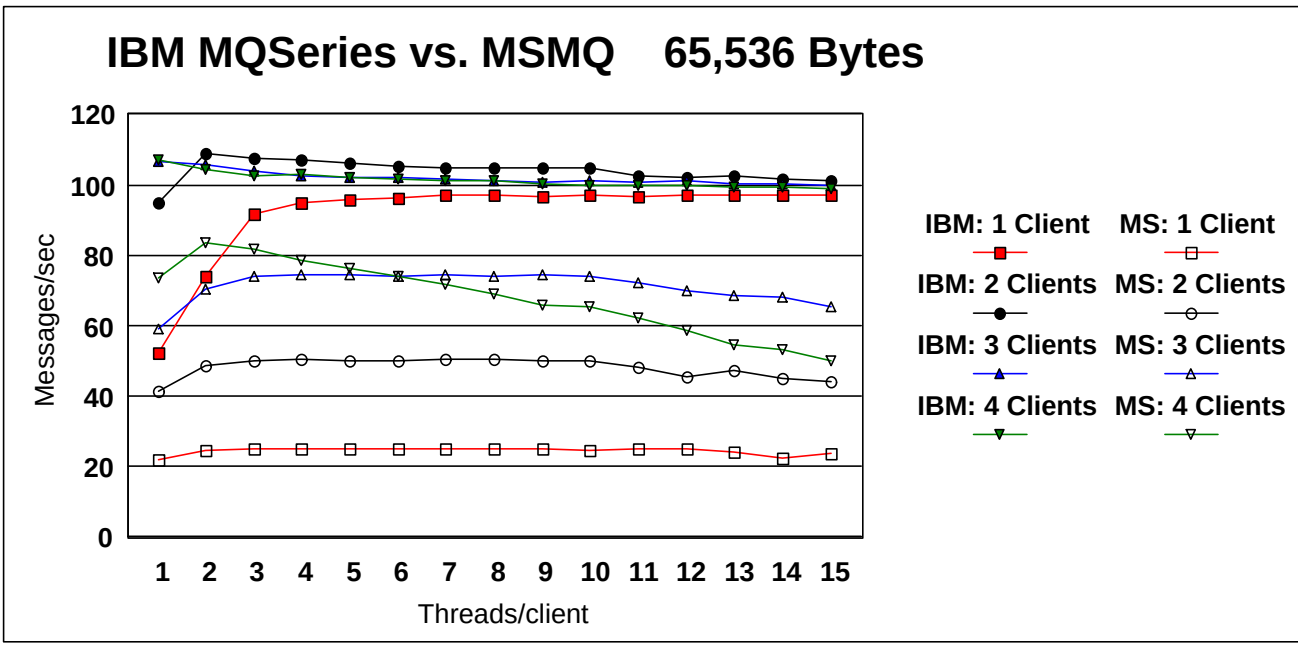

FIGURE 4 IBM MQSERIES vs. MSMQ
MESSAGES/SECOND vs. THREADS/CLIENT
1, 2, 3 and 4 CLIENTS
EXPRESS DELIVERY MODE
65,536 BYTE MESSAGE SIZE

IBM MQSeries and MSMQ messages/second (mps) vs. number of threads/client, for one, two, three and four clients communicating to a server, using a 65,536 byte message size and express delivery mode.

In the legend above: IBM = IBM MQSeries; MS = Microsoft Message Queue (MSMQ)

MQSeries peak performance = 108.9 mps
MSMQ peak performance = 85.7 mps (See Figure 5)

MQSeries single client/single thread response time = 19.2 msec
MSMQ single client/single thread response time = 45.4 msec

IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads/client examined.

At a message length of 65,536 bytes, IBM MQSeries client was so efficient that peak performance was achieved with only 2 clients running 2 threads. Performance was stable as more clients and/or threads were added.

In contrast, a very sudden large percent decrease in MSMQ performance was observed for 4 clients, as the number of threads/client increased above 2 (see also Figure 5, for 5-8 clients). Because MSMQ single client performance on these machines was so much less than the capacity of the server, overall mps increased linearly as up to 3 clients were added.

For both, peak performance was less than that obtained for 1024 byte message sizes due to increased processing times associated with larger message sizes.

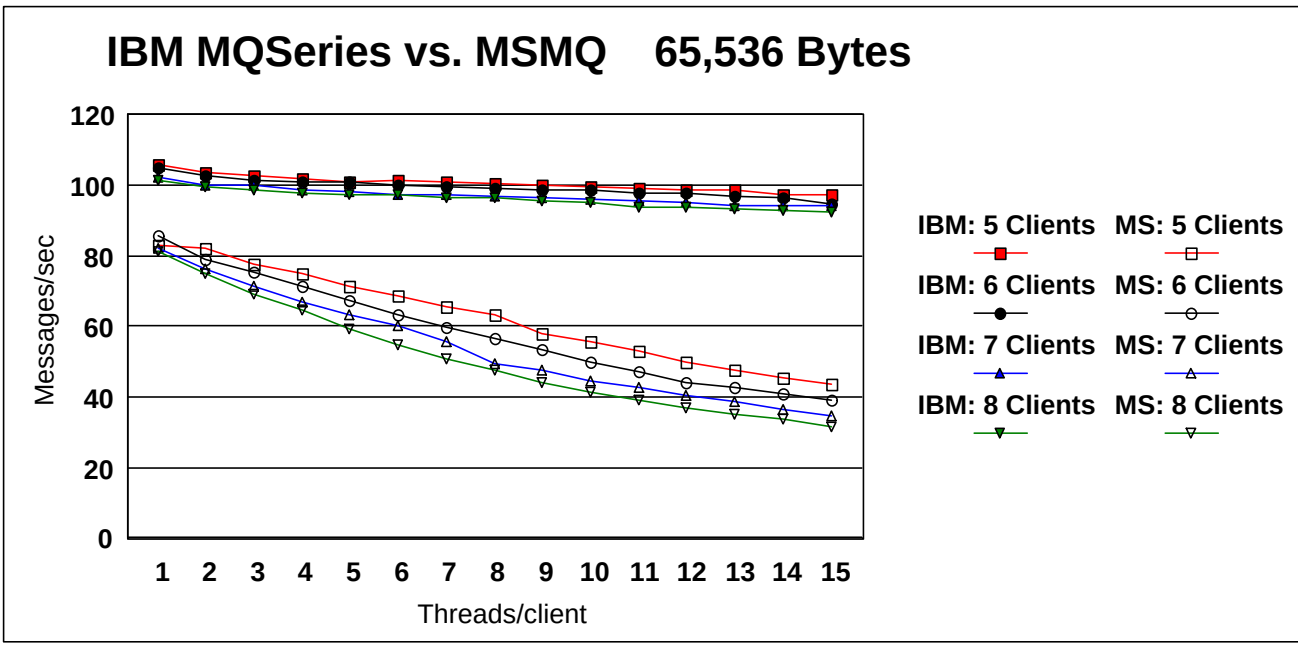

FIGURE 5 IBM MQSERIES vs. MSMQ
MESSAGES/SECOND vs. THREADS/CLIENT
5, 6, 7 and 8 CLIENTS
EXPRESS DELIVERY MODE
65,536 BYTE MESSAGE SIZE

IBM MQSeries and MSMQ messages/second (mps) vs. number of threads/client, for five, six, seven and eight clients communicating to a server, using a 65,536 byte message size and express delivery mode.

In the legend above: IBM = IBM MQSeries; MS = Microsoft Message Queue (MSMQ)

MQSeries peak performance = 108.9 mps (See Figure 4)
MSMQ peak performance = 85.7 mps

IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads/client examined.

In general (as discussed in the text), overall server mps decreased as additional clients and threads/client were added, due to (a) queuing delays at the server (expected as the load was increased), and (b) additional overhead time associated with switching between threads.

For both, peak performance was less than that obtained for 1024 byte message sizes due to increased processing times associated with larger message sizes.

Effects of Preloading the Server Response Queue

In this section, we analyze the effects of preloading the server response queue with a fixed number of messages, prior to measuring performance (using the same setup and test procedures as outlined above). It was observed that when the response queue was not cleared prior to each performance test (for a given parameter pair of number of clients and threads per client), MSMQ performance deteriorated over time. It was determined that occasionally, for a given test, a small number of unretrieved messages remained in the response queue following test termination. These messages were not retrieved in subsequent runs (e.g., messages left by client number 8, used only when 8 clients were tested, could not be retrieved during later tests of 1 through 7 clients). Therefore, unretrieved messages slowly accumulated in the response queue over multiple test runs. When the response queue was systematically cleared prior to each test, MSMQ performance remained stable over time. MQSeries performance was not significantly affected by this process of unretrieved message accumulation.

These initial observations suggested that, over the course of the normal operation of a message queuing server, unretrieved messages should be expected to slowly accumulate with time in the server response queue. These messages would result from their source clients shutting down (perhaps for days or weeks due to vacation), crashing, etc., prior to their retrieval. Our test process simply accelerated the rate at which these unretrieved messages would accumulate.

Therefore, as stated above, in order to optimize MSMQ performance, the server response and request queues were systematically cleared prior to each performance test. This was done for all testing reported in the section **Effects of Varying the Number of Clients and Threads: Messages per Second** above, as well as in the section **Effects of Varying Message Size: Throughput** below. However, given that unretrieved messages would be expected to naturally accumulate over time in any operational message queuing server's response queue, a series of tests was deemed necessary to determine the performance effects of preloading the response queue with a precise number of messages.

For the data presented below, prior to each test, both the request and response queues were cleared. A separate client, not involved in the subsequent performance testing, was then used to directly preload the response queue with a fixed number of messages. The client was then immediately shut off without retrieving the messages. The messages therefore sat in the response queue, awaiting retrieval by the (now unresponsive) client throughout subsequent testing. It was verified that the desired number of preloaded messages was in fact delivered to the response queue, using Administrative Tools for MSMQ and MQSeries Explorer for MQSeries.

A message size of 1024 bytes was used throughout this particular test (for comparison purposes with the performance optimized data previously presented above). The parameter pair of number of clients and threads per client that yielded the peak performance for MSMQ at this message size was 6 clients running a single thread each. This combination was therefore used throughout. The response queue was preloaded with 0, 50, 100, 150, 200, 300, 500 and 1000 messages.

Table 3 and Figure 6 summarize the effects of this preloading on MQSeries and MSMQ performance, presenting messages per second (mps) as a function of the initial response queue size (in messages). Table 3 also presents the ratio of MQSeries to MSMQ mps, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ.

As the data indicates, **MSMQ performance was devastated by even a very small number of initial messages in the response queue**. For an initial response queue size of 50 messages, MSMQ performance dropped to 71.3 mps, a decrease of 41.7% from its peak of 122.3 mps (the latter achieved when no messages were preloaded). **In contrast, MQSeries performance remained unaffected,** yielding 489.9 mps. When the initial response queue size was preloaded with 1,000 messages, MSMQ performance virtually collapsed, yielding barely 7.4 mps. In contrast, MQSeries performance was 376.5 mps.

These results are consistent with the effects that were observed above when the server became congested (i.e., when the product of the number of clients and threads per client became large). Under those conditions, MSMQ performance was observed to decline precipitously.

Table 3. Messages/sec vs. Initial Request Queue Size

Initial Request Queue Size (Messages)	Messages per second (mps)		Ratio of MQSeries to MSMQ mps
	IBM MQSeries	MSMQ	MQSeries/MSMQ
0	498.6	122.3	4.1
50	489.9	71.3	6.9
100	484.7	49.4	9.8
150	482.2	37.6	12.8
200	475.1	30.2	15.7
300	471.2	21.6	21.8
500	441.3	14.1	31.3
1,000	376.5	7.4	50.9

Table 3. MQSeries and MSMQ performance (measured in messages per second) as the server response queue was preloaded, prior to testing, with a fixed number of (unretrievable) messages.

This test scenario mimicked expected conditions within an actual message queuing customer environment. Also included is the ratio of MQSeries to MSMQ performance, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ (e.g., **at 1,000 preloaded messages, MQSeries performance was 50.9 times better than that of MSMQ, or 5,090% better**). This ratio increased as the number of preloaded messages increased, due to the collapse of MSMQ performance. 1024 byte messages were used throughout, with 6 clients running a single thread each. This was the parameter combination that maximized MSMQ peak server performance.

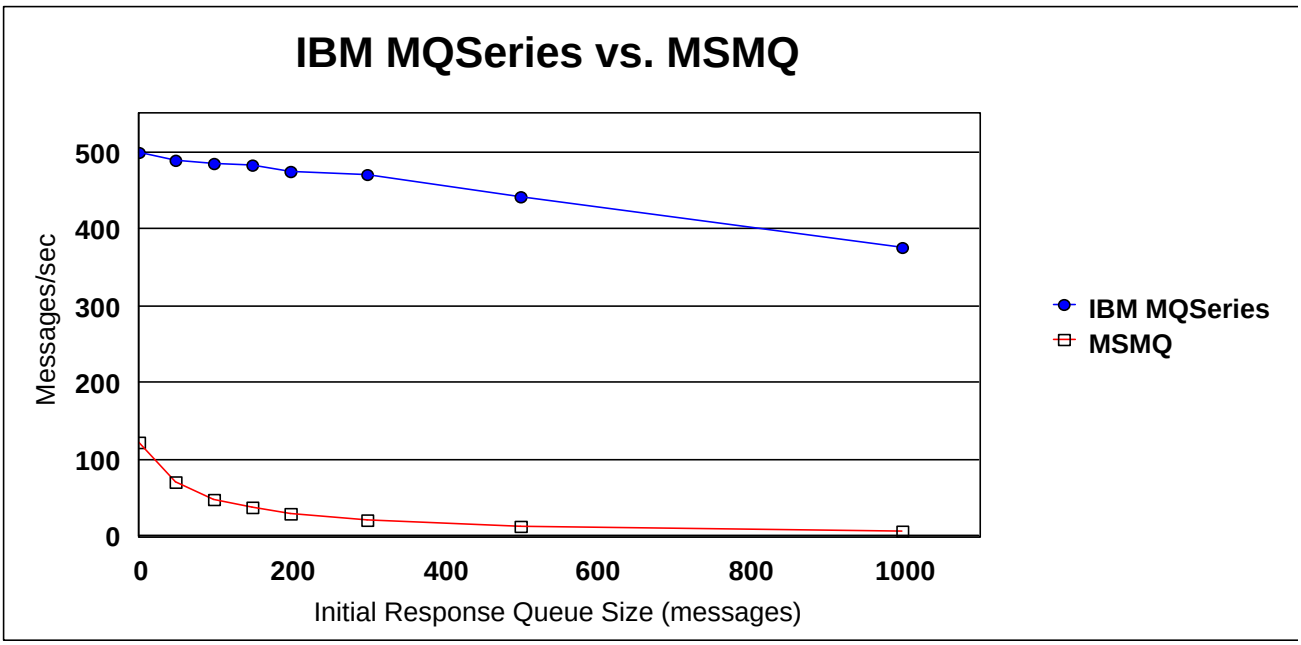

FIGURE 6 IBM MQSERIES vs. MSMQ
MESSAGES/SECOND vs. INITIAL RESPONSE QUEUE SIZE
6 CLIENTS, 1 THREAD/CLIENT
EXPRESS DELIVERY MODE
1024 BYTE MESSAGE SIZE

IBM MQSeries and MSMQ messages/second (mps) vs. initial response queue size, using six clients running a single thread each (the parameter combination that maximized MSMQ peak server performance), a 1024 byte message size and express delivery mode.

For every initial response queue size (Q_0) examined, MQSeries substantially outperformed MSMQ. MSMQ performance completely collapsed as the initial queue size increased.

Q_0 = 0 messages	**MQSeries performance =**	**498.6 mps**
	MSMQ performance =	**122.3 mps**
Q_0 = 100 messages	**MQSeries performance =**	**484.7 mps**
	MSMQ performance =	**49.4 mps**
Q_0 = 1,000 messages	**MQSeries performance =**	**376.5 mps**
	MSMQ performance =	**7.4 mps**

Effects of Varying Message Size: Throughput

In this section, we rigorously analyze the effects of varying the message size on the overall throughput (measured in kilobytes per second, or KBps) that can be processed by the server, using either IBM MQSeries or MSMQ. Throughput was calculated here as the number of messages processed per second (mps) times the message size. As suggested above, mps decreased with increasing message size. Throughput, on the other hand, actually increased with increasing message size (as the data below will show).

The message sizes examined were 100, 512, 1024 5120, 10240, 20480, 30720, 40960 and 65536 bytes. Detailed performance measurements were made for each of these message sizes. These measurements were identical in nature to those taken in the section **Effects of Varying the Number of Clients and Threads: Messages per Second**, i.e., using 1 through 8 clients, each running 1 through 15 threads. The comparative results were the same as those reported in the previous section, i.e., **for every message size examined: (1) IBM MQSeries outperformed MSMQ for every combination of number of simultaneous clients and threads per client examined, and (2) MSMQ performance deteriorated significantly as the number of clients and/or threads per client became sufficiently large**.

To avoid unnecessary repetition, detailed plots of mps vs. threads per client for different numbers of clients are not presented here. (They are, however, available upon request. Please contact the authors.) Instead, in the sections that follow, we examine (a) *peak throughput* obtained for a *single client* (i.e., the maximum throughput observed over 1 through 15 threads), and (b) *overall peak throughput* sustained by the *server* (i.e., the maximum over all combinations of number of clients and threads per client). These two peak throughputs are presented below as functions of message size. Based on all the data collected in these series of tests, performance differences between MQSeries and MSMQ were best characterized by these two peak capacity metrics.

Single Client Performance

Figure 7 represents single client peak throughputs (kilobytes per second, or KBps) for IBM's MQSeries and MSMQ, as a function of message size, for 100, 512, 1024 5120, 10240, 20480, 30720, 40960 and 65536 byte message sizes. All data was collected using express delivery mode. For a given message size, single client peak throughput was defined to be the maximum throughput measured for a single client over all number of threads examined, i.e., over 1 through 15 threads. Single client peak throughputs were always achieved over this range of number of threads.

Table 4 summarizes the individual peak throughput values obtained for each message size examined. The number of threads running on a single client at which the peak performance was achieved is given in parentheses after the throughput value. Table 4 also presents the ratio of MQSeries to MSMQ throughput, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ.

As Figure 7 and Table 4 demonstrate, IBM MQSeries and MSMQ single client peak throughput increased with increasing message size. **They also demonstrate that for every message size examined, single client peak performance for IBM's MQSeries was significantly greater than that of MSMQ.** This was consistent with what was observed above in the more detailed single message size plots given in Figures 2 and 4 (for message sizes of 1024 and 65,536 bytes respectively). MQSeries consistently provided several times the peak single client performance of that offered by MSMQ, for every message size examined.

Please note that throughout, 1 kilobyte per second (1 KBps) of throughput is equal to 1,000 bytes per second (and not 1024 bytes per second).

Table 4. Single Client Peak Throughput vs. Message Size

Message Size (bytes)	Throughput (kilobytes per second)		Ratio of MQSeries to MSMQ Throughput
	IBM MQSeries	MSMQ	MQSeries/MSMQ
100	51.0 (2)	3.0 (4)	17
512	270.0 (1)	15.5 (3)	17.4
1,024	510.6 (1)	31.3 (4)	16.3
5,120	1,575.1 (2)	155.0 (9)	10.5
10,240	2,802.9 (3)	303.3 (3)	9.3
20,480	4,363.1 (3)	591.0 (4)	7.4
30,720	5,548.2 (3)	855.6 (14)	6.5
40,960	6,234.9 (5)	1,104.3 (8)	5.7
65,536	6,372.1 (12)	1,655.2 (11)	3.9

Table 4. IBM MQSeries and MSMQ single client peak throughputs obtained for a given message size. The corresponding number of threads at which the listed peak throughput was obtained is given in parentheses.

MQSeries performance substantially exceeded that of MSMQ, for every situation examined. For a given message size, single client peak throughput was the maximum throughput achieved over all threads examined (i.e., over 1 through 15 threads). The maximum was always achieved over this range of number of threads.

Also included is the ratio of MQSeries to MSMQ performance, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ (e.g., **for a 1024 byte message size, MQSeries performance was 16.3 times better than that of MSMQ, or 1,630% better**).

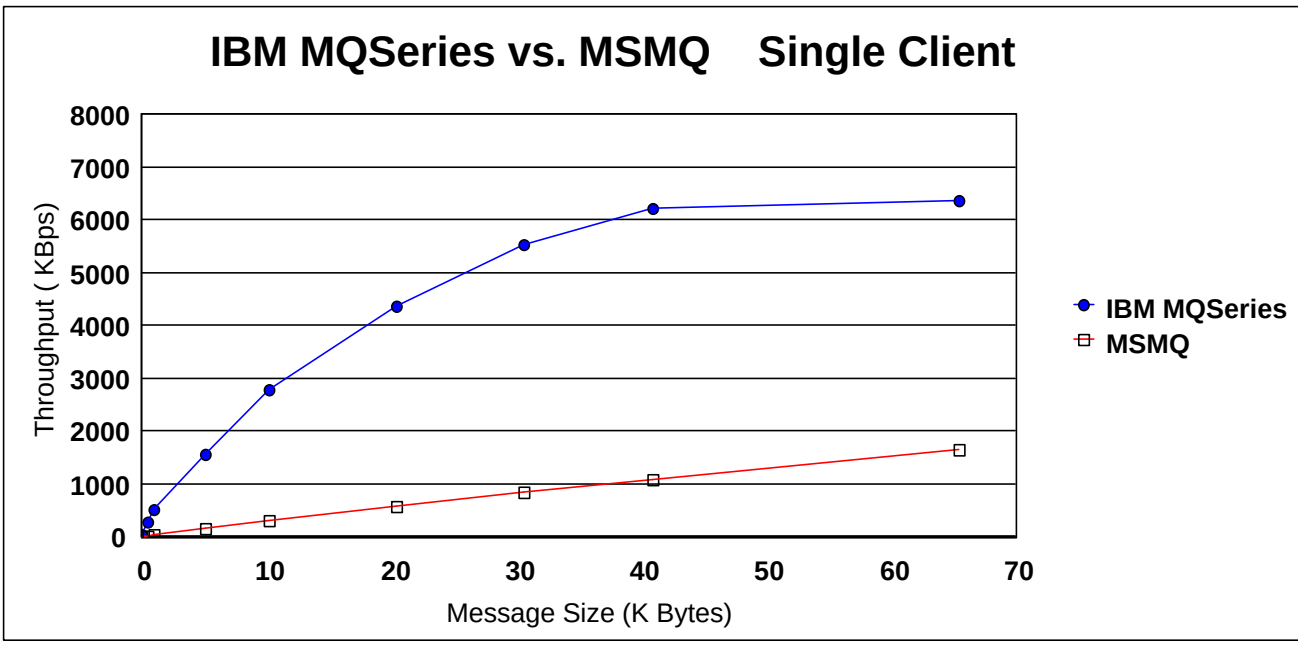

FIGURE 7 IBM MQSERIES vs. MSMQ
PEAK THROUGHPUT vs. MESSAGE SIZE
100 through 64K BYTE MESSAGE SIZES
1 CLIENT
EXPRESS DELIVERY MODE

IBM MQSeries and MSMQ single client peak throughput (kilobytes per second, or KBps) vs. message size (100, 512, 1024 5120, 10240, 20480, 30720, 40960 and 65536 bytes), obtained with a single client communicating to a server using express delivery mode. The throughput plotted for each message size represents the maximum obtained over all examined number of threads (i.e., over 1 through 15 threads), running on a single client. For all message sizes examined, the single client peak throughput always occurred within this range of number of threads.

For every message size examined, MQSeries substantially outperformed MSMQ. For both applications, throughput increased with increasing message size (in contrast to messages per second, which decreased with increasing message size; see Figures 2 and 4).

Overall Server Capacity (Multiple Clients)

Figure 8 represents overall peak server throughputs (kilobytes per second, or KBps) for IBM's MQSeries and MSMQ, as a function of message size, for 100, 512, 1024 5120, 10240, 20480, 30720, 40960 and 65536 byte message sizes. All data was collected using express delivery mode. For a given message size, overall peak server throughput was defined to be the maximum throughput measured over every examined combination of number of clients and threads per client (i.e., over 1 through 8 clients, running 1 through 15 threads per client). Peak throughputs were always achieved over this range of combinations.

Table 5 summarizes the individual peak throughput values obtained for each message size examined. Table 5 also presents the ratio of MQSeries to MSMQ throughput, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ.

As was the case for single client peak throughput, Figures 8 and Table 5 demonstrate that IBM MQSeries and MSMQ overall peak server throughput increased with increasing message size. And as before with single client peak throughput, they also demonstrate that **overall peak server performance for IBM's MQSeries was significantly greater than that of MSMQ for every message size examined**. MQSeries consistently provided several times the peak server performance of that offered by MSMQ.

Table 5. Overall Peak Server Throughput vs. Message Size

Message Size (Bytes)	Throughput (kilobytes per second		Ratio of MQSeries to MSMQ Throughput
	IBM MQSeries	MSMQ	MQSeries/MSMQ
100	51	12	4.3
512	270	60.7	4.4
1,024	510.6	125.3	4.1
5,120	1,587.8	594.8	2.6
10,240	2,802.9	1,129.9	2.5
20,480	4,363.1	2,182.5	2
30,720	5,548.2	3,040.6	1.8
40,960	6,542.1	3,938.2	1.7
65,536	7,135.6	5,617.8	1.3

Table 5. IBM MQSeries and MSMQ overall peak server throughputs obtained for a given message size.

MQSeries performance substantially exceeded that of MSMQ, for every situation examined. For a given message size, overall peak server throughput was the maximum throughput achieved over all combinations of number of clients and threads per client examined (i.e., over 1 through 8 clients, running 1 through 15 threads per client). It represented the overall server capacity to process messages This maximum was always achieved over the range of combinations that were examined.

Also included is the ratio of MQSeries to MSMQ performance, indicating the number of times (not percent) that MQSeries performance was better than that of MSMQ (e.g., (e.g., **for a 1K message size, MQSeries performance was 4.1 times better than that of MSMQ, or 410% better**).

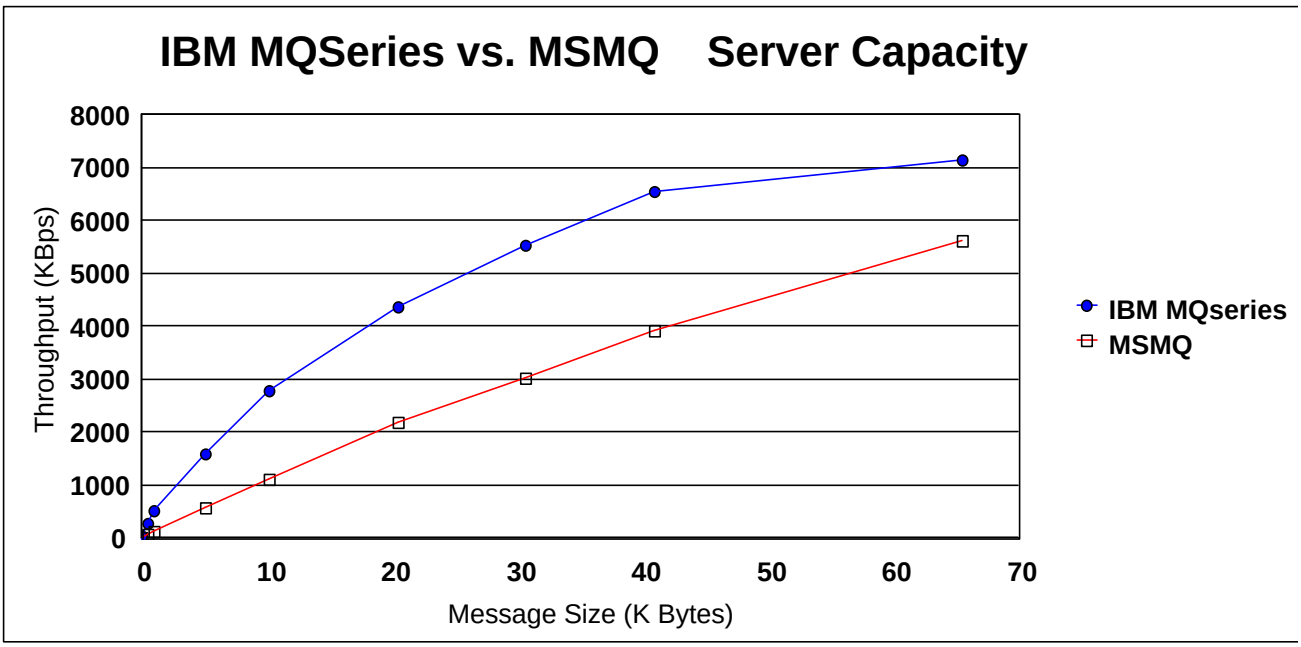

FIGURE 8 IBM MQSERIES vs. MSMQ
PEAK SERVER THROUGHPUT vs. MESSAGE SIZE
100 through 64K BYTE MESSAGE SIZES
EXPRESS DELIVERY MODE

IBM MQSeries and MSMQ overall peak server throughput (kilobytes per second, or KBps) vs. message size (100, 512, 1024 5120, 10240, 20480, 30720, 40960 and 65536 bytes), using express delivery mode. The throughput plotted for each message size represents the maximum obtained over all examined combinations of number of clients and threads/ client (i.e., over 1 through 8 clients, running 1 through 15 threads/client). For all message sizes examined, the overall peak server throughput always occurred within this range of combinations.

For every message size examined, MQSeries substantially outperformed MSMQ. For both applications, throughput increased with increasing message size (in contrast to messages per second, which decreased with increasing message size; see Figures 2 through 5).

Previously Published MSMQ Performance Results

In [4], Microsoft reported substantially larger MSMQ performance numbers (messages per second), than those reported here in our study. No corresponding MQSeries performance numbers were included in [4]. As we will discuss below, the benchmark test procedures employed in Microsoft's analysis were designed to artificially inflate resulting MSMQ performance numbers, while avoiding the very serious performance deficiencies that this current study has uncovered. Microsoft's tests were therefore highly misleading, with potentially disastrous consequences to any e-business solution designer who might base his design choices on that report's conclusions. The deficiencies demonstrated in our study suggest that MSMQ can provide extremely poor performance in many actual customer environments. These deficiencies also seriously call into question MSMQ's ability to scale with the growing customer demands that would arise from a typical thriving e-business.

The differences between this performance study and Microsoft's [4] are the following:

(1) *Message Generation and Processing:* In the Microsoft "Networked Machine Scenario" described in [4] (the relevant scenario here), the client operated independently of the server. It was initially disconnected from the server and allowed to accumulate messages, that were later blasted over to the server when the two machines were reconnected. This procedure was justified, according to Microsoft, because it provided "...consistent, conservative performance information..." [4]. Server processing was minimized after receipt of the message. No return message was subsequently transmitted to the client (as when, in an actual customer environment, processed data or information would be sent back to the client).

In our study, the server was required to transmit a message back to the client, following the receipt of the initial message sent to the server by the client. As such, the full client and server code paths were exercised. This test scenario therefore mimicked a real customer's typical use of a message queuing application, i.e., whereby a message is sent by a client to a (possibly remote) server requesting some service, the results of which are sent back in a return message.

The test scenario chosen by Microsoft therefore minimized any application processing costs incurred by the client and server. Yet, as suggested above, these are the very costs that define the final real performance of any message queuing application in an actual customer environment. Their tests were more indicative of the capacity of their network adapters, than of the capacity of their message queuing server to provide meaningful processing of incoming messages (that would then yield return messages back to the requesting clients).

Using the setup of Figure 1, we were able to generate substantially larger MSMQ performance numbers that were, in fact, greater than those reported in [4]. This was accomplished by simply removing the requirement that a given client thread wait for the acknowledgment of a previously transmitted message before sending another one. MQSeries performance numbers were of course similarly inflated.

(2) *Shared Queues*: More importantly, in the Microsoft study, only a single message-transmitting client was examined. Every message in the server response queue therefore belonged to that client.

In our study, multiple clients (each running one or more threads) shared a single pair of request and response queues. A given client was therefore required to search for its messages from a common queue that contained competing messages from other clients. This also allowed the server response queue to be preloaded with a fixed number of competing messages, thereby increasing the number of messages through which a given client would be required to search.

For both message queuing applications, practical limitations exist on the number of simultaneous independent message queues that can be supported. In a real customer environment containing

thousands of clients, this can preclude assigning a single dedicated set of request and response queues to every client. Therefore, though both applications recommend dedicated queues for performance optimization, some clients realistically will be required to share a common message queue. As a result, the ability of a client to quickly and efficiently locate its message in a shared queue becomes an important determinant in the ability of a message queuing solution to scale with an increasing client population size. (Note that in our study, only up to 8 separate clients sharing a single pair of request and response queues were examined. The number of clients sharing a common queue was therefore not very large. The server response queue preloading tests reported above used only 6 clients, with a seventh providing the preloaded messages.)

As the data presented in our study showed, MSMQ failed miserably when required to search for its messages in a common heterogeneous queue. Its performance collapsed as the size of the sever response queue increased, due either to server congestion (naturally resulting from high user demand) or deliberate preloading. This was because a given MSMQ client had to search through every prior message in the shared queue before it could locate its own message(s). IBM's MQSeries, on the other hand, uses a *correlation identifier* (i.e., a tag) that allows a given client to quickly retrieve its messages from the shared response queue without an exhaustive search. For obvious reasons, Microsoft's tests avoided situations where this weakness would be exposed.

Because MSMQ performance was so sensitive to even a very small number of messages accumulating in its server response queue, our study required that the message queues be cleared prior to each test. Without this, MSMQ performance was observed to significantly degrade over the course of testing. (MQSeries was not appreciably affected.) Unfortunately, the accumulation of messages in a response queue would occur naturally over time in a real customer environment, where Microsoft would not have the system initialization benefits provided by the test procedures used in our study. This seriously compromises MSMQ's ability to scale to the larger client population sizes often found in actual customer environments. Furthermore, MSMQ completely consumed the server processor as a given client searched for its messages in the shared queue. In the real customer world, this would mean that the processor would necessarily be kept from executing important e-business or other applications. The performance of these other applications would therefore suffer.

(3) *Hardware Considerations*: In the Microsoft "Networked Machine Scenario" of [4], their client/server test machines were configured with extremely generous high performance hard drive architectures (i.e., 8 hardware striped disks with a Compaq[5] SmartArray[5] 2 controller on the client machine, and 10 disks on the server, including 3-way and 6-way hardware striped disks with Compaq SmartArray 2 controllers). In [4], Microsoft reported that it "... measured 10 times better message throughput performance from a three-way hardware-striped disk with a 2-MB battery-protected controller as opposed to a standard SCSI II disk."

In our study, the server and clients were configured with a single SCSI drive. However, disk architecture was not the actual performance gating factor in our study. As discussed above, the performance differences reported in our study and in [4] were due primarily to the differences in test procedures. Two separate sets of tests described below confirmed this.

In the first set of tests (not summarized above), preliminary measurements were taken using our more realistically stressful test procedures, on machines configured with disk architectures similar to those used in [4]. These tests yielded performance numbers comparable to those reported above in Figures 2 through 8, and Tables 1 through 5. (These results will be published in a subsequent paper.)

In the second set of tests (previously described in the section above entitled *Message Generation and Processing*), we removed the requirement that a given client thread wait for the acknowledgment of a previously transmitted message before sending another one. Performance measurements obtained using the test setup of Figure 1 were comparable to, and in fact higher than, those reported for the setup used in [4]. The better performance numbers were expected,

since the client and server processors used in our study were significantly more powerful than those used in [4]. (The server in our study contained four 400 MHz Xeon processors, while each client contained a single 300 MHz Pentium II processor. In [4], both server and client contained only a single 200 MHz Pentium Pro processor. It should be noted that, since MQSeries and MSMQ were tested on the identical setup, both applications benefited equally.)

In summary, the benchmarking strategy used by Microsoft (with some regularity) is to create an extremely complex test setup, often using test procedures that are unrepresentative of an actual customer environment. The results are inflated performance numbers that are publicly reported with great fanfare. (Their highly publicized SQL Server[1] TPC-C results measured on the mother-of-all testbeds is a recent example.) Embarrassing head-to-head comparisons (such as this one) are understandably avoided. In the past, the licensing agreements of many Microsoft products have contained prohibitions of published head-to-head performance comparisons, but the Windows 2000 Server (of which MSMQ is a part) has no such prohibition.

CONCLUSIONS

As the data has shown, MSMQ has a number of very serious performance problems which, understandably, were avoided by Microsoft in its previously published benchmark reports. These problems compromise MSMQ's ability to effectively scale for large customer environments. Furthermore, because MSMQ can quite easily consume the server's processor, as it inefficiently searches for a given client's messages in its response queue, other key applications can be left idly waiting for the processor to become available. The overall effect is poor performance of other simultaneously running applications. This is a cost that most e-businesses cannot afford to pay.

In contrast, MQSeries performance was several times that of MSMQ, for every combination of parameters examined. This performance was stable as the response queue size increased. Therefore, MQSeries will scale effectively as the user population grows, without the catastrophic collapse in performance that was observed for MSMQ. This scalable high performance guarantees that your server resources can be used for what they were originally intended, i.e., to provide your users and/or customers with applications and services that run smoothly and continuously, with excellent response times.

REFERENCES

[1] **Advanced Messaging Applications with MSMQ and MQSeries,** Rhys Lewis, QUE Professional (Macmillan), Indianapolis IN, 1999.

[2] "MQSeries for Window NT - V5.1: Capacity planning guidance", Tim Pickrell, Edition 1.0, February 8, 2000.

[3] Getting the most out of MQSeries", Richard G. Nikula, BMC Software White Paper, 2000, see http://www.bmc.com/available_doc.html?item_no=100029505

[4] "Optimizing Windows NT Server performance in a Microsoft Message Queue Server environment", Microsoft TechNet White Paper, January 17, 2000 (last updated), see http://www.microsoft.com/TechNet/winnt/msmqperf.asp .

[5] "HOWTO: Optimize MSMQ performance", Microsoft product Support Services article ID Q19942B, January 23, 1999 (last reviewed), see http://support.microsoft.com/support/kb/articles/Q199/4/28.ASP

NOTES

[1] Trademarks of Microsoft Corp.

[2] Trademarks of IBM Corp.

[3] Trademarks of Intel Corp.

[4] Trademarks of Black Box Corp.

[5] Trademarks of Compaq Computer Corp.