High-Performance JMS Messaging

A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ

Crimson Consulting Group, Inc.
4970 El Camino Real
Los Altos, CA 94022
650.960.3600

650.960.3737 (fax)
www.crimson-consulting.com
info@crimson-consulting.com

© 2003 Crimson Consulting Group

About Crimson Consulting

Crimson provides marketing strategy and implementation consulting to technology companies.

Crimson can manage client engagements and take responsibility for their successful completion, or our clients can manage our expert consultants directly. Many engagements focus on one of our specialized practice areas:

- Market Assessment - evaluating markets, competitive analyses, gathering intelligence
- Partner Programs - identifying optimal partners, providing best practices
- Strategic Sales Tools - developing ROI models, TCO tools, and decision-making maps

For situations where the client would like to manage the consultants directly, we provide interim marketing VPs, Directors, and Managers in the areas of product marketing, product management, channel marketing, marketing communications, market research, etc.

For more information, contact Crimson Consulting Group at 650-960-3600 x117 or info@crimson-consulting.com.

Copyright© 2003 Crimson Consulting Group, Inc. All Rights Reserved.

IBM and WebSphere MQ are registered trademarks of IBM Corporation. Sun Microsystems, Sun ONE, Java, UltraSparc and Solaris are trademarks or registered trademarks of Sun Microsystems Inc.

Table of Contents

Introduction **1**
Sun Java System Message Queue .... **Error! Bookmark not defined.**
IBM WebSphere MQ 1

Test Description and Methodology **2**

Testing Variables **2**
Point-to-Point 2
Publish-and-Subscribe 2
Acknowledgements 3
Persistence 3
Selectors 3
Durable Subscriptions 3

Results **4**
Queue, Non-Persistent, No Selectors 4
Queue, Persistent, No Selectors 4
Queue, Non-Persistent, with Selectors 5
Queue, Persistent, with Selectors 5
Publish/Subscribe, Non-Persistent, No Selectors 6
Publish/Subscribe, DUPS_OK, Persistent, No Selectors 6
Publish/Subscribe, AUTO_ACK, Persistent, No Selectors 7
Publish/Subscribe, Durable, Persistent, with Selectors 7

Conclusion **8**

Appendices **9**
Appendix A—Complete List of Tests 9
Appendix B—Complete Test Results 11
Appendix C—Test Environment 13

© 2003 Crimson Consulting Group

Introduction

Application integration is a mission critical requirement for delivering successful e-business solutions and for maximizing the potential of every new application that is developed and deployed. The application integration required for e-business involves integrating data and business processes among a wide range of legacy systems, ERP solutions, and custom applications both internal and external to the organization.

Over the past several years, enterprise messaging systems, sometimes called message oriented middleware (MOM), have offered a flexible, fast, and well-connected means of exchanging information among applications without the need to build large numbers of individual interfaces. Acting as an intermediary, MOM systems exchange data (messages) between applications within a store-and-forward framework. Enterprise applications need not communicate directly with each other, but communicate only with the MOM, which buffers and routes messages as required.

The disadvantage of early MOM systems was that they required costly and time-consuming integration efforts. Each MOM vendor provided a different Application Programming Interface (API) for its product. Application software vendors had to write messaging components for their applications and special adapters for each application that they wanted to integrate with.

These challenges were addressed with the introduction of the Java Message Services API, which is part of the core J2EE platform. Java Message Services (JMS) provides a standard, portable way for Java programmers to access MOM products. JMS portability is assured through a well-defined set of interfaces along with a standard reference implementation.

Despite their similarities in functionality, different JMS implementations deliver different performance and scalability under heavy workloads. This paper defines real-world scenarios in which messaging may be used, and provides a detailed performance comparison between two JMS implementations: The Java System Message Queue and the IBM WebSphere MQ products.

Sun Java System Message Queue

The Sun Java System Message Queue (formerly Sun ONE Message Queue) is a standards-based, business integration software that connects business applications by enabling them to exchange asynchronous messages over an enterprise message server. The product supports both Point-to-Point and Publish-Subscribe modes of asynchronous messaging. Through its support of the industry-standard Java Message Service specification, and its C Messaging API, the Java System Message Queue allows developers to integrate their Java and native C/C++ applications on multiple platforms. It provides enterprise-strength reliability through features like “once-only” delivery and global transactions; and advanced security features including encryption and authorization to ensure the confidentiality and validity of messages. The Java System Message Queue can scale to manage varying workloads through features such as clustering, configurable message delivery policies, and load balancing. Other features such as client connection failover allow the product to be highly available and suitable for enterprises requiring high uptime. Additional features include multiple message transports, a remote monitoring API, and web services support through SOAP messaging.

IBM WebSphere MQ

IBM WebSphere MQ (formerly known as IBM MQSeries) enables application integration by helping business applications exchange information across platforms by sending and receiving data as messages. WebSphere MQ takes care of network interfaces, ensure “once only” delivery of messages, manage communications protocols, dynamically distribute workload across available resources, handle recovery after system problems, and help make programs portable. WebSphere MQ provides a consistent, multi-platform API. WebSphere MQ Java and an implementation of the Java Messaging Service API are supplied with the product. Additional capabilities include scalability,

© 2003 Crimson Consulting Group

security, system administration, and the ability for applications to operate as publish/subscribe brokers on most platforms and as publishers or subscriber on any platform to automate the distribution of relevant information to all applications that have registered an interest in a particular topic.

Test Description and Methodology

Performance in these tests is defined as the number of 1024-byte messages each product processed per second for each test configuration. All software was installed on a SunFire™ V210 with two 1GHz UltraSPARC IIIi processors and 2GB of RAM running Solaris-8 using the vendor's default installation settings. Tests were conducted under the following conditions:

- All tests were repeated numerous times until the recorded data from multiple runs varied by a 2-sigma (95% confidence interval) of all measured values for each test.
- Each sender and each client used a single JMS connection.
- Results do not include network latencies for client connections.
- 10,000 messages of 1024 bytes each were sent in each test.
- No machine exceeded 75% CPU utilization or 75% memory utilization.

Testing Variables

JMS offers two messaging designs: point-to-point and publish-and-subscribe. A series of benchmark tests were run measuring the performance of the Sun and IBM message queuing software using each of these methods. While testing these two messaging designs, we tested the impact of a number of variables, which are described in this section.

Point-to-Point

The point-to-point message model handles messages intended for a single receiver. Within a point-to-point message system, the messaging provider establishes queues to help ensure that a message is delivered to only one receiver only once. An example of an application using the point-to-point approach might be an employee portal that accesses a product ordering application. In this application, a sales representative logs onto his employee website. His personalized portal channels include a purchase request form. The employee fills out the form and the Purchase Request Servlet places the order information in a queue for later processing. Later, a financial officer logs onto the employee portal website to view all purchase/order requests awaiting approval. A purchase approval servlet pulls all the orders that the finance officer needs to approve off of the Requests Queue. Once the request is approved, it is placed in the Orders queue for processing. The supplier then monitors the Orders queue and processes the order when it arrives on the queue.

Publish-and-Subscribe

Publish-and-subscribe systems handle messages intended for multiple receivers. Publish-and-subscribe systems send messages to a destination called a topic. An example of an application using publish-and-subscribe would be a manufacturer that needs to communicate schedule and quantity changes from its demand forecasting system to suppliers. The manufacturer would use the publish-and-subscribe system to send the messages simultaneously to all vendors supplying components for a particular product.

© 2003 Crimson Consulting Group

Acknowledgements

MOM applications use acknowledgements to notify the messaging provider when its message has been successfully received. This ensures that messages are delivered successfully and that they are delivered only once. Some applications may tolerate duplicate message delivery, such as when an inventory system periodically notifies a manufacturer of current inventory levels. For these applications, JMS supports a DUPS_OK acknowledgement model that allows the messaging provider to speed acknowledgement processing by allowing for the possibility that some duplicate messages may be delivered in the case of provider failure. We compared the IBM and Sun software running several tests with and without the DUPS_OK optimization.

Persistence

Persistent messages are written to stable storage (i.e. to disk) by the provider to ensure message integrity in the event of a provider failure. We compare the IBM and Sun systems running several of our tests with and without persistence.

Selectors

Selectors allow message recipients to specify specific types of messages that they want to see; for example, a user might specify that they only want to see messages from Joe. We ran several tests comparing the performance impact of the use of selectors on each of the two tested systems.

Durable Subscriptions

Nondurable subscriptions last for the lifetime of the subscriber. In other words, a subscriber will only see messages published to a topic while the subscriber is active; if the subscriber is not active, it will miss messages published to the topic. A subscriber can also be made durable—at the cost of higher overhead. When a subscriber is durable, the messaging provider maintains the subscription even if the client is inactive and stores messages for the client. When the client comes back up, it is able to retrieve messages that it would have otherwise missed. A user might want durable messages in an inventory update system where he is required to know what parts are in stock; non-durable messages would be more appropriate for a news clipping service that continually provides the most up-to-date information. We tested the impact of durability on performance in various tests.

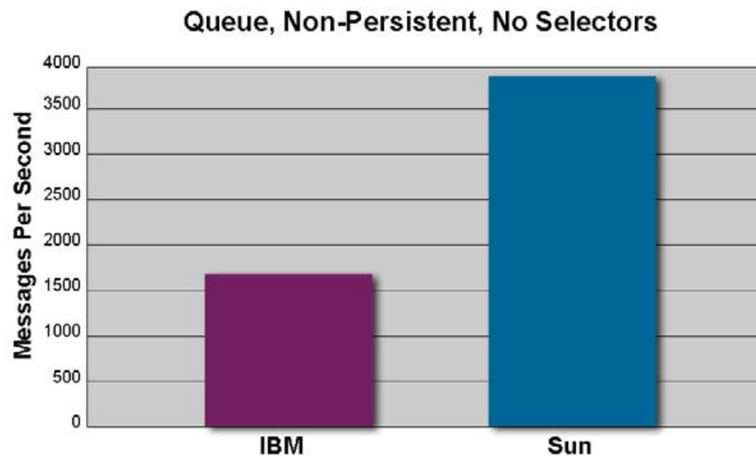
© 2003 Crimson Consulting Group

Results

Test results indicate that Java System Message Queue consistently achieves higher throughput than IBM WebSphere MQ, yielding better performance results. The following bar charts demonstrate how Sun outperforms IBM under a wide range of usage scenarios. For a complete list of tests, see Appendix A.

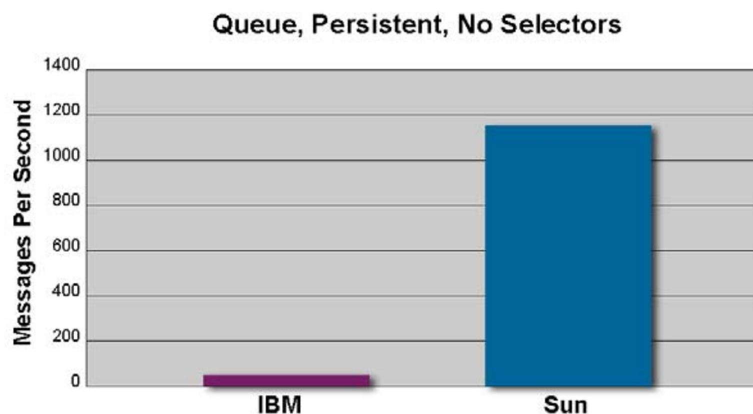
Queue, Non-Persistent, No Selectors

In this test (Q5 in Appendix A), we used a point-to-point scenario with one sender and one receiver. We put on the queue and removed 10K messages of 1024 bytes. This test measured how fast a queue can deliver non-persistent messages with no selectors.



Queue, Persistent, No Selectors

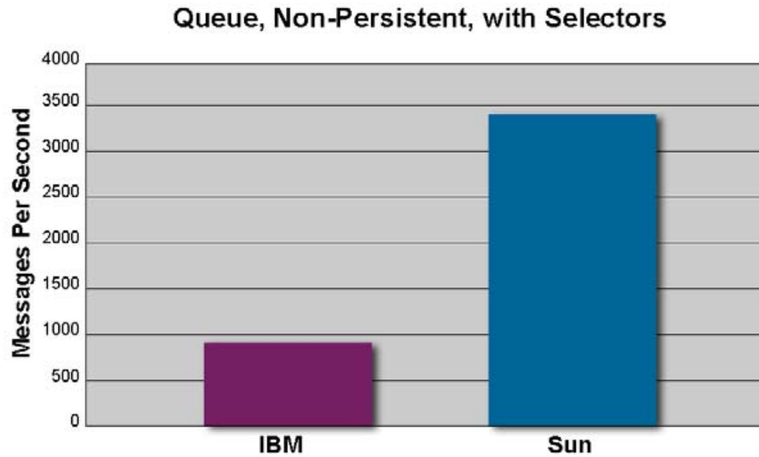
This test (Q7) sent 10K 1024-byte messages to a queue with an active receiver using a DUPS_OK acknowledgement. This test measures the throughput of a persistent queue when the DUPS_OK optimization is allowed.



© 2003 Crimson Consulting Group

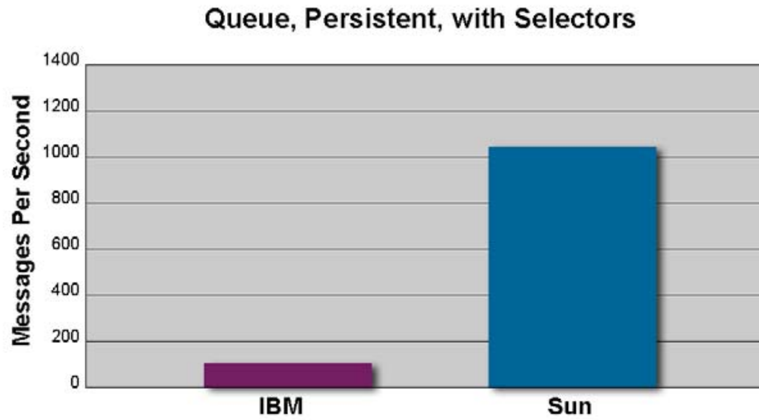
Queue, Non-Persistent, with Selectors

This test (Q9) sent 10K 1024-byte messages to a queue with an active receiver that specifies a selector. This test measures the throughput of a point-to-point queue when the overhead of a selector is introduced.



Queue, Persistent, with Selectors

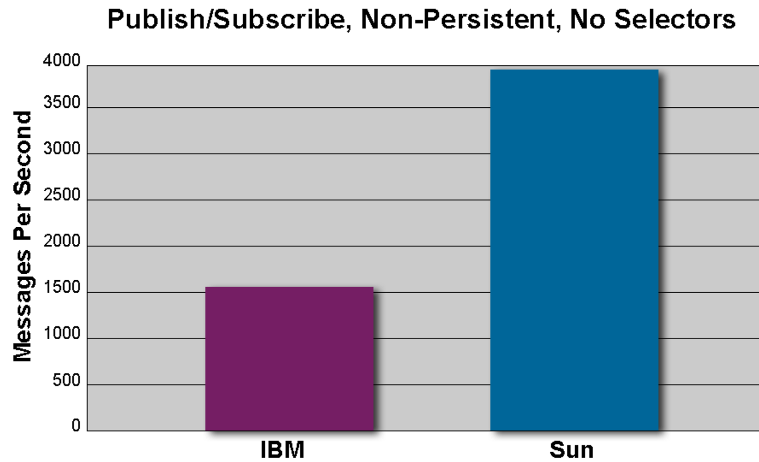
This test (Q12) sent 10K persistent messages to a queue with an active receiver using a selector. This test demonstrates the throughput of a persistent queue when the overhead of a selector is introduced.



© 2003 Crimson Consulting Group

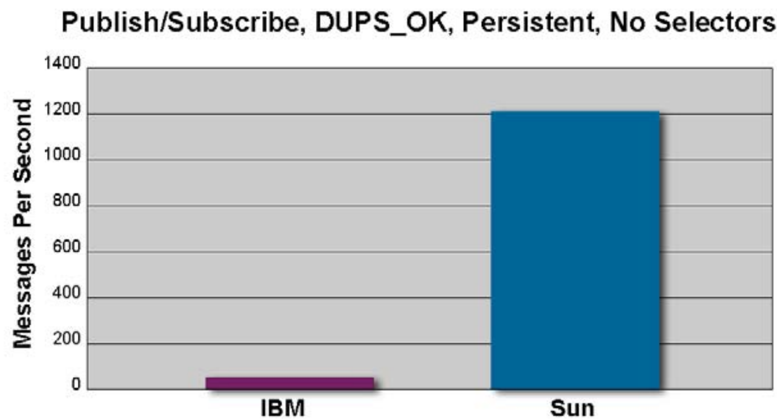
Publish/Subscribe, Non-Persistent, No Selectors

This test (T3) sends 10K non-persistent messages to a publish/subscribe topic. It measures how fast messages can be put into and taken out of the topic with one publisher and one subscriber.



Publish/Subscribe, DUPS_OK, Persistent, No Selectors

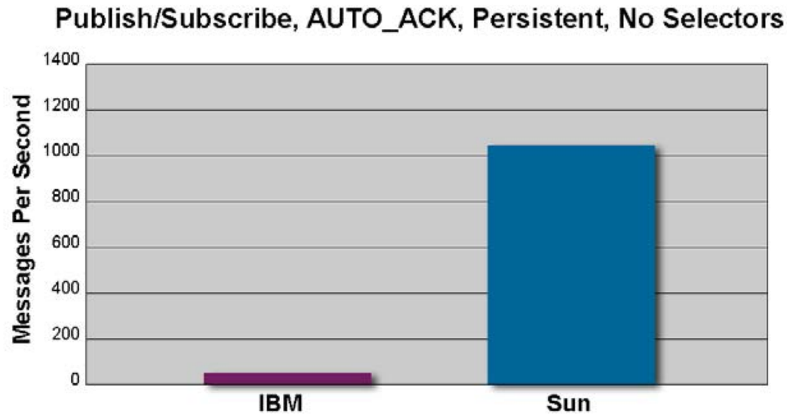
This test (T9) delivers 10K persistent messages to a publish/subscribe topic. It measures how fast a messaging provider can deliver persistent publish/subscribe messages with the lower overhead of Request/Response operations using the DUPS_OK acknowledgement.



© 2003 Crimson Consulting Group

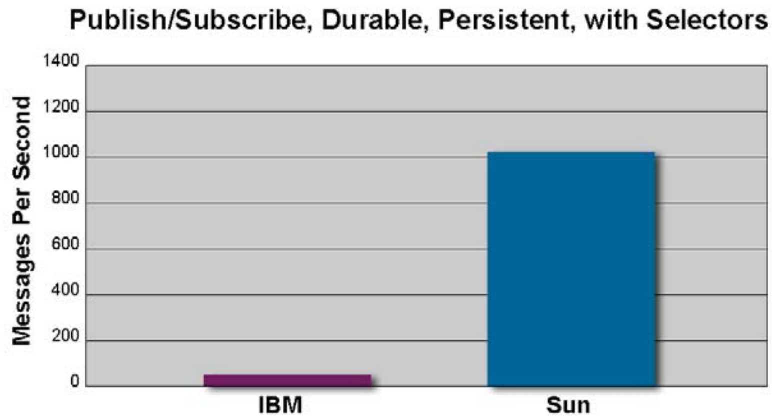
Publish/Subscribe, AUTO_ACK, Persistent, No Selectors

This test (T10) delivers and drains 10K persistent messages to and from a publish/subscribe topic. It measures how fast a messaging provider can deliver persistent messages when the subscriber has specified no selectors but has to confirm processing of the message.



Publish/Subscribe, Durable, Persistent, with Selectors

This test (T13) uses the publish/subscribe semantics and has both persistence and selectors turned on. It measures how fast 10K messages can be published to a persistent topic, selected by a durable subscriber, and drained out of the topic.



© 2003 Crimson Consulting Group

Conclusion

Overall, the Java System Message Queue system had significant performance advantages over IBM WebSphere MQ. Within the point-to-point tests, Sun's solution was 10.5x faster on average, with a maximum advantage of 2,050% on some tests. For the publish/subscribe tests, Sun's product was at least 6.7x faster on average, with a maximum performance advantage of 2,080%.

The test performance figures lead to one remarkable conclusion – Sun Java System Message Queue is significantly more efficient than its counterpart. To quantify the difference, a company would need, depending on queue or topic architecture, from 6 to 10 computers running IBM MQ to equal 1 computer running Sun MQ. The performance difference can lead to staggering disparities in both hardware and software costs.

Sun's higher performance allows organizations to reduce costs by maximizing the efficiency of their network computing hardware. It also enables organizations wishing to develop highly integrated e-business solutions to more effectively scale their solution to meet the peak requirements of their large, mission critical e-business applications.

© 2003 Crimson Consulting Group

Appendices

Appendix A—Complete List of Tests

The following is a complete description of all the benchmark tests that were performed:

Point-to-Point Tests

The following tests were for the point-to-point messaging model.

Q1. Send 10K messages to the queue. This test measures how fast the queue can accept non-persistent messages.

Q2. Deliver 10K messages from a queue. This test measures how fast the queue can deliver non-persistent messages.

Q3. Send 10K persistent messages to the queue. This test measures how fast a queue can deliver persistent messages.

Q4. Deliver 10K persistent messages from a queue. This test measures how fast a queue can deliver persistent messages.

Q5. Send 10K messages to a queue with an active receiver using DUPS_OK acknowledgement. This test measures the throughput of a queue when the DUPS_OK optimization is allowed.

Q6. Send 10K messages to a queue with an active receiver using AUTO_ACK acknowledgement. This test demonstrates the throughput of a queue without the DUPS_OK optimization.

Q7. Send 10K persistent messages to a queue with an active receiver using DUPS_OK acknowledgement. This test measures the throughput of a persistent queue when the DUPS_OK optimization is allowed.

Q8. Send 10K persistent messages to a queue with an active receiver using AUTO_ACK acknowledgement. This test measures the throughput of a persistent queue without the DUPS_OK optimization.

Q9. Send 10K messages to a queue with an active receiver that uses a selector. Use DUPS_OK acknowledgement. This test measures the throughput of a queue when the overhead of a selector is introduced.

Q10. Send 10K messages to a queue with an active receiver that uses a selector. Use AUTO_ACK acknowledgement. This test demonstrates the throughput of a queue without the DUPS_OK optimization when the overhead of a selector is introduced.

Q11. Send 10K persistent messages to a queue with an active receiver using a selector. Use DUPS_OK acknowledgement. This test demonstrates the throughput of a persistent queue when the overhead of a selector is introduced.

Q12. Send 10K persistent messages to a queue with an active receiver using a selector. Use AUTO_ACK acknowledgement. This test measures the throughput of a persistent queue without the DUPS_OK optimization when the overhead of a selector is introduced.

© 2003 Crimson Consulting Group

Publish-and-Subscribe Tests

The following tests were performed for the publish-and-subscribe messaging model.

T1. Send 10K messages to a topic. This test measures how fast a topic can accept non-persistent messages.

T2. Send 10K persistent message to a topic. This test measures how fast a topic can accept persistent messages.

T3. Send 10K messages to a topic with an active subscriber using DUPS_OK acknowledgement. This test measures the throughput of the topic with the DUPS_OK optimization is allowed.

T4. Send 10K messages to a topic with an active subscribers using AUTO_ACK acknowledgement. This test demonstrates the throughput of a topic without the DUPS_OK optimization.

T5. Send 10K persistent messages to a topic with an active subscriber using DUPS_OK acknowledgment. This test measures the throughput of a persistent topic when the DUPS_OK optimization is allowed.

T6. Send 10K persistent messages to a topic with an active subscriber using AUTO_ACK acknowledgement. This test measures the throughput of a persistent topic without the DUPS_OK optimization.

T7. Send 10K messages to a topic with an active durable subscriber using the DUPS_OK acknowledgement. This test measures the throughput of a persistent topic with the DUPS_OK optimization.

T8. Send 10K messages to a topic with an active durable subscriber using the AUTO_ACK acknowledgement. This test measures the throughput of a non-persistent topic without the DUPS_OK optimization.

T9. Send 10K persistent messages to a topic with an active durable subscriber using the DUPS_OK acknowledgement. This test measures the throughput of a persistent topic with the DUPS_OK optimization.

T10. Send 10K persistent messages to a topic with an active durable subscriber with the AUTO_ACK acknowledgement. This test measures the throughput of a persistent topic without the DUPS_OK acknowledgement.

T11. Sends 10K messages to a topic with an active subscriber that uses a selector and the DUPS_OK acknowledgement. This test demonstrates the throughput of a topic with the DUPS_OK optimization when the overhead of a selector is introduced.

T12. Sends 10K persistent messages to a topic with an active subscriber that uses a selector with the DUPS_OK acknowledgement. This test demonstrates the throughput of a persistent topic with the DUPS_OK optimization when the overhead of a selector is introduced.

T13. Sends 10K persistent messages to a topic with an active subscriber that uses a selector with the AUTO_ACK acknowledgement. This test demonstrates the throughput of a persistent topic without the DUPS_OK optimization when the overhead of a selector is introduced.

© 2003 Crimson Consulting Group

Appendix B—Complete Test Results

Test	S / R	P / NP	ACK Mode	Selectors	IBM (Msgs/sec)	Sun (Msgs/sec)	Sun Performance vs. IBM
Q1	S	NP	N/A	No	2,051.54	8,047.44	392%
Q2	R	NP	DUPS_OK	No	2,163.23	6,073.41	281%
Q3	S	P	N/A	No	102.07	1,607.32	1,575%
Q4	R	P	AUTO	No	138.66	1,726.08	1,245%
Q5	SR	NP	DUPS_OK	No	1,682.78	3,870.93	230%
Q6	SR	NP	AUTO	No	1,697.03	1649.74	97%
Q7	SR	P	DUPS_OK	No	54.21	1,167.22	2,153%
Q8	SR	P	AUTO	No	53.41	1,078.90	2,020%
Q9	SR	NP	DUPS_OK	Yes	936.21	3,432.62	367%
Q10	SR	NP	AUTO	Yes	956.84	1,554.37	162%
Q11	SR	P	DUPS_OK	Yes	54.26	1,139.40	2,100%
Q12	SR	P	AUTO	Yes	53.85	1,048.07	1,946%
T1	S	NP	N/A	No	1,906.09	12,148.44	637%
T2	S	P	N/A	No	925.61	2,898.96	313%
T3	SR	NP	DUPS_OK	No	1,526.18	3,946.48	259%
T4	SR	NP	AUTO	No	1,550.40	1,625.71	105%
T5	SR	P	DUPS_OK	No	728.37	1,818.05	250%
T6	SR	P	AUTO	No	734.20	1,432.25	195%
T7	SR	NP	DUPS_OK	No	1,317.69	3,858.88	293%
T8	SR	NP	AUTO	No	1,318.97	1,637.38	124%
T9	SR	P	DUPS_OK	No	55.40	1,205.26	2,176%
T10	SR	P	AUTO	No	54.77	1,062.71	1,940%
T11	SR	NP	DUPS_OK	Yes	1,441.49	3,686.22	256%
T12	SR	P	DUPS_OK	Yes	737.26	1,704.76	231%
T13	SR	P	AUTO	Yes	54.81	1,028.73	1,877%

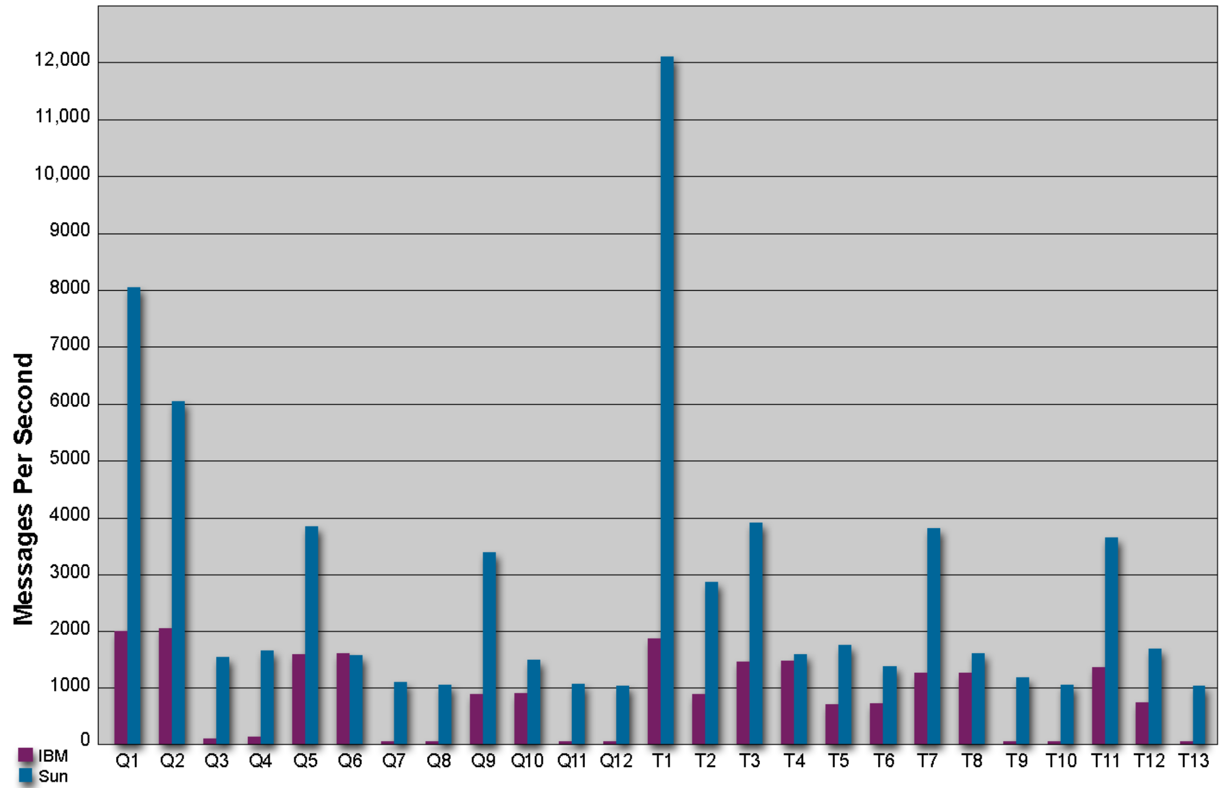
S/R: S = Messages deposited into queue, no receivers
R = Messages retrieved from the queue, no senders
SR = One sender, one receiver

P/NP: P = Persistent
NP = Non-Persistent

ACK Mode: AUTO = Auto acknowledgement,
DUPS_OK = Duplicates OK acknowledgement
N/A = Not Applicable (Sender only)

Selectors: Selectors used

© 2003 Crimson Consulting Group



© 2003 Crimson Consulting Group

Appendix C—Test Environment

The comparative information published in this document reflects laboratory tests undertaken by Crimson Consulting, Los Altos, CA. Performance in individual cases may vary depending on the environment, workload, and any unique characteristics of other software products at other locations.

Crimson tested IBM WebSphere MQ software version 5.3 core product plus the 5.3 JMS client provider for Sun Solaris and Java System Message Queue version 3.5 for Sun Solaris.

All the recommended patches for both packages running on a generic Solaris 8 operating system were installed. The system on which tests were conducted was patched with the standard Sun-recommended patch cluster for Solaris-8 as of November 24, 2003.

We used the Java Software Development Kit (JDK) version 1.4.1 to compile and run the Java source code tests. Bourne shell wrapper scripts ran large sequences of tests automatically.

We installed the products from both vendors using default "out-of-the-box" configurations. We performed no other operating system tuning or product configuration of any kind, except to create the test queues.

We ran the tests on a SunFire V210 with two 1GHz UltraSPARC IIIi processors and 2GB RAM.

We ran the java testing clients locally on the same system as the message queue JMS software to avoid variations in network latencies.

The testing code measured "wall clock" seconds for each test. We defined the performance indicator as the number of messages processed per second in each test.

We ran scripts that repeated all tests numerous times, over several hours until acceptably high confidence values were achieved.

© 2003 Crimson Consulting Group