Message-Driven-Bean Performance using WebSphere MQ v5.3 and WebSphere Application Server v5.0

Version 1.1

25th April 2003

Marc Carter

WebSphere MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Notices

This report is intended to help the reader understand the performance characteristics of WebSphere MQ for Windows V5.3 in conjunction with WebSphere Application Server v5.0. The information is not intended as the specification of any programming interfaces that are provided by WebSphere MQ.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which it operates.

Information contained in this report has **not** been submitted to any formal IBM test and is distributed "**as-is**". The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the reader to evaluate the information and project the results to their operational environment.

The performance measurements included in this report were measured in a controlled environment and the results obtained in other environments may vary significantly.

Trademarks and service marks:

The following terms used in this publication are trademarks of the IBM Corporation in the United States or other countries or both:

IBM

MQSeries

WebSphere

WebSphere MQ

SupportPac

FFST

AIX

Microsoft, Windows, Windows NT and Windows 2000 are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JMS, J2EE and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Preface

The contents of this SupportPac

This SupportPac is intended to:

- Show the performance impact on messaging throughput of using WebSphere Application Server MDBs in place of basic JMS 1.0 applications.
- Demonstrate the advantages of using WebSphere MQ as an external JMS Resource Provider to WebSphere Application Server.
- Provide the reader with some hints on how to configure their messaging scenario.

Feedback on this SupportPac

We welcome constructive feedback on this report. Does it provide the sort of information you want? Do you feel something important is missing? Is there too much technical detail, or not enough? Could the material be presented in a manner more useful to you? Please direct any comments of this nature to: **WMQPG@uk.ibm.com**.

Specific queries about performance problems on your WebSphere MQ system should be directed to your local IBM Representative or Support Center.

Acknowledgements

The author is very grateful to Anthony Tuel for help in producing this report.

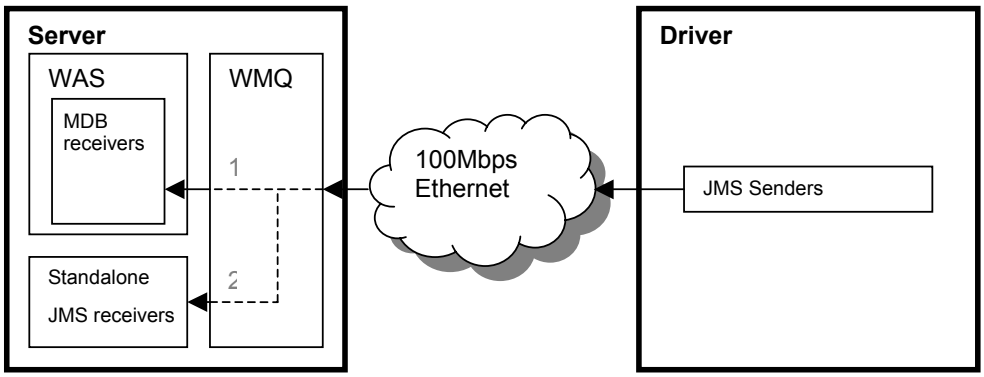
1 Test scenario

1.1 Description

JMSPrim is a benchmark developed by the WebSphere Application Server (WAS) performance team to evaluate an application server's performance in the area of JMS. It is designed to measure a wide range of primitive scenarios, to exercise the individual "building blocks" that would make up a typical enterprise application. *JMSPrim* is made up of a set of Message Driven Beans (MDBs) and contrasting standalone scenarios designed to demonstrate a common customer upgrade scenario. The sender and receiver each run at the same time, with appropriate warm-up intervals, to attempt to measure a typical system under load. In each case the receivers run in a tight loop, performing negligible processing per message. As is typical of primitives, this provides an indication of the upper bound to performance - the best-case performance. This is a good thing for applications which are 100% JMS oriented, but most applications use JMS to drive some task such as updating a database or have operations that are not based in any sort of JMS technology.

Each test herein is carried out with senders threads throttled to a particular rate. The number of these senders is increased until the receivers could not remove messages from the queue faster than they were arriving. All applications connect to a single queue and the messages are used are simple 2048-byte text messages. Unless otherwise specified all tests are exercising WMQ as External JMS Provider and not the default Embedded JMS.

1.2 Architecture



1.3 Terminology and Defaults

np_tns Messages are express (non-persistent) and non-transacted. Each sender is rated at 50 messages per second. MDBs are run with 6 sessions in their Listener Port.

p_tr Messages are persistent and transacted. Each sender is rated at 20 messages per second. MDBs are run with 50 sessions in their Listener Port.

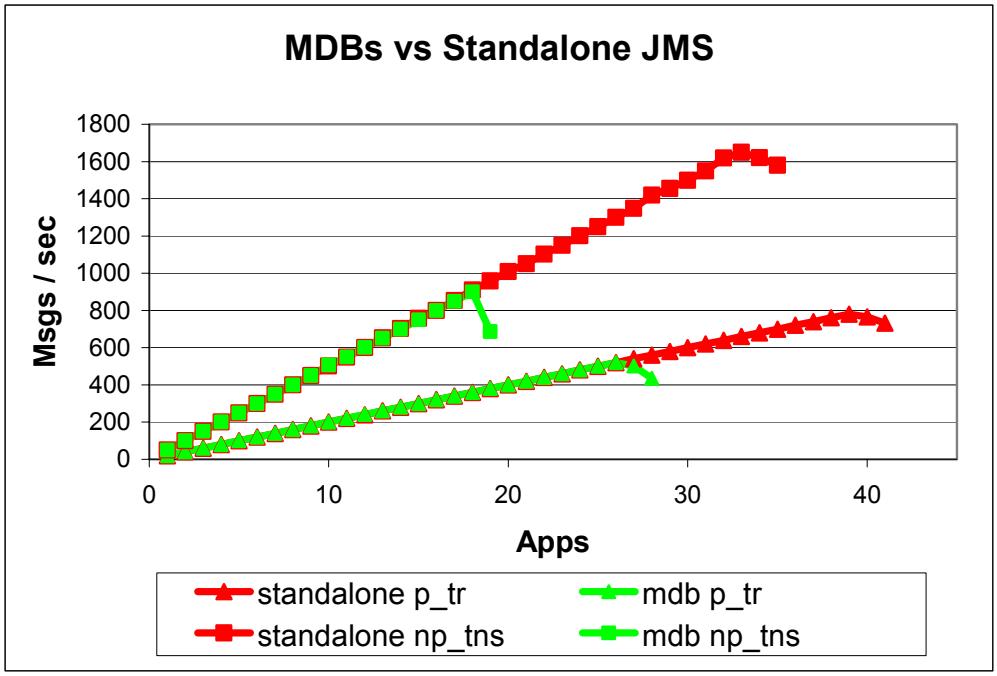
MDB The receivers are MDBs inside of WebSphere Application Server

standalone WAS MDBs are not exercised. The receiving threads are run in their own JVM and WAS is not running

2 Standalone JMS vs. WebSphere MDB

The cost of basic pure-messaging increases when a simple application model, such as our test application, is moved from the world of standalone JMS into an application server providing runtime management and J2EE integration. In this test the application server is not running and a simple JMS application is draining the queue using the same number of threads as the corresponding number of WAS *ListenerPort.maxSessions* (see section 4.3). [6 receivers for non-persistent and 50 receivers for persistent test variations.]

2.1 Chart



2.1.1 Results

Introduction of the application server and the flexible services it provides has decreased the raw throughput (at their respective peaks) of these comparable tests.

Non-persistent results have dropped 45% (from 1650 to 900)

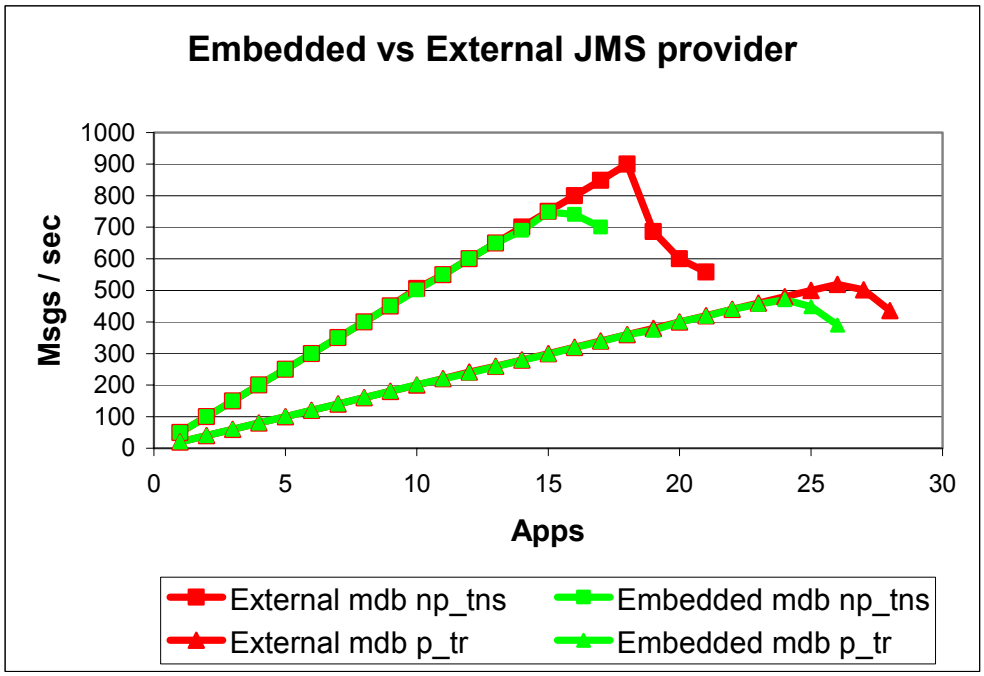
Persistent tests have dropped 36% (from 780 to 500) showing that not as much of their processing is dependant upon raw CPU due to increased locking and serialisation required in persisting a message

Particularly for non-persistent tests, these results demonstrate that increased CPU-cost-per-message can have the controlling effect on the throughput of the system. It also shows that the potential benefits provided by using WMQ as an External JMS Provider to run the queue manager on a separate physical server can be important in a scenario where performance is of concern.

3 Embedded JMS vs. External JMS provided by WMQ v5.3

WebSphere Application Server v5.0 has the option to integrate WebSphere MQ as an external JMS Resource Provider. This gives users more flexibility in their deployment and administration of messaging resources. They can make use of the clustering and high-availability mechanisms in WMQ and communicate with any other WMQ systems in their organisation. It separates the application data from the application server's configuration, allowing different location, security and performance considerations to be applied to each.

3.1 Chart



3.2 Results

Use of a tuned, external Queue Manager has improved throughput over an untuned embedded JMS provider. The CPU provides the constraining point for all tests, which masks some of the improvements of the external Queue Manager, especially in the persistent tests where it showed a 4% increase (from 480 to 500) in peak throughput. This increase is expected to be much larger when placed in a server with more CPU resource available. Non-persistent messaging is, by its nature, limited by CPU. External WMQ outperformed the embedded JMS by 20% in the non-persistent case (going from 750 to 900).

Notes

This test only made use of *Client* connections on the external WMQ. Use of *Bindings* will be shown to improve throughput by another 50% in the next section.

Details on the tuning applied to WMQ are in section 5 of this report.

4 Increasing performance

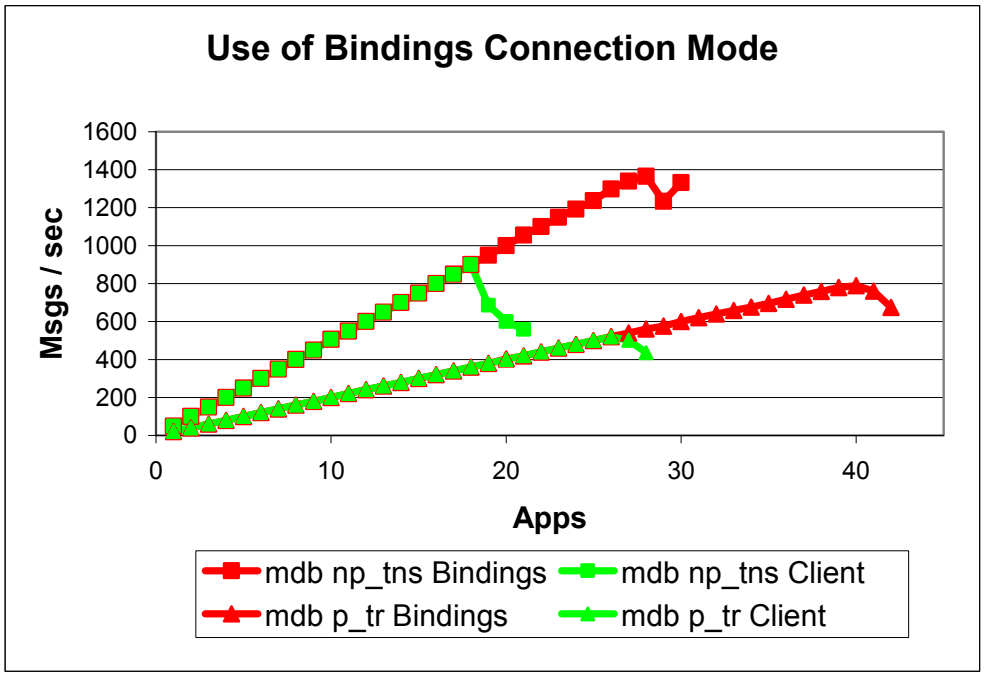
There are many variables pertaining to the topology and workload you are planning for your system. Most of these cannot have generic recommendations applied to them but here are some areas that may benefit WAS messaging performance.

4.1 "Bindings" connections to WMQ

At the cost of decreased safety checking of user-programmed interactions with WMQ, you can choose *Bindings* connections on a *QueueConnectionFactory* used by entities on the same physical box as the Queue Manager. *Bindings* connections will disable several of the security authorisation features in MQ. This setting affects anything which uses that factory, be it an MDB in WAS or an external client using the applications server's JNDI database.

This test changed the MDBs to use *Bindings* connections to WebSphere MQ while leaving the driving applications using *Client* connections from their remote box.

4.1.1 Chart



4.1.2 Results

Use of *Bindings* instead of *Client* connections has increased peak throughput by 50% for both persistent and non-persistent results. This is caused by a reduction in CPU time spent validating the interactions the MDB has with the Queue Manager.

4.1.3 Where to find the Setting

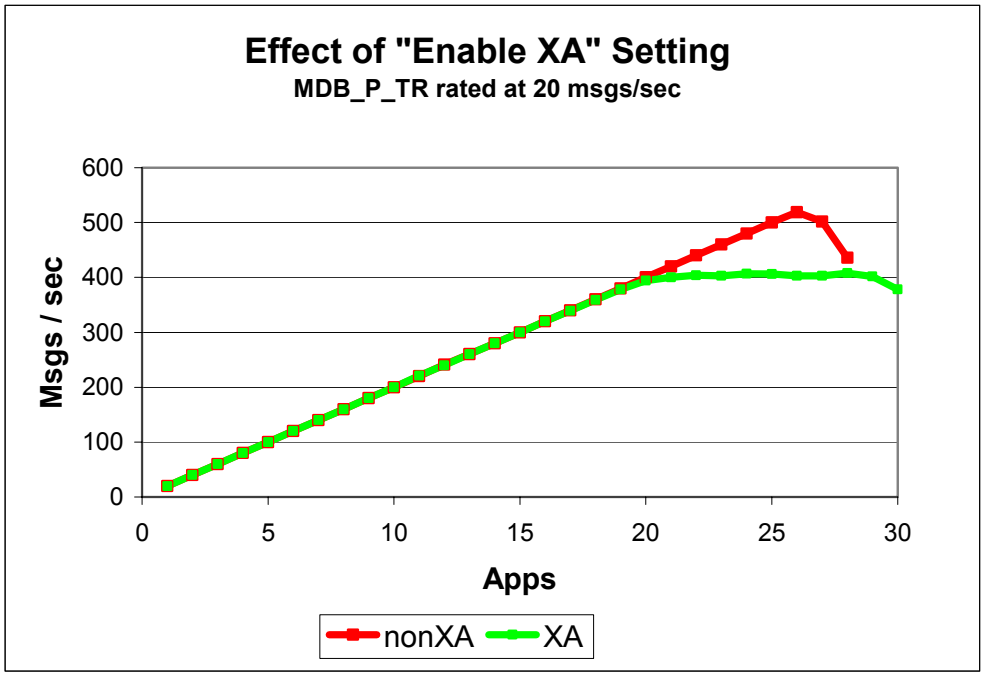
In the WAS AdminConsole GUI under:

Resources
    WebSphere MQ JMS Provider
        Queue Connection Factories
            <Your queue factory name>

4.2 Use of "Enable XA"

Enabling XA co-ordination on Queue and Topic Connection Factories instructs WAS to use two phase commits (2PC) in place of single phase commits (1PC). This is essential for any co-ordinated transactions – those which involve more than one Resource Manager but are logically viewed as a single transaction. WAS will optimise a 2PC when it knows there is only one resource involved. Although no logging will be done, there is still some overhead related to XA processing.

4.2.1 Chart



4.2.2 Results

Turning off XA co-ordination on the persistent, transacted test amounted to a 30% increase in maximum messaging throughput. This demonstrates that the default setting of *enableXA* is sub-optimal if your scenario does not require this additional logic.

The same effect, demonstrated here on *QueueConnectionFactories*, will be seen with *TopicConnectionFactories*

4.2.3 Where to find the Setting

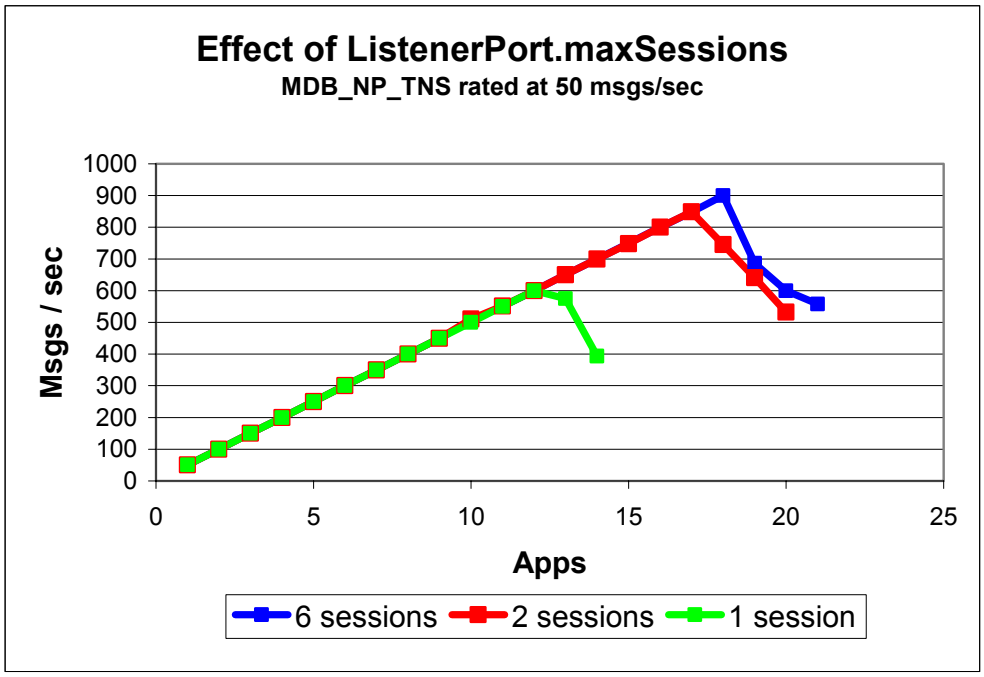
In the WAS AdminConsole GUI under:

Resources
WebSphere MQ JMS Provider
Queue Connection Factories
<Your queue factory name>

4.3 Use of "ListenerPort.maxSessions"

This setting represents the number of concurrent JMS server sessions used by an MDB Listener to process messages for an MDB. This correlates to the number of parallel connections made to WebSphere MQ.

4.3.1 Chart



4.3.2 Results

Message Driven Beans are asynchronous by nature and therefore suffer a significant penalty when forced to run in a serial mode (i.e. with *maxSessions*=1)

In order to make best use of parallelism in your server you need to set realistic values for your maximum number of concurrent sessions. It is also sensible to set the minimum sessions (in the *QueueConnectionFactory*) to a value greater than the default if you expect to handle intermittent, heavy bursts of traffic

4.3.2.1 Non-persistent, non-transacted

It is desirable to have enough sessions such that there is always one spare to make use of each processor as it becomes free.

There are many factors involved in the choice of this number. If, as here, the MDB does not use much CPU time per message, then the optimum value for *maxSessions* is dominated by the number of processors available.

The graph above shows that choosing a suitable number of concurrent sessions has improved non-persistent throughput 33% when compared to the default value.

4.3.2.2 Persistent, transacted

For tests involving use of the WMQ logging facilities, best utilisation of the disk is achieved again by the use of parallelism. In this case a large value for *maxSessions* is necessary. All tests shown in this report involving persistent measurements used a maximum of 50.

Where to find the Setting

In the WAS AdminConsole GUI under:

Servers
Application Servers
<Your server name>
Message Listener Service
Listener Ports
<Your listener port name>

Note

ListenerPort.maxSessions is dependant upon the maximum sessions parameter set in the *QueueConnectionFactory* session pool for the relevant queue. The difference between these two values is that the Factory value encompasses all types of EJB that might request access to that resource. The *ListenerPort* maximum value is for that specific listener port only, when increasing this value – be sure to increase the value in the *QueueConnectionFactory* too.

4.4 Use of "ListenerPort.maxMessages"

Altering of this value with WebSphere MQ v5.3 as JMS provider is not supported at this time.

5 How to Tune External WMQ

Performance reports with tuning information for WebSphere MQ v5.3 on each platform can be found under the official IBM SupportPac webpage at the following URL:

http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html

The main settings used for the tests in this report were:

- Use of multiple physical disk arrays
- Changing log settings to their maximums, for persistent messaging
- FASTPATH Channel Application for *Client* connections
- DefaultQBufferSize = 1MB
- DefaultPQBufferSize = 1MB

6 How to Tune Embedded JMS Provider

WebSphere Application Server may create/delete/start/stop queue managers or queues when it performs administrative procedures on its embedded JMS provider. For this reason it is not recommended for *manual* tuning to be applied to the embedded JMS provider.

For advice on what tuning *can* be applied to this scenario please visit the WAS InfoCenter and search for “tuning parameter list”.

http://publib7b.boulder.ibm.com/webapp/wasinfo1/index.jsp?deployment=ApplicationServer&lang=en

7 Test Environment

7.1 Hardware

Server

- IBM Netfinity 5500 M20, 4 * 500MHz P3 Xeon
- Windows 2000 Server SP3
- 4GB Ram
- 5 * SCSI 7,200 RPM drives
- 100Mb Ethernet card

Driver

- IBM Netfinity 6000R, 4 * 700MHz P3 Xeon
- Windows 2000 Server SP3
- 0.5GB Ram
- 3 * SCSI 10,000 RPM drives
- 1Gb Ethernet card

7.2 Software Levels

WebSphere Application Server v5.0, no PTFs

WebSphere MQ v5.3, CSD 1

IBM Java 1.3.1