WebSphere MQ with JMS:

Get the Best from WebSphere MQ

and Message Broker Publish/Subscribe Processing

Version 1.0.1

MP06

February 2007

Peter Toghill

WebSphere MQ Performance

IBM UK Laboratories

Hursley Park

Winchester

Hampshire

SO21 2JN

Before using this report, please be sure to read the paragraphs on “disclaimers”, “warranty and liability exclusion”, “errors and omissions”, and the other general information paragraphs in the "Notices" section below.

First Edition, February 2007.

This edition applies to WebSphere MQ V6 (and to all subsequent releases and modifications until otherwise indicated in new editions).

Notices

DISCLAIMERS

The performance data contained in this report were measured in a controlled environment. Results obtained in other environments may vary significantly.

You should not assume that the information contained in this report has been submitted to any formal testing by IBM.

Any use of this information and implementation of any of the techniques are the responsibility of the licensed user. Much depends on the ability of the licensed user to evaluate the data and to project the results into their own operational environment.

WARRANTY AND LIABILITY EXCLUSION

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

In Germany and Austria, notwithstanding the above exclusions, IBM's warranty and liability are governed only by the respective terms applicable for Germany and Austria in the corresponding IBM program license agreement(s).

ERRORS AND OMISSIONS

The information set forth in this report could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; any such change will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time and without notice.

INTENDED AUDIENCE

This report is intended for architects, systems programmers, analysts and programmers wanting to understand the performance characteristics of WebSphere MQ V6.0 and Message Broker V6. The information is not intended as the specification of any programming interface that is provided by WebSphere MQ. It is assumed that the reader is familiar with the concepts and operation of WebSphere MQ V6 and Message Broker V6.

LOCAL AVAILABILITY

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES

Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent

product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

USE OF INFORMATION PROVIDED BY YOU
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

TRADEMARKS AND SERVICE MARKS
The following terms used in this publication are trademarks of International Business Machines Corporation in the United States, other countries or both:

- IBM
- WebSphere MQ
- DB2

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

UNIX is a registered trademark and licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

EXPORT REGULATIONS
You agree to comply with all applicable export and import laws and regulations.

Preface

Target audience

This SupportPac is designed for people who want to setup an efficient JMS messaging system with WebSphere Message Broker and WebSphere MQ.

The Qualities of Service used by the customer will determine how the messaging system should be setup. We discuss the meaning of Persistence and how to get the best out of the system so it efficiently processes persistent messages. This continues with configuration steps followed by tuning advice for the Queue Manager and Message Broker.

The reader should have a general awareness of WebSphere MQ in order to make best use of this SupportPac

The contents of this SupportPac

This SupportPac includes:

- Discussion of Persistence as implemented by WebSphere MQ.
- Guidance in setting up an efficient MQ Messaging system together with Message Broker.
- Guidance in tuning an efficient MQ Messaging system together with Message Broker

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 more useful manner?

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 Centre.

1 Essentials to show WebSphere MQ and Message Broker at their best

- This information is intended to help you get an efficient WebSphere MQ and Message Broker System configured and running Publish /Subscribe scenarios.
- Look at the scenarios in Chapter 2 for the description that most closely matches your situation. As well as the four Publish / Subscribe environments, there are two Point – Point Scenarios that only involve WMQ.
- This report does not cover JMS nodes within MB flows.
- We have also provided links to online information.
- Read the rest of this first chapter as it explains some important facts about positioning WebSphere MQ and Message Broker.

1.1 Message persistence

Several JMS settings control the effective QoS of a JMS client's communication. The delivery and acknowledgement modes indicate how many times a given message can be delivered to an application: at-most-once or once-and-only-once. The customer solution relies on a certain level of resilience from the messaging provider.

If the messages are carrying 'inquiry' questions and answers, then it is likely that speed is far more important than resilience, so the architects can make this trade-off and use non-persistent messages.

1.1.1 JMS delivery mode

The JMS API supports two delivery modes for messages to specify whether messages are lost if the JMS provider fails. These are set by the producer via the *deliveryMode* property of javax.jms.MessageProducer#send().

- The PERSISTENT delivery mode, which is the default, instructs the JMS provider to take extra care to ensure that a message is not lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent. Only a hard media failure should cause a PERSISTENT message to be lost. PERSISTENT has the caveat that it does not cover message destruction due to message expiration (which would be considered a normal event), or loss due to "resource restrictions" (which the JMS specification does not define further).
- The NON_PERSISTENT delivery mode does not require the JMS provider to store the message or otherwise guarantee that it is not lost if the provider fails.

1.1.2 JMS acknowledgement mode

The JMS API also supports the ACKNOWLEDGE_MODE property that controls message duplication on non-persistent messaging. It is set via the *acknowdgementMode* property of javax.jms.Session#createSession() on the consuming application.

- Auto acknowledgement (default) means messages will not be delivered more than once
- DUPS_OK acknowledgement means messages may be delivered more than once in certain

circumstances and the client application must be prepared to deal with seeing the same message twice.
- Client acknowledgement leaves control of this feature entirely to the user.

1.1.3 WebSphere MQ Quality-of-Service

The JMS definition of persistence allows considerable scope for different quality of service (QoS).

WebSphere MQ provides QoS that have been appreciated by customers over the last 12 years. Many of the installation defaults provide robustness and small memory footprint rather than maximising good performance.

WebSphere MQ has traditionally provided two QoS: persistent and non-persistent. The WebSphere MQ definitions are similar, but not identical to the JMS requirements. In particular,

- WebSphere MQ does not discard non-persistent messages while the queue manager is running, even in the event of a memory buffer shortage.
- WebSphere MQ provides a persistence and transaction integrity, above and beyond the specification of JMS, which has been industry-proven for a decade.

These QoS are usually paired together with transactionality. If messages are persistent it is expected, though not required, that they should be transactional and if they are non-persistent, they should be non-transactional. Messages carrying 'valuables' should normally be persistent and transactional since that eliminates most causes of failure. The application and system designer needs to consider the levels of resilience and recovery needed in different places, and the complexity needed in each component - the application, the messaging provider, a database, and so on. Using persistent, transactional messaging can remove a lot of complexity from application code.

Non-persistent messages are discarded by WebSphere MQ in the event of a queue manager restart but otherwise are not lost.

WebSphere MQ's use of transactional recovery logs in combination with secondary storage of queues results in resilience against individual failures can be used for high availability and disaster recovery scenarios.

The JMS definition of a persistent message is not precise so application solutions must decide how much dependence is put on the message provider.

- Does the message have to survive if various resource shortages are encountered on the journey?
- Does the message survive if various application, software, or hardware failures are encountered on the journey?
- Greater reliability inevitably means lower run-time performance because of the extra work needed to provide the information needed during recovery.

1.2 Multiprocessor machines

WebSphere MQ is architected for production environments and is able to fully exploit multiprocessor capabilities with a design that uses many processes and threads. These are dynamically managed in response to increasing workloads (a form of self-optimisation). Some

JMS providers use a single-threaded structure that provides good performance on uni-processors but does not scale across CPUs.

Performance tests have shown the WebSphere MQ will perform significantly better with a multiprocessor (SMP) server machine compared with competitors. The relative performance of WebSphere MQ will improve if 4-way SMPs are used in comparisons. Also, note that modern multi-core processors will also show this improvement.

1.3 Measurement Topology

Benchmarks are normally configured with MQ and MB on the server and the JMS message Producers/ Consumers as Clients on a set of driving machines. The objective is normally to find the capacity of the server so it is important to ensure that the Clients are not a system bottleneck. Plan to allocate about 5 times the power to the driving system compared to the server.

1.4 WMQ Bindings

Message Broker and MQ Channels are MQ Applications in that they produce and consume messages. Applications connecting to the QM can execute in a separate process (Normal Bindings use Inter Process Communication with an Agent process altering the Queue Manager infrastructure) or the Application process directly altering the Queue Manager infrastructure (Fastpath Binding). Applications using fastpath bindings must not be cancelled but terminated in a controlled manner. Normal bindings should be used for customer written applications since it protects the Queue Manager from rogue application behaviour but Fastpath bindings can be used to obtain higher throughput from Channels and Message Broker.

2 Scenarios

1. Publish/Subscribe –Persistent, transacted
2. Publish/Subscribe – NP NT ND Real Time
3. Publish/Subscribe - Scalable WMQ MB NP NT ND
4. Publish/Subscribe – High Performance Multicast
5. Point-to-Point Persistent, transacted
6. Point-to-Point – Non-Persistent, Non-Transacted

2.1 Publish/Subscribe -Persistent, transacted

- Configuration Checklist
- WebSphere Queue Manager
- WebSphere Message Broker
- JMS Clients
- Notable Features

This scenario covers the use of Message Broker and Event Broker publish/subscribe using the WebSphere MQ transport with transacted, persistent messaging and durable subscribers. This type of messaging should be used where messages need to be sent with a high level of reliability and to ensure that in the event of a hardware or software failure messages are not lost. With the increased reliability there is some trade-off to performance as messages are logged to disk so they can be recovered following a hardware or software failure. In this scenario we are assuming that the clients will be hosted on a machine separate to the broker and will use the MQ Client transport.

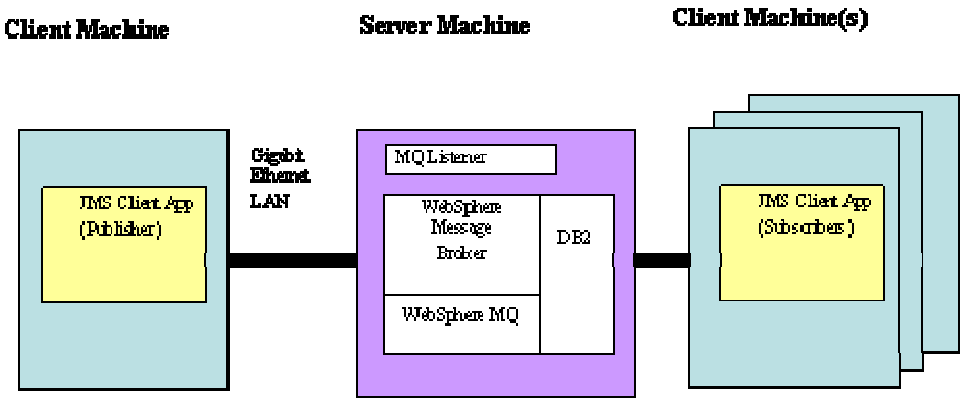
If the client is to be run on the same machine as the broker than the MQ Bindings transport could be used.

2.1.1 Configuration Checklist

Create the default configuration and add the JMS queues to the broker's queue manager.

The following are the steps required to get this scenario working in an efficient manner. In any Message/Event Broker Publish Subscribe system the setup and performance of the following 3 components are critical:

- The Message Broker
- The broker's WebSphere MQ queue manager
- The publishing and subscribing clients (typically JMS clients)



The following sections detail the essential and the optional tuning steps to get this scenario working

2.1.2 The Broker's WebSphere MQ Queue Manager

The following changes are not required but they are advisable if performance is likely to be an issue. Each scenario could use a combination of WMQ Explorer, Registry Editor, AMQMDAIN, or QM.INI to achieve these steps. WMQ Explorer with Service levels from V6:0:2:0 can be used to update the Log, Channels information but not the TuningParameters information.

Given the use of persistent messages in the tests the following MQ parameters were modified:

Log buffer pages. Controls the size of the queue manager buffer to log file writes in number of pages. Use this parameter when high volumes of messages are being sent through a server.

- **How to view or set:** To view the value, display the operating system configuration information for the LogBufferPages setting of the queue manager. For example, on Windows (For Unix see .ini settings in Chapter 2.1.2.2) use the registry editor to navigate to **HKEY_LOCAL_MACHINE > SOFTWARE > IBM > MQSeries > CurrentVision > Configuration > QueueManager >** *QM_name* **> Log > LogBufferPages**.
    - set LogBufferPages to the desired value. Set the value to 4096 pages.

Other settings in the **Log** stanza of interest are LogFilePages, LogPrimaryFiles, LogSecondaryFiles, LogDefaultPath, & LogWriteIntegrity and other interesting stanzas under the *QM_name* are **TuningParameters** and **Channels.**

Log file pages. Controls the size of the individual disk extent used to store the logged data in number of pages. Use this parameter when high volumes of messages are being sent through a server.

- set LogFilePages to the desired value. Set the value to 16384 pages.

Log primary files. Controls the number of primary or permanent log files for the queue manager. Use this parameter when high volumes of messages are being sent through a server.

- Set LogPrimaryFiles to 16 .

Log secondary files. Controls the number of secondary log files for the queue manager. Secondary files are files created when the primary files are not enough and deleted when they are no longer needed.

- set LogSecondaryFiles to 2

Log default path. Controls the location of the queue manager log files. Use this parameter when high volumes of messages are sent through a server.

- It is ideal to have a disk dedicated to this task because WebSphere MQ tries to keep the head of the disk positioned at the place in the file where it needs to write next. A fast RAID volume on SAN or cached disk is best.

Log Write Integrity. Some hardware guarantees that, if a write operation writes a page and fails for any reason, a subsequent read of the same page into a buffer results in each byte in the buffer being either: The same as before the write, or The byte that should have been written in the write operation. On this type of hardware (for example, SAN or SSA write cache enabled), it is safe for the logger to write log records in a single write as the hardware assures full write integrity. This method provides the highest level of performance.

- Set the value SingleWrite

Default persistent queue buffer size. Controls the size in bytes of an in-memory buffer for persistent queues. Use this parameter whenever memory is available. The **TuningParameters** stanza cannot be accessed from the WMQ Explorer.

- *QM_name* > **TuningParameters > DefaultPQBufferSize**.
  - 1MB is enough. But reduce this if you are going to have many subscribers as each subscriber will be allocated its own queue and hence 1MB of memory will be allocated for each subscriber.

Maximum channels. Controls the allowable number of concurrent CLIENT transport clients. Use this parameter when large numbers of clients are being used.

- *QM_name* > **Channels > MaxChannels**.
  - set MaxChannels high enough to contain the maximum number of concurrent JMS clients.

Channel application bind type. Controls if the channel application is an MQ FASTPATH application. Use this parameter at all times.

- *QM_name* > **Channels > MQIBindType**.
  - set MQIBindType to FASTPATH.

Restart the queue manager for the changes to take effect.

2.1.2.1 Other Settings:

The LogType value should be set to “Circular” for benchmarking measurements as that will make the operator management of the logs much simpler.

If many subscribers are to be used then increase the number of open queue handles of the queue manager to allow for the number of subscribers. Set this value using runmqsc e.g. “alter qmgr maxhands(1000)”.

2.1.2.2 Tuning Settings on UNIX

To set the above tuning settings on unix you need to modify the queue managers

.ini file as shown below, and restart the queue manager.

```
#*******************************************************************#
#*                                                                 *#
#* Module Name: qm.ini                                             *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*******************************************************************#

ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfu
   ComponentDataSize=0

Log:
   LogPrimaryFiles=16
   LogSecondaryFiles=2
   LogFilePages=16348
   LogType=CIRCULAR
   LogBufferPages=4096
   LogPath=/var/mqm/log/QMGR/

   LogWriteIntegrity=SingleWrite

Channels:
   MaxChannels=5000
   PipeLineLength=2
   MQIBindType=FASTPATH

TuningParameters:

 DefaultPQBufferSize=1048576
```

2.1.3 WebSphere Message Broker

A message flow consisting of an MQInput node wired to a Publication node needs to be created. By default the transaction parameter on the MQInput node is set to automatic. This is the recommended value to use for transaction mode unless there is a specific requirement to use a particular value since persistent messages will be processed within transactional control and non persistent messages will not. The queue to publish messages needs to be set on the Input Node and the queue created on the brokers QM (Using runmqsc or WMQ Explorer). This flow should be added to a Broker Archive File and deployed to the broker.

At this point the broker is ready for use. However if improved performance is a requirement then the following steps are advisable:

1) Make the broker run as a trusted WebSphere MQ application. To do this stop the broker and run the command "*mqsichangebroker WBRK6_DEFAULT_BROKER -t*".

2) Increase the heap size of the WebSphere Message Broker Java Virtual Machine (JVM) (in which much of the publish subscribe code is executed) to 512MB "*mqsichangeproperties WBRK6_DEFAULT_BROKER -o ComIbmJVMManager -n jvmMaxHeapSize -v 536870912*"

3) If messages over 100k are to be sent then the brokers maxMessageSize property needs to be increased. "*mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o DynamicSubscriptionEngine –n maxMessageSze –v 1000000"*

4) Increase queueCacheMaxSize. Ideally the value should be at least the number of subscribers who will receive the publications. Be aware that increasing the size of the cache will have an effect on the amount of storage used by the execution group. If the number of entries in the cache is too small then one queue has to be closed and a new one opened before the message can be written to required queue. This is repeated for each subscriber over the cache size. This becomes costly if there are many more subscribers than entries in the cache. In order to increase the size of the cache use the following command: "*mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o ComIbmMQConnectionManager -n queueCacheMaxSize -v <size>*"

5) Consider the use of additional instances. This enables multiple copies of the same flow to run within the execution group and can offer improved performance for multiprocessor machines. This is set using the configure tab when editing a Broker Archive file.

2.1.4 JMS Client (Publisher and Subscribers)

2.1.4.1 JNDI Definitions

The relevant JNDI definitions will need to be defined for your broker for use by your JMS Client application. An example of settings to use with JMSAdmin is shown below. (WMQ Explorer with Service levels from V6:0:2:0 can also be used to update this information).

```
# Delete any previously created definitions
DEL TCF(TCF)
DEL T(TestTopic)
#
# Define a TopicConnectionFactory
# Only parameters being overridden from their default values are specified.
# This sets up a MQ client binding. For a server binding remove the last 3
DEF TCF(TCF) +
QMANAGER(WBRK6_DEFAULT_QUEUE_MANAGER) +
BROKERVER(V2) +
CLIENTID(JMSCLIENT_TESTING) +
TRANSPORT(CLIENT) +
HOSTNAME(localhost.hursley.ibm.com) +
PORT(2414) +
BROKERPUBQ(PUBLISHQ) +
BROKERQMGR(WBRK6_DEFAULT_QUEUE_MANAGER) +
CHANNEL(SYSTEM.DEF.SVRCONN) +
```

```
BROKERDURSUBQ(SYSTEM.JMS.D.*)+
PUBACKINT(100)+
# Define a Topic
DEF T(TestTopic) +
TOPIC(TestTopic) +
BROKERVER(V2) +
#
END
```

More information on JMSAdmin is available in the *WebSphere MQ Using Java* manual. Also SupportPac MS0N offers a GUI to the JMS Admin tool and available here:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24004691&loc=en_US&cs=utf-8&lang=en

2.1.4.2 JMS Client Applications

If you need to write your own specific PubSub Client Application then a simple JMS example is shown in the appendix in Chapter 7 .

If you are only interested in driving the broker to a high throughput or sending a few simple messages the following tools will be useful:

The JMS Performance Harness tool can be very useful as a client application to drive pubsub message flows, the readme contains examples how running the tool against Message Broker. It provides a fast flexible way to drive messages through the broker and it reports simple throughput statistics, it can also be used with requiring JNDI definitions:

http://www.alphaworks.ibm.com/tech/perfharness

If you wish to publish a single message at a time or examine the message contents then a better tool may be IH03: WebSphere Message Broker V6 - Message display, test and performance utility:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24000637&loc=en_US&cs=utf-8&lang=en

2.1.5 Notable Features

When using the WebSphere MQ transport the publish rate is throttled by the MQ acknowledgement protocol to a rate which was sustainable by the broker. The publisher acknowledgement interval controls this throttling mechanism. This is set in the client either programmatically or using JNDI with the JMSAdmin setting “PUBACKINT”. This value should be increased to ensure messages were always available on the brokers input queue. This ensures the broker is never waiting for input and will show better throughput rates.

Each subscriber should be allocated its own temporary dynamic queue to store its messages on the broker. This is set in the client either programmatically or using JNDI with the JMSAdmin setting BROKERDURSUBQ.

2.1.6 Hints and Tips

- Consider the use of batching messages i.e. sending more than one message per transaction. This will reduce the overhead per message of transactions and increase performance.
- For ease of use when using JNDI it can be helpful to place the JNDI definitions on a network share and map this as a shared drive on the client machines. This avoids the need to copy .bindings files between boxes and any changes that are made will be picked up by all clients.
- To view current Queue Depths of the Publish Queue and any Subscriber Queues use WebSphere MQ Explorer, or the runmqsc command. Remember to enable it to show System and temporary queues.
- If many clients are to be used then consider spreading them over multiple JVMs on multiple machines. This is to ensure that in any testing the clients do not become the bottle neck in the system.

2.1.7 Problem Determination

- To view subscriptions in the DB2 database, useful if you think you have a subscriber left over but the subscription does not show up in the GUI: "*mqsibrowse WBRK6_DEFAULT_BROKER -t BSUBSCRIPTIONS"*
- If the clients are unable to connect to the broker, check the broker is up and running, that the QM listener is running, and the clients are configured to use the correct port.
- If the publish queue is high during testing then the broker may be struggling to keep up with the current rate, reducing the Publisher acknowledgement interval may help.

Technotes: For current information about known problems and available fixes, look at: http://www.ibm.com/software/integration/wbiMessage Broker/support/

Either enter a search string and check **Solve a Problem** before clicking **Submit** Or click 'Technotes' in the section called Self Help: Solve a Problem. All relevant Technotes are displayed, which can be ordered and searched further.

2.1.7.1 Getting Trace

MQ QM trace

To start QM trace, in a cmd shell enter:

strmqtrc

To end QM trace

endmqtrc

The trace is output to <MQinstallDir>\errors or <MQinstallDir>\trace depending on the MQ Version, all files end in .trc , this is default trace there are ~20files generated.

MQ JMS Client trace

Run the client as normal adding the flag in bold, which sets the relevant java properties to enable client

trace.

This creates trace file in ..\sub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\sub***

This creates trace file in ..\pub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\pub***

Broker trace

Turn on tracing:

mqsichangetrace WBRK6_DEFAULT_BROKER -e default -l debug -r -t -m safe -c 20000

This creates debugtrace.xml in current dir:

mqsireadlog WBRK6_DEFAULT_BROKER -e default -o debugtrace.xml -t

This creates debugtrace.log in current dir:

mqsiformatlog -i debugtrace.xml -o debugtrace.log

Turn off tracing

mqsichangetrace photon -e default -l none -r -t -m safe -c 20000

On Unix trace is logged to /var/mqsi/log

2.1.8 Performance Characteristics

Refer to the WebSphere Message Broker Performance (starting with IP**) or the WebSphere MQ Performance (starting with MP**) reports for detailed performance information.

http://www-306.ibm.com/software/integration/support/supportpacs/product.html

http://www-1.ibm.com/support/docview.wss?rs=171&context=SSFKSJ&dc=DA400&uid=swg27007150&loc=en_US&cs=UTF-8&lang=en&rss=ct171websphere

Detailed JMS Client performance information can be seen in Support Pac MP7G: JMS Performance with WebSphere MQ for Windows V6.0:

http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24010028&loc=en_US&cs=utf-8&lang=en

2.1.9 References

The stand-alone information centre for WebSphere Message Broker and WebSphere Event Broker is available on the WebSphere Documentation CD, and is also accessible from:
http://www.ibm.com/software/integration/wbiMessage Broker/library/

It is also available for download from:
ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/wmb_help_lin.zip (Linux version) and
ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/wmb_help_win.zip (Windows version).

2.1.9.1 Useful Articles:

2.1.9.2 http://www.ibm.com/developerworks/websphere/library/techarticles/0403_dunn/0403_dunn.html

http://www-128.ibm.com/developerworks/websphere/library/techarticles/0505_withers/0505_withers.html

Determining How Many Copies of a Message Flow Should Run:

http://www-128.ibm.com/developerworks/websphere/library/techarticles/0311_dunn/dunn.html

2.2 Publish/Subscribe - NP NT ND Real Time

- Configuration Checklist
- WebSphere Message Broker
- JMS Clients
- Notable Features

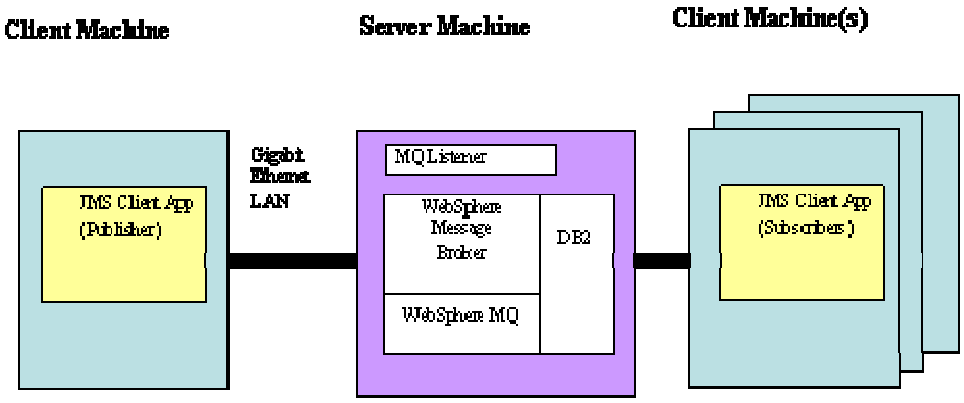
The WebSphere MQ Real-time (also known as IP) Transport is a lightweight protocol optimized for use with non-persistent messaging. This transport is used exclusively by Java Message Service clients and provides high levels of scalability and message throughput. The IP Transport is suited for use in environments where large numbers of messages need to be sent or where messages are to be sent to large numbers of client applications. The IP transport should be used by applications that can rely on the quality of service provided by TCP/IP and are not in need of persistent delivery. For example it could be used in situations where a piece of data is updated very frequently such as updating a scoreboard for a sporting event or updating a share price on a stock ticker. As this is a lightweight protocol it offers higher levels of performance for non-persistent messaging than the MQ Transport. The IP Transport does not provide any facilities for persistent messaging or durable subscriptions.

2.2.1 Configuration Checklist

The following are the steps required to get this scenario working in an efficient manner. In any Message Broker or Event Broker publish/subscribe system the setup and performance of the following three components are critical:

- The Message Broker

- The broker's WebSphere MQ queue manager
- The publishing and subscribing clients (typically JMS clients)



The following sections detail the essential and optional tuning steps for getting this scenario working. The following article shows how to setup a broker and client to use the realtime transport:

http://www-128.ibm.com/developerworks/websphere/library/techarticles/0410_bicheno/0410_bicheno.html

2.2.2 The Broker's WebSphere MQ Queue Manager

The Real-time transport does not use MQ infrastructure to transfer messages, hence tuning and queue manager settings are irrelevant in this scenario.

2.2.3 WebSphere Message Broker

To use the IP Transport your broker must be deployed with a message flow containing the RealtimeOptimizedFlow Node. The RealtimeOptimizedFlow Node is configured with a TCP/IP port number e.g 2029. This is the port on which the broker will listen for incoming connections. Client applications that use the IP Transport connect to this port. The IP Transport is a non-queued transport, applications communicate with the broker by writing data directly to TCP/IP.

At this point the broker is ready for use. However if improved performance is a requirement then the following parameters should be configured.

JVM Size

Increase the heap size of the WebSphere Message Broker Java Virtual Machine (JVM) (in which much of the publish subscribe code is executed) to 512MB "*mqsichangeproperties WBRK6_DEFAULT_BROKER -o ComIbmJVMManager -n jvmMaxHeapSize -v 536870912*"

Client Pinging

The ping protocol implements a "keep alive" protocol where the broker is periodically verifying that connected clients are alive. This process allows the broker to detect disconnected clients and

maintain an updated subscription list. In situations where there are a large number of clients connected to a broker, this pinging process may account for a large proportion of the messages exchanged between the broker and clients and can impact the broker's throughput . In such circumstances, the ping interval can be turned off or alternatively increased to reduce the amount of traffic generated by pinging.

Client Queue Size

The broker employs a set of internal queues which are used to regulate the delivery of messages to subscribers. These queues are not related to the queues used by the MQ Transport. The size of the queue specifies the number of bytes of data that the broker will store for one client. If this maximum is exceeded, the broker will take action which is determined by the next parameter "Client disconnection due to queue overflow". The default queue size is 100,000 bytes. Setting the value to zero will allow the broker to grow the queue size as required. In this case, the queue size will only be limited by the available system memory. Hence set this value with care, especially if you have limited memory available.

Client disconnection due to queue overflow

When the depth of the client queue exceeds the Client Queue Size value, the broker can choose between two courses of action. The default action is to disconnect the client; in this case, all the queued messages are lost. The other course of action is to keep the client connection alive but remove any access messages from the client's queue. The default value is true.

Maximum message size

If the broker receives a message that is bigger than the maximum message size value, it will disconnect the client that sent the message. This feature is useful for protecting the broker from applications sending excessively large messages. The default maximum message size is 100,000 bytes. If messages over 100k are to be sent then the brokers maxMessageSize property needs to be increased. *"mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o DynamicSubscriptionEngine –n maxMessageSize –v 1000000"*

Maximum number of client connections

The broker has the ability to limit the number of client connections that it will handle. This is useful in situations where the number of client applications that will connect to the broker is unknown. In such cases limiting the number of connections will allow the broker to maintain a particular level of service (this level will depend upon the particular environment in which the broker is being used). The default setting is unlimited. To modify the maximum number of connections use the following command. mqsichangeproperties <broker name> -e <execution group> -o DynamicSubscriptionEngine -n maxConnections -v <number>

Read and Write Threads

The RealtimeOptimized flow node will not scale like a traditional message flow with the use of additional instances. The broker properties read and write threads can be beneficial if you are using large number of clients. The default values for these are 10 each and these are usually sufficient.

Typical values for the above settings are shown below, these could be placed in a script to tune the broker:

mqsichangeproperties WBRK6_DEFAULT_BROKER -o DynamicSubscriptionEngine -n statsInterval -v 10000

mqsichangeproperties WBRK6_DEFAULT_BROKER -o DynamicSubscriptionEngine -n enableClientDiscOnQueueOverflow -v false

mqsichangeproperties WBRK6_DEFAULT_BROKER -o DynamicSubscriptionEngine -n maxClientQueueSize -v 0

mqsichangeproperties WBRK6_DEFAULT_BROKER -o DynamicSubscriptionEngine -n clientPingInterval -v 0

mqsichangeproperties WBRK6_DEFAULT_BROKER -o ComIbmJVMManager -n jvmMaxHeapSize -v 536870912

2.2.4 JMS Client (Publisher and Subscribers)

2.2.4.1 JNDI Definitions

The relevant JNDI definitions will need to be defined for your broker for use by you JMS Client application. An example of settings to use with JMSAdmin is shown below. (WMQ Explorer with Service levels from V6:0:2:0 can also be used to update this information).

```
# Delete any previously created definitions
DEL TCF(TCF)
DEL T(TestTopic)
#
# Define a TopicConnectionFactory
# Only parameters being overridden from their default values are specified.
# This sets up a Realtime client.
DEF TCF(TCF) +
QMANAGER(WBRK6_DEFAULT_QUEUE_MANAGER) +
BROKERVER(V2) +
CLIENTID(JMSCLIENT_TESTING) +
TRANSPORT(DIRECT) +
HOSTNAME(localhost.hursley.ibm.com) +
PORT(2029) +
# Define a Topic
DEF T(TestTopic) +
TOPIC(TestTopic) +
BROKERVER(V2) +
#
END
```

More information on JMSAdmin is available in the *WebSphere MQ Using Java* manual. Also, SupportPac MS0N offers a GUI to the JMS Admin tool and available here:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24004691&loc=en_US&cs=utf-8&lang=en

2.2.4.2 JMS Client Applications

If you need to write your own specific PubSub Client Application then a simple JMS example is shown in the appendix in Chapter 7 .

If you are only interested in driving the broker to a high throughput or sending a few simple messages the following tools will be useful:

The JMS Performance Harness tool can be very useful as a client application to drive pubsub

message flows, the readme contains examples how running the tool against Message Broker. It provides a fast flexible way to drive messages through the broker and it reports simple throughput statistics, it can also be used with requiring JNDI definitions:

http://www.alphaworks.ibm.com/tech/perfharness

2.2.5 Notable Features

The WebSphere MQ Real-time transport does not have a self throttling protocol like that of MQ. As a result the publish rate was manually adjusted until the optimum message throughout for the broker is found. The optimum level of message throughput can be determined by monitoring the ClientBytesQueued value in broker statistics.

The value for ClientBytesQueued shows the number of bytes waiting to be delivered to subscriber clients. When the broker becomes overloaded it is unable to service this buffer fast enough and so the number of bytes that are queued increases. Hence ideally the buffer size must not continually increase during the test run. Constant growth of this buffer indicates too high a publish rate. The point at which the buffer starts to fill is dependent on a combination of factors such as network bandwidth, system memory and client performance. To enable broker stats see the section later in this document.

2.2.6 Hints and Tips

- The current state of all configurable parameters for the broker can be viewed at any time by issuing the following command. "*mqsireportproperties WBRK6_DEFAULT_BROKER -e <execution group> -o DynamicSubscriptionEngine –a*". The following command will produce the same information but with more detailed reports about the subscriptions currently held by the broker. "*mqsireportproperties WBRK6_DEFAULT_BROKER -e <execution group> -o DynamicSubscriptionEngine -r* "

- Start Simple by running the client on the same box as the broker where possible. Once the setup is working move the clients of the broker machine to maximize resources available to the broker.

- If performance is important use a Gigabit Ethernet LAN if possible. On 100Mb LAN Using IP transport the network is likely to become fully utilized before the limit of the broker. Tuning of the operating system tcpip stack is also likely to provide performance improvements.

- You may wish to see the subscriptions made to the broker using the WebSphere MQ Real-time transport. To show the subscriptions, set the property to tell the broker to display non-durable subscriptions. With the broker still running, from a command prompt, run: "`mqsichangeproperties` *WBRK6_DEFAULT_BROKER* `-e default -DynamicSubscriptionEngine -n nondurableSubscriptionEvents -v true`"

- For ease of use when using JNDI it can be helpful to place the JNDI definitions on a network share and map this as a shared drive on the client machines. This avoids the need to copy .bindings files between boxes and any changes that are made will be picked up by all clients.

- If many clients are to be used then consider spreading them over multiple JVMs on multiple machines. This is to ensure that in any testing the clients to not become the bottle neck in the system.

2.2.7 Broker Statistics

2.2.7.1 Enabling Statistics Reporting

WebSphere Message Broker is capable of generating statistics reports that provide information about the performance of the broker. These statistics reports can be invaluable when trying to detect performance problems. By default, statistics reporting is disabled; it can be enabled using the command shown below. It is not necessary to stop and start the broker after running this command.

"*mqsichangeproperties <broker name> -e <execution group> -o DynamicSubscriptionEngine -n statsInterval -v <time interval>* "

There is a small performance overhead introduced by enabling statistics reporting, therefore the time interval should be chosen with care. If the statistics report is set to a high frequency (small value), the broker will spend too much time in generating statistics data. A time interval of 10000 (10 seconds) is sensible. This proves sufficiently frequent to diagnose any performance problems may occur. Should it be necessary to disable statistics reporting, it can be done by running the above command with a time interval of zero.

2.2.7.2 Subscribing to statistics reports

When statistics reporting is enabled, the broker will publish a statistics report over a pre defined time interval. This publication will be distributed to all subscribers that subscribed to the appropriate topic. The topic hierarchy for statistics reports is as follows. $SYS/Broker/<broker name>/ExecutionGroup/<execution group>/Statistics <broker name> is the name of the broker from which you want to collect statistics <execution group> is the name of the execution group deployed to the broker

Like all subscriptions, it is possible to use wildcards when subscribing for statistics data. This is particularly useful when accessing statistics reports in a multi broker environment where we can collect statistics from all participating brokers. For example, to receive statistics reports for all brokers regardless of the name of the deployed execution group, the following subscription would be used. $SYS/Broker/+/ExecutionGroup/+/Statistics It is important to note that a client can only receive statistics publication from brokers that were enabled to generate statistics, regardless if they are single or multi brokers. The statistics publication is a JMS Bytes Message containing the statistics report in an XML format. Upon delivery of this message, a suitable client application can extract the data and use it appropriately.

2.2.7.3 Understanding Statistics Reports

WebSphere Message Broker is capable of generating statistics reports that provide information about the performance of the broker. These statistics reports can be invaluable when trying to detect performance problems. The statistics data is published by the broker in XML format. The report is split into three main sections: broker, client and neighbour statistics. Each of these sections and their contents is detailed below. The information contained in the statistics report only relates to JMS IP clients. There is no equivalent report for MQ clients although it is possible to collect similar information using MQ standard reporting facilities.

This section reports general information about the broker.

Data Label	Description
Broker	The name of the broker that generated the statistics report

Execution Group	The name of the execution group running on the broker. This will normally be default unless the execution group has been renamed
Client Count	The total number of clients that connected to the broker
Neighbour Count	The number of neighbour brokers that the broker is currently connected to (this only applies in multi broker environments)
Subscription Count	The number of subscriptions currently held by the broker
Timestamp	The time at which the broker completed gathering statistics and sent the data for publication

2.2.7.4 Client Statistics

The client statistics section of the report contains data regarding message throughput between the broker and the clients, currently connected to the broker.

Data Label	Description
Bytes Queued	The number of bytes of data currently queued by the broker for delivery to clients
Messages Sent	The number of messages the broker has delivered to clients
Bytes Sent	As above but measured in bytes
Bytes Cut Through	The number of bytes of data sent immediately to clients, without being queued internally by the broker
Messages Received	The number of messages received by the broker from clients
Bytes Received	As above but measured in bytes
Messages Dropped	The number of messages dropped due to queue overflow, where the client was not subsequently disconnected from the broker
Bytes Dropped	As above but measured in bytes

2.2.7.5 Problem Determination

- If the clients are unable to connect to the broker check the broker is up and running and the clients are configured to use the correct port.
- Real-time clients are restricted currently to messages sizes not exceeding 100k bytes.
- If the Subscriber has “Overflow Exceptions”. The subscriber, when using WebSphere MQ Real-time client, contains a message buffer to protect itself from message rate spikes. If you are likely to use high message increase from the default (1000) to hold 3000 messages. For information on how to set the subscriber max buffer size see in the WebSphere MQ "Using Java" manual.

Technotes: For current information about known problems and available fixes, look at:
http://www.ibm.com/software/integration/wbiMessage Broker/support/

Either enter a search string and check **Solve a Problem** before clicking **Submit** Or click 'Technotes' in the section called Self Help: Solve a Problem. All relevant Technotes are displayed, which can be ordered and searched further.

2.2.7.6 Getting Trace

MQ JMS Client trace

Run the client as normal adding the flag in bold, which sets the relevant java properties to enable client trace.

This creates trace file in ..\sub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\sub***

This creates trace file in ..\pub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\pub***

Broker trace

Turn on tracing:

mqsichangetrace WBRK6_DEFAULT_BROKER -e default -l debug -r -t -m safe -c 20000

This creates debugtrace.xml in current dir:

mqsireadlog WBRK6_DEFAULT_BROKER -e default -o debugtrace.xml -t

This creates debugtrace.log in current dir:

mqsiformatlog -i debugtrace.xml -o debugtrace.log

Turn off tracing

mqsichangetrace photon -e default -l none -r -t -m safe -c 20000

On Unix trace is logged to /var/mqsi/log

2.3 Publish/ Subscribe - Scalable WMQ MB NP NT ND

- Configuration Checklist
- WebSphere MQ Queue Manager
- WebSphere Message Broker
- JMS Clients
- Notable Features

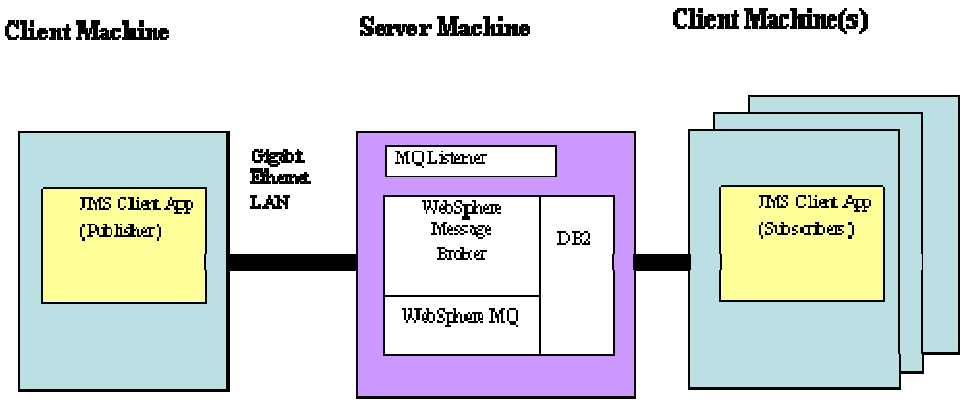
This scenario covers the use of Message Broker and Event Broker publish/subscribe using the WebSphere MQ transport with non-transacted, non-persistent messaging and non-durable subscribers. This type of messaging should be used where messages need to be sent with a high level of reliability but without the need for persistence, that is, in the event of software or a hardware failure, message loss can be tolerated. In this scenario we are assuming that the clients will be hosted on a machine separate to the broker and will use the MQ Client transport, if the client is to be run on the same machine as the broker than the MQ Bindings transport could be used. This scenario would be used instead of the real-time transport if reliability rather than performance is more important or if the future requirement for persistent messaging is likely.

2.3.1.1 Configuration Checklist

Create the default configuration during installation and add the JMS Queues to the brokers QM.

The following steps are required to get this scenario working in an efficient manner. In any Message Broker or Event Broker publish/subscribe system, the setup and performance of the following three components are critical:

- The Message Broker
- The broker's WebSphere MQ queue manager
- The publishing and subscribing clients (typically JMS clients)



The following sections details the essential and the optional tuning steps to get this scenario working:

2.3.2 The Broker's WebSphere MQ Queue Manager

Chapter 2.1.2 discuss various ways of setting significant MQ Parameters.

The following changes are not required but they are advisable if performance is likely to be an issue:

Given the use of non-persistent messages, the following MQ parameters should be modified:

Default queue buffer size. Controls the size in bytes of an in-memory buffer for nonpersistent queues. Use this parameter when large message sizes are used, or large bursts of messages cause the queue to back up. If the queue backs up past this buffer, messages are flushed out to the disk.

QM_name > **TuningParameters > DefaultQBufferSize**.

- Set this parameter to accommodate the typical number of messages sitting on the queue at any given time. This should be numberOfMessages*(messageSizeInBytes+500). The maximum value is 100MB, but typically 1MB is enough.

Maximum channels. Controls the allowable number of concurrent CLIENT transport clients. Use this parameter when large numbers of clients are being used.

QM_name > **Channels > MaxChannels**.

- set MaxChannels high enough to contain the maximum number of concurrent JMS clients.

Channel application bind type. Controls if the channel application is an MQ FASTPATH application. Use this parameter at all times.

- *QM_name* > **Channels > MQIBindType**.
  - set MQIBindType to FASTPATH.

2.3.2.1 Other Settings:

If many subscribers are to be used then increase the number of open queue handles of the queue manager to allow for the number of subscribers. Set this value using runmqsc e.g. “alter qmgr maxhands(1000)”.

2.3.2.2 Tuning Settings on UNIX

To set the above tuning settings on unix you need to modify the queue managers .ini file as shown below, and restart the queue manager.

```
#*******************************************************************#
#*                                                                 *#
#* Module Name: qm.ini                                             *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*******************************************************************#

ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

Service:
   Name=AuthorizationService
```

```
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfu
   ComponentDataSize=0

Log:
   LogPrimaryFiles=16
   LogSecondaryFiles=2
   LogFilePages=16348
   LogType=CIRCULAR
   LogBufferPages=4096
   LogPath=/var/mqm/log/QMGR/

   LogWriteIntegrity=SingleWrite

Channels:
   MaxChannels=5000
   PipeLineLength=2
   MQIBindType=FASTPATH



TuningParameters:

   DefaultQBufferSize=1048576
```

2.3.3 WebSphere Message Broker

A message flow consisting of an MQInput node wired to a Publication node needs to be created. By default the transaction parameter on the MQInput node is set to automatic. This is the recommended value to use for transaction mode unless there is a specific requirement to use a particular value since persistent messages will be processed within transactional control and non persistent messages will not. The queue to publish messages needs to be set on the Input Node and the queue created on the brokers QM (Using runmqsc or WMQ Explorer). This flow should be added to a Broker Archive File and deployed to the broker.

At this point the broker is ready for use. However if improved performance is a requirement then the following steps are advisable:

1) Make the broker run as a trusted WebSphere MQ application. To do this stop the broker and run the command "*mqsichangebroker WBRK6_DEFAULT_BROKER -t*".

2) Increase the heap size of the WebSphere Message Broker Java Virtual Machine (JVM) (in which much of the publish subscribe code is executed) to 512MB "*mqsichangeproperties WBRK6_DEFAULT_BROKER -o ComIbmJVMManager -n jvmMaxHeapSize -v 536870912*"

3) If messages over 100k are to be sent then the brokers maxMessageSize property needs to be increased. "*mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o DynamicSubscriptionEngine –n maxMessageSze –v 1000000"*

4) Increase queueCacheMaxSize. Ideally the value should be at least the number of subscribers who will receive the publications. Be aware that increasing the size of the cache will have an

effect on the amount of storage used by the execution group. If the number of entries in the cache is too small then one queue has to be closed and a new one opened before the message can be written to required queue. This is repeated for each subscriber over the cache size. This becomes costly if there are many more subscribers than entries in the cache. In order to increase the size of the cache use the following command: "*mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o ComIbmMQConnectionManager -n queueCacheMaxSize -v <size>*"

5) Consider the use of additional instances. This enables multiple copies of the same flow to run within the exeeution group and can offer improved performance for multiprocessor machines. This is set using the configure tab when editing a Broker Archive file.

2.3.4 JMS Client (Publisher and Subscribers)

2.3.4.1 JNDI Definitions

The relevant JNDI definitions will need to be defined for your broker for use by you JMS Client application. An example of settings to use with JMSAdmin is shown below. (WMQ Explorer with Service levels from V6:0:2:0 can also used to update this information).

```
# Delete any previously created definitions
DEL TCF(TCF)
DEL T(TestTopic)
#
# Define a TopicConnectionFactory
# Only parameters being overridden from their default values are specified.
# This sets up a MQ client binding. For a server binding remove the last 3
DEF TCF(TCF) +
QMANAGER(WBRK6_DEFAULT_QUEUE_MANAGER) +
BROKERVER(V2) +
PERSISTENCE(NON) +
CLIENTID(JMSCLIENT_TESTING) +
TRANSPORT(CLIENT) +
HOSTNAME(localhost.hursley.ibm.com) +
PORT(2414) +
BROKERPUBQ(PUBLISHQ) +
BROKERQMGR(WBRK6_DEFAULT_QUEUE_MANAGER) +
CHANNEL(SYSTEM.DEF.SVRCONN) +
BROKERSUBQ(SYSTEM.JMS.ND.*)+
PUBACKINT(100)+
# Define a Topic
DEF T(TestTopic) +
TOPIC(TestTopic) +
BROKERVER(V2) +
#
END
```

More information on JMSAdmin is available in the *WebSphere MQ Using Java* manual. Also, SupportPac MS0N offers a GUI to the JMS Admin tool and available here:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24004691&loc=en_US&cs=utf-8&lang=en

2.3.4.2 JMS Client Applications

If you need to write your own specific PubSub Client Application then a simple JMS example is shown in the appendix in Chapter 7 .

If you are only interested in driving the broker to a high throughput or sending a few simple messages the following tools will be useful:

The JMS Performance Harness tool can be very useful as a client application to drive pubsub message flows, the readme contains examples how running the tool against Message Broker. It provides a fast flexible way to drive messages through the broker and it reports simple throughput statistics, it can also be used with requiring JNDI definitions:

http://www.alphaworks.ibm.com/tech/perfharness

If you wish to publish a single message at a time or examine the message contents then a better tool may be IH03: WebSphere Message Broker V6 - Message display, test and performance utility:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24000637&loc=en_US&cs=utf-8&lang=en

2.3.5 Notable Features

When using the WebSphere MQ transport the publish rate is throttled by the MQ acknowledgement protocol to a rate which was sustainable by the broker. The publisher acknowledgement interval controls this throttling mechanism. This is set in the client either programmatically or using JNDI with the JMSAdmin setting “PUBACKINT”. This value should be increased to ensure messages were always available on the brokers input queue. This ensures the broker is never waiting for input and will show better throughput rates.

Each subscriber should be allocated its own temporary dynamic queue to store its messages on the broker. This is set in the client either programmatically or using JNDI with the JMSAdmin setting BROKERSUBQ.

2.3.6 Hints and Tips

- For ease of use when using JNDI it can be helpful to place the JNDI definitions on a network share and map this as a shared drive on the client machines. This avoids the need to copy

bindings files between boxes and any changes that are made will be picked up by all clients.

- To view current Queue Depths of the Publish Queue and any Subscriber Queues use WebSphere MQ Explorer, or the runmqsc command. Remember to enable it to show System and temporary queues.

- If many clients are to be used then consider spreading them over multiple JVMs on multiple machines. This is to ensure that in any testing the clients do not become the bottle neck in the system.

2.3.7 Problem Determination

- If the clients are unable to connect to the broker check the broker is up and running, that the QM listener is running and the clients are configured to use the correct port.

- If the publish queue is high during testing then the broker may be struggling to keep up with the current rate, reducing the Publisher acknowledgement interval may help.

Technotes: For current information about known problems and available fixes, look at:
http://www.ibm.com/software/integration/wbiMessage Broker/support/

Either enter a search string and check **Solve a Problem** before clicking **Submit** Or click 'Technotes' in the section called Self Help: Solve a Problem. All relevant Technotes are displayed, which can be ordered and searched further.

2.3.7.1 Getting Trace

MQ QM trace

To start QM trace, in a cmd shell enter:

strmqtrc

To end QM trace

endmqtrc

The trace is output to <MQinstallDir>\errors or <MQinstallDir>\trace depending on the MQ Version, all files end in .trc , this is default trace there are ~20files generated.

MQ JMS Client trace

Run the client as normal adding the flag in bold, which sets the relevant java properties to enable client trace.

This creates trace file in ..\sub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\sub***

This creates trace file in ..\pub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\pub***

Broker trace

Turn on tracing:

mqsichangetrace WBRK6_DEFAULT_BROKER -e default -l debug -r -t -m safe -c 20000

This creates debugtrace.xml in current dir:

mqsireadlog WBRK6_DEFAULT_BROKER -e default -o debugtrace.xml -t

This creates debugtrace.log in current dir:

mqsiformatlog -i debugtrace.xml -o debugtrace.log

Turn off tracing

mqsichangetrace photon -e default -l none -r -t -m safe -c 20000

On unix trace is logged to /var/mqsi/log

2.4 Publish/Subscribe – High Performance Multicast

- Configuration Checklist
- WebSphere Message Broker
- JMS Clients
- Notable Features

WebSphere MQ Multicast Transport is a service that connects dedicated JMS application clients and is optimized for high volume, one-to-many publish/subscribe topologies. This transport is used exclusively by Java Message Service clients and provides high levels of scalability and message throughput. The Multicast Transport is suited for use in environments where large numbers of messages need to be sent or where messages are to be sent to large numbers of subscriber applications. The Multicast transport should be used by applications that can rely on the quality of service provided by multicast and are not in need of persistent delivery. For example it could be used in situations where a piece of data is updated very frequently such as updating a scoreboard for a sporting event or updating a share price on a stock ticker. As this is a lightweight protocol it offers higher levels of performance for non-persistent messaging than the MQ Transport. The Multicast Transport does not provide any facilities for persistent messaging or durable subscriptions.

In a publish/subscribe system normally a separate message is sent to each subscriber of a publication. However, with *multicast*, regardless of how many subscribers to a topic there are on a subnet, only one message is sent. This improves network utilization. The more subscribers there are in your publish/subscribe system, the greater the improvement to network utilization there might be if you use multicast.

Message Broker Multicast based on the PTL protocol has 2 modes of operation:

- **Unreliable** – this is based on the UDP protocol and offers the lowest quality of service.

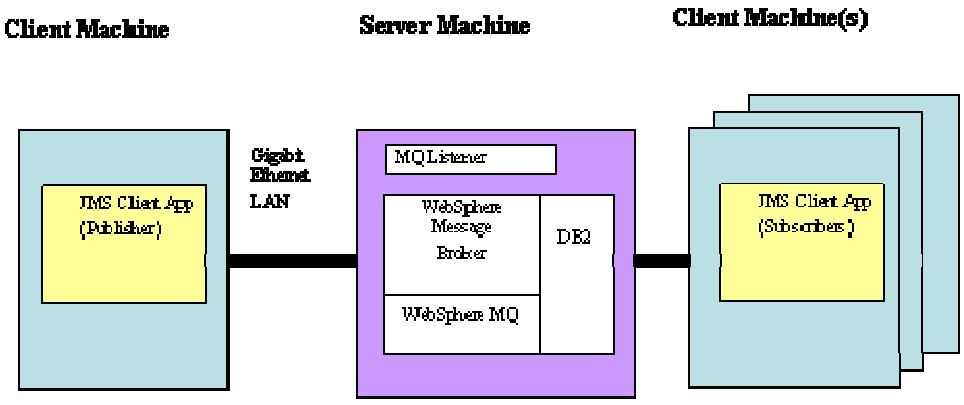
- **Reliable** – this adds a layer of reliability on the UDP multicast protocol and uses a history of sent messages to resend to subscribers to ensure a better quality of service. This improved reliability comes with a performance cost. The mode used is set by setting up the client connection factory and the broker topics appropriately.

2.4.1 Configuration Checklist

Create the default configuration and add the JMS queues to the broker's queue manager.

The following are the steps required to get this scenario working in an efficient manner. In any Message Broker or Event Broker publish/subscribe system the setup and performance of the following three components are critical:

- The Message Broker
- The broker's WebSphere MQ queue manager
- The publishing and subscribing clients (typically JMS clients)



The following sections detail the essential and optional tuning steps for getting this scenario working.

2.4.2 The Broker's WebSphere MQ Queue Manager

The Multicast transport does not use MQ infrastructure to transfer messages, hence tuning and queue manager settings are irrelevant in this scenario.

2.4.3 WebSphere Message Broker

The multicast transport is only actually used as a transport between the broker and subscribers. The publishers will typically use the real-time transport and subscribers will use the multicast transport. Hence the real-time setup and tuning will be required, see Chapter 2.2 on Publish/Subscribe High Performance Real-time.

To use the multicast transport your broker must be deployed with a message flow containing the RealtimeOptimizedFlow Node. The RealtimeOptimizedFlow Node is configured with a port number, for example, 2029. This is the port on which the broker will listen for incoming

connections. Client applications that use the multicast transport connect to this port.

Now the broker must be enabled for multicast. To make a broker capable of handling multicast requests, complete the following tasks from within the workbench:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double-click on the Topology item to open the Broker Topology editor.
4. In the Broker Topology editor, right-click the broker that you want to modify, and select Properties.
5. In the left panel of the properties window, select Multicast.
6. Select the Multicast Enabled check box.

Note: There is a tooling bug for v6 that was fixed in WMB V6 fixpac3: it deploys the value "None;None" to the broker which is expecting the value "None" or a valid ip or hostname for the multicast interface.

There is a workaround, which is to override the setting on the broker from the command line. The first thing to do is run the command to check the value that is deployed to the broker and check whether it has the erroneous ";None"

mqsireportproperties <brokername> -e <exegrpname> -o DynamicSubscriptionEngine -r

Check the value of the multicastMulticastInterface, then assuming it is wrong, replace it with:

mqsichangeproperties <brokername> -e <exegrpname> -o DynamicSubscriptionEngine -n multicastMulticastInterface -v none

If you use this method of changing broker properties, you need to bear in mind that, if you modify the broker properties in the toolkit and redeploy, the command line changes are overwritten so the "none;none" value will be reset.

The topics that you wish to publish/subscribe to must also be enabled for multicast. To make individual topics, or groups of topics, capable of being multicast you need to make changes to the topic hierarchy:

1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the appropriate broker domain.
3. Double-click on the Topics item to open the Topics Hierarchy editor.
4. In the Topics Hierarchy editor, right-click the topic, or group of topics, that you want to make capable of being multicast, and select properties.
5. In the left panel of the properties window, select Multicast.
6. Choose the Multicast Enabled required.

- For the topic root, the choice is either Enabled or Disabled. The default is Disabled.
- For a child topic root, the choice can be Inherit, Enabled, or Disabled. The default is Inherit.

7. Check the Automatic Multicast Address box, or type in the name of the MC Group Address.
8. Choose the Quality of Service required. The choice is between Reliable or Unreliable. The default is Reliable.
9. Optional: Select the Encrypted check box.
10. Click OK.

At this point the broker is ready for use. However, if improved performance is a requirement then the following parameters must be configured:

JVM Size

Increase the heap size of the WebSphere Message Broker Java Virtual Machine (JVM) (in which much of the publish subscribe code is executed) to 512MB "*mqsichangeproperties WBRK6_DEFAULT_BROKER -o ComIbmJVMManager -n jvmMaxHeapSize -v 536870912*"

Maximum message size

If the broker receives a message that is bigger than the maximum message size value, it will disconnect the client that sent the message. This feature is useful for protecting the broker from applications sending excessively large messages. The default maximum message size is 100,000 bytes. If messages over 100k are to be sent then the brokers maxMessageSize property needs to be increased. "*mqsichangeproperties WBRK6_DEFAULT_BROKER -e <execution group > -o DynamicSubscriptionEngine –n maxMessageSize –v 1000000"*

Maximum number of client connections

The broker has the ability to limit the number of client connections that it will handle. This is useful in situations where the number of client applications that will connect to the broker is unknown. In such cases limiting the number of connections will allow the broker to maintain a particular level of service (this level will depend upon the particular environment in which the broker is being used). The default setting is unlimited. To modify the maximum number of connections use the following command. mqsichangeproperties <broker name> -e <execution group> -o DynamicSubscriptionEngine -n maxConnections -v <number>

The following properties are some of the most important broker properties that will impact the performance of multicast. These can be set from the toolkit when you enable the broker for multicast. There are other properties that can also be set and are detailed in the help.

Broker Packet Size

The size, in bytes, of multicast packets. This can be in the range 500 through 32000. The default value is 7000. If you are sending only small messages this value should be reduced.

Broker Multicast TTL

The maximum number of hops that a multicast packet can make between the client and the broker.

This value is one more than the maximum number of routers that there can be between the client and the broker. The default value is 1, which means that the multicast packet must remain local to its originator and does not pass through any routers. The maximum value is 255. Adjust this value according to your local network setup.

Client Packet Buffer Number

The number of memory buffers that are created at startup for packet reception. Having a high number of buffers available improves the reception performance and minimizes packet loss at high delivery rates, but requires increased memory use. Each buffer is 33 KB; having 500 buffers (the default value) uses approximately 15 MB of main memory. If memory use is important, try using different values for this parameter and look at the effect on the overall performance of your application when transmission rates are high. This value can be in the range 1 through 5000. The default value is 500. Increase this to 5000 for high message rates assuming you have sufficient memory.

Client Socket Buffer Size

The size, in kilobytes, of the client's socket receiver buffer. Increasing this value reduces the number of data packets that might be dropped by the client receiver. This value can be in the range 65 through 10000. The default value is 3000. If memory is available set this value to the maximum.

Broker History Cleaning Time (only for reliable mode)

The time, in seconds, that is defined for cleaning the retransmission buffer. This value can be in the range 1 through 20. The default value is 7. Increase this number for improved reliability as the history of sent messages will be stored for longer.

Broker Minimal History Size (only for reliable mode)

The minimum size, in kilobytes, of a buffer that is allocated as an archive for all transmitted packets. This buffer is shared by all reliable topics, and can be used to recover lost packets. This value can be in the range 1000 through 1,000,000. The default value is 60,000. Increase this number for improved reliability as a larger history of sent messages will be stored for retransmission.

2.4.4 JMS Client (Publisher and Subscribers)

2.4.4.1 JNDI Definitions

The relevant JNDI definitions will need to be defined for your broker for use by you JMS Client application. An example of settings to use with JMSAdmin is shown below. (WMQ Explorer with Service levels from V6:0:2:0 can also be used to update this information).

```
# Delete any previously created definitions
DEL TCF(TCF)
DEL T(TestTopic)
#
# Define a TopicConnectionFactory
# Only parameters being overridden from their default values are specified.
# This sets up a Multicast client in reliable mode
DEF TCF(TCF) +
QMANAGER(WBRK6_DEFAULT_QUEUE_MANAGER) +
```

```
BROKERVER(V2) +
CLIENTID(JMSCLIENT_TESTING) +
TRANSPORT(DIRECT) +
#use the value NOTR for unreliable mode
MULTICAST(RELIABLE)+
HOSTNAME(localhost.hursley.ibm.com) +
PORT(2029) +
# Define a Topic
DEF T(TestTopic) +
TOPIC(TestTopic) +
BROKERVER(V2) +
#
END
```

More information on JMSAdmin is available in the *WebSphere MQ Using Java* manual. Also, SupportPac MS0N offers a GUI to the JMS Admin tool, which is available here:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24004691&loc=en_US&cs=utf-8&lang=en

2.4.4.2 JMS Client Applications

If you need to write your own specific PubSub Client Application then a simple JMS Example is shown in the appendix in Chapter 7.

If you are only interested in driving the broker to a high throughput or sending a few simple messages the following tools will be useful:

The JMS Performance Harness tool can be very useful as a client application to drive pub sub message flows, the read me contains examples how running the tool against Message Broker. It provides a fast flexible way to drive messages through the broker and it reports simple throughput statistics. It can also be used with requiring JNDI definitions. This tool is available here:

http://www.alphaworks.ibm.com/tech/perfharness

2.4.5 Notable Features

2.4.5.1 Hints and Tips

- The current state of all configurable parameters for the broker can be viewed at any time by issuing the following command. "*mqsireportproperties WBRK6_DEFAULT_BROKER -e <execution group> -o DynamicSubscriptionEngine –a*". The following command will produce the same information but with more detailed reports about the subscriptions currently held by the broker. "*mqsireportproperties WBRK6_DEFAULT_BROKER -e <execution group> -o DynamicSubscriptionEngine -r* "

- Start Simple by running the client on the same box as the broker where possible. Once the setup is working move the clients of the broker machine to maximize resources available to the broker.

- If performance is important use a Gigabit Ethernet LAN if possible. On 100Mb LAN Using IP

transport the network is likely to become fully utilized before the limit of the broker. Tuning of the operating system tcpip stack is also likely to provide performance improvements.

- You may wish to see the subscriptions made to the broker using the WebSphere MQ Multicast transport. To show the subscriptions, set the property to tell the broker to display non-durable subscriptions. With the broker still running, from a command prompt, run:
  "`mqsichangeproperties` *WBRK6_DEFAULT_BROKER* `-e default -DynamicSubscriptionEngine -n nondurableSubscriptionEvents -v true`"

- For ease of use when using JNDI it can be helpful to place the JNDI definitions on a network share and map this as a shared drive on the client machines. This avoids the need to copy .bindings files between boxes and any changes that are made will be picked up by all clients.

- If many clients are to be used then consider spreading them over multiple JVMs on multiple machines. This is to ensure that in any testing the clients to not become the bottle neck in the system.

- To check that the topics are correct in the broker database:
  mqsibrowse <broker>-t BMULTICASTTOPICS

2.4.5.2 Broker Statistics

When statistics reporting is enabled for a broker that is enabled for multicast, the broker publishes statistics reports at regular intervals (as determined by the value specified for the statsInterval property of the broker). The statistics reports are distributed, as publications, to all subscribers that subscribed to the following topics:

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Topics`

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Groups`

`$SYS/Broker/`*`broker name`*`/ExecutionGroup/`*`execution group`*`/Statistics/Multicast/Summary`

where *broker name* is the name of the broker, and *execution group* is the name of the execution group that is deployed to that broker.

2.4.6 Problem Determination

- If the clients are unable to connect to the broker, check that the broker is up and running and the clients are configured to use the correct port.

- If the Subscriber has "Overflow Exceptions". The subscriber, when using WebSphere MQ Multicast client, contains a message buffer to protect itself from message rate spikes. If you are likely to use high message increase from the default (1000) to hold 3000 messages. For information on how to set the subscriber max buffer size see in the WebSphere MQ *Using Java* manual.

- Check Unix boxes in the multicast network should have a flag set showing that multicast is enabled. On most systems, the command to display this is "ifconfig -a".

- On AIX we found an additional issue due to the box being enabled for IPv6 addresses when we were using IPv4 addresses.
  This can cause the following style of error when running a client:

com.ibm.rmm.util.exceptions.FailedMulticastInterfaceSet: RMM Error: Multicast interface setting to achillea.hursley.ibm.com/9.20.139.232 failed
at com.ibm.rmm.util.RmmLogger.checkFatalProblem(RmmLogger.java:201)
at com.ibm.rmm.util.RmmLogger.baseLog(RmmLogger.java:234)
at com.ibm.rmm.ptl.mstp.transmitter.PacketFireout.run(PacketFireout.java:93)
Caused by: java.net.SocketException: The socket name is not available on this system.
at java.net.PlainDatagramSocketImpl.socketSetOption(Native Method)
at java.net.PlainDatagramSocketImpl.setOption(PlainDatagramSocketImpl.java:295)
at java.net.MulticastSocket.setInterface(MulticastSocket.java:440)
at com.ibm.rmm.ptl.mstp.transmitter.PacketFireout.run(PacketFireout.java:89)
There are two methods to resolve this. Firstly, you can run a Java program with the following flag:
-Djava.net.preferIPv4Stack=true -Djava.net.preferIPv6Addresses=false
and/or you can try setting the following on your profile:
export IBM_JAVA_OPTIONS="-Djava.net.preferIPv4Stack=true -Djava.net.preferIPv6Addresses=false"

- Valid IPv4 ranges. The broker allows topics to be assigned an address between 224.0.0.0 to 239.255.255.255. Note that 224.0.0.x is a reserved address range, used for IGMP routing traffic. Some values in 224.0.1.x may also be reserved.
  If you assign addresses in the reserved range to topics, multicast messages may well be bounced back from the router and will give the impression that the network is not functioning. IBM Haifa recommend that the following addresses are used:
  - The best range for site-local multicast address is 239.255.x.y
  - The best range for organization-local multicast address is 239.192-251.x.y
  However we do not need to confine ourselves to this range in testing.

- Crossing the Router - Time to Live parameter. When a multicast message is transmitted, it is allowed to perform a certain number of router hops on the network before the message expires, expressed by the TimeToLive parameter, which can be set both on the broker GUI and the UdpR test connectivity program.
  In theory a TimeToLive value of 0 could be used to check that multicast messages are not transmitted from a broker. However, some operating systems such as Linux and some Windows variants cannot cope with 0 and default to a value of 1 instead. Try not to use a TimeToLive of 0. A TimeToLive of 1 should allow multicast messages to travel within a subnet, but not outside it. A TimeToLive value of 2 will allow a multicast message to hop across 1 router.

Technotes: For current information about known problems and available fixes, see:
http://www.ibm.com/software/integration/wbiMessage Broker/support/

Either enter a search string and check **Solve a Problem** before clicking **Submit** Or click 'Technotes' in the section called Self Help: Solve a Problem. All relevant Technotes are displayed, and they can be ordered and searched further.

2.4.6.1 Getting Trace

MQ JMS Client trace

Run the client as normal adding the flag in bold, which sets the relevant java properties to enable client trace.

This creates trace file in ..\sub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\sub***

This creates trace file in ..\pub dir, make sure this exists

java ***-DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=..\pub***

Broker trace

To turn on tracing:

mqsichangetrace WBRK6_DEFAULT_BROKER -e default -l debug -r -t -m safe -c 20000

This creates debugtrace.xml in current dir:

mqsireadlog WBRK6_DEFAULT_BROKER -e default -o debugtrace.xml -t

This creates debugtrace.log in current dir:

mqsiformatlog -i debugtrace.xml -o debugtrace.log

Turn off tracing

mqsichangetrace photon -e default -l none -r -t -m safe -c 20000

On Unix trace is logged to /var/mqsi/log

2.5 Point-to-Point Persistent, transacted

- Configuration Checklist
- Websphere MQ Queue Manager
- Notable Features

Persistent point-to-point, transactional messaging is generally used where messages need to be sent with a high level of reliability in the scenario that, in the possibility that a hardware or software, failure messages are not lost. With the increased reliability there is some trade-off to performance to ensure messages are logged to disk so they can be recovered following a hardware or software failure.

2.5.1 Configuration Checklist

WebSphere MQ Server

- On a heavily loaded Linux server, you may find that you need to increase the maximum number of open files from the default value. If the Linux distribution that you are using supports the proc filesystem you can do this by altering the value for /proc/sys/fs/file-max. There are also limits that apply to System V IPC resources that may need to be increased.
- To improve the rate at which new connections are accepted by the listener on the server, specify the environment variable MQNOREMPOOL=YES before starting the listener. This specifies that channel agents on the queue manager are run within threads within the runmqlsr process, and this scales well up to around 100 concurrently running channels. By default, channel agents on the server are distributed between a pool of amqrmppa channel processes to achieve maximum scalability. However, there is some trade-off against channel startup performance using this method.

- By default, listeners use process pooling which scales well given the appropriate hardware resources up to a maximum of approximately 10,000 concurrent channels.
- Specifying that server channel agents run as trusted applications can also significantly increase performance. This can be configured by setting the MQIBindType tuning parameter to FASTPATH in the Channels stanza of the queue manager ini file.
- By default, the queue manager allocates enough resources to start up to a maximum of 100 concurrent channel connections. If more than 100 channels attempt to run concurrently then the startup request is denied. This limit can be increased to allow more than 100 concurrent channels to run by altering the MaxChannels attribute in the Channels stanza of the queue manager ini file to a value greater than 100.
- Pipelining of channels can also be used to give a marginal improvement in performance. In pipelining, the channel agent utilizes two threads simultaneously to send and receive data from the network. Pipelining can be enabled by altering the PipeLineLength attribute in the Channels stanza of the queue manager ini file to the value of 2.
- Where a queue manager's logs are stored on a storage device (for example, an SSA write cache enabled device) that guarantees if a write operation writes a page and fails for any reason that a subsequent read of the same page returns either the original page or the byte that should have been written, configuring the LogWriteIntegrity tuning parameter to SingleWrite in the Log stanza of the queue manager ini file provides the highest level of performance for logging.
- Increasing the size of the LogBufferPages tuning parameter above the default of 128 (x 4Kb) pages can increase persistent performance for large messages. The maximum value that this can be increased to is 4096 pages.

WebSphere MQ Client

To set up a JMS point-to-point non-transacted message application:

1. Create a QueueConnectionFactory, or a more generic ConnectionFactory object with valid connection attributes. Then create a Queue object with the Persistence attribute set to Non Persistence. These objects will be used within your application to connect to your queue manager and queue.
2. Define a QueueConnectionFactory Object from within JMSAdmin, with the required attributes. (WMQ Explorer with Service levels from V6:0:2:0 can also be used to update this information). If you want to use the more generic Connection Factory object, substitute 'qcf' with 'cf' in the example below. (Text from within the quotes should be used in the examples below):
    - 'DEFINE QCF(myQCF) QMGR(QM1)' - defines a Queue Connection Factory in JNDI assigning its Queue Manager Attribute to 'QM1'
    - 'ALTER QCF(myQCF) TRANSPORT(CLIENT)' - if you want to run in Client Connection Mode
    - 'ALTER QCF(myQCF) HOSTNAME(MQServerMachineName)' - if you want to run in Client Connection Mode

- 'ALTER QCF(myQCF) PORT(1414)' - if you want to run in Client Connection Mode (assuming '1414' is the MQ Server listener port, you will have to manually start a listener (runmqlsr) on the desired port)
- 'ALTER QCF(myQCF) CHANNEL(SYSTEM.DEF.SVRCONN)' - if you want to run in Client Connection Mode (SYSTEM.DEF.SVRCONN should exist on the QM by default)
- 'DEFINE Q(myQ) QMGR(QM1) QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE) persistence(NON)' - defines a Queue in JNDI for MQ Queue 'SYSTEM.DEFAULT.LOCAL.QUEUE', using non persistent messages

3. Set 'Client Mode' options if your JMS Client is on another machine to your MQ Server
   - To define your session as non-transacted you will need to do this programatically, see the sample code in the appendix in Chapter 8.
     more specifically the line of interest is:
     - session = connection.createQueueSession(transacted, Session.AUTO_ACKNOWLEDGE);
       (where 'transacted' equals 'false')

2.5.2 Configuration Scripts

Chapter 2.1.2 discuss various ways of setting significant MQ Parameters.

WebSphere MQ Server

Windows:
(Previous scenarios used REGEDIT to amend the Windows registry but this example uses AMQMDAIN.)

```
set MQNOREMPOOL=Yes

amqmdain reg QMGR -c add -s Channels -v MQIBindType=FASTPATH
amqmdain reg QMGR -c add -s Channels -v MaxChannels = 200
amqmdain reg QMGR -c add -s Channels -v PipeLineLength=2
amqmdain reg QMGR -c display -s Channels -v *

amqmdain reg QMGR -c add -s Log -v LogWriteIntegrity=SingleWrite
amqmdain reg QMGR -c add -s Log -v LogBufferPages=512
amqmdain reg QMGR -c display -s Log -v *
```

Linux:

```
echo 32768 > /proc/sys/fs/file-max
echo 268435456 > /proc/sys/kernel/shmmax
export MQNOREMPOOL=Yes
#*******************************************************************#
#*                                                                 *#
#* Module Name: qm.ini                                             *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
```

```
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*****************************************************************#

ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfu
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=512
   LogPath=/var/mqm/log/QMGR/

Channels:
   MaxChannels=200
   PipeLineLength=2
   MQIBindType=FASTPATH
```

WebSphere MQ Client

- Use the sample Java code to execute a simple yet configurable point-to-point messaging scenario rererenced in the appendix in Chapter 8, this will need to be compiled against the WebSphere MQ Java jars. You will also require these jars in your classpath when running this code. Usage is contained within the sample.

2.5.3 Notable Features

2.5.3.1 Problem Determination

JMS Problem Determination

Common basic problem areas:

- Unable to connect to JNDI store from within JMS application (so cannot retreive JNDI objects).
  - Have you specified the correct url and initial context factory for your JNDI store? Check the JMSAdmin.config file, you should be using the same options as this file.
- Unable to connect to qmgr for some reason.
  If you are trying to connect in local/bindings mode:
  - Are you specifying the correct queue manager name on your Queue Connection Factory?

- Are you sure you are using 'TRANSPORT(BIND)' on your Queue Connection Factory? In JMSAdmin 'display qcf(myQCF)' should tell you.

If you are trying to connect in remote/client mode:

- Are you specifying the correct queue manager name on your Queue Connection Factory?
- Are you sure you are using 'TRANSPORT(CLIENT)' on your Queue Connection Factory? In JMSAdmin 'display qcf(myQCF)' should tell you.
- Have you specified the same listener port on your Queue Connection Factory, as the Listener you have running against your queue manager? In JMSAdmin type 'display qcf(myQCF)', then check the PORT attribute.
- Have you specified the correct hostname/ip address corresponding to the ip of the MQ Server Machine? In JMSAdmin type 'display qcf(myQCF)', then check the HOSTNAME attribute.
- Have you specified a valid Server Connection Channel that exists on your MQ Server Queue Manager? In JMSAdmin type 'display qcf(myQCF)', then check the CHANNEL attribute.

- A JMSException is thrown from your application when messaging.
  - Make sure your application logs what is does as it goes, so when an exception occurs you know what you were trying to do.
  - Make sure you catch and subsequently output all JMSExceptions, output the reason code nested in the exception and cross check this is the explanations for all JMS reason codes within the *WebSphere MQ Using Java* manual.
  - At worst case, take a JMS Trace, 'java -DMQJMS_TRACE_LEVEL=base MyJMSProg'. Your trace should be a file called, mqjms.trc in your current working directory.
- Messages are sent but none are received, no exceptions are thrown.
  - Make sure you have started your connection within your application ('connection.start()'). Until your connection is started message receiving is blocked.

Queue Manager Problem Determination

If a channel fails to start for any reason, the queue manager error logs can often provide an appropriate message that explains the cause and resolution to the failure.

2.5.4 Performance Characteristics

Refer to the WebSphere MQ for Windows V6.0 Performance Evaluations & WebSphere MQ for Linux V6.0 Performance Evaluations document for in-depth performance characteristics.
http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html

2.5.5 References

WebSphere MQ Intercommunications Guide

WebSphere MQ Clients

These publications can be found at

http://www.ibm.com/software/integration/mqfamily/library/manualsa

2.6 Point-to-Point - Non-Persistent, Non-Transacted

- Configuration Checklist
- WebSphere MQ Queue Manager
- Notable Features

Non-persistent point-to-point, non transactional messaging is generally used where messages need to be sent with a very high level of performance but with the possibility that a hardware or software failure may result in messages being lost.

This is used for high message throughput when senders and receivers have a one to one mapping, where message recovery and rollback of system state in an exceptional event is not required. This scenario uses non-persistent messaging, so no MQ logging is done for the messages. In addition, the JMS sessions are non transacted, so no work can be done within sync-point. The use of non-persistent messages along with non-transacted sessions works well together to maximise the throughput of your messages. Using only one of the two would negate some of the benefits of the other. For example, you could potentially cause a bottleneck on committing messages if you use non-persistent messaging.

2.6.1 Configuration Checklist

WebSphere MQ Server

To set up a queue manager for point-to-point non-persistent, non-transacted throughput:

- On a heavily loaded Linux server, you may find that you need to increase the maximum number of open files from the default value. If the Linux distribution that you are using supports the proc filesystem you can do this by altering the value for /proc/sys/fs/file-max. There are also limits that apply to System V IPC resources that may need to be increased.
- To improve the rate at which new connections are accepted by the listener on the server, specify the environment variable MQNOREMPOOL=YES before starting the listener. This specifies that channel agents on the queue manager are run within threads within the runmqlsr process, and this scales well up to around 100 concurrently running channels. By default channel agents on the server are distributed between a pool of amqrmppa channel processes to achieve maximum scalability. However there is some trade-off against channel startup performance using this method.
- By default, listeners use process pooling which scales well given the appropriate hardware resources up to a maximum of approximately 10,000 concurrent channels.
- Specifiying that server channel agents run as trusted applications can also significantly increase performance. This can be configured by setting the MQIBindType tuning parameter to FASTPATH in the Channels stanza of the queue manager ini file.

- By default, the queue manager allocates enough resources to start up to a maximum of 100 concurrent channel connections. If more than 100 channels attempt to run concurrently then the startup request is denied. This limit can be increased to allow more than 100 concurrent channels to run by altering the MaxChannels attribute in the Channels stanza of the queue manager ini file to a value greater than 100.

- Pipelining of channels can also be used to give a marginal improvement in performance, in pipelining the channel agent utilizes two threads simultaneously to send and receive data from the network. Pipelining can be enabled by altering the PipeLineLength attribute in the Channels stanza of the queue manager ini file to the value of 2.

WebSphere MQ Client

To set up a JMS point-to-point non-transacted message application:

1. Create a QueueConnectionFactory, or a more generic ConnectionFactory object with valid connection attributes. Then create a Queue object with the Persistence attribute set to Non Persistence. These object wil be used within your Application to connect to your Queue Manager and Queue.

2. Define a QueueConnectionFactory Object from within JMSAdmin, with the required attributes. (WMQ Explorer with Service levels from V6:0:2:0 can also be used to update this information). If you want to use the more generic Connection Factory object, substitute 'qcf' with 'cf' in the example below. (Text from within the quotes should be used in the examples below):

    - 'DEFINE QCF(myQCF) QMGR(QM1)' - defines a Queue Connection Factory in JNDI assigning its Queue Manager Attribute to 'QM1'

    - 'ALTER QCF(myQCF) TRANSPORT(CLIENT)' - if you want to run in Client Connection Mode

    - 'ALTER QCF(myQCF) HOSTNAME(MQServerMachineName)' - if you want to run in Client Connection Mode

    - 'ALTER QCF(myQCF) PORT(1414)' - if you want to run in Client Connection Mode (assuming '1414' is the MQ Server listener port, you will have to manually start a listener (runmqlsr) on the desired port)

    - 'ALTER QCF(myQCF) CHANNEL(SYSTEM.DEF.SVRCONN)' - if you want to run in Client Connection Mode (SYSTEM.DEF.SVRCONN should exist on the QM by default)

    - 'DEFINE Q(myQ) QMGR(QM1) QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE) persistence(NON)' - defines a Queue in JNDI for MQ Queue 'SYSTEM.DEFAULT.LOCAL.QUEUE', using non persistent messages

3. Set 'Client Mode' options if your JMS Client is on another machine to your MQ Server.

    - To define your session as non-transacted you will need to do this programatically, see the sample code in the appendix in Chapter 8,
      more specifically the line of interest is:

- session = connection.createQueueSession(transacted, Session.AUTO_ACKNOWLEDGE);
  (where 'transacted' equals 'false')

2.6.2 Configuration Scripts

Chapter 2.1.2 discuss various ways of setting significant MQ Parameters.

WebSphere MQ Server

Windows:

(Previous scenarios used REGEDIT to amend the Windows registry but this example uses AMQMDAIN.)

```
set MQNOREMPOOL=Yes

amqmdain reg QMGR -c add -s Channels -v MQIBindType=FASTPATH
amqmdain reg QMGR -c add -s Channels -v MaxChannels = 200
amqmdain reg QMGR -c add -s Channels -v PipeLineLength=2
amqmdain reg QMGR -c display -s Channels -v *
```

Linux:

```
echo 32768 > /proc/sys/fs/file-max
echo 268435456 > /proc/sys/kernel/shmmax
export MQNOREMPOOL=Yes
#*******************************************************************#
#*                                                                 *#
#* Module Name: qm.ini                                             *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*******************************************************************#

ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfu
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=0
   LogPath=/var/mqm/log/QMGR/

Channels:
```

```
MaxChannels=200
PipeLineLength=2
MQIBindType=FASTPATH
```

WebSphere MQ Client

- Use the sample Java code referenced in the appendix in Chapter 8 to execute a simple yet configurable PTP Messaging scenario, this will need to be compiled against the WebSphere MQ Java jars. You will also require these jars in your classpath when running the code. Usage is contained within the sample.

2.6.3 Notable Features

2.6.3.1 Problem Determination – see Chapter 2.5.3.1

2.6.4 Performance Characteristics

Refer to the *WebSphere MQ for Windows V6.0 Performance Evaluations & WebSphere MQ for Linux V6.0 Performance Evaluations* document for in-depth performance characteristics: http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html

2.6.5 References

WebSphere MQ Intercommunications Guide
WebSphere MQ Clients

These publications can be found at http://www-1.ibm.com/support/docview.wss?uid=swg27007065

3 Tuning for JMS Applications

- Tuning the Queue manager
- Tuning minimum heap size for Java

3.1 Tuning the Queue manager

Performance reports with tuning information for WebSphere MQ v6.0 on each platform can be found on the IBM SupportPac webpage at the following URL:
http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html

The main tuning actions taken for the scenarios in this report were:

- Use of multiple physical disk arrays (with IBM SSA battery-backed cache or SAN)
- Log / LogBufferPages = 4096
- Log / LogFilePages = 16348
- Log / LogPrimaryFiles = 16
- Channels / MQIBindType = FASTPATH
- TuningParameters / DefaultQBufferSize = 1MB
- TuningParameters / DefaultPQBufferSize = 1MB

3.2 Tuning minimum heap size for Java

During operation, current garbage collectors (GC) will normally interrupt the execution of all other threads in a JVM to some extent. The level of interruption depends on the amount and the type of work the GC is doing. This is largely dependant on how the memory is being used by the application and the GC settings currently in operation.

Messaging in Java has characteristics such that fixed memory requirements are low and transient memory requirements are high. Without tuning, or with incorrect tuning, the automatic garbage collection policies of Java do not favour messaging.

The most common GC settings are:

- –Xms Minimum heap size.
- –Xmx Maximum heap size.
- –verbose:gc Display garbage collection events.

As an example, the following line sets the heap limits to 64MB and 256MB and enables verbose garbage collection.

```
java –Xms64M –Xmx256M –verbose:gc
```

Recommendations

- Use –verbose:gc to monitor the frequency of your application's garbage collection under different loads and adjust the minimum and maximum heap sizes accordingly.
- A garbage collection interval of less than one second is detrimental to performance. A sensible minimum GC interval is 1-2 seconds.
- Do not leave the minimum heap size unset. If left unset, the heap will not be expanded. With a small (the default) minimum heap size, the GC will operate very frequently, reclaiming transient memory but not extending the heap (since we have already stated that most of the memory is released immediately after it is requested when messaging).
- It is a common mistake to fix both minimum and maximum heap settings to a single large value. When the minimum heap size is set too high, the GC can operate very infrequently, have more work to do and can create a noticeable pause in response times. If you cannot avoid this situation, consider having multiple independent JMS applications serving the same destinations. When one application is stalled on garbage collection another might still be serving.

4 WebSphere MQ Queue Manager Tuning

- Queue Disk, Log Disk, and Message Persistence
- Nonpersistent and Persistent Queue Buffer
- Log Buffer Size, Log File Size, and Number of Log Extents
- Channels: Process or Thread, Standard or Fastpath?
- Applications: Design and Configuration
- Standard (Shared or Isolated) or Fastpath?
- Parallelism, Batching, and Triggering
- Storage

This section highlights the tuning activities that are known to give performance benefits for WebSphere MQ V6.0; some of these can be applied to Version 5.3. Note that the following tuning recommendations may not necessarily need to be applied, especially if the message throughput and/or response time of the queue manager system already meets the required level. Some tuning recommendations that follow may degrade the performance of a previously balanced system if applied inappropriately. You should carefully monitor the results of tuning the queue manager to be satisfied that there have been no adverse effects.

Customers should test that any changes have not used excessive real resources in their environment and make only essential changes. For example, allocating several megabytes for multiple queues reduces the amount of shared and virtual memory available for other subsystems, as well as over committing real storage.

Note: The 'TuningParameters' stanza is not a documented external interface and may change or be removed in future releases.

4.1.1 Queue Disk, Log Disk, and Message Persistence

To avoid potential queue and log I/O contention due to the queue manager simultaneously updating a queue file and log extent on the same disk, it is important that queues and logs are located on separate and dedicated physical devices. With the queue and log disks configured in this manner, careful consideration must still be given to message persistence: persistent messages should only be used if the message needs to survive a queue manager restart (forced by the administrator or as the result of a power failure, communications failure, or hardware failure). In guaranteeing the recoverability of persistent messages, the pathlength through the queue manager is three times longer than for a nonpersistent message. This overhead does not include the additional time for the message to be written to the log, although this can be minimised by using cached disks.

4.1.2 Nonpersistent and Persistent Queue Buffer

Each queue in the queue manager is assigned two buffers to hold messages, one for non persistent messages and one for persistent messages. After messages spill out of the buffer they will move to the operating system file system. It is possible to change the buffer sizes to limit this overspill so

that data is more readily available to the queue manager.
The buffer for the non persistent messages has a default size of 64K for the 32 bit queue managers (Windows, Linux32) and 128K for 64 bit Queue Managers (AIX, Solaris, HPUX, Linux64). The buffer for persistent messages has a default size of 128K for 32 bit Queue Managers and 256K for 64 bit Queue Managers. The maximum size supported for both queue buffers is 100MB.
(For more details see SupportPac MP01: MQSeries – Tuning Queue Limits.) Increasing the queue buffer provides the capability to absorb peaks in message throughput at the expense of real storage. Once these queue buffers are full, the additional message data is given to the file system that will eventually find its way to the disk. Defining queues using large nonpersistent or persistent queue buffers can degrade performance if the system is short of real memory either because a large number of queues have already been defined with large buffers, or for other reasons - for example, a large number of channels being defined.

Note: The queue buffers are allocated in shared storage so consideration must be given to whether the agent process or application process has the memory addressability for all the required shared memory segments.

Queues can be defined with different values of DefaultQBufferSize and DefaultPQBufferSize. The value for DefaultQBufferSize is taken from the TuningParameters stanza in use by the queue manager when the queue was defined. When the queue manager is restarted existing queues will keep their earlier definitions and new queues will be created with the desired parameters. When a queue is opened, resources are allocated according to the definition held on disk from when the queue was created. The value for DefaultQBufferSize is taken from the TuningParameters stanza currently in use by the queue manager

4.1.3 Log Buffer Size, Log File Size, and Number of Log Extents

The log buffer is a piece of main memory where the log records are appended so that multiple log records can be written to disks together. The default size of the log buffer is 128 pages with a maximum size of 4096 pages. To improve persistent message throughput the LogBufferPages should be increased to 512 x 4K pages = 2MB, or larger. LogFilePages (that is, crtmqm –lf <LogFilePages>) defines the size of one physical disk extent and should be configured to a large size, for example: 16384 x 4K pages = 64MB, with the maximum size being 65535 pages. The number of LogPrimaryFiles (that is, crtmqm -lp <LogPrimaryFiles>) should be configured to a large number and the maximum number of Primary plus Secondary extents is 255(Windows) and 511(UNIX). The cumulative effect of this tuning will:

- Improve the throughput of persistent messages (permitting a possible 2MB of log records to be written from the log buffer to the log disk in a single write).
- Reduce the frequency of log switching (permitting a greater amount of log data to be written into one extent).
- Allow more time to prepare new linear logs or recycle old circular logs (especially important for long-running units of work).

Changes to the queue manager LogBufferPages stanza take effect at the next queue manager restart. The number of pages can be changed for all subsequent queue managers by changing the LogBufferPages parameter in the product default Log stanza.

It is unlikely that poor persistent message throughput will be attributed to a 2MB queue manager log but processing of large messages will be helped by these enhanced limits. It is possible to fill and empty the log buffer several times each second and reach a CPU limit writing data into the log buffer, before a log disk bandwidth limit is reached.

4.1.4 Channels: Process or Thread, Standard or Fastpath?

Threaded channels are used for all the measurements in this report ('runmqlsr', and for server channels an MCATYPE of 'THREAD') the threaded listener 'runmqlsr' can now be used in all scenarios with client and server channels. Additional resource savings are available using the 'runmqlsr' listener rather than 'inetd', including a reduced requirement on: virtual memory, number of processes, file handles, and System V IPC.

Fastpath channels, and/or fastpath applications can increase throughput for both nonpersistent and persistent messaging (see Standard (Shared or Isolated) or Fastpath? for further information). For persistent messages, the improvement is only for the path through the queue manager, and does not affect performance writing to the log disk.

Note: Since the greater proportion of time for persistent messages is in the queue manager writing to the log disk, the performance improvement for fastpath channels is less apparent with persistent messages than with nonpersistent messages.

4.2 Applications: Design and Configuration

4.2.1 Standard (Shared or Isolated) or Fastpath?

You should be aware of the issues associated with writing and using fastpath applications, as described in the *MQSeries Application Programming Guide*. Although it is recommended that customers use fastpath channels, it is not recommended to use fastpath applications. If the performance gain offered by running fastpath is not achievable by other means, it is essential that applications are rigorously tested running fastpath, and never forcibly terminated (that is, the application should always disconnect from the queue manager). Fastpath channels are documented in the *MQSeries Intercommunication Guide*.

4.2.2 Parallelism, Batching, and Triggering

An application should be designed wherever possible to have the capability to run multiple instances or multiple threads of execution. Although the capacity of a multi-processor (SMP) system can be fully utilised with a small number of applications using nonpersistent messages, more applications are typically required if the workload is mainly using persistent messages. Processing messages inside syncpoint can help reduce the amount of time the queue managers takes to write a group of persistent messages to the log disk. The performance profile of a workload will also be subject to variability through cycles of low and heavy message volumes, therefore a degree of experimentation will be required to determine an optimum configuration.

Queue avoidance is a feature of the queue manager that allows messages to be passed directly from an 'MQPUTer' to an 'MQGETer' without the message being placed on a queue. This feature only applies for processing messages outside of syncpoint. In addition to improving the performance of a workload with multiple parallel applications, the design should attempt to ensure that an application or application thread is always available to process messages on a queue (that is, an 'MQGETer'), then messages outside of syncpoint do not need to ever be physically placed on a queue.

Note that as more applications are processing messages on a single queue there is an increasing likelihood that queue avoidance will not be maintainable. The reasons for this have a cumulative and exponential effect, for example, when messages are being placed on a queue quicker than they can be removed. The first effect is that messages begin to fill the queue buffer—and MQGETers need to retrieve messages from the buffer rather than being received directly from an MQPUTer. A

secondary effect is that as messages are spilled from the buffer to the queue disk, the MQGETers must wait for the queue manager to retrieve the message from the queue disk rather than being retrieved from the queue buffer. While these problems can be addressed by configuring for more MQGETers (that is, processing threads in the server application), or using a larger queue buffer, it may not be possible to avoid a performance degradation.

Processing persistent messages inside syncpoint (that is, in batches) can be more efficient than outside of syncpoint. As the number of messages in the batch increases, the average processing cost of each message decreases. For persistent messages the queue manager can write the entire batch of messages to the log disk in one go while outside of syncpoint control, the queue manager must wait for each message to be written to the log before returning control to the application.

Only one log record per queue can be written to the disk per log I/O when processing messages outside of syncpoint. This is not a bottleneck when there are a lot of different queues being processed. When there are a small number of queues being processed by a large number of parallel application threads, it is a bottleneck. By changing all the messages to be processed inside syncpoint, the bottleneck is removed because multiple log records per queue can share the same log I/O for messages processed within syncpoint.

A typical triggered application follows the performance profile of a short session (refer to in the next release of this document. The ‘runmqlsr’ has a much smaller overhead of connecting to and disconnecting from the queue manager because it does not have to create a new process. The programmatical implementation of triggering is still worth consideration with regard to programming a disconnect interval as an input parameter to the application program. This can provide the flexibility to make tuning adjustments in a production environment, if for instance, it is more efficient to remain connected to the queue manager between periods of message processing, or disconnect to free queue manager and operating system resources.

4.3 Virtual Storage, Real memory, and Paging

VMSTAT reports on 'in use' and 'free' memory as seen by the operating page manager. In AIX, the 'avm' and 'fre' fields report in terms of 4K pages. Starting a Queue manager generated using default values increases the AVM and reduces the FRE by 32M bytes. Each client channel uses 264K - 410K bytes for processing 2K byte messages depending on traffic rate, type of 'Reply-Queue', and MQIBINDTYPE which also assumes an appropriate size is specified on the BUFFERLENGTH parameter of the MQGet. (Chapter 6 of the MQ V6 Performance reports provides an estimate of the storage needed when clients either share a predefined queue with other clients or have a dynamic queue per client). 100K byte messages will use up to 700K bytes per client. The power of a machine used to process a workload needs to handle the peaks of troughs. The peak hourly rate cannot be divided by 3600 because the peak rate per second will be 2-3 times higher. It is important to prevent the queue depths increasing because they will occupy memory from the 'fre' pool or be spilled out to disk. Over commitment of real memory is handled by the page manager but sudden large jumps possibly due to queues becoming deep can cause the throughput to break down completely if the page disk is used. Gradual over commitment enables the page manager to shuffle out those pages that are not part of the working set.

5 General Tuning/WebSphere MQ Message Broker

- WebSphere MQ
- TCP/IP
- Database
- Miscellaneous
- Additional Tuning Information

Source: IP74:WebSphere Message Broker V6.0 Peformance Report for Windows

---

Message Broker

The Message Broker used in the measurements was configured in the following ways for all tests:

1. The broker ran as a trusted WebSphere MQ application. This was achieved by use of the '-t' flag on broker creation (with the mqsicreatebroker command) and by ensuring that the environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker was started. Note: There is a potential integrity exposure to the Message Broker queue manager as the level of isolation between the Message Broker and queue is reduced. This is where the improved performance comes from.

2. Transactional support was used where appropriate: when processing persistent messages it was used, with non-persistent messages it was not used. The use of transaction control means that message processing takes place within a WebSphere MQ unit of work. This involves additional CPU and I/O processing by WebSphere MQ because the unit of work is recoverable. The result is inevitably a reduction in message throughput for persistent messages. By default the transaction parameter on the MQInput node was set to automatic. This is the recommended value to use for transaction mode unless there is a specific requirement to use a particular value since persistent messages will be processed within transactional control and non-persistent messages will not.

Additional tuning was performed for the publish/subscribe tests. This was as follows:

1. Heap size - The heap size of the WebSphere Message Broker Java Virtual Machine (JVM) (in which much of the publish/subscribe code is executed) was set to 512MB. For the non publish/subscribe tests the default value of 128MB was used.

2. Thread settings - The thread settings of the RealtimeOptimizedNode used a default value of 10 read and write threads. These were sufficient to cater for the test cases run in this report. However, if more clients are used increasing these values could be beneficial.

3. Client pinging - The ping protocol implements a "keep alive" protocol where the broker is periodically verifying that connected clients are alive. This process allows the broker to detect disconnected clients and maintain an updated subscription list. In situations where there are a large number of clients connected to a broker, this pinging process may account for a large proportion of the messages exchanged between the broker and clients and can impact the broker's message throughput. In such circumstances, the ping interval can be turned off or alternatively increased to reduce the amount of traffic generated by pinging. For the tests in the report the value was set to 0.

4. Client queue size - The broker employs a set of internal queues which are used to regulate the delivery of messages to subscribers. Note: these are not the queues used by the MQ Transport. The size of the queue specifies the number of bytes of data that the broker will store for one client. If this maximum is exceeded, the broker will take action which is determined by the value of the parameter "Client disconnection due to queue overflow". The default queue size is 100,000 bytes.
   Setting the value to zero allows the broker to grow the queue size as required. In this case, the queue size will only be limited by the available system memory. In the tests detailed in this report a value of 0 was used.

5. Client disconnection due to queue overflow - When the depth of the client queue exceeds the Client Queue Size value, the broker can choose between two courses of action. The default action is to disconnect the client; in this case, all the queued messages are lost. This is specified through a value of true for the parameter. The alternative course of action, specified with a value of false, is to keep the client connection alive but remove any excess messages from the client's queue. In the tests detailed in this report a value of false was used.

6. Maximum message size - If the broker receives a message that is bigger than the maximum message size value, it will disconnect the client that sent the message.
   This feature is useful for protecting the broker from applications sending excessively large messages. The default maximum message size is 100,000 bytes and this was adequate for the tests run in the report so the value was left unchanged.

7. Maximum number of client connections - The broker has the ability to limit the number of client connections that it will handle. This is useful in situations where the number of client applications that will connect to the broker is unknown. In such cases limiting the number of connections will allow the broker to maintain a particular level of service (this level will depend upon the particular environment in which the broker is being used). The default setting is unlimited. This was also the value used for the tests run in the report.

8. Interval statistics reporting - This was enabled and set to an interval of 10000 (10 seconds) so that the value of ClientBytesQueued could be monitored.

Additional tuning was performed for the JMS nodes. These was as follows: The performance of the JMS Nodes is dependant on the performance of the JMS Provider and the JMS Providers client code. You should check the JMS Providers documentation for tuning details, in particular look for details of how to tune the client code which is supplied.

In the tests run for this report the max buffer size for the TopicConnectionFactory used by the JMS Input Node was increased to a value of 3000. This is important for a subscriber using WebSphere MQ Real-time as it offers protection from message rate spikes. For information on how to set the subscriber max buffer size see the WebSphere MQ "Using Java" manual.

There were no error processing or error conditions in any of the measurements. All messages were successfully passed from one node to another through the out or true terminal. No messages were passed through the failure terminal of a node.

5.1 WebSphere MQ

The Queue Manager was tuned as shown in Chapter 3.1

The environment variable MQ_CONNECT_TYPE=FASTPATH was present in the environment in which the broker queue manager was started.

5.2 TCP/IP

No specific tuning was performed for TCP/IP. All machines used the operating system default values.

5.3 Database

The DB2 instance used with the message broker was a default configuration and the only tuning performed on the instance was placement of the database data and log files on different disks.

5.4 Miscellaneous

Although not implemented in all cases the following additional tuning changes are recommended

- Locate the log of any WebSphere MQ queue manager through which persistent messages pass on a dedicated disk.
- Locate the WebSphere MQ queue manager log on a very fast disk such as one with a non-volatile fast write cache. Such disks are consistently capable of I/O times of 1ms compared with a time of 6 ms for a 10,000 RPM SCSI disk. When using a disk with a fast write cache it is essential that it has a non-volatile capability as the log data is critical to the integrity of your queue manager.
- Note that there is no need to locate the WebSphere queue manager queue file on a fast disk. It is advisable to locate it on a dedicated disk in order to improve the efficiency of queue manager checkpoint processing.
- Locate the log of the Message Broker database on a dedicated disk.
- Locate the log of the Message Broker database on a very fast disk such as one with a non-volatile fast write cache.
- When performing BLOB inserts to a database locate the data portion of the database on a very fast disk such as one with a non-volatile fast write cache. BLOB I/O is not buffered by a database such as DB2 and is written to disk immediately.
- When using the aggregation node follow the message flow coding advice provided in Supportpac IP05, WebSphere MQ Integrator V2.1 - Optimizing Use of Aggregation Nodes. This is available at

Note: When using WebSphere Message Broker V6 there is no need to follow the recommendations in the document about Message Broker database configuration. This is because the aggregation mechanism is now based on the use of WebSphere MQ queues, rather than a database table as with previous versions of the Message Broker.

5.5 Additional Tuning Information

In order to obtain the maximum message rate for your implementation it is important that you understand the current best practices for WebSphere Message Broker. These practices cover the architecture of message flow processing, the coding of message flows as well as the configuration and tuning of the message broker and associated components. Such information can be found in the Business Integration Zone of WebSphere Developer Domain. A suggested starting place is the article
http://www.ibm.com/developerworks/websphere/library/techarticles/0403_dunn/0403_dunn.html
which highlights the information available and where it may be found.

6 On Line Resources

- WebSphere MQ 6.0 Books
  - www-306.ibm.com > Software > Integration > Wmq > Library > Index

http://www-306.ibm.com/software/integration/wmq/library/index.html

- WebSphere MQ 5.3 Books
  - www-306.ibm.com > Software > Integration > Wmq > Library > Library53

http://www-306.ibm.com/software/integration/wmq/library/library53.html

- Redbooks

http://www.redbooks.ibm.com/

  - WebSphere MQ V6 Fundamentals

http://www.redbooks.ibm.com/abstracts/sg247128.html

Configuring and tuning WebSphere MQ for performance on Unix and Windows
http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html

7 Appendix – JMS Examples

Some sample applications, such as the Installation Verification Programs (IVPs) are supplied with WebSphere MQ. The location of the JMS examples are shown in the table below.

Platform	**Directory**
AIX	/usr/mqm/samp/java/jms
HP-UX, Linux, and Solaris	/opt/mqm/samp/java/jms
I5/OS	/QIBM/ProdData/mqm/java/samples/jms
Windows	*install_dir*\Tools\java\jms
z/OS	*install_dir*/mqm/V6R0M0/java/samples/jms
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

Example Filename	**Purpose**
JMSPubSub.java	Demonstrates Publish/Subscribe Function
PTPSample01.java	Demonstrates Point to Point • how to retrieve administered objects from JNDI namespace • how to create administered objects at runtime if no JNDI is available • how to create connections, sessions, senders and receivers • how to send and receive a message
PTPSample02.java	Demonstrates Point to Point • how to use a MessageListener for asynchronous message delivery • how to use message selectors to filter the input stream
PTPSample03.java	Demonstrates Point to Point • use of a QueueRequester.