Performance Bootcamp: WebSphere MQ & Message Broker

Morag Hughson (WebSphere MQ Development)
Dave Gorman (WebSphere MB Development)
IBM Hursley

3rd March 2011

Agenda

- MQ
  - Concepts
  - Application Programming
  - Channels
  - Clients
  - Distributed Specifics
  - z/OS Specifics
- MB
  - What are the main costs?
  - What other considerations are there?
  - Understanding your broker’s behaviour.
- Summary

Agenda

- The performance of a system depends on how the application uses real resources and how much real resource is allocated to the run time system. The application designers need to understand which virtual resources exposed to the programmer are heavyweight in their consumption of real system resources.

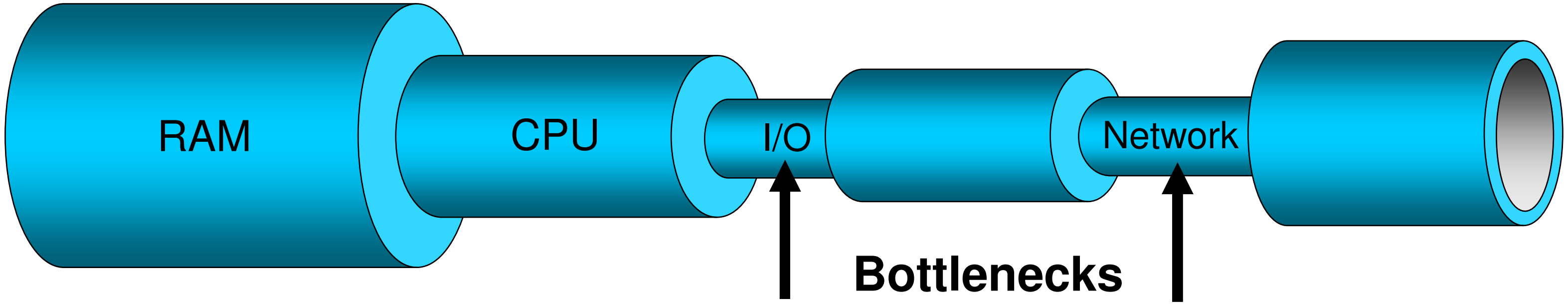
- The application determines which WMQ resources are used to solve the business problem. Some of these resources are heavyweight.

- WMQ product design objectives back in the early '90s were to deal with low volume, high value messages. These persistent messages must be logged to disk as necessary to ensure 'once only assured delivery'.

- Customers adopted WMQ and wanted to combine 'enquiry' messages alongside the valuable messages and were prepared to tradeoff total reliability for speed.

- Alternative ways of achieving the customer goals are examined using some lighter weight resources.

Performance Bottlenecks



- Business Systems are complex
  - Often no single bottleneck limiting performance
  - Can be tricky to tune due to limited effect
  - Performance can mean different things to different people
  - Throughput
    - Scalability
    - Low resource usage
- Not only limited by physical resources
  - Application design, such as parallelism can have major effect

Performance Bottlenecks

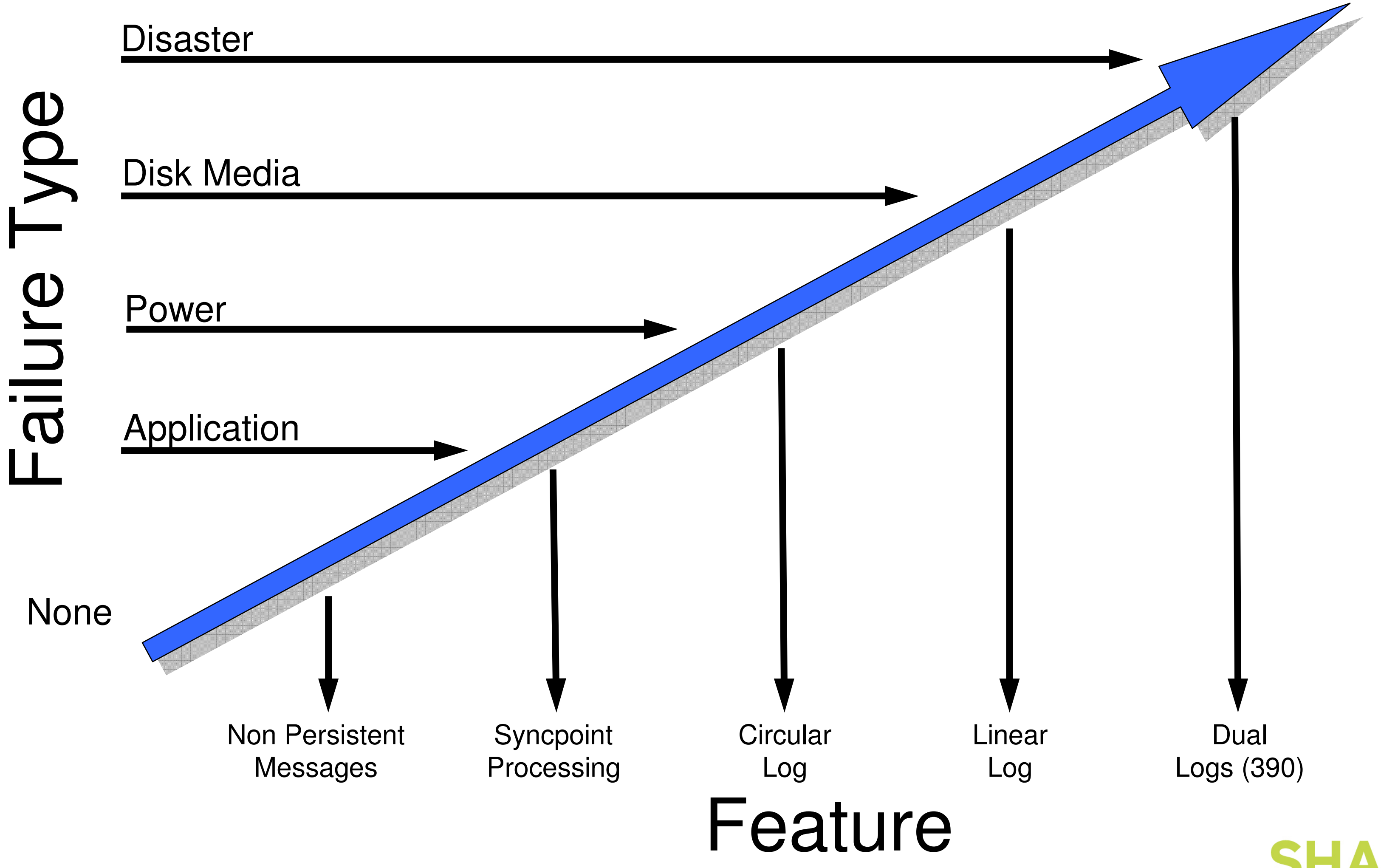
N O T E S

- Modern systems are complex and there are many factors which can influence the performance of a system. The hardware resources available to the application as well as the way that application is written all affect the behavior.

- Tuning these environments for maximum performance can be fairly tricky and requires fairly good knowledge of both the application and the underlying system. One of the key points to make is that the simple trial and error approach of changing a value and then measuring may not yield good results. For example, a user could just measure the throughput of messages on the previous foil. They could then double the speed of the disk and re-measure. They would see practically no increase of speed and could wrongly deduce that disk I/O is not a bottleneck.

- Of course throughput is not the only metric that people want to tune for. Sometimes it is more important that a system is scalable or that it uses as little network bandwidth as possible. Make sure you understand what your key goal is before tuning a system. Often increasing one metric will have a detrimental affect on the other.

Recovery Requirement



Recovery Requirement

N O T E S

- How much recovery does the application need from the queue manager. If the messages are carrying 'enquiry' questions and answers, then it is likely that speed is far more important than resilience, so the architects can make this tradeoff and use non persistent messages. Non Persistent messages are discarded in the event of a queue manager restart.

- The higher up this arrow the less likelihood of the occurrence of errors but the higher the cost of protection.

- Is it important that the database and/or message queues have 'atomic' changes in the event of application failure? If so, then using syncpoint coordination and possibly an XA Coordinator are needed.

- Is it important that power failure or software failure can recover messages? Circular logging will be sufficient (and required).

- Is it important that DISK media failure results in message recovery? Linear logging is necessary.

- Is it important that disaster recovery results in message recovery? Then consider systems - in particular z/OS - with remote site dual logging since distributed platforms depend on the operating systems 'mirrored' disks.

Persistence – the choice

Persistent messages are *much* more expensive to process than nonpersistent

So why do we use persistent messages?

Cost of losing the messages ✗

Cost of developing the application ✓

Persistence – the choice

NOTES

- Many people assume, incorrectly, that you must use persistent messages for "important" information -- nonpersistent is for when you don't care.

- The real reason for WebSphere MQ persistent message support is to *reduce application complexity*.

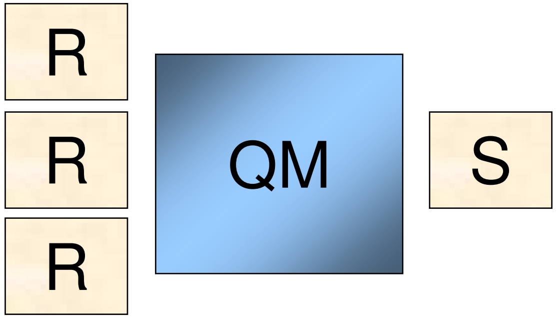
- If you use persistent messages your application does not need logic to detect and deal with lost messages.

- If your application (or operational procedures) *can* detect and deal with lost messages, then you do not need to use persistent messages.
- Consider:
  - A bank may have sophisticated cross checking in its applications and in its procedures to ensure no transactions are lost or repeated.
  - An airline might assume (reasonably) that a passenger who does not get a response to a reservation request within a few seconds will check what happened and if necessary repeat or cancel the reservation.

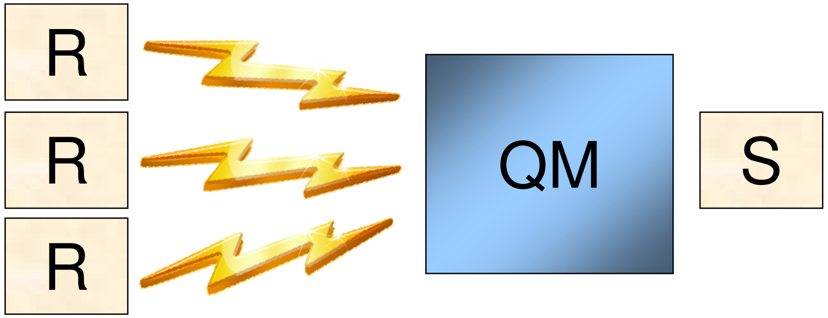
- In both cases the messages are important but the justification for persistence may be weak.

Application Design

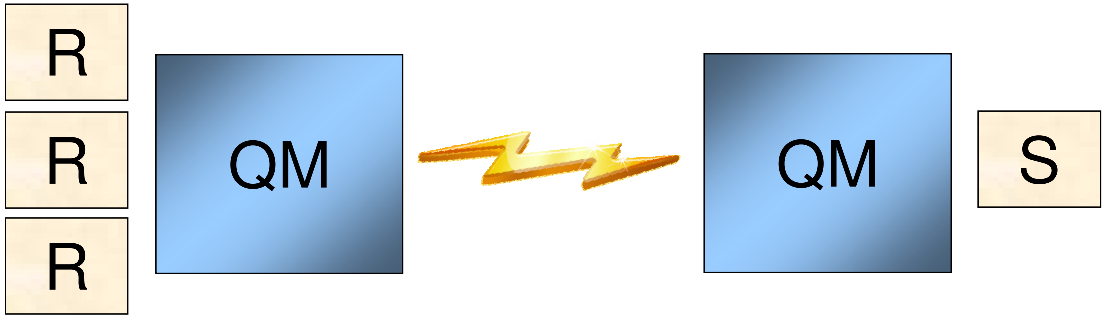
Network Configuration
Location Determines Cost



Local Messaging



Client Messaging



Inter System (DQ) Messaging

Key
R – Requesting Application
S – Server Application

Network Configuration
Location Determines Cost

N

O

T

E

S

- The network location of requester and responder applications will determine the resources necessary to accomplish the tasks.
- Requester Connection to local queue manager is the basic cost
- Responder in local or remote queue manager will determine whether the MQ Channels are used
- Requester Client Connection will involve an IP cost together with a channels in the receiving queue manager.
- Responder in same or remote queue manager will determine if further channels costs involved

Heavyweight MQI Calls

- MQCONN is a “heavy” operation
  - Don’t let your application do lots of them!
  - Think about OO programming and encapsulation
    - IssueRequest(…)
  - Lots of MQCONNs could cause throughput to go from 1000s Msgs/Sec to 10s Msgs/Sec
- MQOPEN is also ‘heavy’ compared to MQPUT/MQGET
  - Depends on the type of queue and whether first use
  - It’s where we do the security check
    - try to cache queue handles if more than one message
  - If you’re only putting one message consider using MQPUT1
    - Particularly client bindings
- Try to avoid exclusive access to the Queue
  - Makes it harder to scale the solution
    - For example adding more instances of application
  - Implies that reliance on message order is not required
    - Partition the data to allow parallel processing?

Connecting and disconnecting

N O T E S

- MQCONN is a very heavyweight operation. Doing lots of these calls could cause throughput to suffer. Make sure that you don't connect and disconnect a lot in your application, rather, connect once and then use this connection for all subsequent operations. Think carefully about any encapsulation you might do in your OO applications, make sure that the encapsulation does cause you to do lots of MQCONNs and MQDISCs.

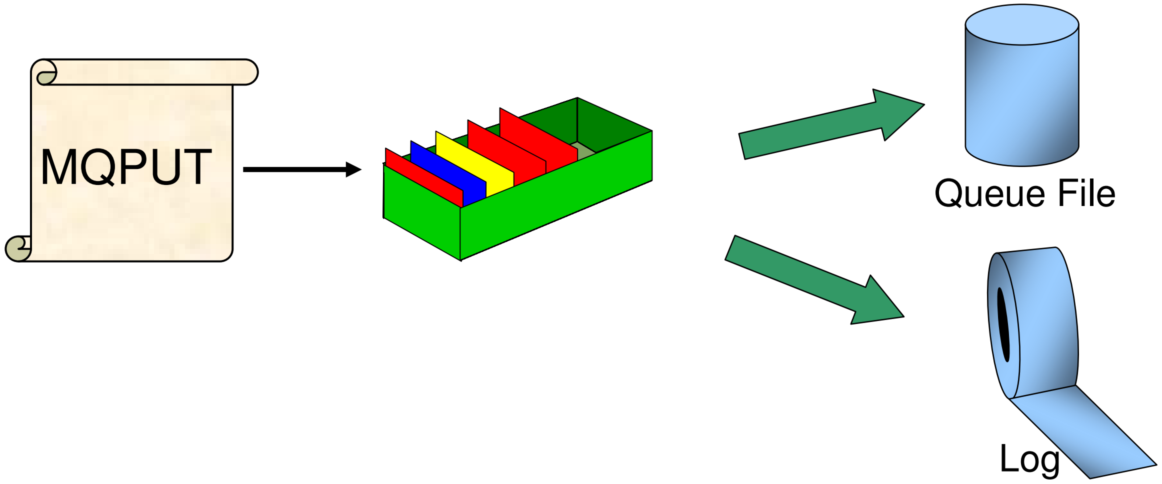
- MQPUT1
  If just putting a single message to a queue, MQPUT1 is going to be cheaper than MQOPEN, MQPUT and MQCLOSE. This is because it is only necessary to cross over from the application address space into the queue manager address space once, rather than the three times required for the separate calls. Under the covers inside the queue manager the MQPUT1 is implemented as an MQOPEN followed by MQPUT and finally the MQCLOSE. The cost saving for a single put to a queue is the switching from application to queue manager address space. Of course, if multiple messages need to be put to the queue then the queue should be opened first and them MQPUT used. It is a relatively expensive operation to open a queue.

- Exclusive use of queues
  Opening queues for exclusive use can help with sequencing issues, but it is a good idea to investigate whether other solutions are available. Exclusive use will make it harder to add extra tasks to process more work if needed in the future. Possible solutions are partitioning the data on the queue so that different tasks can work on different parts of the queue data (get by CorrelID can be used for this). This will enable more tasks to process the queue while maintaining ordering within the partitioned part of the data.

Persistence

- Log bandwidth is going to restrict throughput
  - Put the log files on the fastest disks you have

- Persistent messages are the main thing that requires recovery after a queue manager outage
  - Can significantly affect restart times



Persistence

N

O

T

E

S

- If persistent messages are used then the maximum rate that messages can be put is typically going to be limited by the logging bandwidth available. This is normally the over riding factor as to the throughput available when using persistent messages.

- As persistent messages need to be made available when a queue manager is restarted, they may need to be recovered if there has been a failure (could be queue manager or system etc). The persistent workload that has been done is the main key as to how long it is going to take to restart the queue manager after a failure. There are other factors involved which include the frequency of checkpoints etc, but ultimately it all comes down to the fact that persistent messages have been used. If there has been a failure then no recovery is required on non-persistent messages, the pages that contained them are simply marked as not used.

Syncpoint

- Do you need it?
  - Set of work needs to either all be performed, or all not performed
- Maximum Size of UOW can be limited
  - QMGR MAXUMSGS parm
  - Set to sensible value to avoid runaway applications
- Make sure you keep the size of your UOWs small
  - Don’t forget to end the UOW!
- Cheaper to process in syncpoint for persistent messages
  - Up to a point, not huge UOWs
  - Log not forced after every MQPUT/MQGET

Syncpoint

N

O

T

E

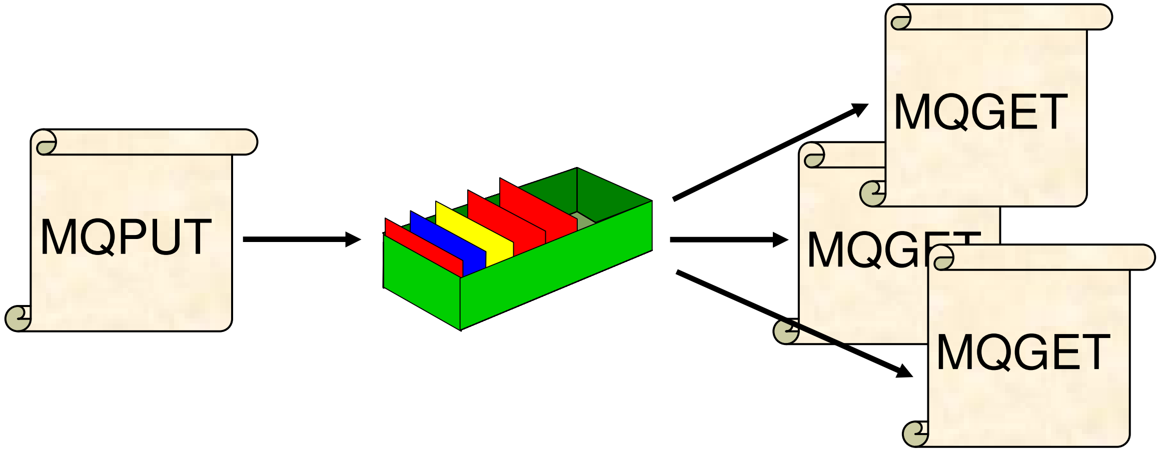
S

- The size of a UOW (number of messages in it) can be controlled via the MAXUMSGS queue manager parameter. This however has no impact on the duration of the UOW, it simply controls the maximum number of messages that a UOW can contain. It prevents runaway applications

- It can be considerably cheaper to process multiple persistent messages inside syncpoint rather than processing them outside syncpoint. This is because if persistent messages are being used outside of syncpoint, it is necessary to force them to the log as soon as they are put, to ensure that they are available if a failure occurs. If they are processed inside syncpoint it is only necessary to force the log when the UOW is committed. This means that we will spend less time waiting for the pages to be forced out to disk. In effect the cost of forcing the UOW out to disk is shared between all of the messages put and got, rather than each one having to bear the cost. Syncpoint should not be considered as an 'overhead'.

Put to a waiting getter

- MQPUT most efficient if there is getting application waiting
  - Having multiple applications processing queue increases percentage
- Only for out of syncpoint messages
  - and non-persistent on z/OS
- No queuing required
  - Removes a lot of processing of placing the message onto the queue
- Significantly reduces CPU cost and improves throughput
  - 40% CPU saving for 1K messages has been seen on z/OS
  - Also applies to shared queues if getter on the same queue manager
  - Particularly significant for large shared queue messages



Put to a waiting getter

N

O

T

E

S

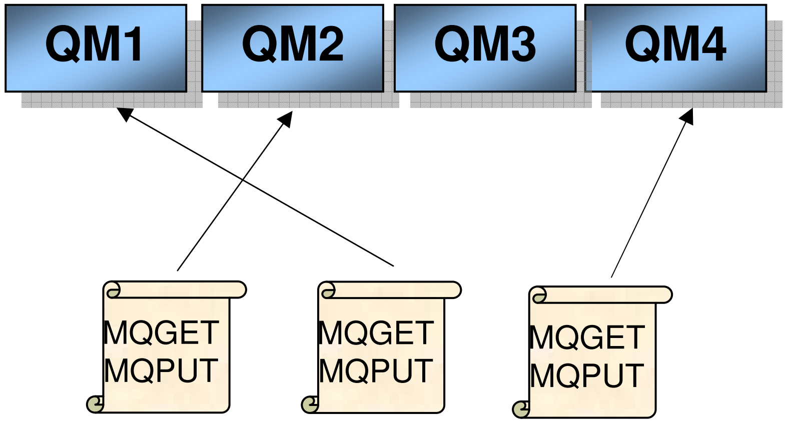
•"Put to a waiting getter" is a technique whereby a message may not actually be put onto a queue if there is an application already waiting to get the message. Certain conditions must be satisfied for this to occur, in particular the putter and getter must be processing the message outside syncpoint control (and on z/OS the message must also non-persistent). If these conditions are met then the message will be transferred from the putters buffer into the getters buffer without actually touching the MQ queue. This removes a lot of processing involved in putting the message on the queue and therefore leads to increased throughput and lower CPU costs.

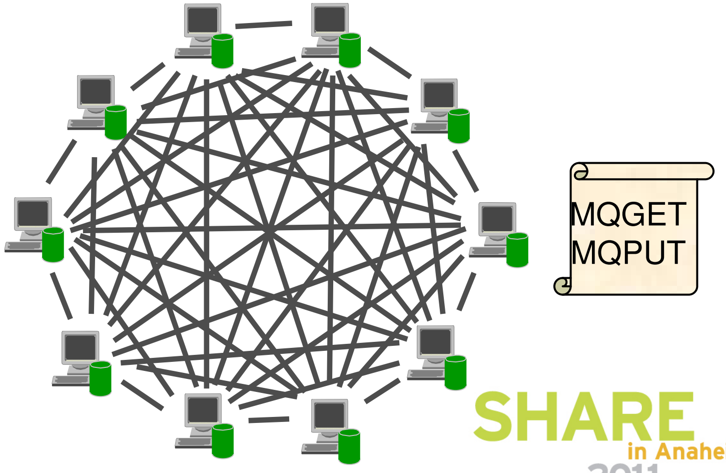
•When in “put to waiting getter” mode the Queue Manager will try to keep one thread ‘hot’.

- Distributed always tries to keep one thread ‘hot’

Workload balancing

- A single server might not be sufficient to support the required workload
- Need to look at method for balancing the workload within the network to increase overall processing capabilities
- Possibilities include:
  - CCDT or network sprayer for client connections
  - Clustering





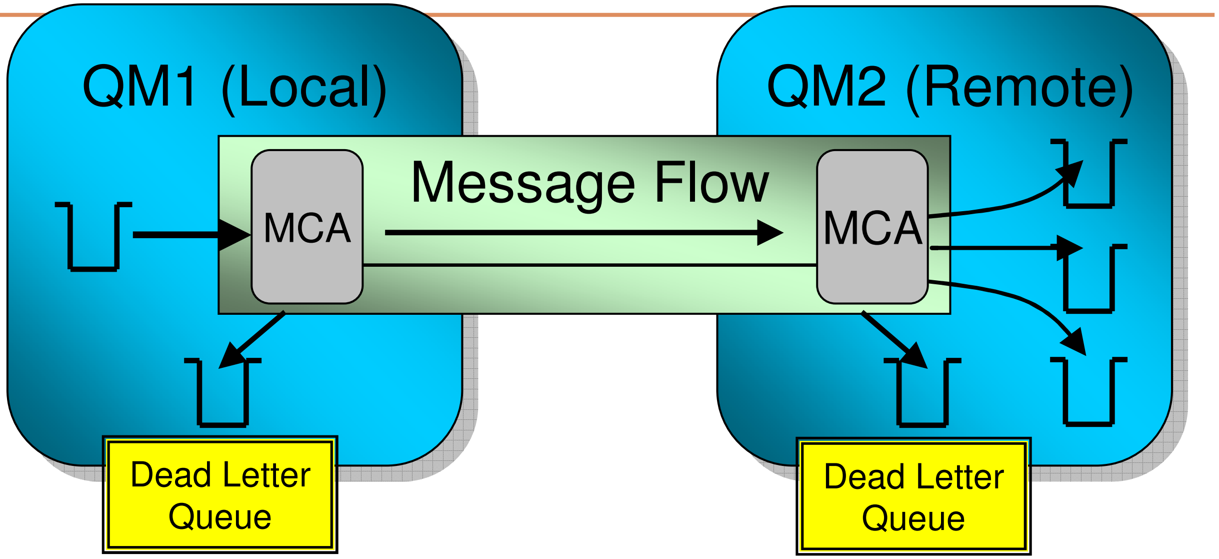
Workload balancing

N O T E S

- There may come a point in time when a single server isn't capable of meeting the workload demands. At this point it will be necessary to spread the workload across multiple servers.
- You may want to consider this eventuality upfront so that any design put in place can be modified to accommodate additional workload balancing that may need to take place in the future.
- The two main areas to consider are connecting into the MQ infrastructure and then the destination of messages once connected to MQ.
- If using server bindings then the application needs to connect to a queue manager on the same operating system image, however, for client bindings the application does not that the same restriction. In this case, workload balancing can be carried out using a variety of techniques. These include configuring the CCDT to have multiple queue managers that could be chose, or alternatively the distribution could be carried out using a network spraying technique to spread the connections amongst available queue managers.
- Once connect into the MQ infrastructure, clustering can be employed to spread the workload around available servers. This enables the system to be modified dynamically to provide more processing capability when the workload increases.

Channels

Channels



- Send messages from Transmit Q until
  - Batchsize or Empty & Batchint expired.
- Store 'indoubt' record and send 'End-of-Batch' to Remote MCA.
- Remote MCA updates Batch Sequence number, MQCMIT, sends acknowledge.
- Local MCA updates Batch Sequence number and issues MQCMIT.
- Pipeline Length =2 provides additional thread that will start processing next batch after 'End-of-Batch' sent to Remote MCA

Channels

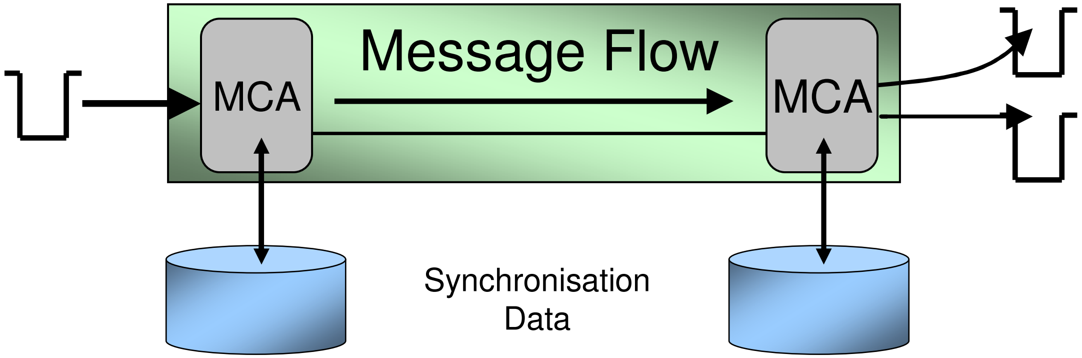
NOTES

- The channel operation conforms to a quite simple model:

```
Do until batchsize or (no more messages and batchint) expired
    Local MCA gets a message from the transmission queue
    A header is put on the data and sent using APPC, TCP etc.
End
Harden the message ids/indoubt flag
Send "End of batch flag"
Remote end commits
Remote end sends "OK" flag back
Local end updoubt synchronisation record to non-indoubt state and commits
```

- If there is any failure in the communications link or the MCA processes, then the protocol allows for re-synchronisation to take place and messages to be appropriately recovered.
- Probably the most misunderstood part of the message exchange protocol is Batchsize. Batchsize controls the frequency of commit flows used by the sending MCA. This, in turn, controls how often the communications line is turned around and - perhaps more importantly - how quickly messages at the receiving side are committed on the target application queues. The value for Batchsize that is negotiated at channel start-up is the maximum Batchsize only - if the transmission queue becomes empty then a batch of messages is automatically committed. Each batch containing Persistent messages uses the Scratchpad. The larger the effective batch size, the smaller is the resource cost per message on the channel. Batchint can increase the effective batch size and can reduce cost per message in the server.
- Pipelinelength=2 enable overlap of putting messages onto TCP while waiting for acknowledgment of previous batch. This enables overlap of sending messages while waiting for Batch synchronization at remote system.

NPMSpeed (FAST) Channels



Synchronisation Data

- No commit processing for non-persistent messages
  - No logging of synchronisation data
  - Reduced processing requirement for non-persistent messages
  - Faster throughput

- Persistent messages
  - Persistent messages may share the channel
  - Standard commit processing

- Fast Messages are optional
  - Alter Channel(X) NPMSPEED(FAST¦NORMAL)

NPMSpeed (FAST) Channels

N O T E S

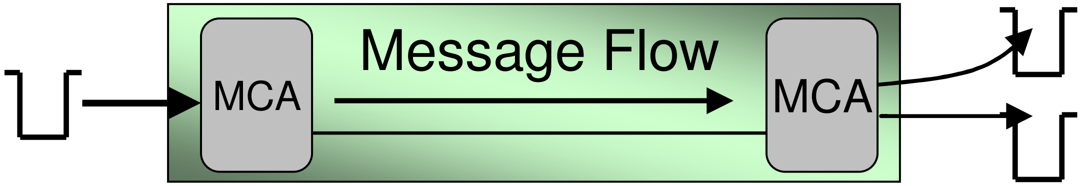
- Fast Non-Persistent Message speed channels make an additional tradeoff for speed over reliability for non persistent messages. Persistent messages are processed in just the same way as 'Normal' channels.

- When only non-persistent messages use these channels, there is no disk logging involved during batch synchronization. At end of batch (caused by empty XMIT Q or Batchsize), a round trip is made between the Sender and Receiver mover so that the channel returns to 'heartbeat' mode.

- For a NPMSpeed(FAST) channel If the channel fails (ie TCP failure) while a non-persistent message is in flight, the message is lost. The channel recovery will be unable to find the message because it was got 'outside' of syncpoint. The definition of a 'non-persistent' message is that it will not survive queue manager restart so we are using this attribute to provide faster message delivery.
- Since no attempt is made to re-deliver non-persistent messages if the channel fails NPMSpeed(FAST) should probably not be used if the network is unreliable.

- Non-persistent messages may overtake Persistent messages. Persistent messages are got and put within syncpoint. They are only visible at the receiver when the MQCMIT completes. Non-persistent messages are got and put outside of syncpoint. They are visible immediately they arrive at the receiver.

Channel Batch Size



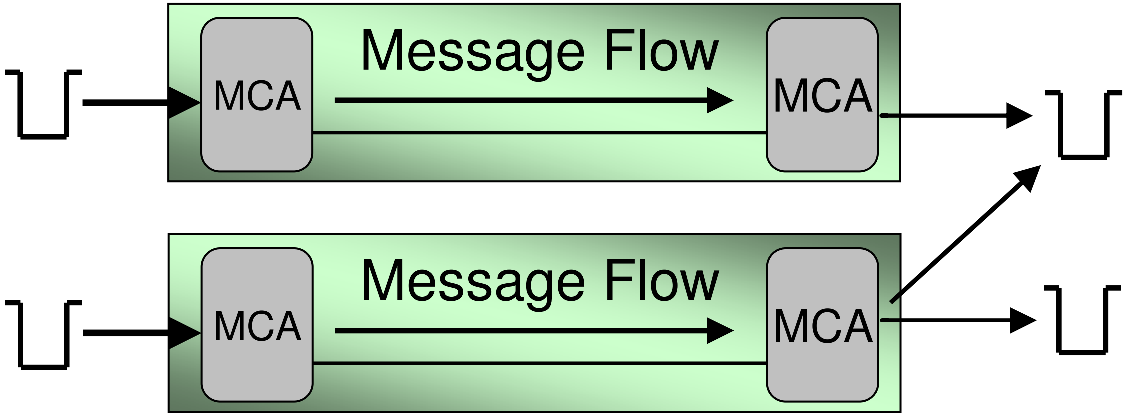
- Batchsize parameter determines maximum batch size
  - Actual batchsize depends on
    - Arrival rate
    - Processing speed
- Low Batchsize may cause XmitQ to build up
  - Scratchpad housekeeping uses equiv of 3 Persistent messages
  - High Batchsize will make little difference - self throttling
- Use high (~50) Batchsize unless there are issues with:
  - Data visibility - throughput
  - Communication link reliability
  - Message Size
- Batchint can increase effective batch size towards Batchsize
  - Delays message availability to application
  - Reduces CPU cost per message

Channel Batch Size

N O T E S

- Actual batchsize is #messages/ #batches on particular channel.
- Setting an appropriate Batchsize (BATCHSZ) is a difficult issue and is related to the following factors:
  - Applications write messages to XMIT queues for moving over channels to remote systems. Channels take batches of messages from XMIT queues and move to destination. The overhead per batch (commit, CPU and disk activity) is divided by the #messages in the batch to give the cost per msg If the commit process occurs less often then the message transfer rate is increased.
  - A batch is ended when one of two things happen - either the number of messages transferred has reached the maximum allowed for the batch or the transmission queue is empty and the Batchint has expired. If the message arrival rate is lower than the message transfer rate then the effective batch size is dynamically reduced as a drained transmission queue implies 'end of batch'. This reduction in batch size will reduce the message transfer rate as commit processing is more frequent and increases the cost of processing each message by the channel.
  - Messages arriving at the receiving MCA are placed on the target application queues under syncpoint control. This means that they are not visible to any receiving applications until the commit is performed. If the batch size is large, messages may not be made available to receiving applications for some time which may have a severe impact on the message throughput of the overall system.
- Because the batch size is so greatly influenced by the message arrival rate on the transmission queue, it is generally recommended to set the Batchsize quite high(ie leave at default of 50) - unless there are contrary factors, which are:
  - Data visibility - due to (outstanding) commit processing.
  - Unreliable, slow or costly communication links, making frequent commit processing a necessity.
  - Large Messages. Upon restart it may be necessary to resend the last batch.
- Entirely Non persistent message batches do not use disk for hardening batch information (NPMSpeed(FAST)) but still cause a line turnaround.

One or Multiple Channels



- Use one channel if it can handle the throughput
  - Monitor depth of XMIT queue
  - One high-use channel is more efficient than two low-use channels
    - The actual batch size will be larger
  - Multiple channels can yield some performance benefit
    - Depends on network and arrival rate
- Multiple channels for separate classes of work
  - Large messages only delay large message
  - Encryption cost on taken on worthwhile messages
  - Small interactive messages do not get delayed

One or Multiple Channels

N

O

T

E

S

- For the reasons previously outlined, it is most appropriate to have as high a channel utilization as possible. This means that it is appropriate to have as few channels as can handle the load between any two queue managers.

- However, where there are different classes of message being moved around the WMQ network, it may be appropriate to have multiple channels to handle the different classes.

- Messages deemed to be 'more important' may be processed by a separate channel. While this may not be the most efficient method of transfer, it may be the most appropriate. Note that a similar effect may be achieved by making the transmission queue a priority order queue and placing these message at the head of the transmission queue.

- Very large messages may hold up smaller messages with a corresponding deterioration in message throughput. In this instance, providing a priority transmission queue will not solve the problem as a large message must be completely transferred before a high priority message is handled. In this case, a separate channel for large messages will enable other messages to be transferred faster.

- If it is appropriate to route message traffic through different parts of the underlying network, multiple channels will enable those different network routes to be utilized by allowing different (APPC) classes of service or different target nodes to be specified in the channel definitions.

Clients

WebSphere MQ Client Architectures

- Large systems have been built with clients
  - Up to 50,000 clients per server

- Similar considerations to synchronous architectures
  - Request / response time critical
  - WMQ Transport cost can be < 1 milli second
  - 256 byte user message causes 1420 bytes flow

- Scalability considerations
  - Large number of processes on server
  - Trusted bindings (Channel programs)
  - Overheads of per-client ReplyToQ
    - Share Q's with CorrelId/MsgId

WebSphere MQ Client Architectures

N

O

T

E

S

- WMQ thin clients offer lightweight, low overhead, low cost and low administration access to WMQ services. Clients reduce the requirements for machine resources on the client machine, but there are tradeoffs: Resources on the server are required for the MCAs to handle the client connections - 1 per client connection (MQCONN).

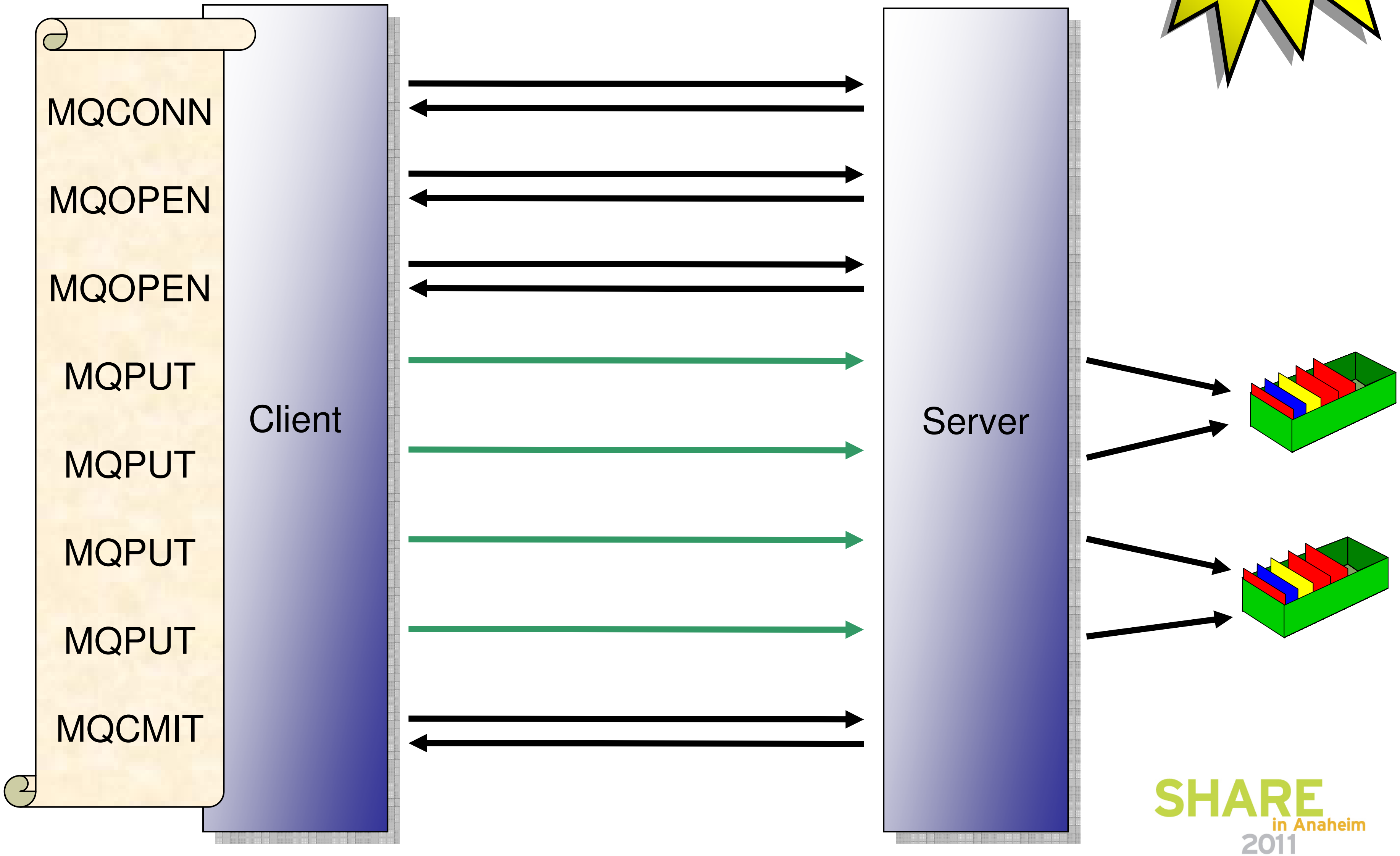
- Application architectures built around thin clients often feature large numbers of connections. WMQ has been proven with large configurations of up to 32,000 clients concurrently attached to a single AIX server. However, there are some points to consider to achieve the best performance with thin clients:
  - Large configurations (ie many client attachments) result in a large number of WMQ processes:
  - Each client connection requires a channel. Each channel requires a receiver and an agent.

- The number of processes can be reduced by using trusted bindings for the receiver, eliminating the agent processes.

- Since each queue requires control structures in memory, having a ReplyToQ for each client will result in a large number of queues and high memory usage. You can reduce the number of queues, and therefore memory requirements, by sharing a ReplyToQ between some (or all) of the clients, and referencing reply messages using MsgId and/or Correll_id.
- Each API call is transferred (without batching) to the server, where the call is executed and the results returned to the client. The MQMD has to be passed on input and output flow. Similarly the MQGMO/MQPMO.

Asynchronous Put Response

New in MQ V7



SHARE in Anaheim 2011

Asynchronous Put Response

N O T E S

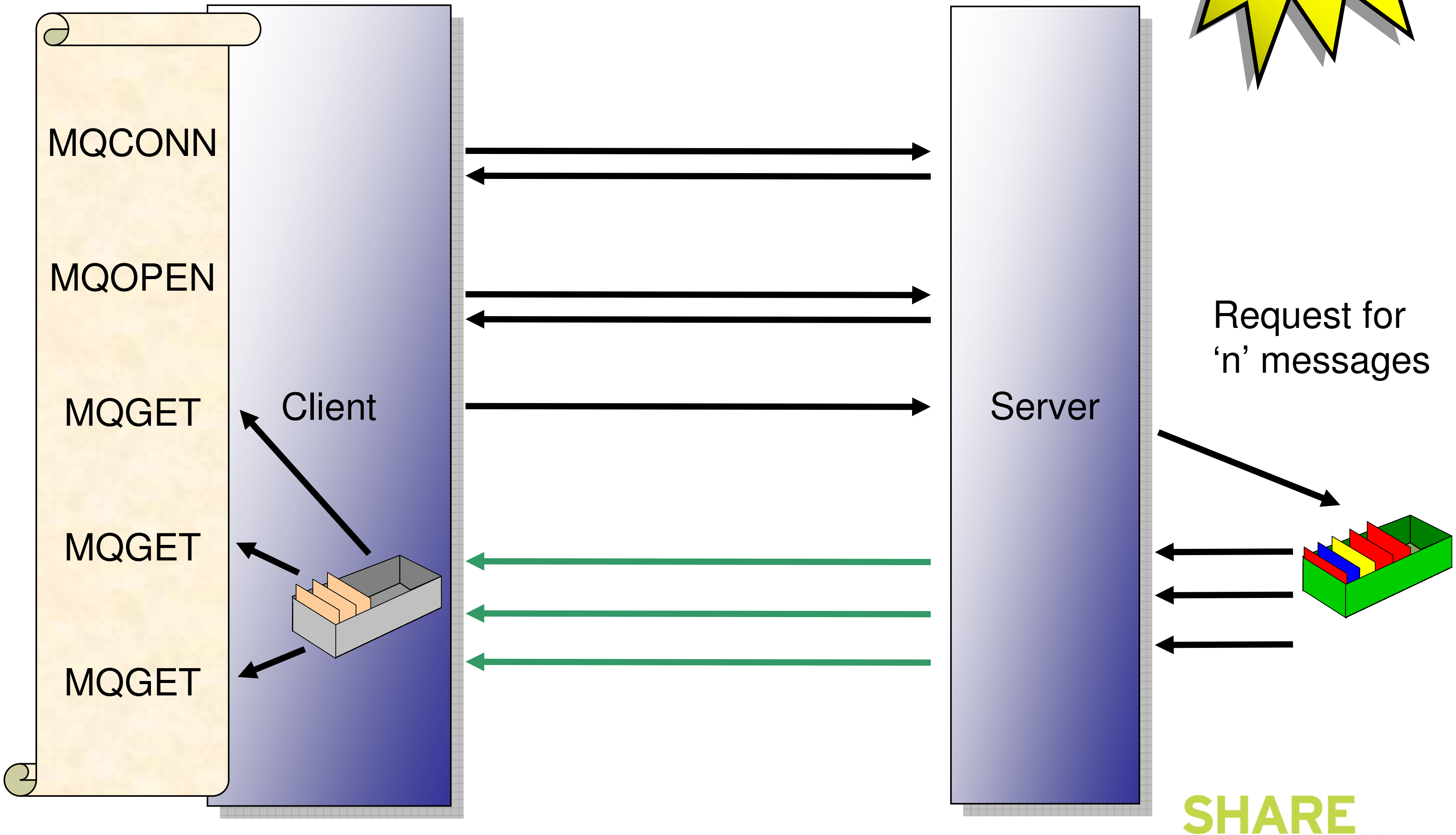
- Asynchronous Put (also known as 'Fire and Forget') is a recognition of the fact that a large proportion of the cost of an MQPUT from a client is the line turnaround of the network connection. When using Asynchronous Put the application sends the message to the server but does not wait for a response. Instead it returns immediately to the application. The application is then free to issue further MQI calls as required. The largest speed benefit will be seen where the application issues a number of MQPUT calls and where the network is slow.

- Once the application has competed it's put sequence it will issue MQCMIT or MQDISC etc which will flush out any MQPUT calls which have not yet completed.

- Because this mechanism is designed to remove the network delay it currently only has a benefit on client applications. However, it is recommended that applications that could benefit from it, use it for local bindings as well since in the future there is the possibility that the server could perform some optimisations when this option is used.

Read-ahead of messages

New in MQ V7



Read-ahead of messages

N O T E S

- Read Ahead (also known as 'Streaming') is a recognition of the fact that a large proportion of the cost of an MQGET from a client is the line turnaround of the network connection. When using Read Ahead the MQ client code makes a request for more than one message from the server. The server will send as many non-persistent messages matching the criteria (such as MsgId) as it can up to the limit set by the client. The largest speed benefit will be seen where there are a number of similar non-persistent messages to be delivered and where the network is slow.

- Read Ahead is useful for applications which want to get large numbers of non-persistent messages, outside of syncpoint where they are not changing the selection criteria on a regular basis. For example, getting responses from a command server or a query such as a list of airline flights.

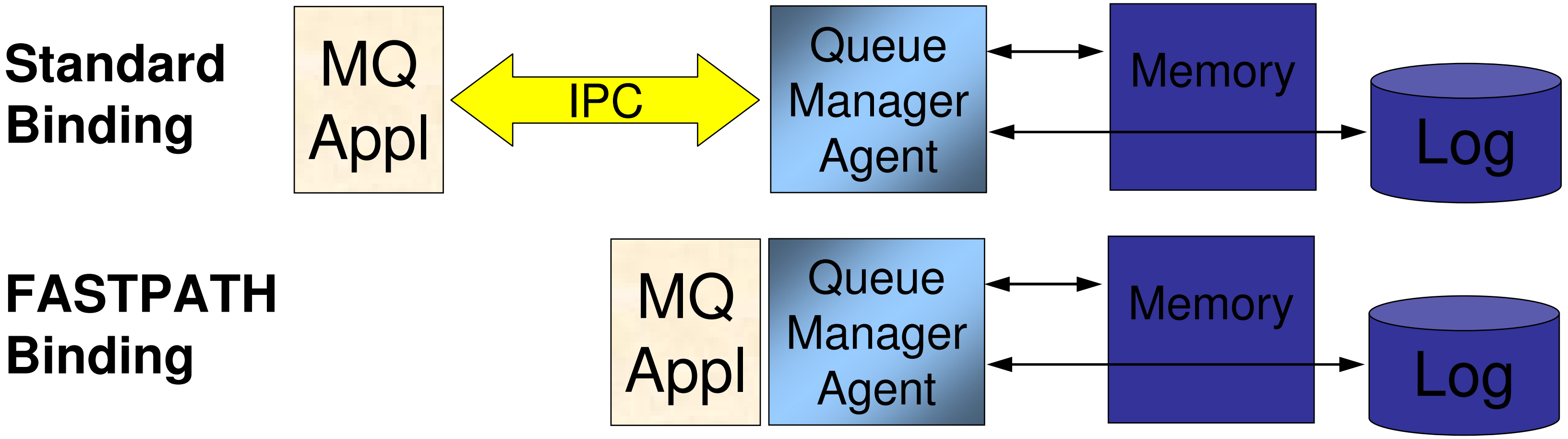
- If an application requests read ahead but the messages are not suitable, for example, they are all persistent then only one message will be sent to the client at any one time. Read ahead is effectively turned off until a sequence of non-persistent messages are on the queue again.

- The message buffer is purely an 'in memory' queue of messages. If the application ends or the machine crashes these messages will be lost.

- Because this mechanism is designed to remove the network delay it currently only has a benefit on client applications. However, it is recommended that applications that might benefit from it, use it for local bindings as well since in the future there is the possibility that the server could perform some optimisations when this option is used.

Distributed Specifics

Connection Binding



- Fastpath binding removes IPC component
  - Implies that the application is 'trusted' !!
  - MQCONNX option MQCNO_FASTPATH_BINDING
  - Application failure can corrupt queue manager
- Primary benefit is for non-persistent message processing
  - Use for MCAs, Broker
    - - 30% CPU saving

Connection Binding

N O T E S

- Two of the major overheads in the processing path for WMQ are the Inter-Process Communication component and the I/O subsystem.
- For non-persistent messages, the I/O subsystem is rarely used. Therefore there is substantial benefit to be gained from by-passing the IPC component. This is what the Trusted Binding provides.
- Depending upon the efficiency of the IPC component for a particular platform, the use of a Trusted Binding will provide anything up to an 3 times reduction in the pathlength for non-persistent message processing.
- There is a price to pay for this improvement in pathlength. The Standard Binding for applications provides separation of user code and WMQ code (via the IPC component). The actual Queue Manager code runs in a separate process from the application, known as an agent process (AMQZLAA0). Using standard binding it is not possible for a user application to corrupt queue manager internal control blocks or queue data. This will NOT be the case when a Trusted Binding is used, and this implies that ONLY applications which are fully tested and are known to be reliable should use the Trusted Binding.
- The Trusted Binding applies to the application process and will also apply to persistent message processing. However, the performance improvements are not so great as the major bottleneck for persistent messages is the I/O subsystem.

- Use for Channel programs
  - MQ_CONNECT_TYPE=FASTPATH env variable
  - qm.ini under the Channels: section
    - MQIBindType=FASTPATH
  - Do not issue Stop Channel mode(TERMINATE)
  - Exit code
- Write exits so that Channels and Broker can run trusted
- See integrity discussion in Application Programming Reference

Queue Choice

- Predefined Queues and Dynamic Queues

- Share between small number of applications
  - Specific Queue can only have one update outside syncpoint per Log I/O
  - Multiple updates inside syncpoint per Log I/O
  - Try not to have reply queue each if large number of clients

- Single Q on 4 way system
  - can process half thruput of n(>4) Queues
  - Plan for an active queue per CPU(WMQ only)
  - More like 4 CPU's per Q for Message Broker

- Multiple applications with own queues
  - Can each update queue per Log I/O

- Empty Queue requires…..
  - Dynamic Queue 50K Virtual storage
  - Predefined Queue 300K Virtual storage

Queue Choice

N

O

T

E

S

- An empty queue needs 12 pages from the FREEPAGE list to build an infrastructure to contain non persistent buffer, persistent buffer, SPACE map about all messages in the queue and a table of recently accessed messages.
- The creation and deletion of this infrastructure is expensive and any messages in the queue are read(header) to build up space map.
- When the last handle to a queue is closed, the queue is unloaded at the second checkpoint.
- The queue infrastructure (including the disk file) becomes a GHOST queue that can be quickly reused when another queue is opened

z/OS Specifics

Long running UOWs

- Bad for the health of your queue manager
- Require storage inside the queue manager
- Hold locks on pages (stopping them being reused)
  - Might effect the size of pagesets required
- Could cause a qmgr outage
  - Need to be able to access all log records to backout the UOW
  - Particular problem if not archiving logs
  - Need to access archive from tape?
  - Less of an issue at V6 with Log Shunting – but still don't do it!

Long running UOWs

N

O

T

E

S

- Long running UOWs are generally a bad idea! The queue manager is optimised for short UOWs (both time and size). Each message that is put or got within a UOW is going to require both storage and locks inside the queue manager. This will reduce the resources available to other users of the queue manager. For example, if there are 100000 messages on a queue, and they are all got within a single UOW, there will be locks on every single message (held in storage in the queue manager) to stop any one else from getting the messages, and making sure that their pages used can't be freed until the UOW is committed. This will mean that the pageset space can't be reclaimed until the commit has taken place. This will mean that larger pagesets are required to cope with the pages being locked for a long time.

- In an extreme case it is possible to cause a queue manager outage by having a long running UOW. If an application backs out then the queue manager needs to read all the log records back to the start of the UOW. If the queue manager can't read all these records it will abend as otherwise it could cause a data integrity issue. This is particularly an issue of archive logging is not being used and long running UOWs don't fit completely in the active logs.

- Log Shunting was added in V6 which helps alleviate this to a certain extent, this is done by copying a small part of each log record for a UOW to a more recent point in the logs. This will mean that we try to keep the records in the active logs. Having said that, it is still not a good idea to have long running UOWs and the root cause should be addressed rather than relying on Log Shunting.

Shared Queues

- Need to take into account CF CPU and link usage
- CPU cost is typically higher than for private queues
- Because work can be spread across sysplex, higher throughput can be achieved
  - If bottleneck is logging of persistent message, adding an extra queue manager will get more log bandwidth

Shared Queues

N

O

T

E

S

- When looking at the cost of using shared queues, you need to consider the CF CPU and link costs in addition to the CPU cost on the image where the queue manager is running. As a very rough guide, persistent shared queue message CPU cost is approximately 160% more than for the best case local queue messages (best case being kept in bufferpool etc).

- Because for persistent messages the bottleneck is typically log bandwidth rather than CPU usage, an increased throughput can be achieved by adding additional queue managers. Each queue manager can effectively have its own log bandwidth, so persistent message through put can scale well.

Large shared queue messages

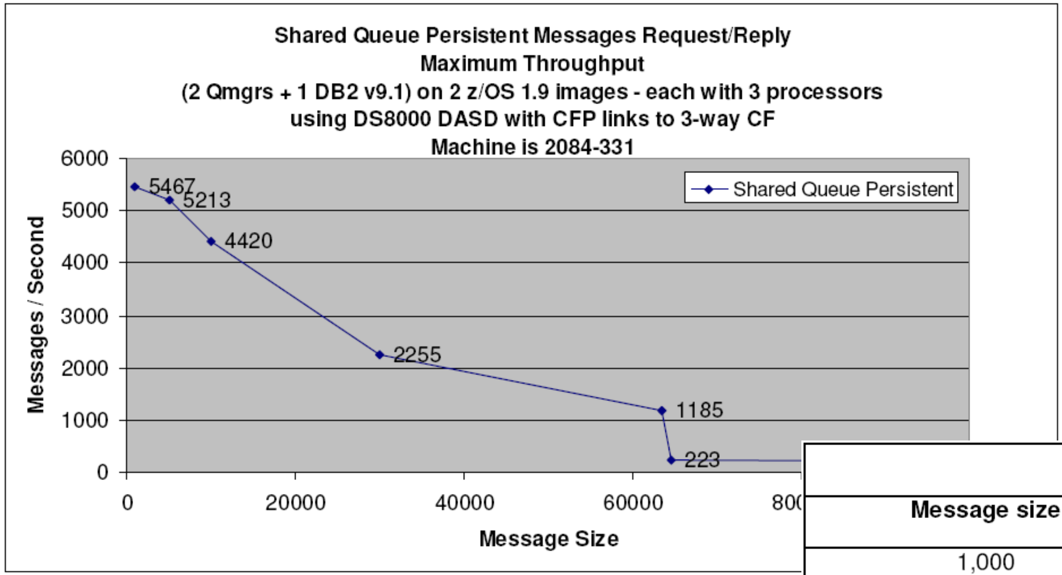
- If message is over 64512 bytes data is stored in DB2 LOB table
  - Some data still in CF
- Drop in throughput when going through 63KB message size
  - 1185 msgs/sec to 223 msgs/sec
- CPU cost increases when DB2 is needed to store message data

Large shared queue messages

N O T E S

- Messages up to 100MB can be put to shared queues. There are however performance implications of putting large messages to shared queues.
- The most noticeable impact is when changing from a message size of 64512 to 64513 bytes. In the tests performed by the performance team at Hursley a throughput of 1185 msgs/sec was achieved with messages of 64512 bytes. This dropped to 223 msgs/sec when the message size was increased to 64513 bytes.
- The CPU cost for the messages also increased significantly as the message size became greater than 63KB.
- For messages 63KB or smaller all of the data for the message is stored in the CF. When the message size exceeds 63KB then a small amount of data is stored in the CF still (about 1KB) and the whole of the message is stored in DB2. The increased cost of using large messages is due to the use of different technology for small and large messages.

Large shared queue messages



Shared Queue Persistent Messages Request/Reply
Maximum Throughput
(2 Qmgrs + 1 DB2 v9.1) on 2 z/OS 1.9 images - each with 3 processors
using DS8000 DASD with CFP links to 3-way CF
Machine is 2084-331

2084-303 CPUmillisecs/msg	
Message size	CPUmillisecs/msg
1,000	0.81
5,000	0.84
10,000	0.89
30,000	1.04
63,512	1.33
63,513	6.79
100,000	8.97
522,000	13.1
524,000	21.46

Large shared queue messages

N

O

T

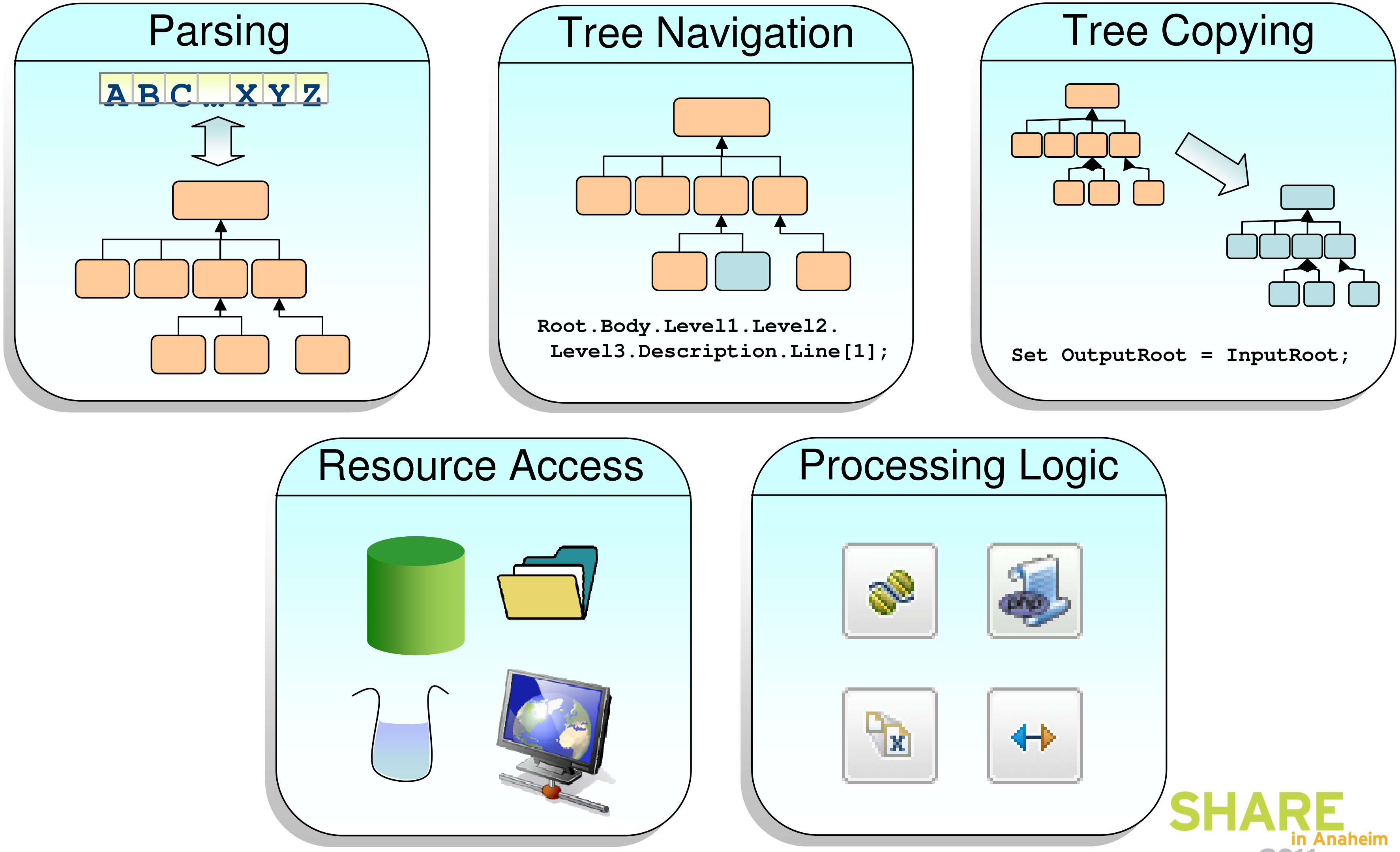
E

S

- The charts show details of measurements made by the performance team. As can be noticed on the line graph there is a discontinuity in the throughput around the 64512 byte message size.
- The CPU cost can also be seen to increase.
- Further details can be obtained from the MP16 SupportPac from which these graphs were "borrowed".

Message Broker:
What are the main costs?

What are the main performance costs in message flows?



```
Root.Body.Level1.Level2.
 Level3.Description.Line[1];
```

```
Set OutputRoot = InputRoot;
```

Slide 1 of 2

N O T E S

- This foils shows a simple message flow. When this or any other message flow processes messages costs arise. These costs are:
  - Parsing. This has two parts. The processing of incoming messages and the creation of output messages. Before an incoming message can be processed by the nodes or ESQL it must transformed from the sequence of bytes, which is the input message, into a structured object, which is the message tree. Some parsing will take place immediately such as the parsing of the MQMD, some will take place, on demand, as fields in the message payload are referred to within the message flow. The amount of data which needs to be parsed is dependent on the organization of the message and the requirements of the message flow. Not all message flows may require access to all data in a message. When an output message is created the message tree needs to be converted into an actual message. This is a function of the parser. The process of creating the output message is referred to as serialization or flattening of the message tree. The creation of the output message is a simpler process than reading an incoming message. The whole message will be written at once when an output message is created. We will discuss the costs of parsing in more detail later in the presentation.
  - Message/Business Processing. It is possible to code message manipulation or business processing in any one of a number of transformation technologies. These are ESQL, Java, PHP, Mapping node, XSL and WebSphere TX mappings. It is through these technologies that the input message is processed and the tree for the output message is produced. The cost of running this is dependent on the amount and complexity of the transformation processing that is coded.

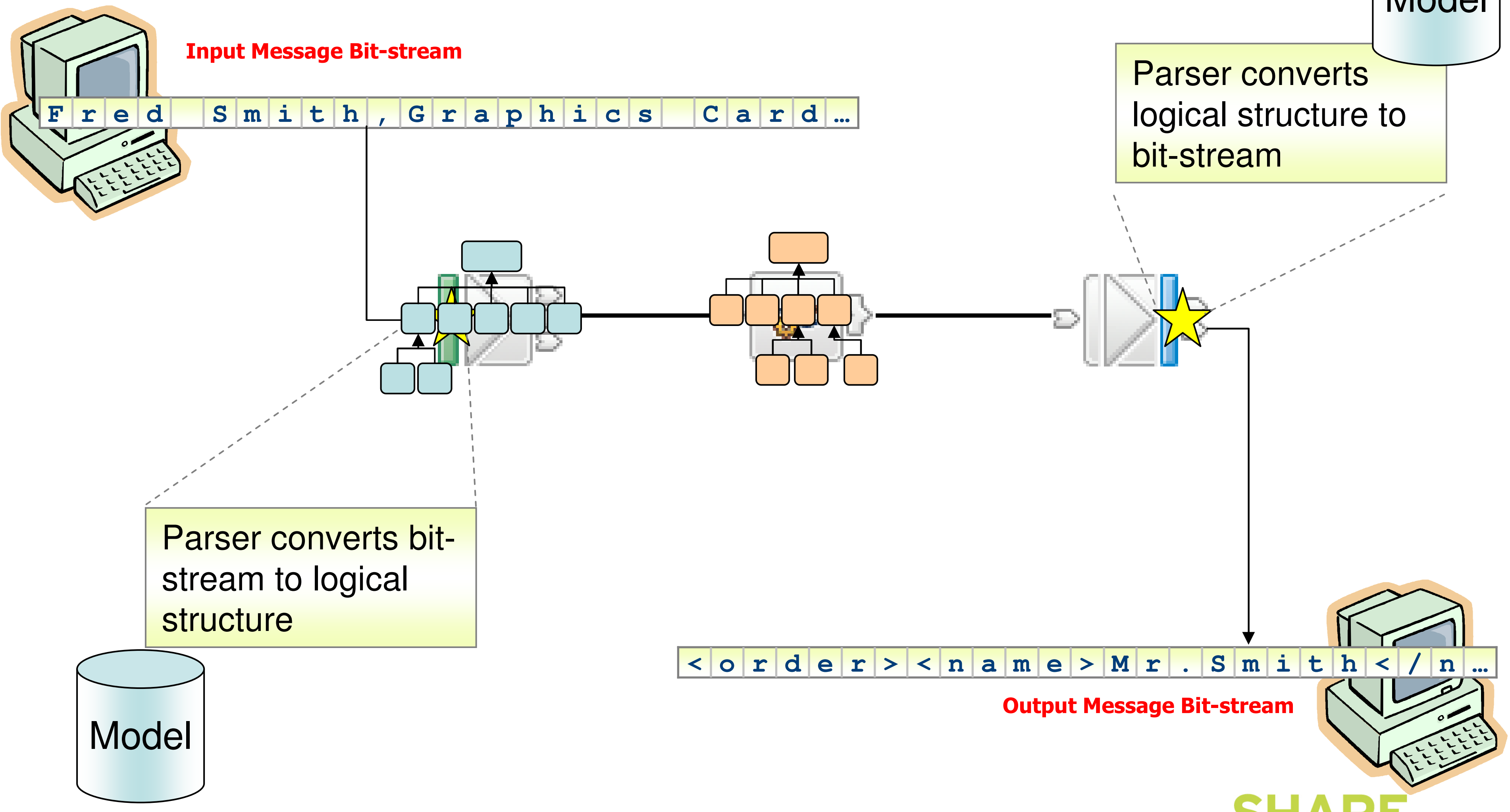
Slide 2 of 2

N
O
T
E
S

- Navigation. This is the process of "walking" the message tree to access the elements which are referred to in the ESQL or Java. The cost of navigation is dependent on the complexity and size of the message tree which is in turn dependent on the size and complexity of the input messages and the complexity of the processing within the message flow.
- Tree Copying. This occurs in nodes which are able to change the message tree such as Compute nodes. A copy of the message tree is taken for recovery reasons so that if a compute node makes changes and processing in node incurs or generates an exception the message tree can be recovered to a point earlier in the message flow. Without this a failure in the message flow downstream could have implications for a different path in the message flow. The tree copy is a copy of a structured object and so is relatively expensive. It is not a copy of a sequence of bytes. For this reason it is best to minimize the number of such copies, hence the general recommendation to minimize the number of compute nodes in a message flow. Tree copying does not take place in a Filter node for example since ESQL only references the data in the message tree and does not update it.
- Resources. This is the cost of invoking resource requests such as reading or writing WebSphere MQ messages or making database requests. The extent of these costs is dependent on the number and type of the requests.

- The parsing, navigation and tree copying costs are all associated with the population, manipulation and flattening of the message tree and will be discussed together a little later on.
- Knowing how processing costs are encountered puts you in a better position to make design decisions which minimize those costs. This is what we will cover during the course of the presentation.

Parsers



Input Message Bit-stream
F r e d   S m i t h , G r a p h i c s   C a r d …
Model
Parser converts logical structure to bit-stream
Parser converts bit-stream to logical structure
Model
< o r d e r > < n a m e > M r . S m i t h < / n …
Output Message Bit-stream

Parsing

- The means of populating and serializing the tree
  - Can occur whenever the message body is accessed
  - Multiple Parsers available: XMLNSC, MRM XML, CWF, TDS, MIME, JMSMap, JMSStream, BLOB, IDOC, RYO
  - Message complexity varies significantly …and so do costs!
- Several ways of minimizing parsing costs
  - Use cheapest parser possible, e.g. XMLNSC for XML parsing
  - Identify the message type quickly
  - Use Parsing strategies
    - Parsing avoidance
    - Partial parsing
    - Opaque parsing

Slide 1 of 2

N O T E S

- In order to be able to process a message or piece of data within Message Broker we need to be able to model that data and build a representation of it in memory. Once we have that representation we can transform the message to the required shape, format and protocol using transformation processing such as ESQL or Java. That representation of a sequence of bytes into a structured object is the message tree. It is populated though a process called parsing.
- There are two parts to parsing within Message Broker:
  - The first is the most obvious. It is the reading and interpretation of the input message and recognition of tokens within the input stream. Parsing can be applied to message headers and message body. The extent to which it is applied will depend on the situation. As parsing is an expensive process in CPU terms it is not something that we want to automatically apply to every part of every message that is read by a message flow. In some situations it is not necessary to parse all of an incoming message in order to complete the processing within the message flow. A routing message flow may only need to read a header property such as user identifier or ApplIdentifyData in the MQMD for example. If the input message is very large parsing the whole message could result in a large amount of additional CPU being unnecessarily consumed if that data is not required in the logic of the message flow.
  - The second, is the flattening, or serialisation of a message tree to produce a set of bytes which correspond to the wire protocol of a message in the required output format and protocol.

Slide 2 of 2

N

O

T

E

S

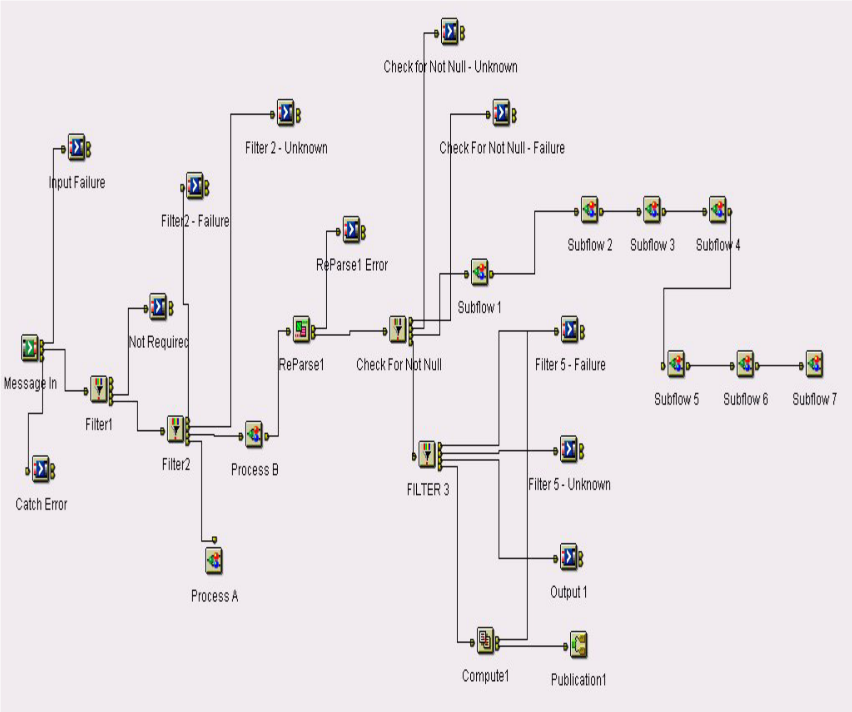
- Message Broker provides a range of parsers. The parsers are named on the foil. The parsers cover different message formats. There is some overlap – XMLNSC and MRM XML are both able to parse a generic XML message but there are also differences. MRM XML can provide validation. XMLNSC does not. Other parsers such as JMSStream are targeted at specific message formats. In this case it is a JMS stream message.

- Messages vary significantly in their format and the level of complexity which they are able to model. Tagged Delimited string (TDS) messages for example can support nested groups. More complex data structures make parsing costs higher. The parsing costs of different wire formats is different. You are recommended to refer to the performance reports in order to get information on the parsing costs for different types of data.

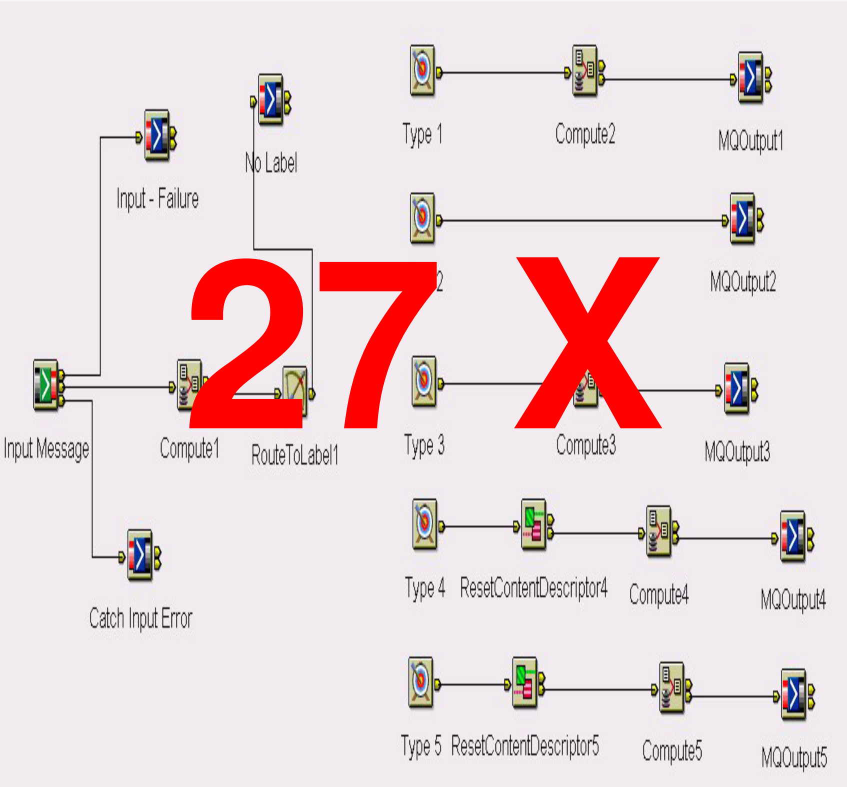
- Whatever the message format there are a number of techniques that can be employed to reduce the cost of parsing. We will now look at the most common ones. First though we will discuss the reasons for the different cost of parsing.

Identifying the message type quickly

- Avoid multiple parses to find the message type



VS



27 X

5 msgs/sec

138 msgs/sec

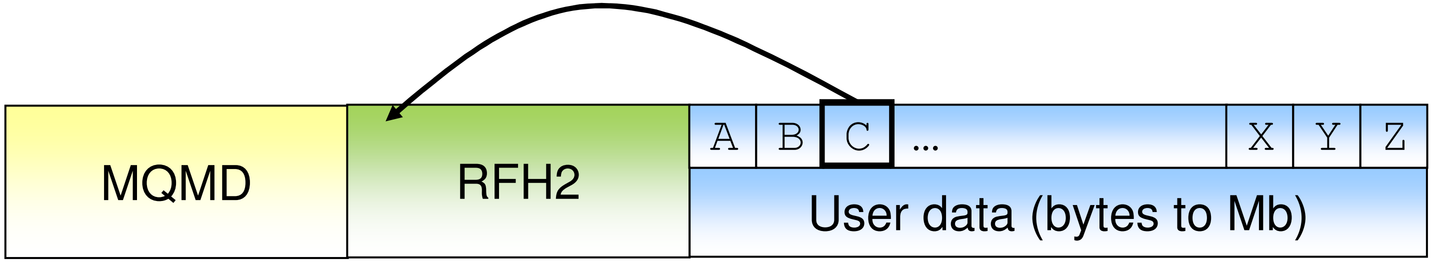
Notes

N O T E S

- It is important to be able to correctly recognise the correct message format and type as quickly as possible. In message flows which process multiple types of message this can be a problem. When often happens is that the message needs to be parsed multiple times in order to ensure that you have the correct format. Dependent on the particular format and the amount of parsing needed this can be expensive.
- This example shows how a message flow which processed multiple types of input message was restructured to provide a substantial increase in message throughput.
- Originally the flow was implemented as a flow with a long critical path. Some of the most popular messages were not processed until late in the message flow resulting in a high overhead. The flow was complex. This was largely due to the variable nature of the incoming messages, which did not have a clearly defined format. Messages had to be parsed multiple times.
- The message flow was restructured. This substantially reduced the cost of processing. The two key features of this implementation were to use a very simple message set initially to parse the messages and the use of the RouteToLabel and Label nodes within the message flow in order to establish processing paths which were specialized for particular types of message.
- Because of the high processing cost of the initial flow it was only possible to process 5 non persistent message per second using one copy of the message flow. With the one copy of the restructured flow it was possible to process 138 persistent messages per second on the same system. This is a 27 times increase in message throughput plus persistent messages where now being processed whereas previously they were non persistent.
- By running five copies of the message flow it was possible to:
  - Process around 800 non persistent messages per second when running with non persistent messages.
  - Process around 550 persistent messages per second when running with persistent messages.

Parser avoidance

- If possible, avoid the need to parse at all!
  - Consider only sending changed data
  - Promote/copy key data structures to MQMD, MQRFH2 or JMS Properties
    - May save having to parse the user data
    - Particularly useful for message routing



Notes

N O T E S

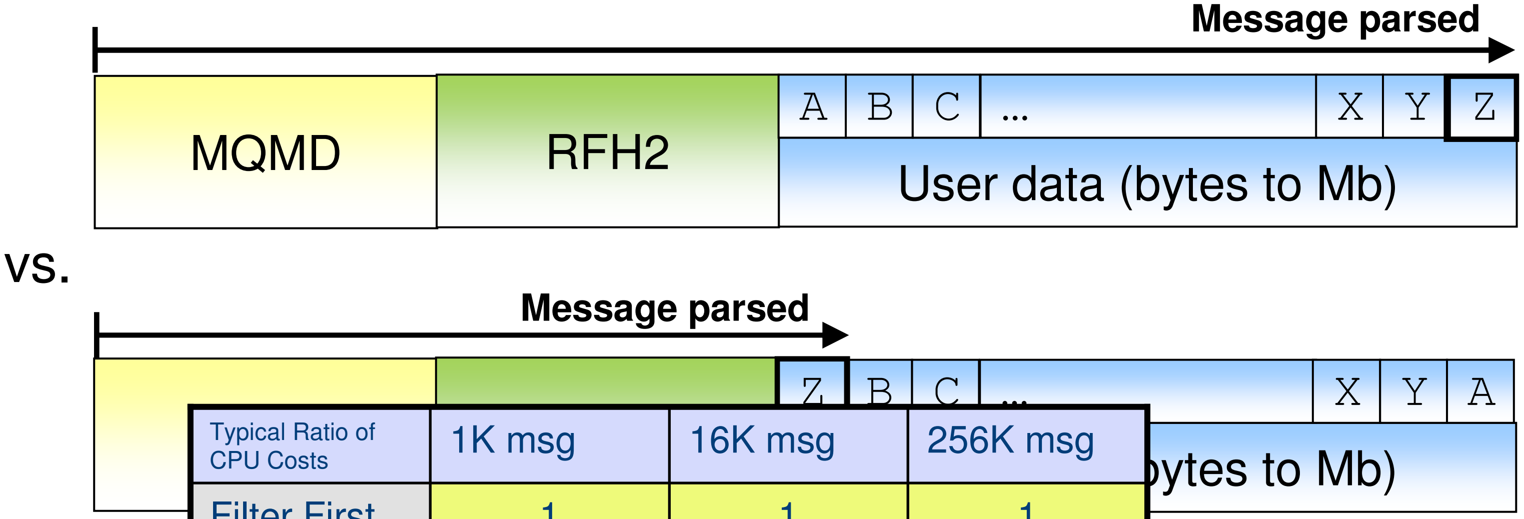
- One very effective technique to reduce the cost of parsing is not to do it. A couple of ways of doing this are discussed in this foil.
- The first approach is to avoid having to parse some parts of the message. Suppose we have a message routing message flow which needs to look at a field in order to make a routing decision. If that field is in the body of the message the body of the incoming message will have to be parsed to get access to it. The processing cost will vary dependent on which field is needed. If it is field A that is OK as it is at the beginning of the body and would be found quickly. If it is field Z then the cost could be quite different, especially if the message is several megabytes in size. A technique to reduce this cost would be to have the application which creates this message copy the field that is needed for routing into a header within the message, say in an MQRFH2 header for an MQ message or as a JMS property if it is a JMS message. If you were to do this it would no longer be necessary to parse the message body so potentially saving a large amount of processing effort. The MQRFH2 heard or JMS Properties folder would still need to be parsed but this is going to be smaller amount of data. The parsers in this case are also more efficient then the general parser for a message body as the structure of the header is known.

- A second approach to not parsing data is to not send it in the first place. Where two applications communicate consider sending only the changed data rather than sending the full message. This requires additional complexity in the receiving application but could potentially save significant parsing processing dependent on the situation. This technique also has the benefit of reducing the amount of data to be transmitted across the network.

Partial parsing

- Typically, the Broker parses elements up to and including the required field
  - Elements that are already parsed are not reparsed
- If possible, put important elements nearer the front of the user data

```
Set MyVariable = <Message Element Z>
```



Message parsed

MQMD | RFH2 | A | B | C | ... | X | Y | Z

User data (bytes to Mb)

vs.

Message parsed

Z | B | C | ... | X | Y | A

bytes to Mb)

Typical Ratio of CPU Costs	1K msg	16K msg	256K msg
Filter First	1	1	1
Filter Last	1.4	3.4	5.6

Slide 1 of 2

N O T E S

- Given parsing of the body is needed the next technique is to only parse what is needed rather than parse the whole message just in case. The parsers provided with Message Broker all support partial parsing.
- The amount of parsing which needs to be performed will depend on which fields in a message need to accessed and the position of those fields in the message. We have two example here. One with the fields ordered Z to A and the other with them ordered A to Z. Dependent on which field is needed one of the cases will be more efficient than the other. Say we need to access field Z then the first case is best. Where you have influence over message design ensure that information needed for routing for example is placed at the start of the message and not at the end of the message.
- As an illustration of the difference in processing costs consider the table in the foil. The table shows a comparison of processing costs for two versions of a routing message flow when processing several different message sizes.
  - For the filter first test the message flow routes the message based on the first field in the message. For the filter last the routing decision is made based on the last field of the message.
  - The CPU costs of running with each message size have been normalised so that the filter first test for each message represents a cost of 1. The cost of the filter last case is then given as a ratio to the filter first cost for that same message size.
  - For the 1K message we can see that by using the last field in the message to route the CPU cost of the whole message flow was 1.4 times the cost of using the first field in the same message. This represents the additional parsing which needs to be performed to access the last field.
  - For the 16K message, the cost of using the last field in the message had grown to 3.4 times that of the first field in the same message. That is we can only run at one third of the message that we can when looking at the first field of the message.
  - These measurements were taken on a pSeries 570 Power 5 with 8 * 1.5 GHZ processors, using WebSphere Message Broker V6.

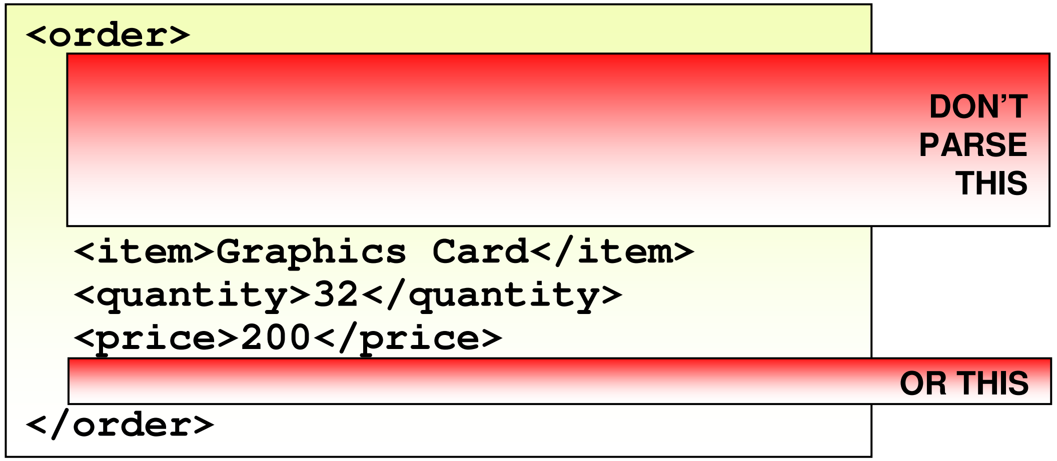
Slide 2 of 2

N O T E S

- For the 256K message, the cost of using the last field in the message was 5.6 times that of the first field in the same message. That is we can only run at one fifth of the message that we can when looking at the first field of the message.
- You can see how processing costs quickly start to grow and this was a simple test case. With larger messages the effect would be even more pronounced. Where the whole message is used in processing this is not an issue as all of the data will need to be parsed at some point. For simple routing cases though the ordering of fields can make a significant difference.
- When using ESQL, and Mapping nodes the field references are typically explicit. That is we have references such as InputRoot.Body.A. Message Broker will only parse as far as the required message field to satisfy that reference. It will stop at the first instance. When using XPath, which is a query language the situation is different. By default an XPath expression will search for all instances of an element in the message which implicitly means a full parse of the message. If you know there is only one element in a message there is the chance to optimise the XPath query to say only retrieve the first instance. See the ESQL and Java coding tips later in the presentation for more information.

Opaque parsing

- Treat elements of an XML document as an unparsed BLOB
- Reduces message tree size and parsing costs
- Cannot reference the sub tree in message flow processing
- Configured on input nodes (“Parser Options” tab)



```
<order>

  <item>Graphics Card</item>
  <quantity>32</quantity>
  <price>200</price>

</order>
```

DON’T PARSE THIS

OR THIS

Slides 1 of 2

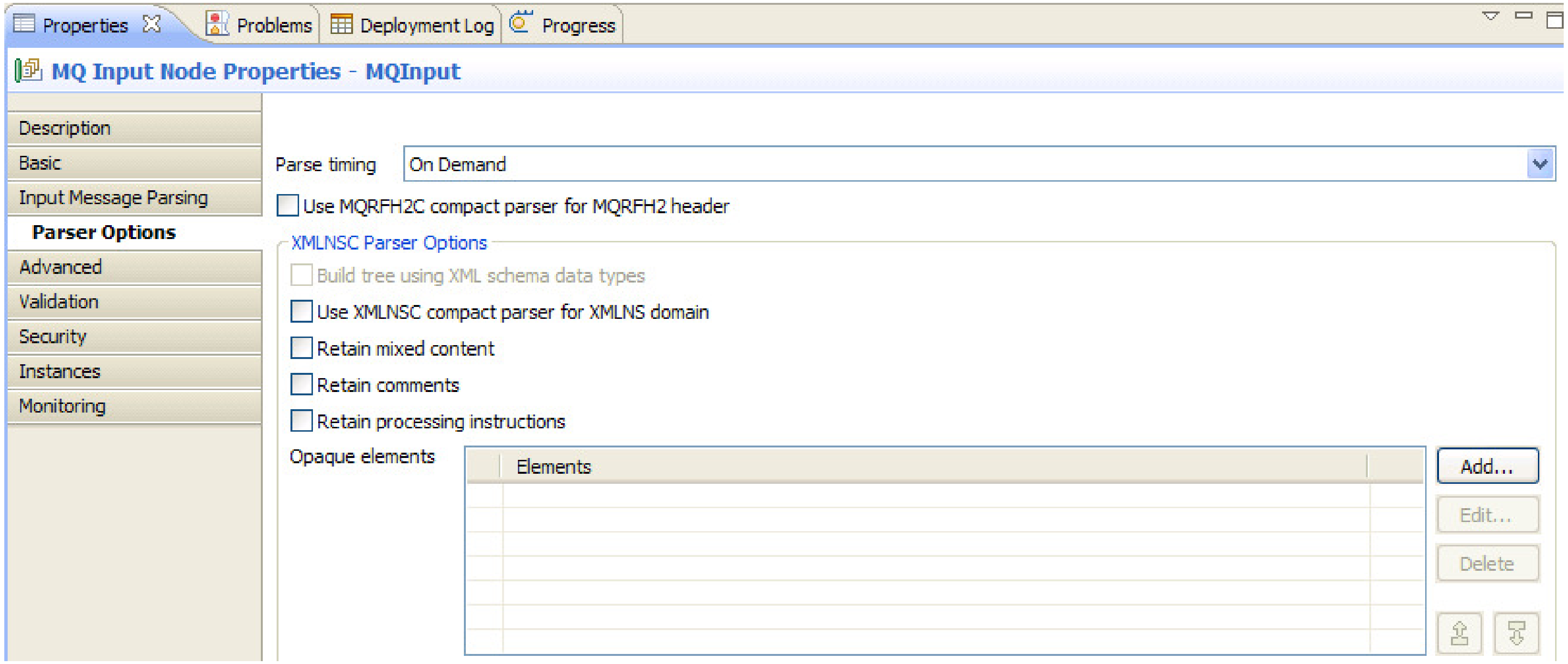
N O T E S

- Opaque parsing is a technique that allows the whole of an XML sub tree to placed in the message tree as a single element. The entry in the message tree is the bit stream of the original input message. This technique has two benefits:
  - It reduces the size of the message tree since the XML sub tree is not expanded into the individual elements.
  - The cost of parsing is reduced since less of the input message is expanded as individual elements and added to the message tree.
- Use opaque parsing where you do not need to access the elements of the sub tree, for example you need to copy a portion of the input tree to the output message but may not care about the contents in this particular message flow. You accept the content in the sub folder and have no need to validate or process it in any way.
- Opaque parsing is supported for the XMLNS and XMLNSC domains only.
  - The support for the XMLNS domain was introduced in Message Broker V6 fixpack 1 and was an early version of opaque parsing and as such could be changed in the future.
  - The support for the XMLNSC domain was added in Message Broker V6.1. It has a different interface from the XMLNS domain to specify the element names which are to be opaquely parsed.
- The support for the XMLNS domain continues to be available in its original form. Indeed this is the only way in which the messages in the XMLNS domain can be opaquely parsed. It is not possible to use the same interface that is provided for the XMLNSC to specify element names for the XMLNS domain.
- The following pages of notes explain how opaque parsing is performed for messages in the XMLNS and XMLNSC domains.

Slides 1 of 2

NOTES

- The message domain must be set to XMLNSC.
- The elements in the message which are to be opaquely parsed are specified on the 'Parser Options' page of the input node of the message flow. See the picture below of the MQInput node.
- Enter those element names that you wish to opaquely parse in the 'Opaque Elements' table.
- Be sure not to enable message validation as it will automatically disable opaque parsing. Opaque parsing in this case does not make sense since for validation the whole message must be parsed and validated.
- Opaque parsing for the named elements will occur automatically when the message is parsed.
- It is not currently possible to use the CREATE statement to opaquely parse a message in the XMLNSC domain.



MQ Input Node Properties - MQInput

Parse timing: On Demand

☐ Use MQRFH2C compact parser for MQRFH2 header

XMLNSC Parser Options

☐ Build tree using XML schema data types

☐ Use XMLNSC compact parser for XMLNS domain

☐ Retain mixed content

☐ Retain comments

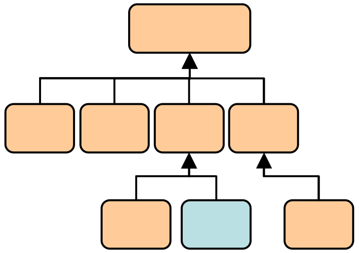
☐ Retain processing instructions

Opaque elements

Elements

Navigation

- The logical tree is walked every time it is evaluated
  - This is not the same as parsing!



```
SET Description =
          Root.Body.Level1.Level2.Level3.Description.Line[1];
```

- Long paths are inefficient
  - Minimise their usage, particularly in loops
  - Use reference variables/pointers (ESQL/Java)
  - Build a smaller message tree if possible
    - Use compact parsers (XMLNSC, MRM XML, RFH2C)
    - Use opaque parsing

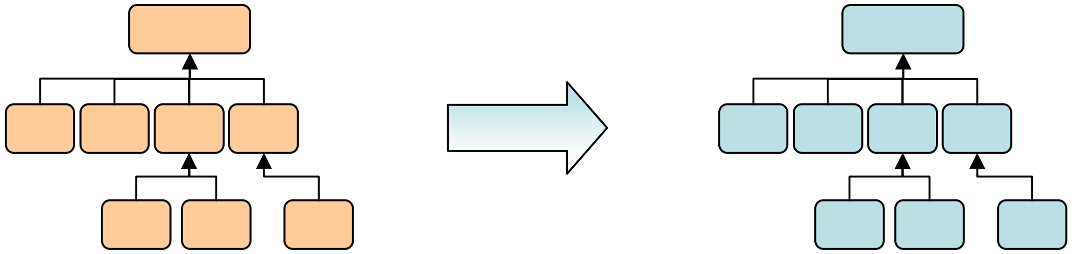
Notes

N O T E S

- Navigation is the process of accessing elements in the message tree. It happens when you refer to elements in the message tree in the Compute, JavaCompute, PHP and Mapping nodes.
- The cost of accessing elements is not always apparent and is difficult to separate from other processing costs.
- The path to elements is not cached from one statement to another. If you have the ESQL statement SET Description = Root.Body.Level1.Level2.Level3.Description.Line[1]; the broker runtime will access the message tree starting at the correlation Root and then move down the tree to Level1, then Level2, then Level3, then Description and finally it will find the first instance of the Line array. If you have this same statement in the next line of your ESQL module exactly the same navigation will take place. There is no cache to the last referenced element in the message tree. This is because access to the tree is intended to be dynamic to take account of the fact that it might change structure from one statement to another. There are techniques available though to reduce the cost of navigation.
- With large message trees the cost of navigation can become significant. There are techniques to reduce which you are recommended to follow:
  - Use the compact parsers (XMLNSC, MRM XML and RFH2C). The compact parsers discard comments and white space in the input message. Dependent on the contents of your messages this may have an effect of not. By comparison the other parsers include data in the original message, so white space and comments would be inserted into the message tree.
  - Use reference variables if using ESQL or reference pointers if using Java. This technique allows you to save a pointer to a specific place in the message tree. Say to Root.Body.Level1.Level2.Level3.Description for example.
- For an example of how to use reference variables and reference pointers see the coding tips given in the Common Problems section.

Message tree copying

```
SET OutputRoot.XML.A = InputRoot.XML.A;
```



- Message tree copying causes the logical tree to be duplicated in memory… and this is computationally expensive
- Reduce the number of times the tree is copied
  - Reduce the number of *Compute* and *JavaCompute* nodes in a message flow
  - See if “Compute mode” can be set to not include “message”
  - Copy at an appropriate level in the tree (copy once rather than for multiple branch nodes)
  - Copy data to the Environment (although changes are not backed out)
- Minimize the effect of tree copies
  - Produce a smaller message tree (again)!

Notes

N

O

T

E

S

- Tree Copying is the process of copying the message tree either in whole or part.
- Copying occurs in nodes which are able to change the message tree such as Compute nodes. A copy of the message tree is taken for recovery reasons so that if a compute node makes changes and processing in node incurs or generates an exception the message tree can be recovered to a point earlier in the message flow. Without this a failure in the message flow downstream could have implications for a different path in the message flow. The tree copy is a copy of a structured object and so is relatively expensive. It is not a copy of a sequence of bytes. For this reason it is best to minimize the number of such copies, hence the general recommendation to minimize the number of compute nodes in a message flow. Tree copying does not take place in a Filter node for example since ESQL only references the data in the message tree and does not update it.
- There are a few ways to reduce the costs of tree copying. These are:
  - Produce a smaller message tree in the first place. A smaller tree will cost less to copy. Ways to achieve this are to use Smaller messages, use Compact Parsers (XMLNSC, MRM XML, RFH2C), use Opaque parsing partial parsing (all discussed previously).
  - Reduce the number of times the whole tree is copied. Reducing the number of compute nodes (ESQL or Java) will help to reduce the number of times that the whole tree needs to be copied. In particular avoid situations where you have one compute node followed immediately by another. In many cases you may need multiple computes across a message flow. What we want to do is to optimise the processing not have to force everything into one compute node.
  - Copy portions of the tree at the branch level if possible rather than copying individual leaf nodes. This will only work where the structure of the source and destination are the same but it is worth doing if possible.
  - Copy data to the Environment correlation (remember it is not backed out) and work with it in Environment. This way the message tree does not have to be copied every time you run a compute node. Environment is a scratchpad area which exists for each invocation of a message flow. The contents of Environment are accessible across the whole of the message flow. Environment is cleared down at the end of each message flow invocation.
    - The first compute node in a message flow can copy InputRoot to Environment. Intermediate nodes then read and update values in the Environment instead of using the InputRoot and OutputRoot correlations as previously.
    - In the final compute node of the message flow OutputRoot must be populated with data from Environment. The MQOutput node will then serialize the message as usual. Serialization will not take place from Environment.
  - Whilst use of the Environment correlation is good from a performance point of view be aware that the any updates made by a node which subsequently generates an exception will remain in place. There is no back-out of changes as there is when a message tree copy was made prior to the node exception

Message Broker:
What other considerations are there?

Resource Access

- **MQ**
  - Tune the QM and associated QM's, logs, buffer sizes
  - Consider running as a trusted application (but be aware of the risks)
  - Avoid unnecessary use of persistent messages, although always use persistent messages as part of a co-ordinated transaction
  - Use fast storage if using persistent messages
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html

- **JMS**
  - Follow MQ tuning if using MQ JMS
  - Follow provider instructions
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0604_bicheno/0604_bicheno.html

- **SOAP/HTTP**
  - Use persistent HTTP connections
  - Use embedded execution group-level listener for HTTP nodes (V7.0.0.1)
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0608_braithwaite/0608_braithwaite.html

- **File**
  - Use cheapest delimiting option (see reports)
  - Use fast storage

Slide 1 of 3

N O T E S

- There are a wide variety of ways of reading data into Message Broker and sending it out once it is has been processed. The most common methods are MQ messages, JMS messages, HTTP and files. Collectively for this discussion lets refer to them as methods of transport although we recognise that is pushing it where file, VSAM and database are concerned.
- Different transports have different reliability characteristics. An MQ persistent message has assured delivery. It will be delivered once and once only. MQ non persistent messages are also pretty reliable but are not assured. If there is a failure of machine or power outage for example data will be lost with these non persistent messages. Similarly data received over a raw TCP/IP session will be unreliable. With WebSphere MQ Real-time messages can be lost if there is a buffer overrun on the client or server for example.
- In some situations we are happy to accept the limitations of the transport in return for its wide availability or low cost for example. In others we may need to make good the shortcoming of the transport. One example of this is the use of a WebSphere MQ persistent message (or database insert) to reliably capture coming over a TCP/IP connection as the TCP/IP session is unreliable – hey this is why MQ is popular because of the value it builds on an unreliable protocol.
- As soon as data is captured in this way the speed of processing is changed. We now start to work at the speed of MQ persistent messages or the rate at which data can be inserted into a database. This can result in a significant drop in processing dependent on how well the queue manager or database are tuned.
- In some cases it is possible to reduce the impact of reliably capturing data if you are prepared to accept some additional risk. If it is acceptable to have a small window of potential loss you can save the data to an MQ persistent message or database asynchronously. This will reduce the impact of such processing on the critical path. This is a decision only you can make. It may well vary by case by case dependent on the value of the data being processed.
- It is important to think how data will be received into Message Broker. Will data be received one message at a time or will it arrive in a batch. What will be involved in session connection set-up and tear-down costs for each message or record that is received by the message flow. Thinking of it in MQ terms will there be an MQCONN, MQOPEN, MQPUT, MQCMT and MQDISC by the application for each message sent to the message flow. Lets hope not as this will generate lot of additional processing for the broker queue manager [assuming the application is local to the broker queue manager]. In the HTTP world you want to ensure that persistent sessions (not to be confused with MQ persistence) are used for example so that session set-up and tear-down costs are minimised. Make sure that you understand the sessions management behaviour of the transport that is being used and that you know how to tune the transport for maximum efficiency.
- Message Broker give you the ability to easily switch transport protocols, so data could come in over MQ and leave the message flow in a file or be sent as a JMS message to any JMS 1.1 provider. In your processing you should aim to keep a separation between the transport used and the data being sent over that transport. This will make it easier to use different transports in the future.

Slide 2 of 3

NOTES

- As with other MQ applications it is possible to run a broker as a trusted application on Windows, Solaris, HP-UX and AIX. The benefit of running in trusted mode is that the cost of an application communicating with the queue manager is reduced and MQ operations are processed more efficiently. It is thus possible to achieve a higher level of messaging with a trusted application. However when running in trusted mode it is possible for a poorly written application to crash and potentially corrupt the queue manager. The likelihood of this happening will depend on user code, such as a plugin node, running in the message flow and the extent to which it has been fully tested. Some users are happy to accept the risk in return for the performance benefits. Others are not.
- Using the MBExplorer it is possible to increase the number of instances of a message flow which are running. Similarly you can increase the number of execution groups and assign message flows to those execution groups. This gives the potential to increase message throughput as there are more copies of the message flow. There is no hard and fast rule as to how many copies of a message flow to run. For guidelines look at the details in the Additional Information section
- Tuning is available for non MQ transports. For example:
  - When using HTTP you can tune the number of threads in the HTTP listener
  - When using JMS nodes follow the tuning advice for the JMS provider that you are connecting to.
- As the broker is dependent on the broker queue manager to get and put MQ messages the performance of this component is an important component of overall performance.
- When using persistent messages it is important to ensure the queue manager log is efficient. Review log buffer settings and the speed of the device on which the queue manager log is located.
- It is possible to run the WebSphere queue manager channels and listener as trusted applications. This can help reduce CPU consumption and so allow greater throughput.
- In practice the broker queue manager is likely to be connected to other queue managers. It is important to examine the configuration of these other queue managers as well in order to ensure that they are well tuned.

Slide 3 of 3

N

O

T

E

S

- The input messages for a message flow do not magically appear on the input queue and disappear once they have been written to the output queue by the message flow. The messages must some how be transported to the broker input queue and moved away from the output queue to a consuming application.
- Let us look at the processing involved to process a message in a message flow passing between two applications with different queue manager configurations.
  - Where the communicating applications are local to the broker, that this they use the same queue manager as the broker, processing is optimized. Messages do not have to move over a queue manager to queue manager channel. When messages are non persistent no commit processing is required. When messages are persistent 3 commits are required in order to pass one message between the two applications. This involves a single queue manager.
  - When the communicating applications are remote from the broker queue manager, messages passed between the applications must now travel over two queue manager to queue manager channels (one on the outward, another on the return). When messages are non persistent no commit processing is required. When messages are persistent 7 commits are required in order to pass one message between the two applications. This involves two queue managers.
- If the number of queue managers between the applications and broker where to increase so would the number of commits required to process a message. Processing would also become dependent on a greater number of queue managers.
- Where possible keep the applications and broker as close as possible in order to reduce the overall overhead.
- Where multiple queue managers are involved in the processing of messages it is important to make sure that all are optimally configured and tuned.

Consider your operational environment

- Network topology
  - Distance between broker and data

- What hardware do I need?
  - Including high availability and disaster recovery requirements
  - IBM can help you!

- Transactional behaviour
  - Early availability of messages vs. backout
  - Transactions necessitate log I/O – can change focus from CPU to I/O
  - Message batching (commit count)

- What are your performance requirements?
  - Now… and in five years…

Notes

N O T E S

- Allocating the correct resources to the broker is a very important implementation step.
- Starve a CPU bound message flow of CPU and you simply will not get the throughput. Similarly neglect I/O configuration and processing could be significantly delayed by logging activities for example.
- It is important to understand the needs of the message flows which you have created. Determine whether they are CPU bound or I/O bound.
- Knowing whether a message flow is CPU bound or I/O bound is key to knowing how to increase message throughput. It is no good adding processors to a system which is I/O bound. It is not going to increase message throughput. Using disks which are much faster such as a solid state disk or SAN with a fast-write non-volatile cache will have a beneficial effect though.
- Message Broker has the potential to use multiple processors by running multiple message flows as well as multiple copies of those message flows. By understanding the characteristics of the message flow it is possible to make the best of that potential.
- In many message flows parsing and manipulation of messages is common. This is CPU intensive activity. As such message throughput is sensitive to processor speed. Brokers will benefit from the use of faster processors. As a general rule it would be better to allocate fewer faster processors than many slower processors.
- We have talked about ensuring that the broker and its associated components are well tuned and it is important to emphasize this. The default settings are not optimized for any particular configuration. They are designed to allow the product to function not to perform to its peak.
- It is important to ensure that software levels are as current as possible. The latest levels of software such as Message Broker and WebSphere MQ offer the best levels of performance.

Message Flow Deployment Algorithm

How many Execution Groups should I have?
How many Additional Instances should I add?

Execution Groups

- Results in a new process/address-space
- Increased memory requirement
- Multiple threads including management
- Operational simplicity
- Gives process level separation
- Scales across multiple EGs

Additional Instances

- Results in more processing threads
- Low(er) memory requirement
- Thread level separation
- Can share data between threads
- Scales across multiple EGs

Recommended Usage

- Check resource constraints on system
  - How much memory available?
  - How many CPUs?
- Start low (1 EG, No additional instances)
- Group applications in a single EG
- Assign heavy resource users to their own EG
- Increment EGs and additional instances one at a time
  - Keep checking memory and CPU on machine
- Don't assume configuration will work the same on different machines
  - Different memory and number of CPUs

Ultimately
have
to
balance
Resource
Manageability
Availability

Slide 1 of 2

N O T E S

- Within a broker it is possible to have one or more execution groups.
- Execution group is a broker term and is a container in which one or more message flows can run.
- The implementation of an execution group varies with platform. On Windows and UNIX platforms it is implemented as an operating system process. On z/OS it is implemented as an address space.
- The execution group provides separation at the operating system level. If you have message flows which you must separate for some reason, you can achieve this by assigning them to different execution groups.
- An individual message flow runs as an operating system thread on the Windows and UNIX platforms and as a Task Control Block (TCB) on z/OS.
- It is possible to run more than one copy of a message flow in an execution group, in which case there will be multiple threads or TCBs running the same message flow. Equally you can run a message flow in more than one execution group. In which case there will be one or more threads or TCBs running the message flow in each of the processes or address spaces to which the message flow has been assigned.
- A significant benefit of using Message Broker is that the threading model is provided as standard. The message flow developer does not need to explicitly provide code in order to cope with the fact that multiple copies of the message flow might be run.
- How many copies and where they are run is an operational issue and not a development one. This provides significant flexibility.
- The ability to use multiple threads or TCBs and also to replicate this over multiple processes or address spaces means that there is excellent potential to use and exploit multiprocessor machines.
- In testing and production you will need to decide the which message flows are assigned to which execution groups and how many execution groups are allocated. There are a number of possible approaches: Allocate all of the message flows in to one or two execution groups: Have an execution group for each message flow; Split message flows by project and so on. Before considering what the best approach is, lets look at some of the characteristics of execution groups and additional instances.

Slide 2 of 2

N O T E S

- Execution Group:
  - Each execution group is an operating system process or address space in which one or more message flows run. Each new execution group will result in an additional process or address space being run.
  - An execution group might typically require ~150MB of memory to start and load executables. Its size after initialisation will then depend on the requirements of the message flows which run within it. Each additional execution group which is allocated will need a minimum of another ~150MB so you can see that using a high number of execution groups is likely to use a large amount of memory quickly.
  - As well as the threads needed to run a message flow, an execution group also has a number of additional threads which are use to perform broker management functions. So allocating many execution groups will result in more management threads being allocated overall.
  - An execution group provides process level separation between different applications. This can be useful for separating applications.
- Additional Instance:
  - Each new additional instance results in one more thread being allocated in an existing execution group. As such the overhead is low. There will be some additional memory requirements but this will be significantly less than the ~150MB that is typically needed to accommodate a new execution group.
  - Message flows running as instances in a single execution group can access common variables called shared variables. This gives a cheap way to access low volumes of common data. The same is not true of messages flows assigned to different execution groups.
- When assigning message flows to execution groups there is no fixed algorithm imposed by Message Broker. There is significant flexibility in the way in which the assignment can be managed. Some users allocate high numbers of execution groups and assign a few message flows to each. We have seen several instance of 100+ execution groups being used. Others allocate many message flows to a small number of execution groups.
- A key factor in deciding how many execution groups to allocate is going to be how much free memory you have available on the broker machine.
- A common allocation pattern is to have one or possibly two execution groups per project and to then assign all of the message flows for that project to those execution groups. By using two execution groups you build in some additional availability should one execution fail. Where you have message flows which require large amounts of memory in order to execute, perhaps because the input messages are very large or have many elements to them, then you are recommended to keep all instances of those message flows to a small number of execution groups rather than allocate over a large number of execution groups [It is not uncommon for an execution group to use 1GB+ of memory.]. Otherwise every execution group could require a large amount of memory as soon as one of the large messages was processed by a flow in that execution group. Total memory usage would rise significantly in this case and could be many gigabytes.
- As with most things this is about achieving a balance between the flexibility you need against the level of resources (and so cost) which have to be assigned to achieve the required processing

Message Broker: Understanding your broker's behaviour.

Some tools to understand your broker's behaviour

- *PerfHarness* – Drive realistic loads through the broker
  - http://www.alphaworks.ibm.com/tech/perfharness

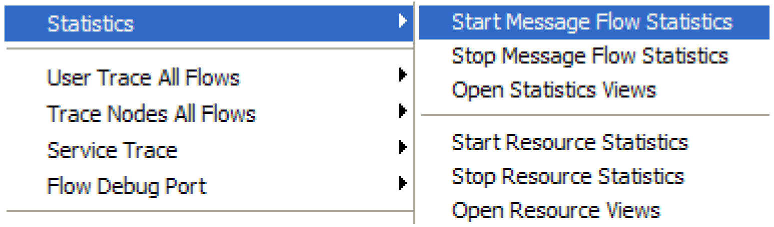
- OS Tools
  - Run your message flow under load and determine the limiting factor.
  - Is your message flow CPU, memory or I/O bound?
  - e.g. "perfmon" (Windows), "vmstat" or "top" (Linux/UNIX) or "SDSF" (z/OS).
  - This information will help you understand the likely impact of scaling (e.g. additional instances), faster storage and faster networks

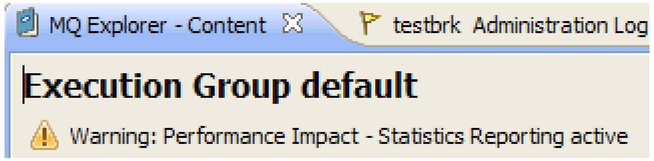
- Third Party Tools
  - *RFHUtil* – useful for sending/receiving MQ messages and customising all headers
  - *NetTool* – useful for testing HTTP/SOAP
  - *Filemon* – Windows tool to show which files are in use by which processes
  - *Java Health Center* – to diagnose issues in Java nodes

- MQ Explorer
  - Queue Manager administration
  - Useful to monitor queue depths during tests

- Message Broker Explorer
  - Understand what is running
  - Offload WS-Security processing onto XI50 appliance
  - performance and resource statistics

Broker performance statistics

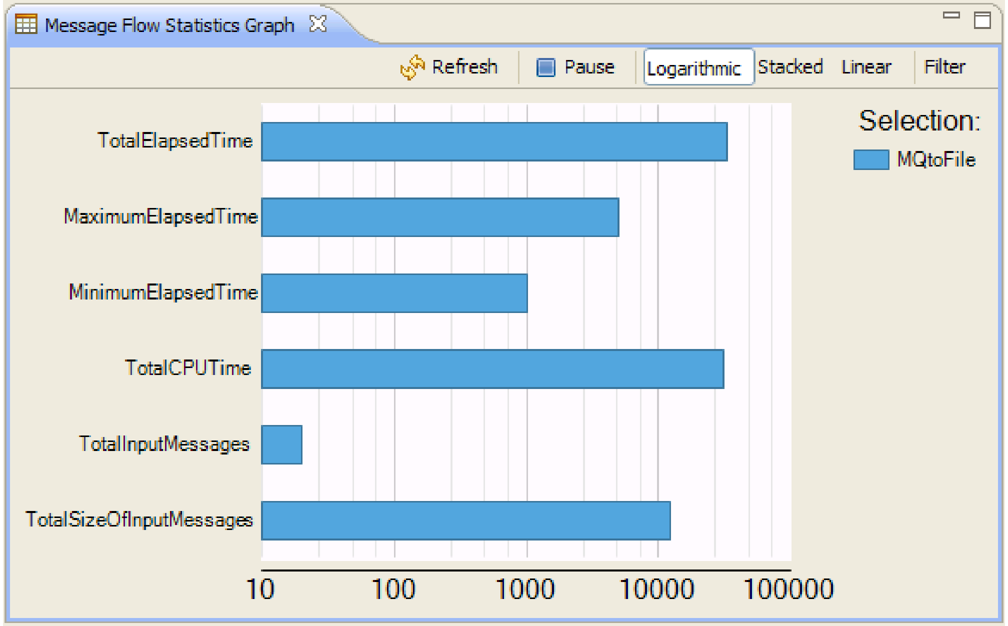


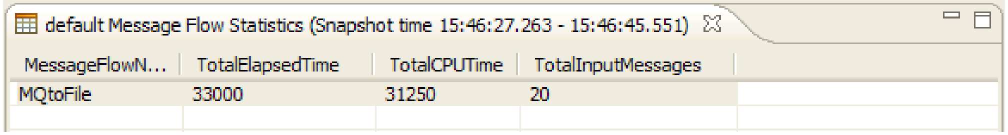


Execution Group default

Warning: Performance Impact - Statistics Reporting active

- The Message Broker Explorer enables you to start/stop message flow statistics on the broker, and view the output.
- New in V7 (although supportpac IS02 available for V6.1)
- Warnings are displayed advising there may be a performance impact (typically ~3%)

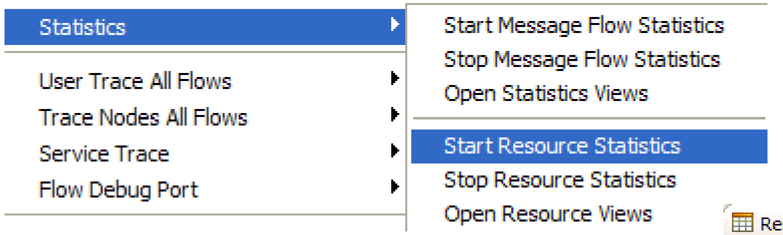


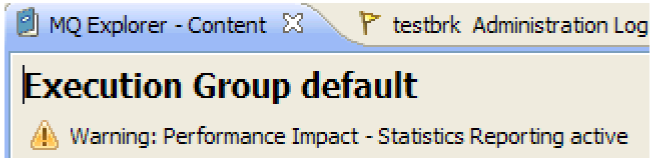


default Message Flow Statistics (Snapshot time 15:46:27.263 - 15:46:45.551)

MessageFlowN...	TotalElapsedTime	TotalCPUTime	TotalInputMessages
MQtoFile	33000	31250	20

Broker resource statistics

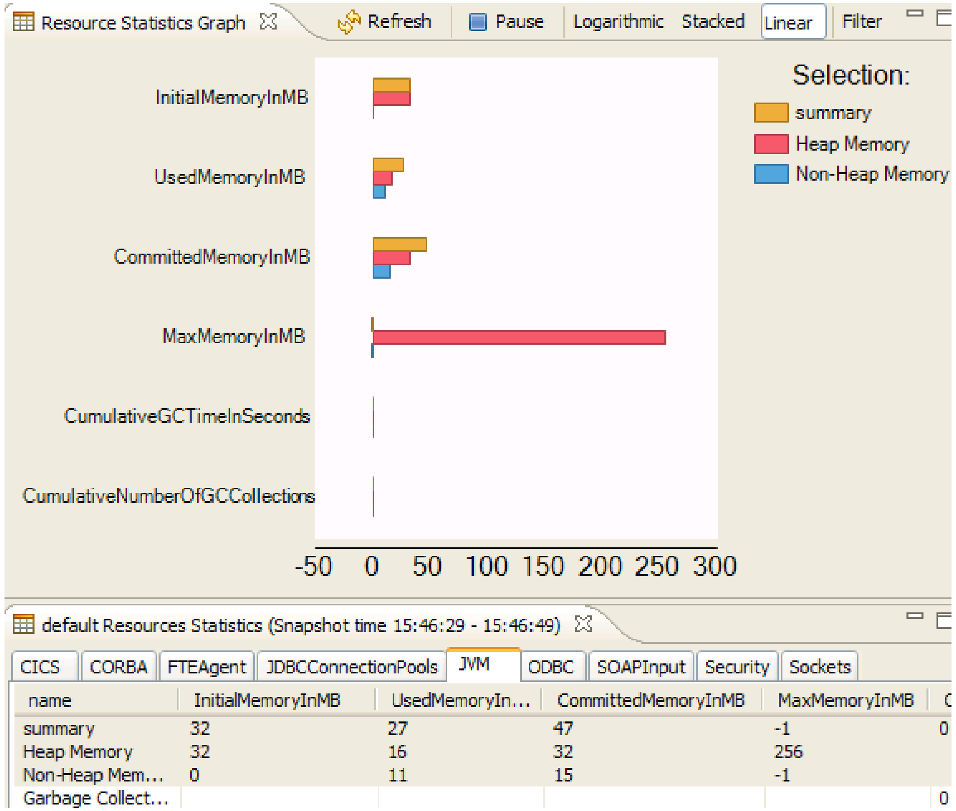




Execution Group default

Warning: Performance Impact - Statistics Reporting active

- The Message Broker Explorer enables you to start/stop resource statistics on the broker, and view the output.
- New in V7
- Warnings are displayed advising there may be a performance impact (typically ~1%)

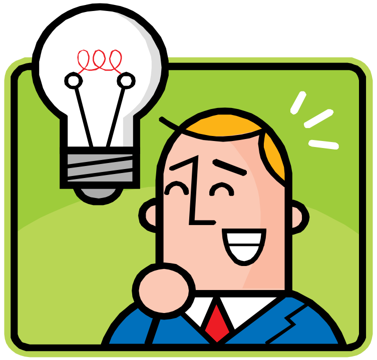


default Resources Statistics (Snapshot time 15:46:29 - 15:46:49)

name	InitialMemoryInMB	UsedMemoryIn...	CommittedMemoryInMB	MaxMemoryInMB	C
summary	32	27	47	-1	0
Heap Memory	32	16	32	256	
Non-Heap Mem...	0	11	15	-1	
Garbage Collect...					0

Testing and Optimising Message Flows

- Run driver application (e.g. *PerfHarness*) on a machine separate to broker machine
- Test flows in isolation
- Use real data where possible
- Avoid pre-loading queues
- Use a controlled environment – access, code, software, hardware
- Take a baseline before any changes are made
- Make ONE change at a time.
- Where possible retest after each change to measure impact
- Start with a single flow instance/EG and then increase to measure scaling and processing characteristics of your flows



Performance Reports

- See http://www-1.ibm.com/support/docview.wss?uid=swg27007150

- Lots of V6.1 reports.
- V7.0 AIX (IP6N), xLinux (IPL6) Windows 64bit (IP76)

- Consists of 2 Sections:
  - High level release highlights and use case throughput numbers
    - Use to see what new improvements there are
    - Use to see what kind of rates are achievable in common scenarios
    - Numbers in this section can be replicated in your environment using product samples and perfharness
  - Detailed low level metrics (nodes/parsers)
    - Use to compare parsers
    - Use to compare persistent vs non-persistent
    - Use to compare transports
    - Use to compare transformation options
    - Use to see scaling and overhead characteristics
    - Shows tuning and testing setup

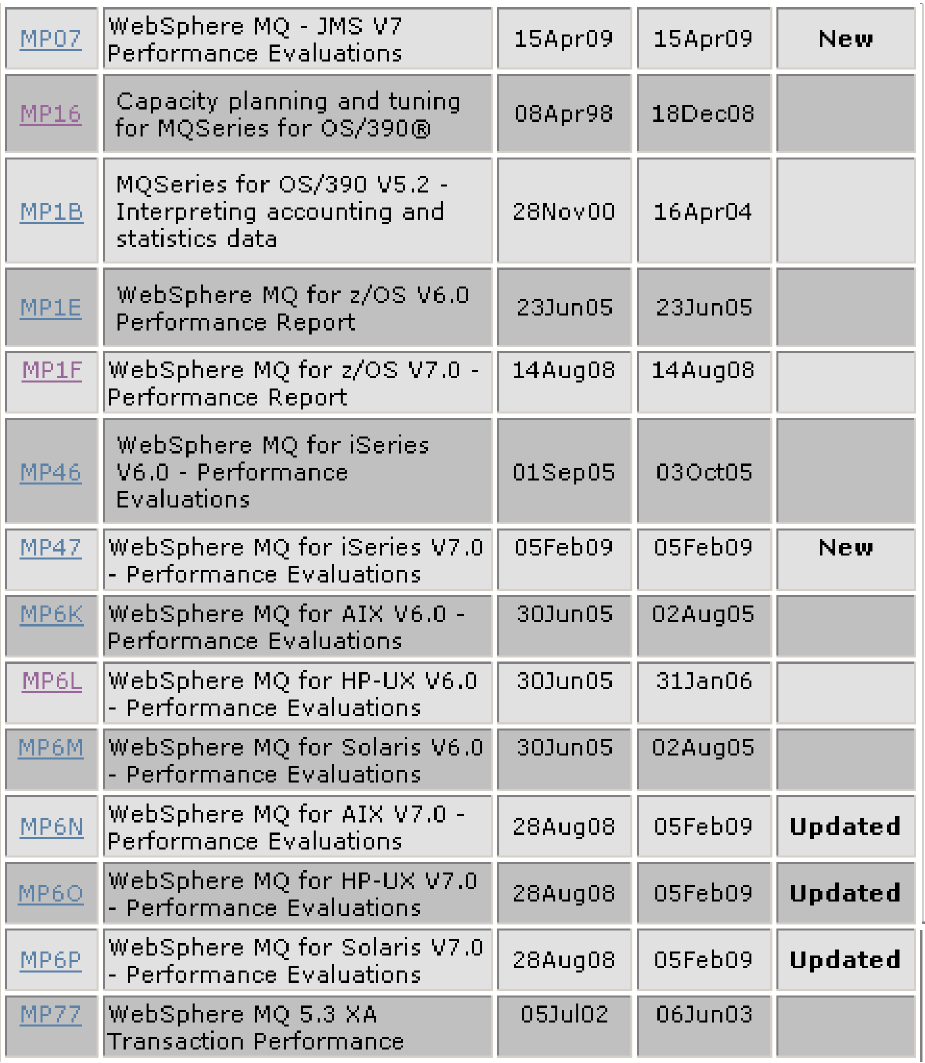
- Please send us your feedback!

Message Broker V7 – Performance highlights

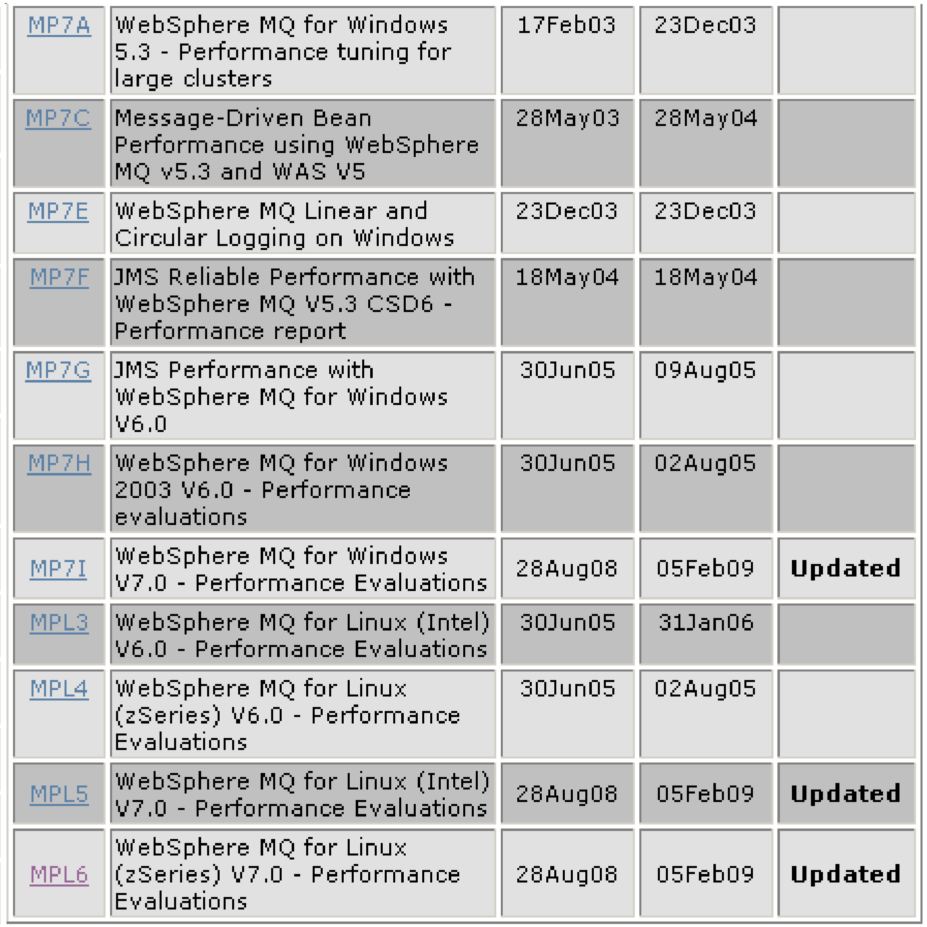
- Performance continues to be a key focus
- Overall performance increase of 5% over v6.1
- 64-bit on all platforms
- Removal of the Configuration Manager and User Name Server
- Significant improvement in deployment times, with v7 in some scenarios taking only 23% of the time taken in v6.1 to complete the same deployment
- Significant improvement in toolkit connect times, with v7 in some scenarios taking only 8% of the time taken in v6.1 to complete the same connection
- Business-level monitoring improvements of between 30% and 50% over v6.1.0.3
- Improvements to Java through optimised 1.6 JRE
- Smaller installation and base memory footprint

Summary

SupportPacs



MP07	WebSphere MQ - JMS V7 Performance Evaluations	15Apr09	15Apr09	**New**
MP16	Capacity planning and tuning for MQSeries for OS/390®	08Apr98	18Dec08	
MP1B	MQSeries for OS/390 V5.2 - Interpreting accounting and statistics data	28Nov00	16Apr04	
MP1E	WebSphere MQ for z/OS V6.0 Performance Report	23Jun05	23Jun05	
MP1F	WebSphere MQ for z/OS V7.0 - Performance Report	14Aug08	14Aug08	
MP46	WebSphere MQ for iSeries V6.0 - Performance Evaluations	01Sep05	03Oct05	
MP47	WebSphere MQ for iSeries V7.0 - Performance Evaluations	05Feb09	05Feb09	**New**
MP6K	WebSphere MQ for AIX V6.0 - Performance Evaluations	30Jun05	02Aug05	
MP6L	WebSphere MQ for HP-UX V6.0 - Performance Evaluations	30Jun05	31Jan06	
MP6M	WebSphere MQ for Solaris V6.0 - Performance Evaluations	30Jun05	02Aug05	
MP6N	WebSphere MQ for AIX V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**
MP6O	WebSphere MQ for HP-UX V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**
MP6P	WebSphere MQ for Solaris V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**
MP77	WebSphere MQ 5.3 XA Transaction Performance	05Jul02	06Jun03	



MP7A	WebSphere MQ for Windows 5.3 - Performance tuning for large clusters	17Feb03	23Dec03	
MP7C	Message-Driven Bean Performance using WebSphere MQ v5.3 and WAS V5	28May03	28May04	
MP7E	WebSphere MQ Linear and Circular Logging on Windows	23Dec03	23Dec03	
MP7F	JMS Reliable Performance with WebSphere MQ V5.3 CSD6 - Performance report	18May04	18May04	
MP7G	JMS Performance with WebSphere MQ for Windows V6.0	30Jun05	09Aug05	
MP7H	WebSphere MQ for Windows 2003 V6.0 - Performance evaluations	30Jun05	02Aug05	
MP7I	WebSphere MQ for Windows V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**
MPL3	WebSphere MQ for Linux (Intel) V6.0 - Performance Evaluations	30Jun05	31Jan06	
MPL4	WebSphere MQ for Linux (zSeries) V6.0 - Performance Evaluations	30Jun05	02Aug05	
MPL5	WebSphere MQ for Linux (Intel) V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**
MPL6	WebSphere MQ for Linux (zSeries) V7.0 - Performance Evaluations	28Aug08	05Feb09	**Updated**

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

Message Broker Articles

- All Performance Reports are available at
  - http://www.ibm.com/support/docview.wss?rs=171&uid=swg27007150&loc=en_US&cs=utf-8&lang=en

- developerWorks - WebSphere Message Broker Home Page
  - http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html

- Some developerWorks Articles
  - What's New in WebSphere Message Broker V7.0
  - Determining How Many Copies of a Message Flow Should Run Within WBI Message Broker V5
  - How to Estimate Message Throughput For an IBM WebSphere MQ Integrator V2.1 Message Flow
  - Optimising DB2 UDB for use with WebSphere Business Integration Message Broker V5
  - How to monitor system and WebSphere Business Integration Message Broker V5 resource use
  - The Value of WebSphere Message Broker V6 on z/OS
  - JMSTransport Nodes in WebSphere Message Broker V6
  - Connecting the JMS Transport Nodes for WebSphere Message Broker v6 to Popular JMS Providers
  - HTTP transport nodes in WebSphere Message Broker V6
  - Testing and troubleshooting message flows with WebSphere Message Broker Test Client

Copyright and Trademarks