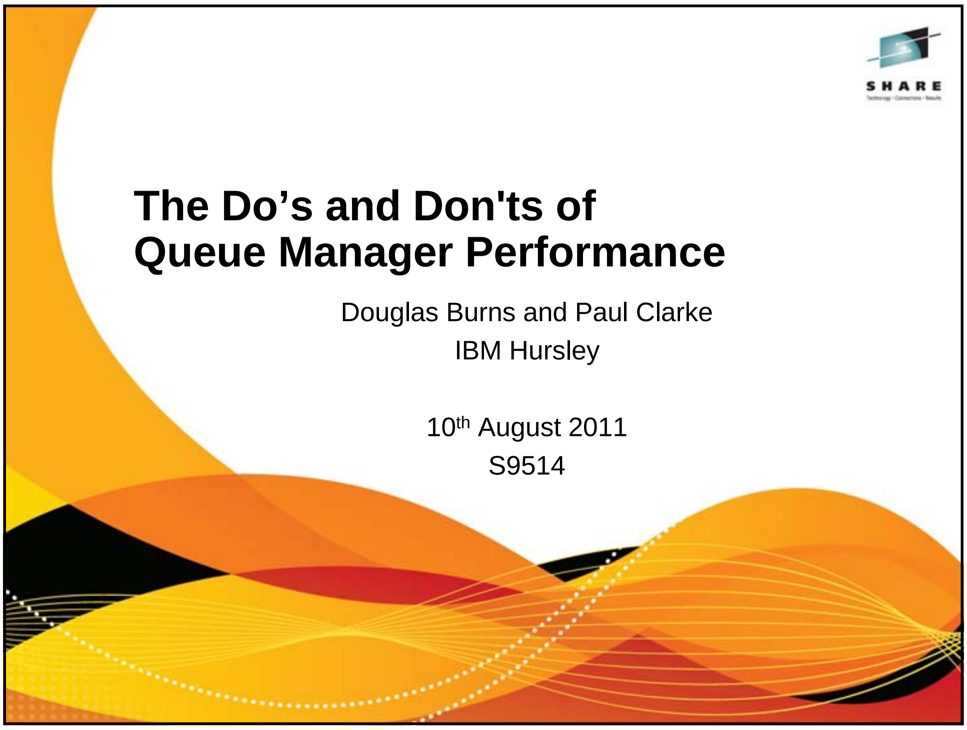


The Do’s and Don'ts of Queue Manager Performance

Douglas Burns and Paul Clarke
IBM Hursley

10th August 2011
S9514

Agenda

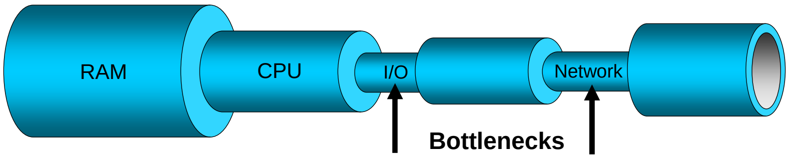
- Concepts
- Application Programming
- Channels
- Clients
- Distributed Specifics
- z/OS Specifics
- Summary

Agenda

NOTES

- 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



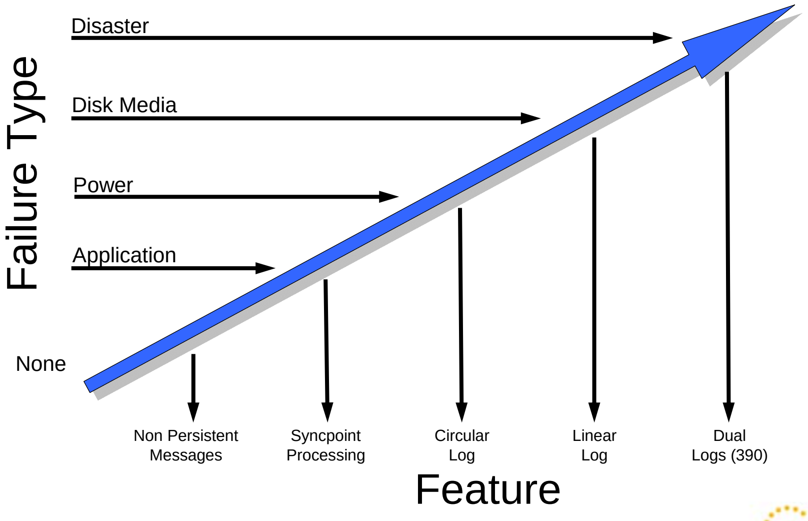
- Business Systems are complex
  - Often no single bottleneck limiting performance
  - Can be tricky to tune due to limited effect
  - Performance can mean different things to different people
  - Throughput
    - Scalability
    - Low resource usage
- Not only limited by physical resources
  - Application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

Performance Bottlenecks

NOTES

- 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



Failure Type

Disaster

Disk Media

Power

Application

None

Non Persistent Messages

Syncpoint Processing

Circular Log

Linear Log

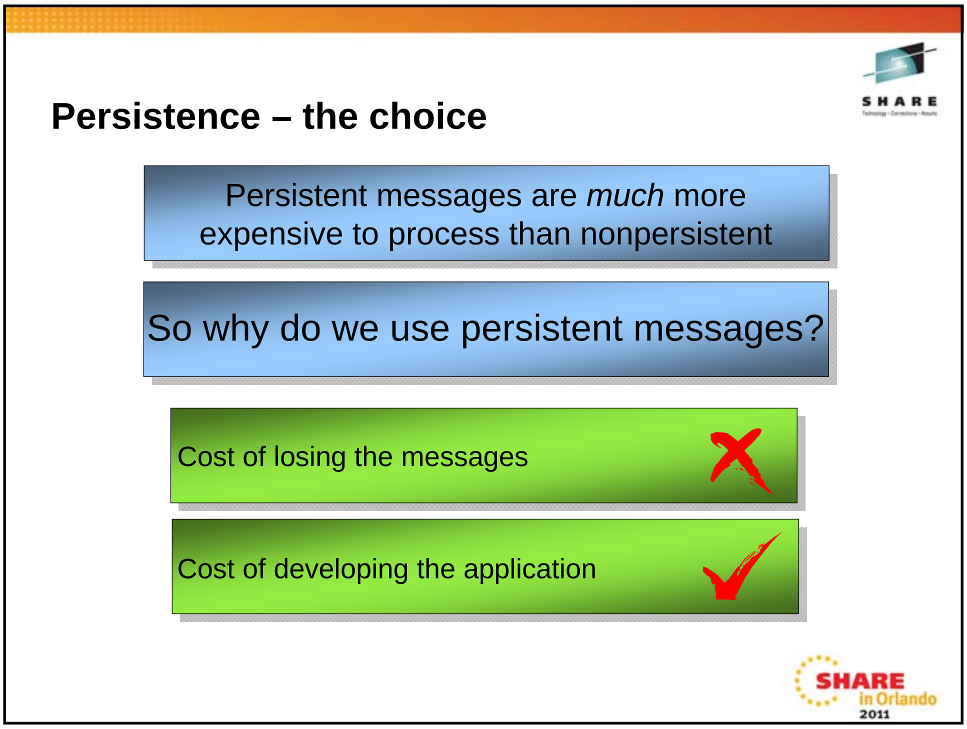
Dual Logs (390)

Feature

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

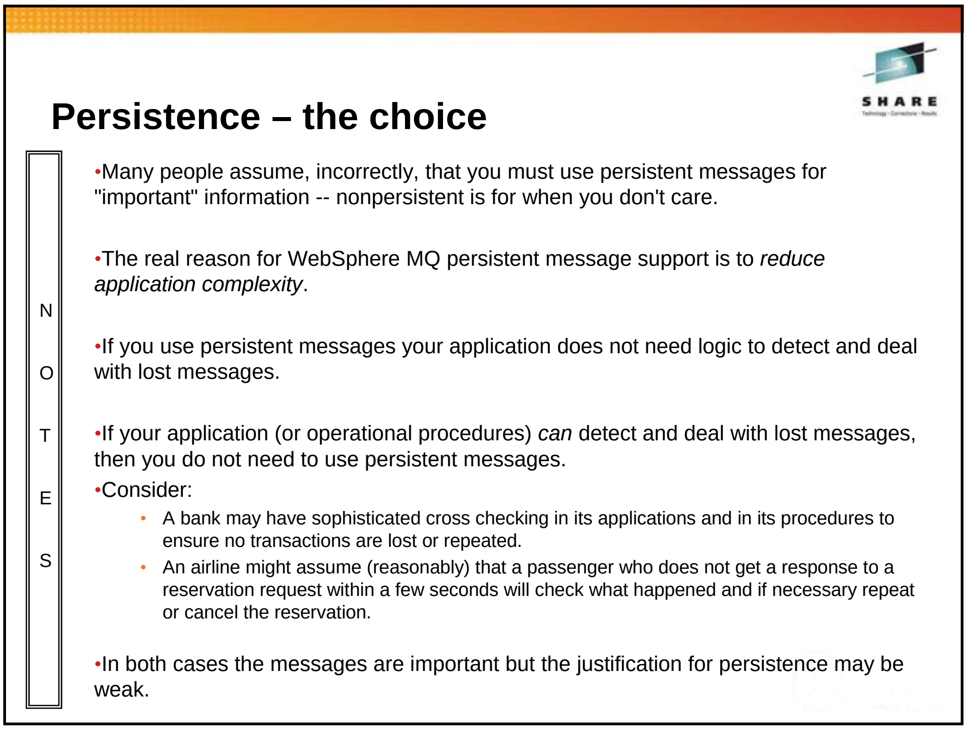
Persistent messages are *much* more expensive to process than nonpersistent

So why do we use persistent messages?

Cost of losing the messages ✗

Cost of developing the application ✓

SHARE in Orlando 2011



Persistence – the choice

N O T E S

- 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*.

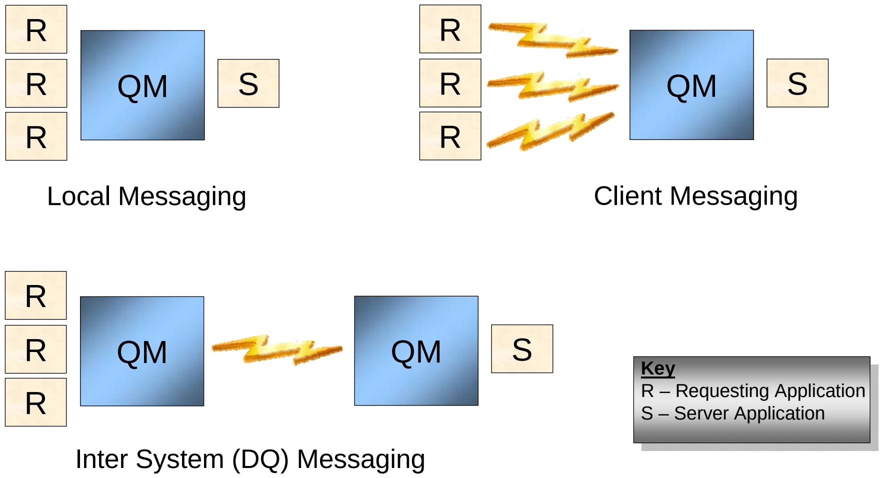
- If you use persistent messages your application does not need logic to detect and deal with lost messages.

- 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



Key
R – Requesting Application
S – Server Application

Network Configuration
Location Determines Cost

NOTES

- 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?

SHARE in Orlando 2011

Connecting and disconnecting

N O T E S

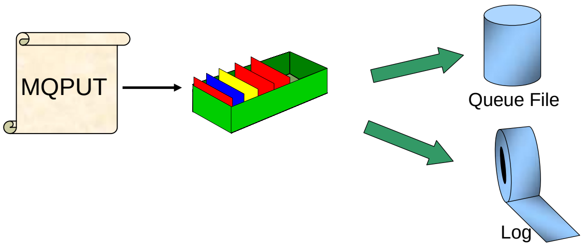
- MQCONN is a very heavyweight operation. Doing lots of these calls could cause throughput to suffer. Make sure that you don’t connect and disconnect a lot in your application, rather, connect once and then use this connection for all subsequent operations. Think carefully about any encapsulation you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

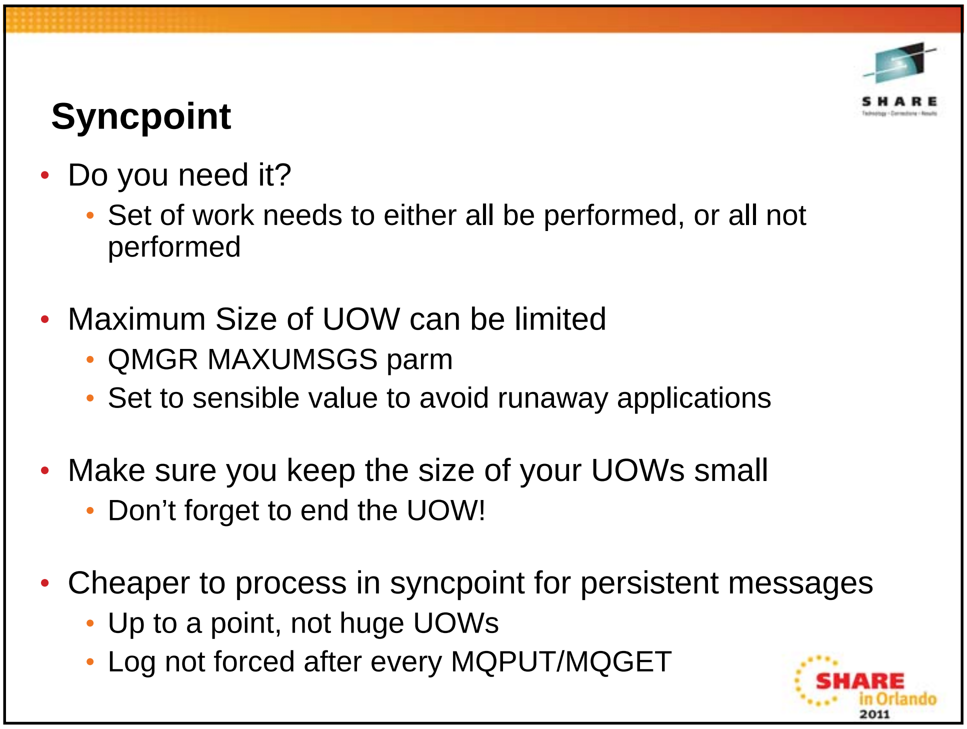


Persistence

N O T E S

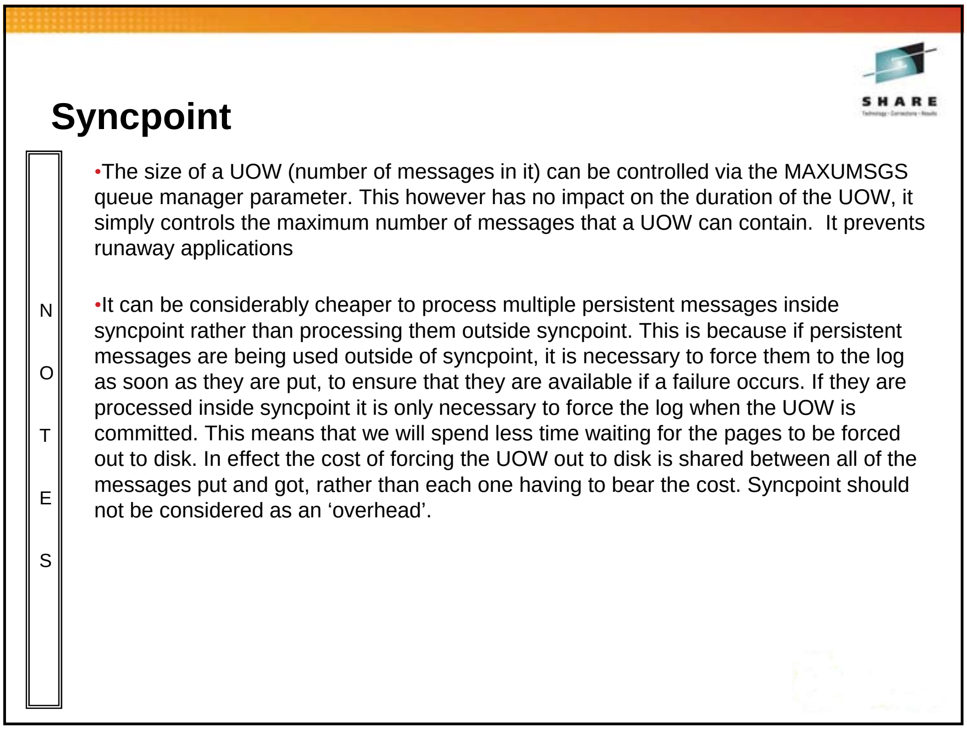
•If persistent messages are used then the maximum rate that messages can be put is typically going to be limited by the logging bandwidth available. This is normally the over riding factor as to the throughput available when using persistent messages.

•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

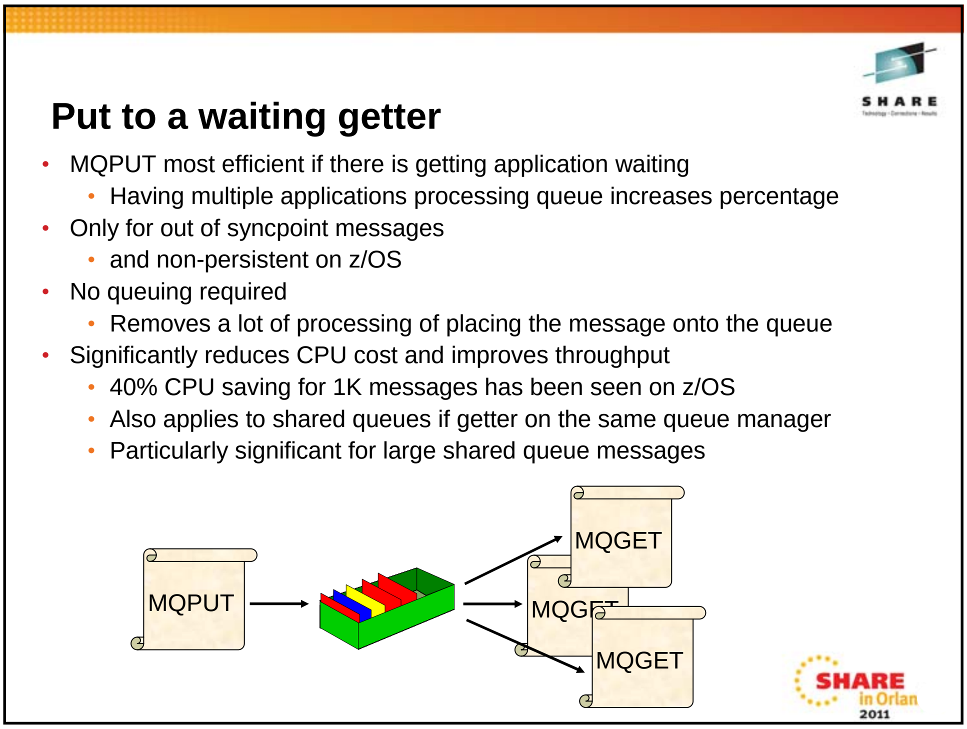


Syncpoint

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

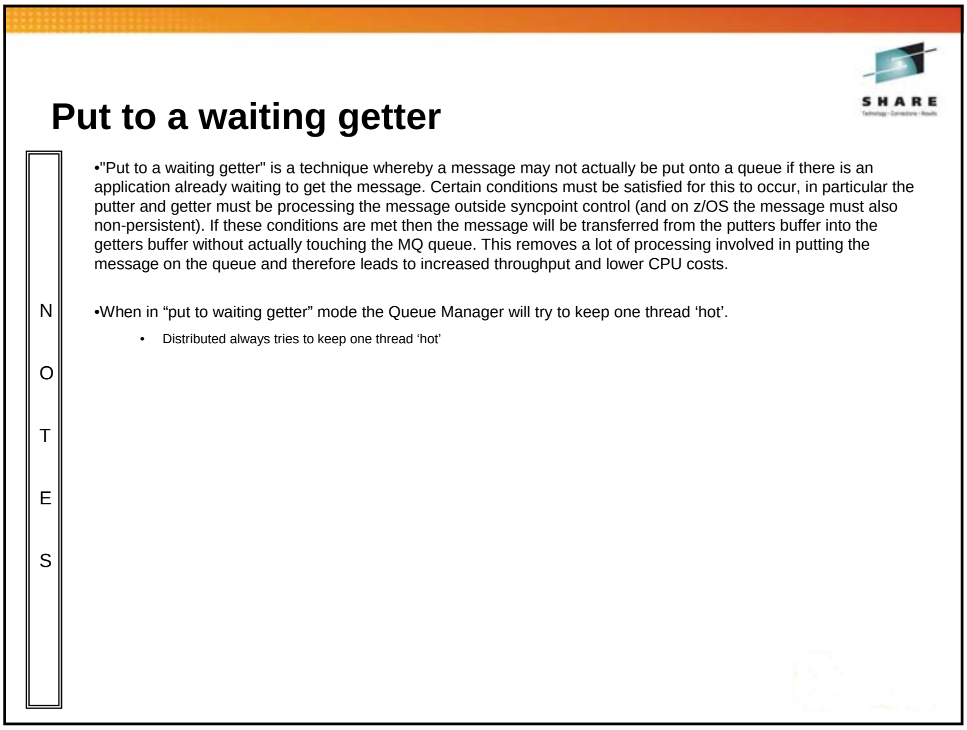
•It can be considerably cheaper to process multiple persistent messages inside syncpoint rather than processing them outside syncpoint. This is because if persistent messages are being used outside of syncpoint, it is necessary to force them to the log as soon as they are put, to ensure that they are 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

MQPUT
MQGET
MQGET
MQGET



Put to a waiting getter

NOTES

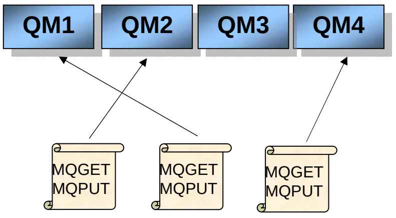
•"Put to a waiting getter" is a technique whereby a message may not actually be put onto a queue if there is an application already waiting to get the message. Certain conditions must be satisfied for this to occur, in particular the putter and getter must be processing the message outside syncpoint 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.

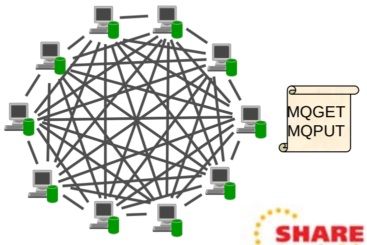
•When in “put to waiting getter” mode the Queue Manager will try to keep one thread ‘hot’.

- 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





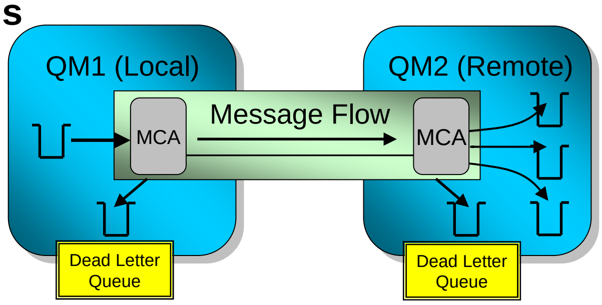
Workload balancing

NOTES

- 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 have 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 chosen, or alternatively the distribution could be carried out using a network spraying technique to spread the connections amongst available queue managers.
- Once connected 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

N O T E S

- The channel operation conforms to a quite simple model:

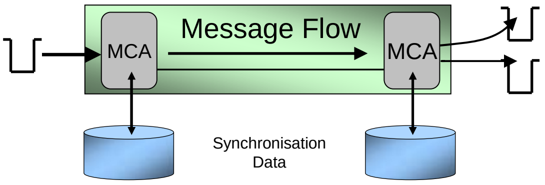
```
    Do until batchsize or (no more messages and batchint) expired
          Local MCA gets a message from the transmission queue
          A header is put on the data and sent using APPC, TCP etc.
       End
       Harden the message ids/indoubt flag
       Send "End of batch flag"
       Remote 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



- 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

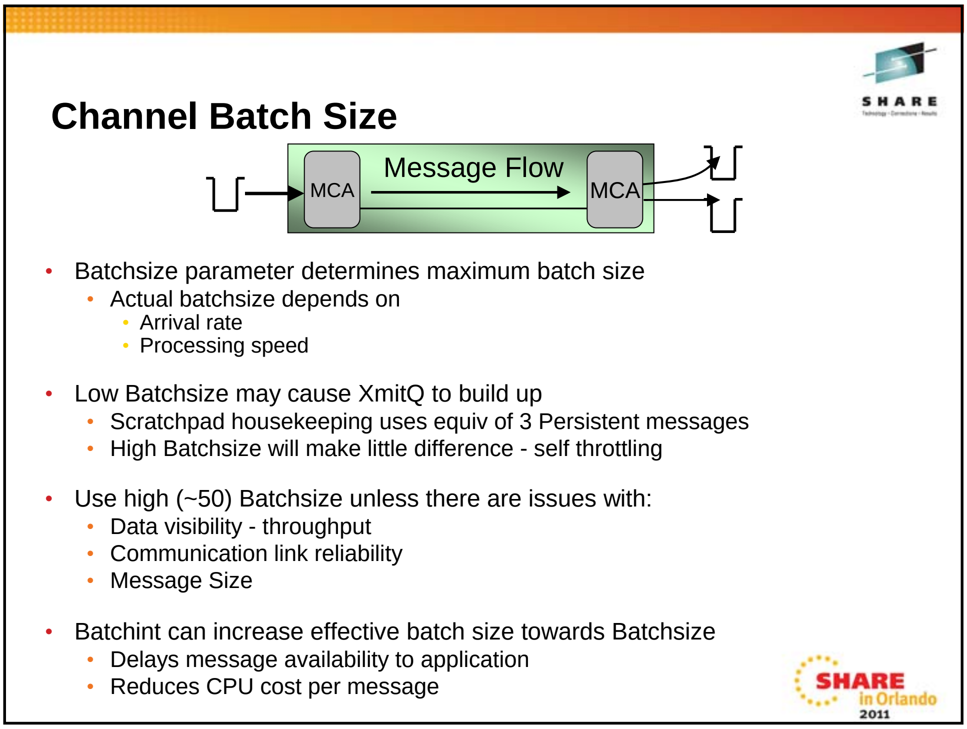
N O T E S

- 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.

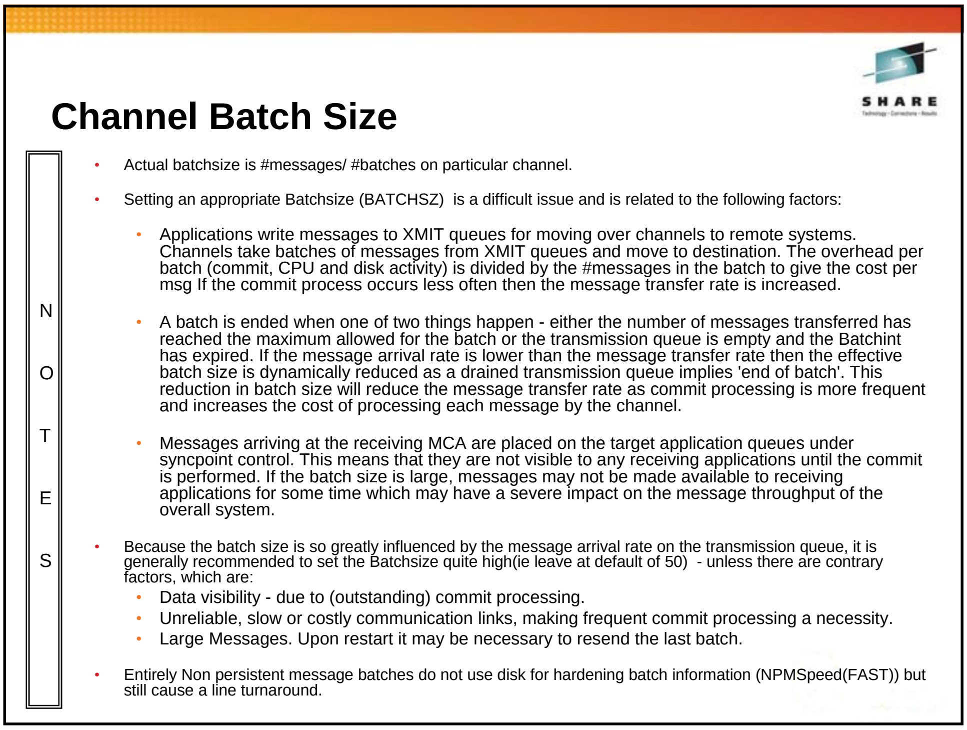
- Non-persistent messages may overtake Persistent messages. Persistent messages are got and put within syncpoint. They are only visible at the receiver when the MQCMIT completes. Non-persistent messages are got and put outside of syncpoint. They are visible immediately they arrive at the receiver.



Channel Batch Size

MCA
Message Flow
MCA

- 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

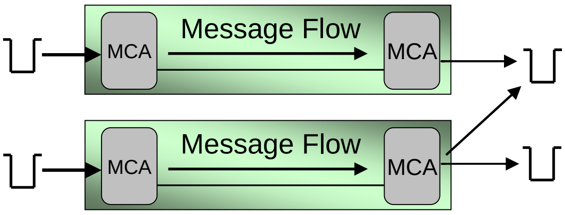


Channel Batch Size

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

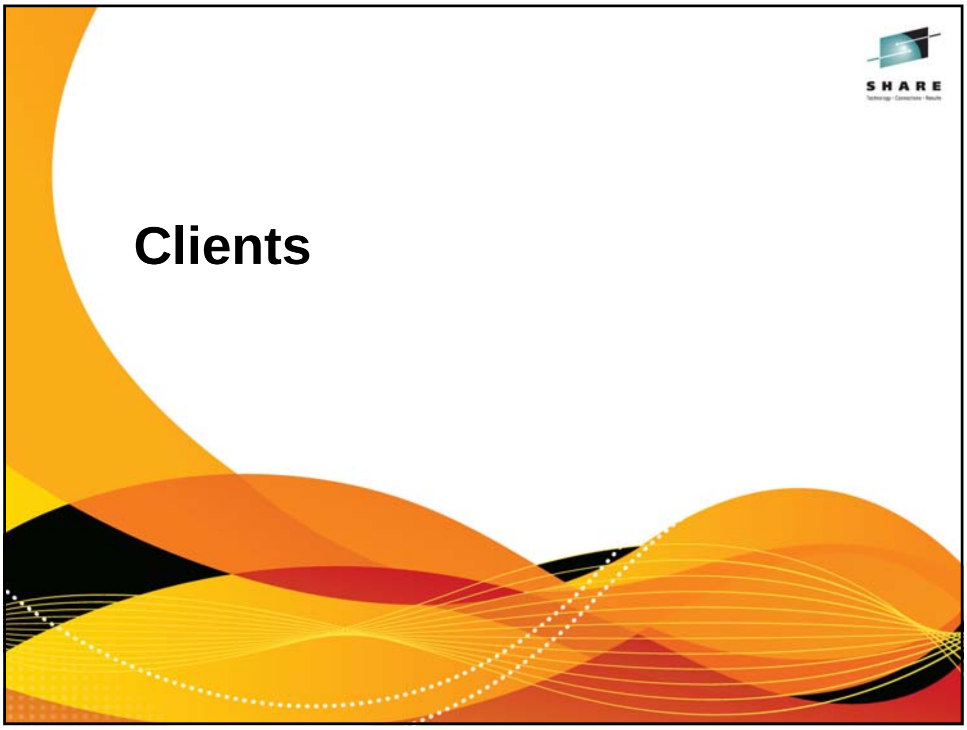


- Use one channel if it can handle the throughput
  - Monitor depth of XMIT queue
  - One high-use channel is more efficient than two low-use channels
    - The actual batch size will be larger
  - Multiple channels can yield some performance benefit
    - Depends on network and arrival rate
- Multiple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

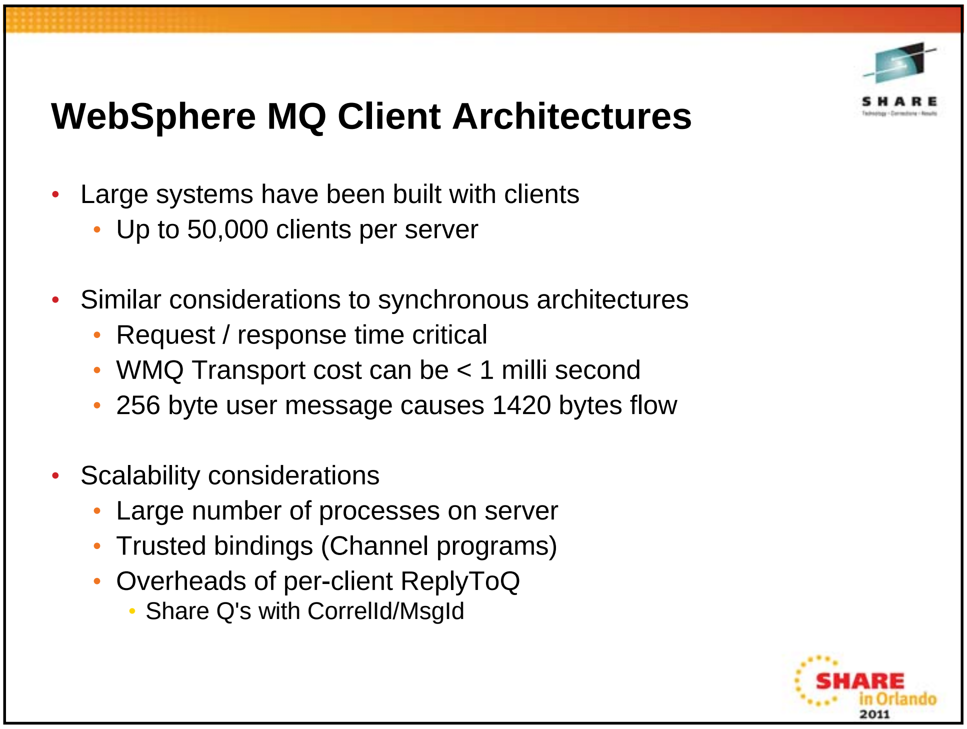
One or Multiple Channels

NOTES

- 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.

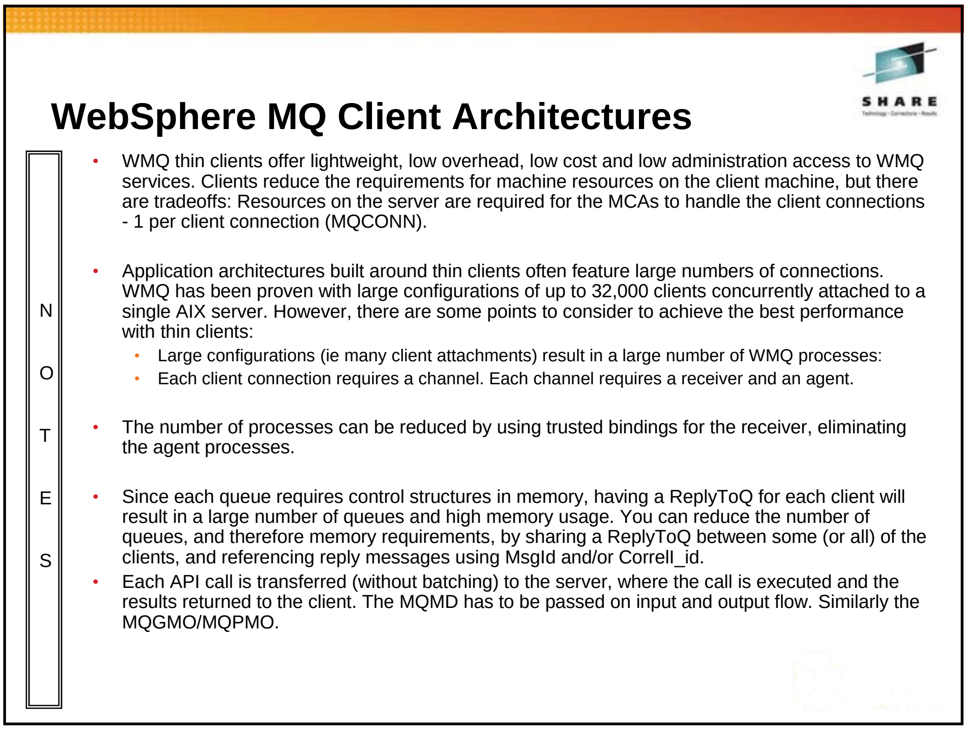


Clients



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

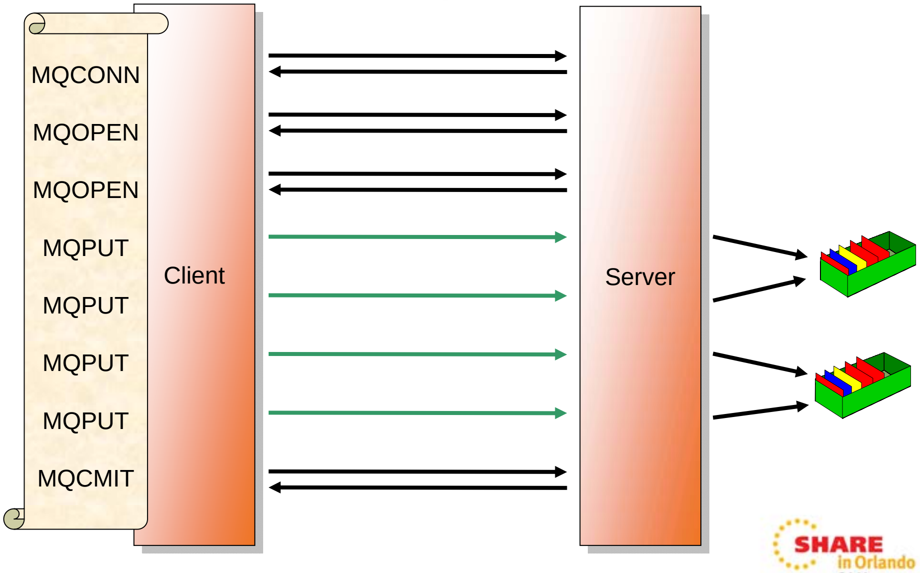


WebSphere MQ Client Architectures

NOTES

- 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



Asynchronous Put Response

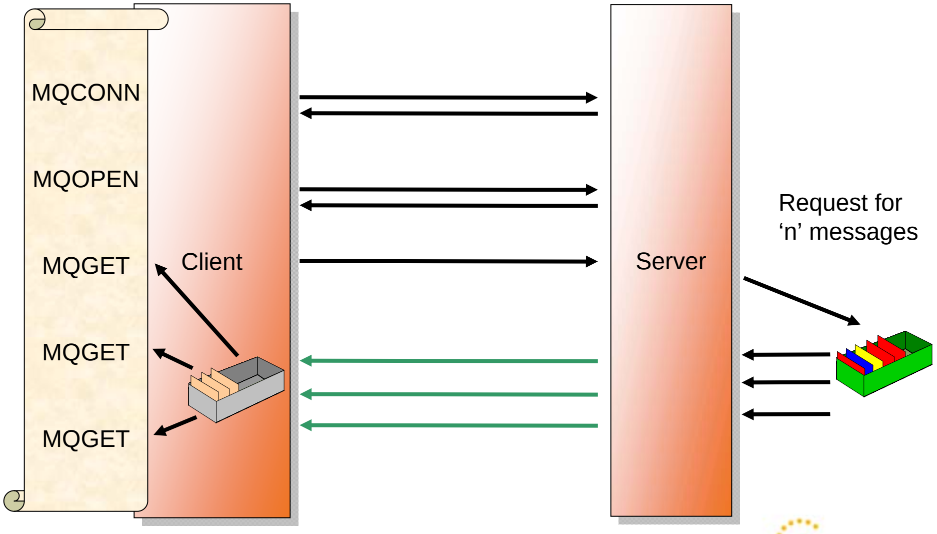
NOTES

- 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



Read-ahead of messages

NOTES

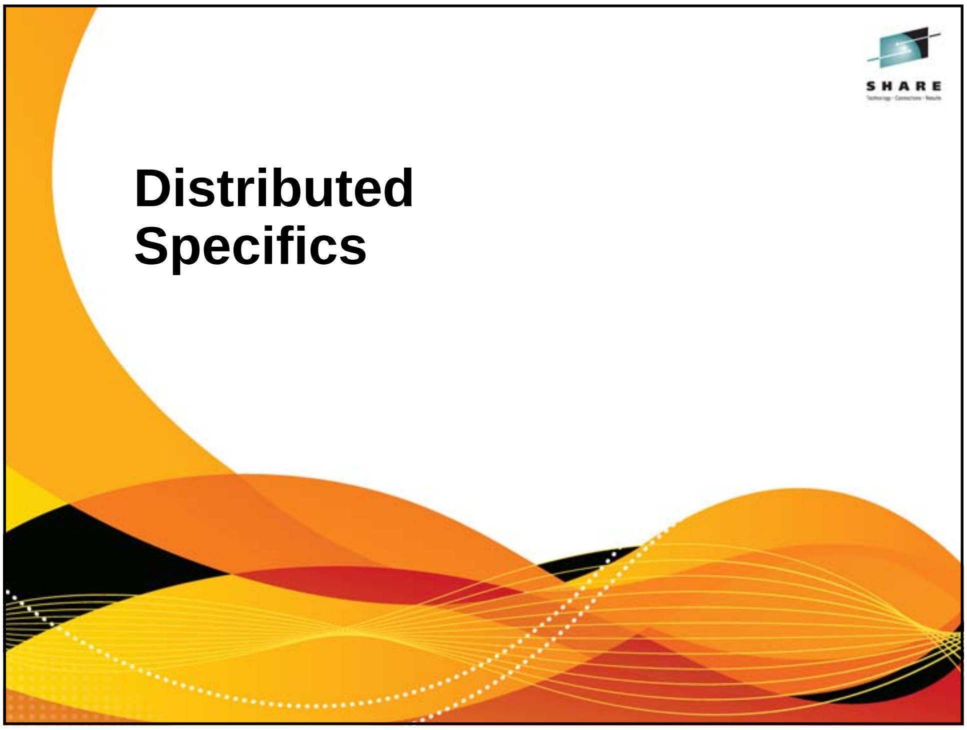
- Read Ahead (also known as 'Streaming') is a recognition of the fact that a large proportion of the cost of an MQGET from a client is the line turnaround of the network connection. When using Read Ahead the MQ client code makes a request for more than one message from the server. The server will send 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.

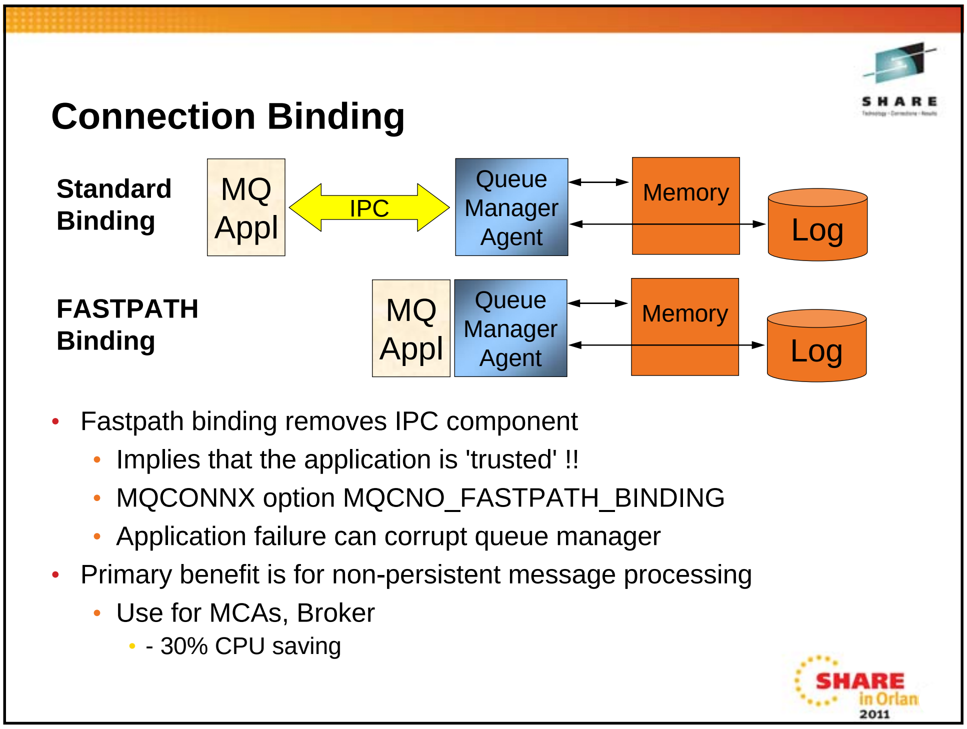
- If an application requests read ahead but the messages are not suitable, for example, they are all persistent then only one message will be sent to the client at any one time. Read ahead is effectively turned off until a sequence of non-persistent messages are on the queue again.

- 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.

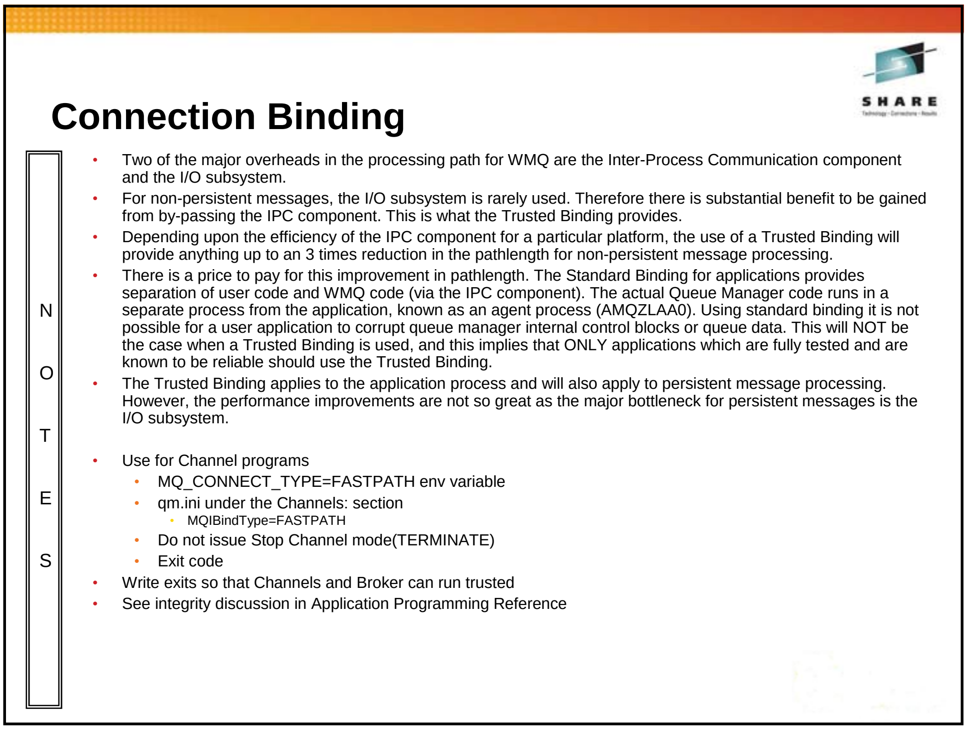


Distributed Specifics



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

NOTES

- 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

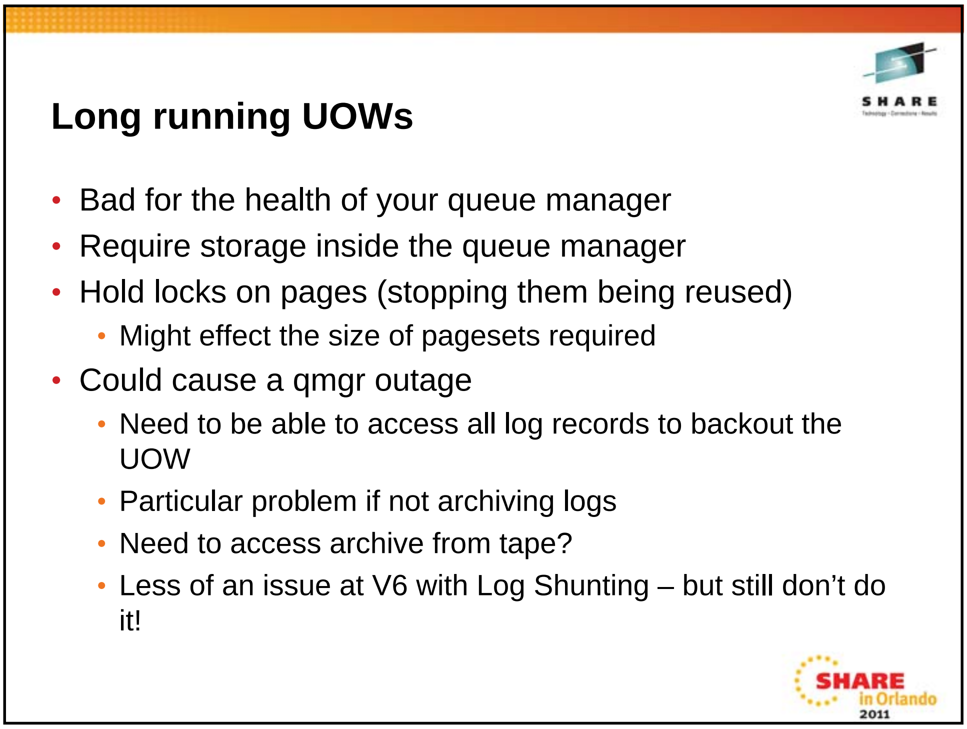
- Predefined Queues and Dynamic Queues
- Share between small number of applications
  - Specific Queue can only have one update outside syncpoint per Log I/O
  - Multiple updates inside syncpoint per Log I/O
  - Try not to have reply queue each if large number of clients
- Single Q on 4 way system
  - 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

NOTES

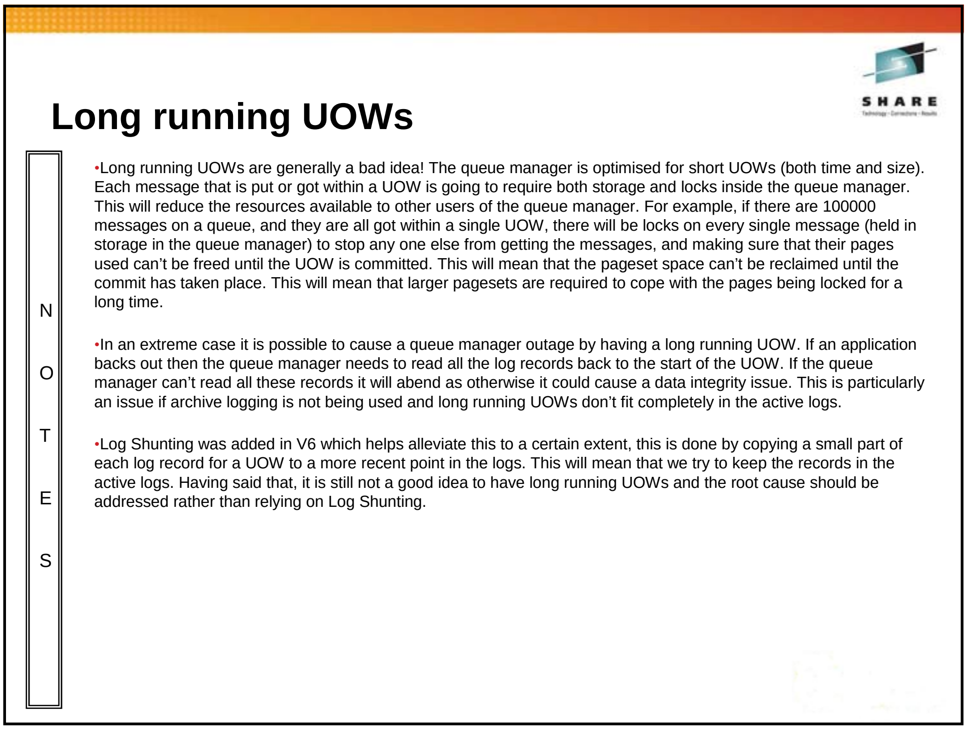
- An empty queue needs 12 pages from the FREEPAGE list to build an infrastructure to contain non persistent buffer, persistent buffer, SPACE map about all messages in the queue and a table of recently accessed messages.
- The creation and deletion of this infrastructure is expensive and any messages 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!



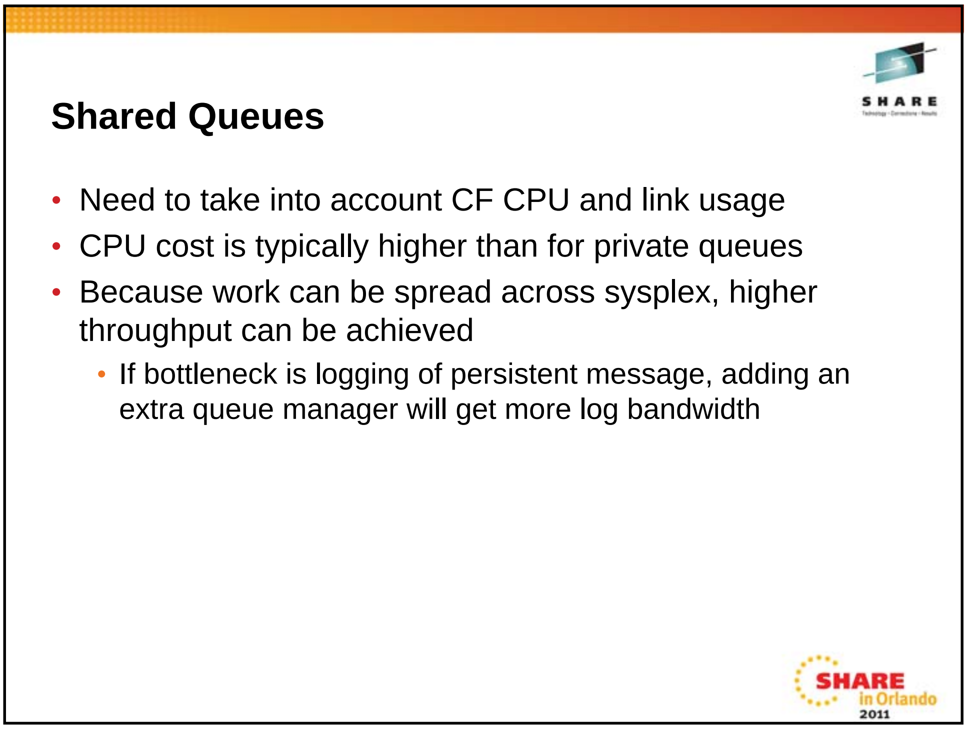
Long running UOWs

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.

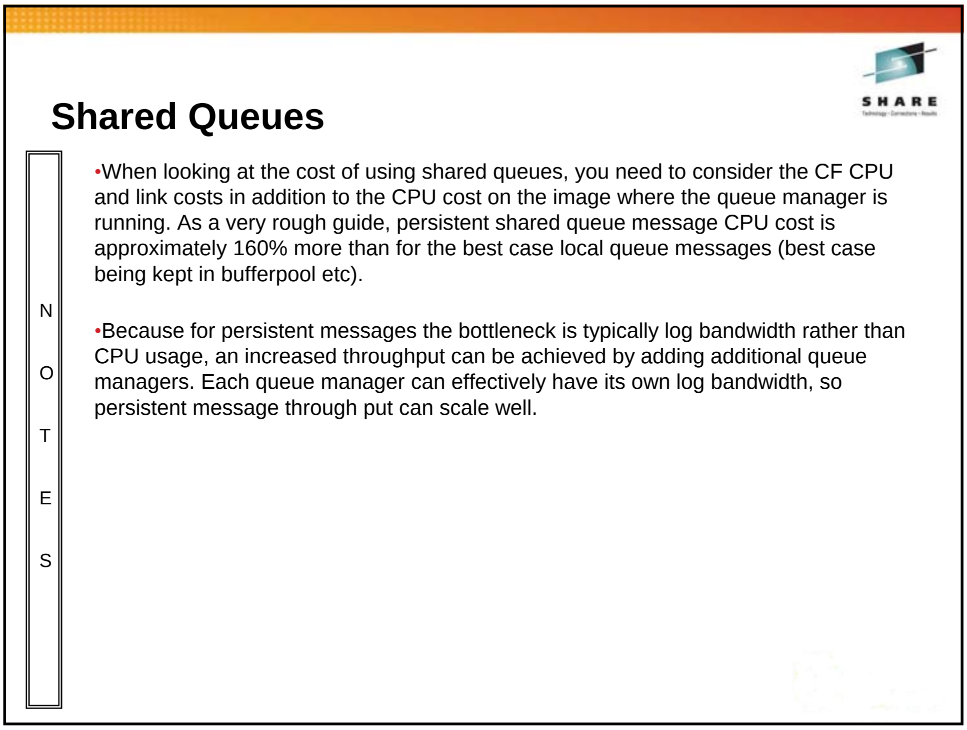
•In an extreme case it is possible to cause a queue manager outage by having a long running UOW. If an application backs out then the queue manager needs to read all the log records back to the start of the UOW. If the queue manager can't read all these records it will abend as otherwise it could cause a data integrity issue. This is particularly an issue if 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

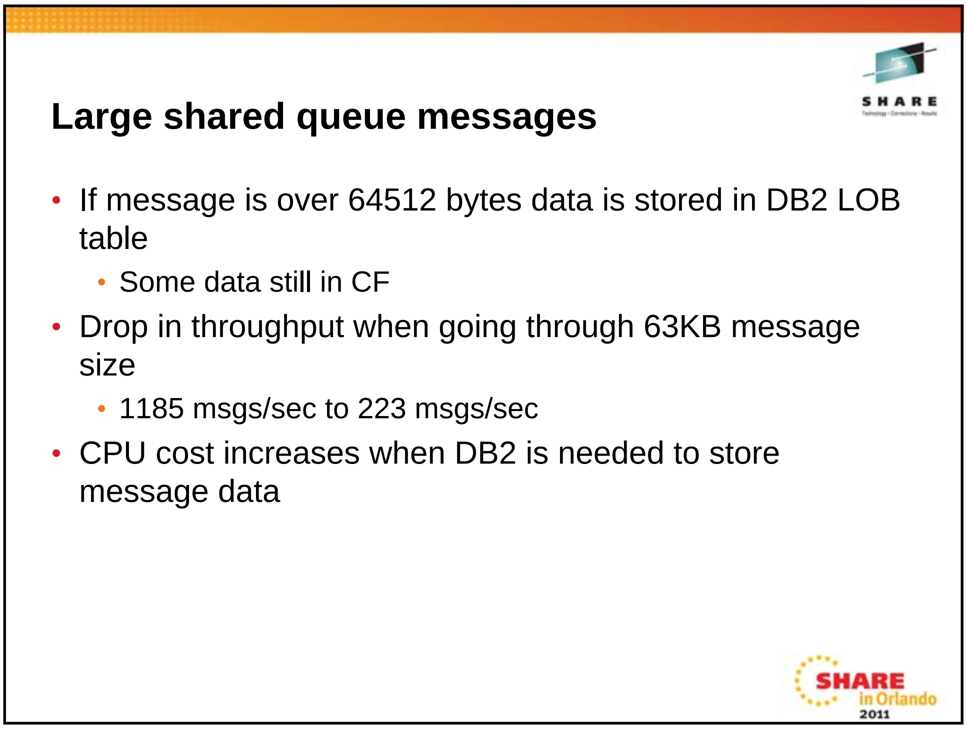


Shared Queues

NOTES

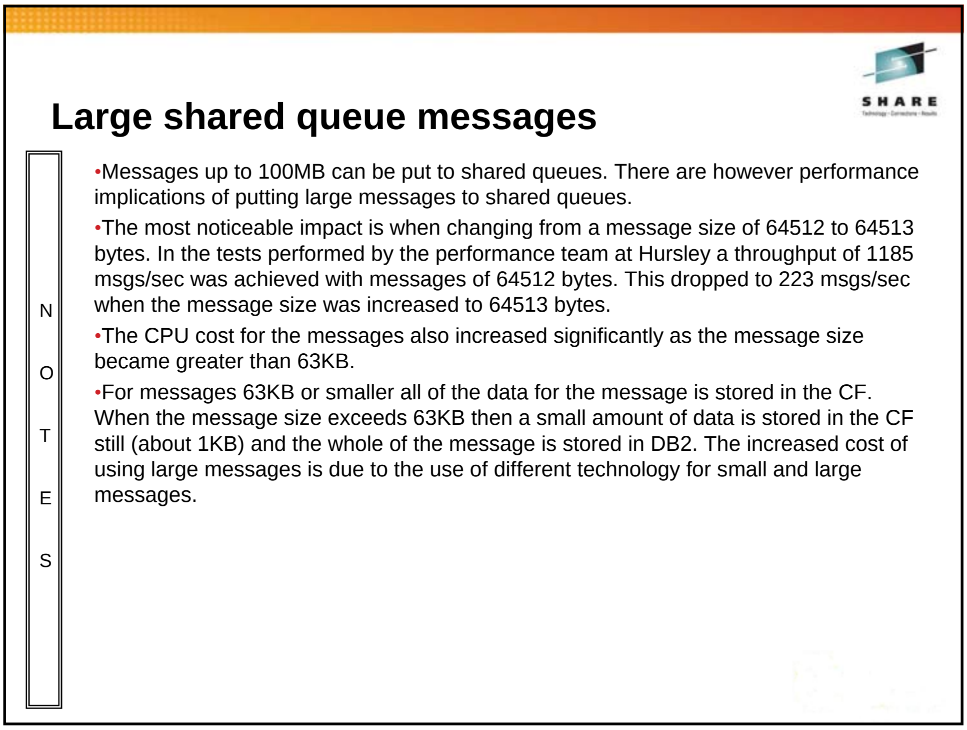
- When looking at the cost of using shared queues, you need to consider the CF CPU and link costs in addition to the CPU cost on the image where the queue manager is running. As a very rough guide, persistent shared queue message CPU cost is approximately 160% more than for the best case local queue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

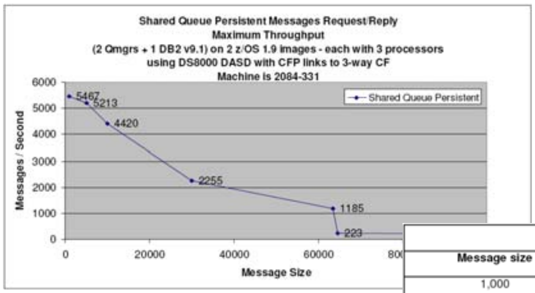


Large shared queue messages

NOTES

- 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



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

SHARE in Orlando 2011

Large shared queue messages

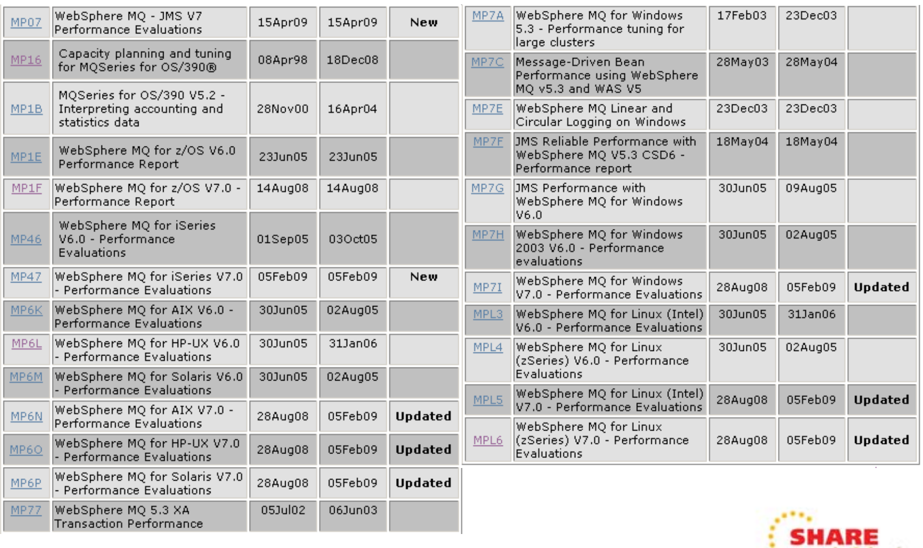
N O T E S

- 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".

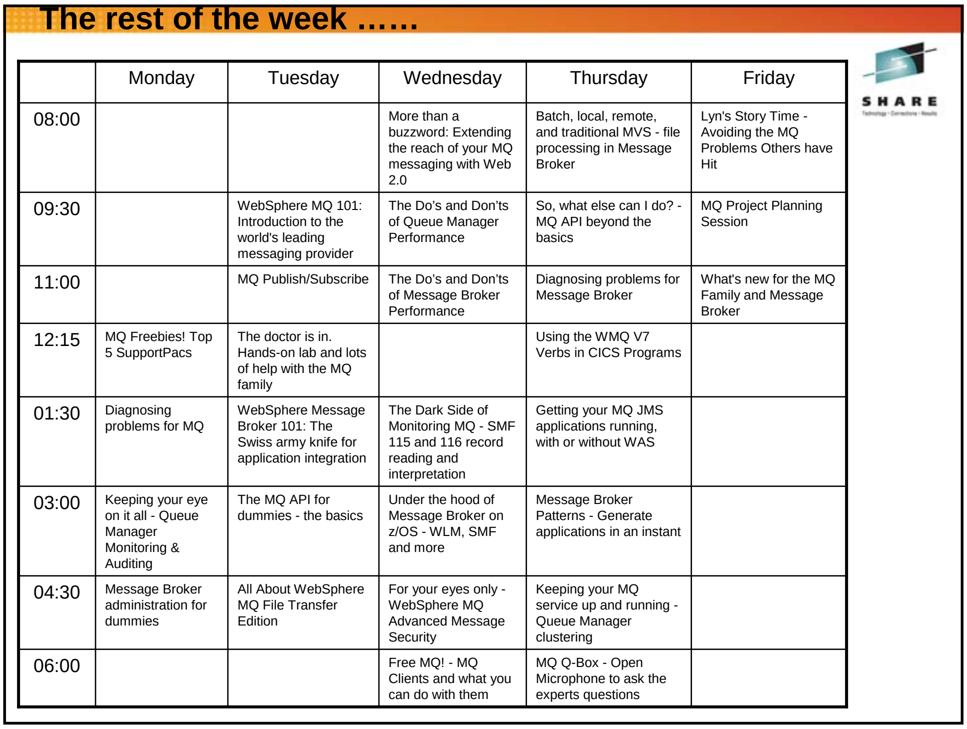
Summary

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

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**



The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00			More than a buzzword: Extending the reach of your MQ messaging with Web 2.0	Batch, local, remote, and traditional MVS - file processing in Message Broker	Lyn's Story Time - Avoiding the MQ Problems Others have Hit
09:30		WebSphere MQ 101: Introduction to the world's leading messaging provider	The Do's and Don'ts of Queue Manager Performance	So, what else can I do? - MQ API beyond the basics	MQ Project Planning Session
11:00		MQ Publish/Subscribe	The Do's and Don'ts of Message Broker Performance	Diagnosing problems for Message Broker	What's new for the MQ Family and Message Broker
12:15	MQ Freebies! Top 5 SupportPacs	The doctor is in. Hands-on lab and lots of help with the MQ family		Using the WMQ V7 Verbs in CICS Programs	
01:30	Diagnosing problems for MQ	WebSphere Message Broker 101: The Swiss army knife for application integration	The Dark Side of Monitoring MQ - SMF 115 and 116 record reading and interpretation	Getting your MQ JMS applications running, with or without WAS	
03:00	Keeping your eye on it all - Queue Manager Monitoring & Auditing	The MQ API for dummies - the basics	Under the hood of Message Broker on z/OS - WLM, SMF and more	Message Broker Patterns - Generate applications in an instant	
04:30	Message Broker administration for dummies	All About WebSphere MQ File Transfer Edition	For your eyes only - WebSphere MQ Advanced Message Security	Keeping your MQ service up and running - Queue Manager clustering	
06:00			Free MQ! - MQ Clients and what you can do with them	MQ Q-Box - Open Microphone to ask the experts questions	