#SHAREorg

SHARE
Technology · Connections · Results

MQ Performance and Tuning on Distributed - Including Internals

Craig Both (bothcr@uk.ibm.com)
IBM UK, Hursley

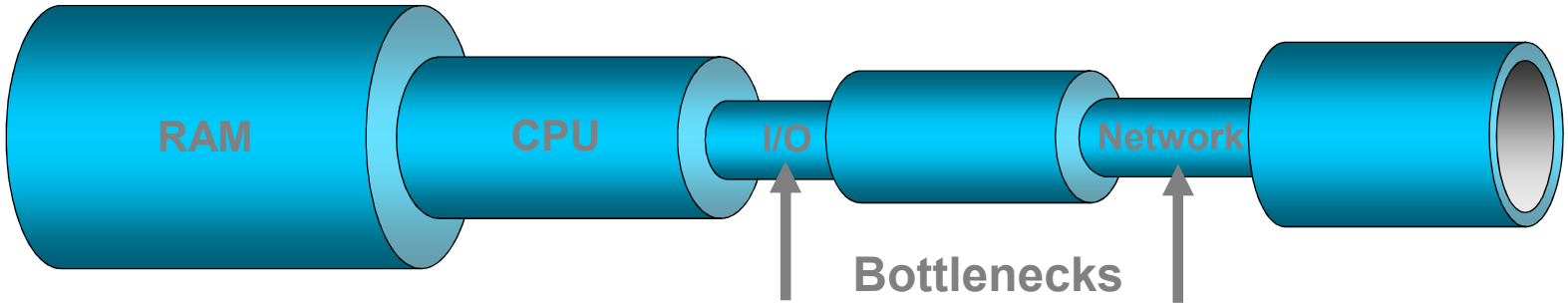
9th August 2012
11863

SHARE in Anaheim 2012

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

Performance Bottlenecks



- Business Systems are complex
  - Often no single bottleneck limiting performance
  - 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 Reports (from SupportPac site) show range of scenarios

Notes: 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.

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

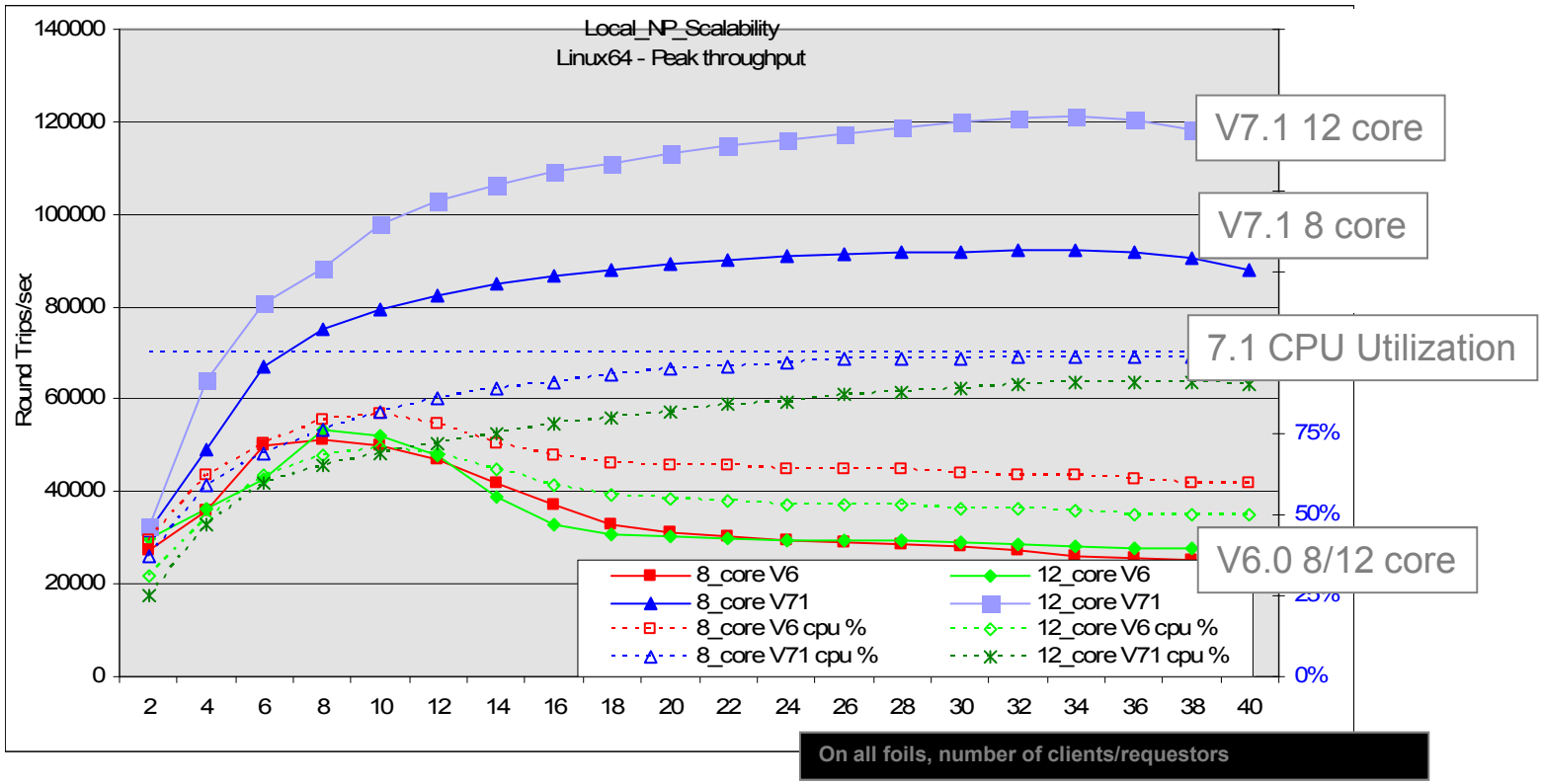
CPU Scaling

- The local non persistent tests really try to show what a qmgr is technically capable of
  - No disk i/o for logging, No waits for network traffic / bandwidth
    - Not a real world scenario but more a proof of technology
  - Single queue, so lots of queue contention

- What we find is pre-7.1, multiple cores may show little benefit and can make performance worse
  - As the waiting chain size increases so does the cost of managing it
  - This test is single queue – lots of queue contention

- 7.1 made many improvements to the internal locking
  - How they were used, types and scope of locks, less global locks
  - Handles contention considerably better

Scaling 7.1 (Linux, NP, 2Kb messages)



V7.1 can now make use of more cores than v6(or v7) could by handling the queue contention better

Notes: Scaling Chart

N O T E S

- Whilst the test itself is limited by the queue contention caused by the single queue being used its worth bearing in mind there are many places in the MQ system where there is contention like this, such as cluster and channel transmit queues.The real benefits come out once the system is heavily loaded to the point where contention starts to occur
- In tests on specific hardware, we were finding 12 CPUS performed worse than 8 for pre-7.1, but on the same hardware there was incremental benefit of adding cores up to about 20 cores.
- Results vary by platform but the locking changes shown here show benefit any time a lock is taken in the product and account for a considerable portion of the performance improvements that 7.1 shows.
- The CPU use is considerably higher to handle the larger messages being sent around in the system. As more CPUs are involved, the overall CPU use drops because a portion of the time is used to manage the context switching and lock chain management.

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

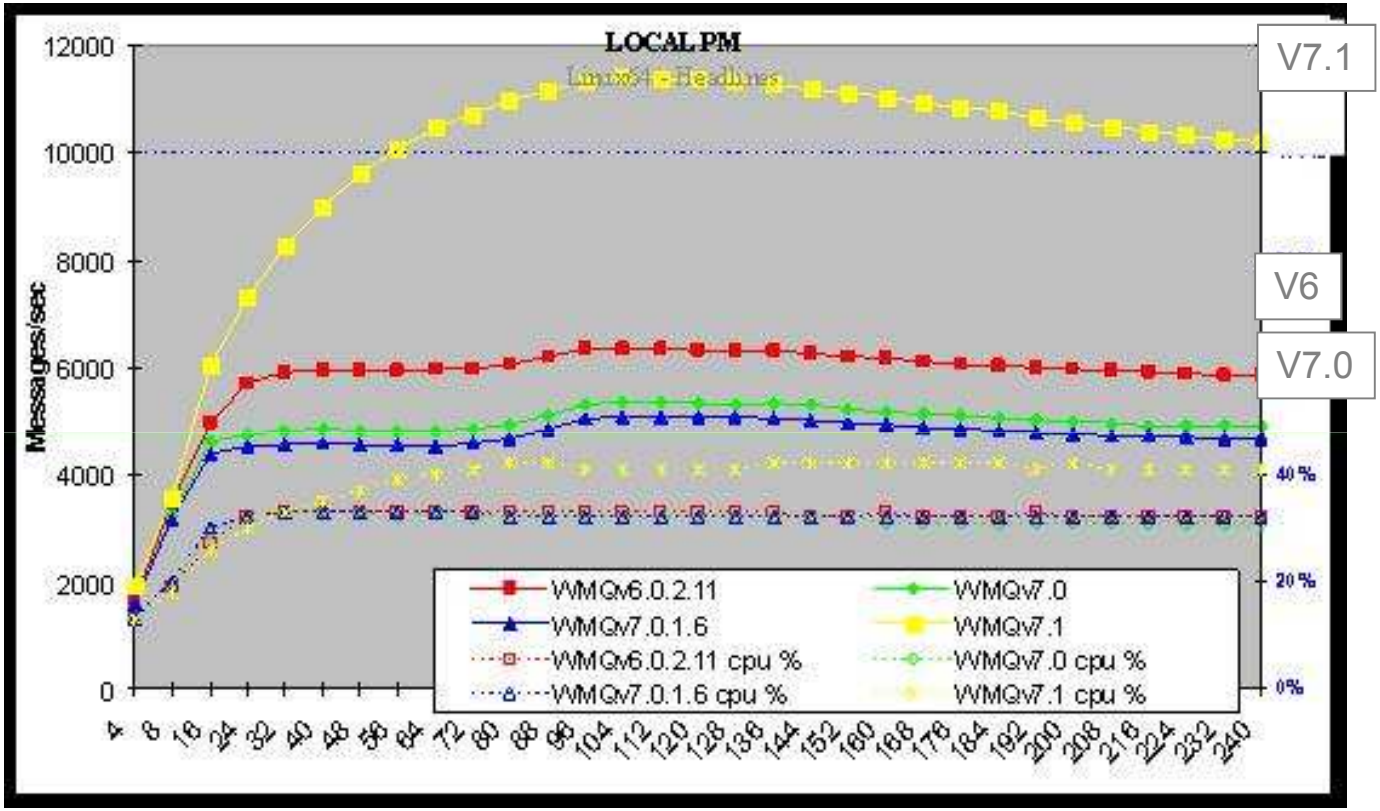
Persistent Messaging Improvements in 7.1

- Highlights (Averages across all the various test types)
  - AIX 40% improvement
  - HP 33% improvement
  - Linux x86 80% improvement
  - Solaris 36% improvement
  - Windows 100% improvement
- Especially visible in Local persistent message tests
  - Logger and Locking improvements with no network I/O
- Using the same standard performance tests MQ each release
  - No attempt to show better figures by modifying the tests!

Faster Persistent Messages

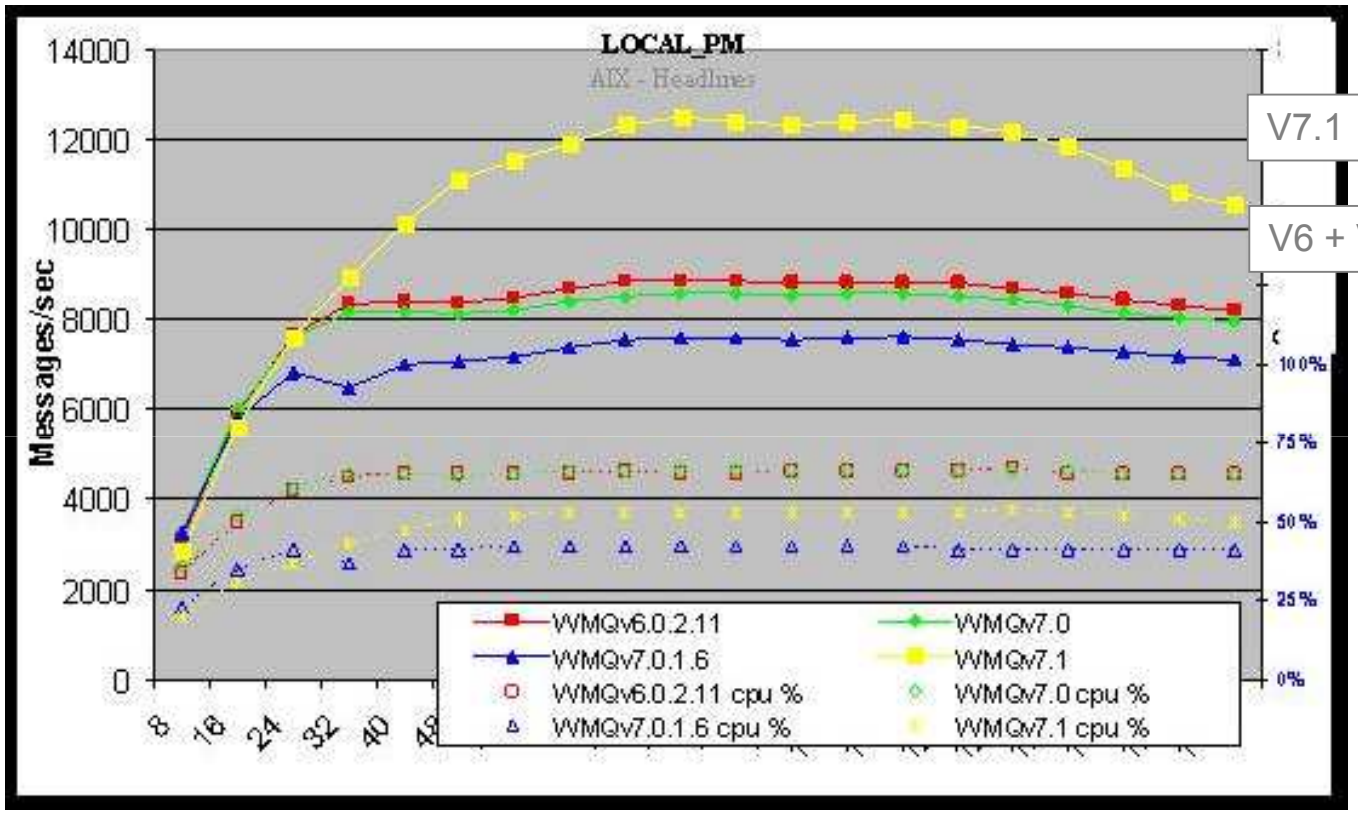
- The **local** persistent message tests try to show how fast a queue manager could perform with reliable messages
  - No network i/o – Put and Get on same machine
    - Not a real world scenario but more a proof of technology
- What we usually find is the bottleneck is around recording the information to persist the messages reliably (logging)
- Significant changes in 7.1 around allowing multiple applications to update in memory buffers in parallel. Along with the locking improvements
  - reduce logger memory buffer contention
  - utilizes more CPU and getting better throughput

Faster Persistent Messages (2K,Linux x86)



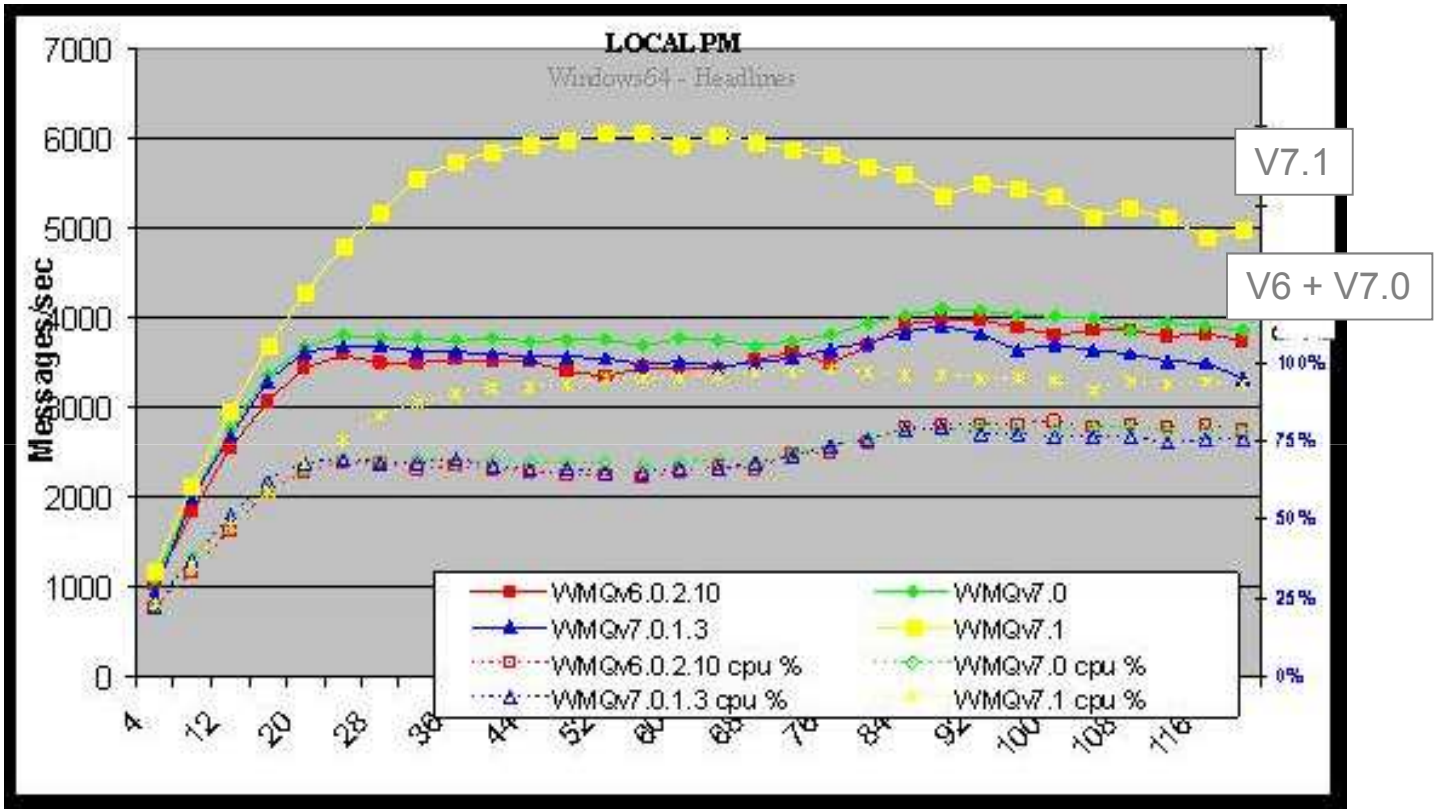
Local Persistent Messaging on Linux x86 improved 113% compared to v7.0.1

Faster Persistent Messages (2K,AIX)



Local Persistent Messaging on AIX improved 52% compared to v7.0.1

Faster Persistent Messages (2K,Windows)



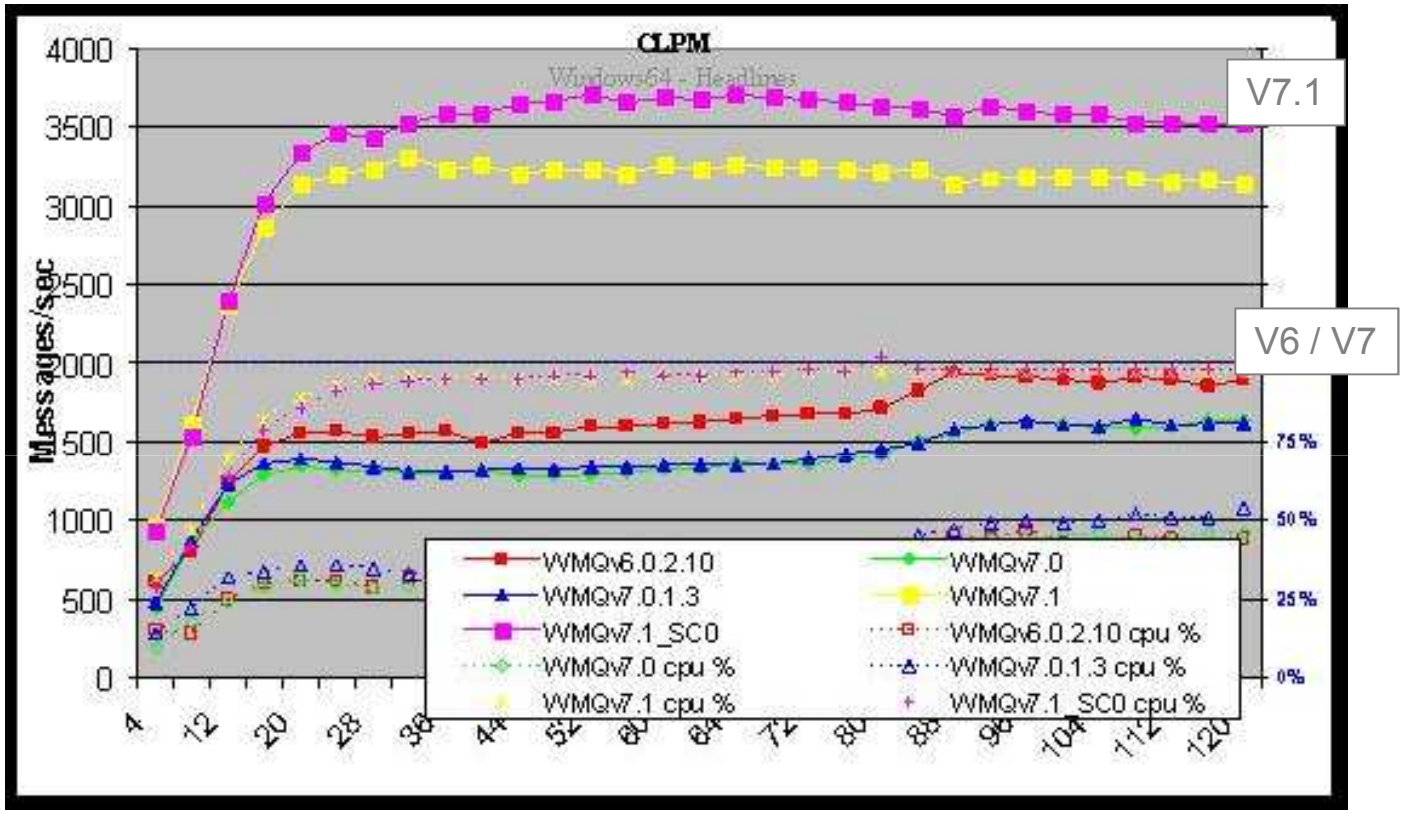
Local Persistent Messaging on Windows improved 49% compared to v7.0.1

Faster Persistent Messages – More real world

- The client channel persistent message tests show a system where clients deliver / retrieve the messages
- The distributed queuing message tests shows where messages are put on a channel and delivered back over another channel
  - Network i/o waits involved
  - Much more a real world scenario

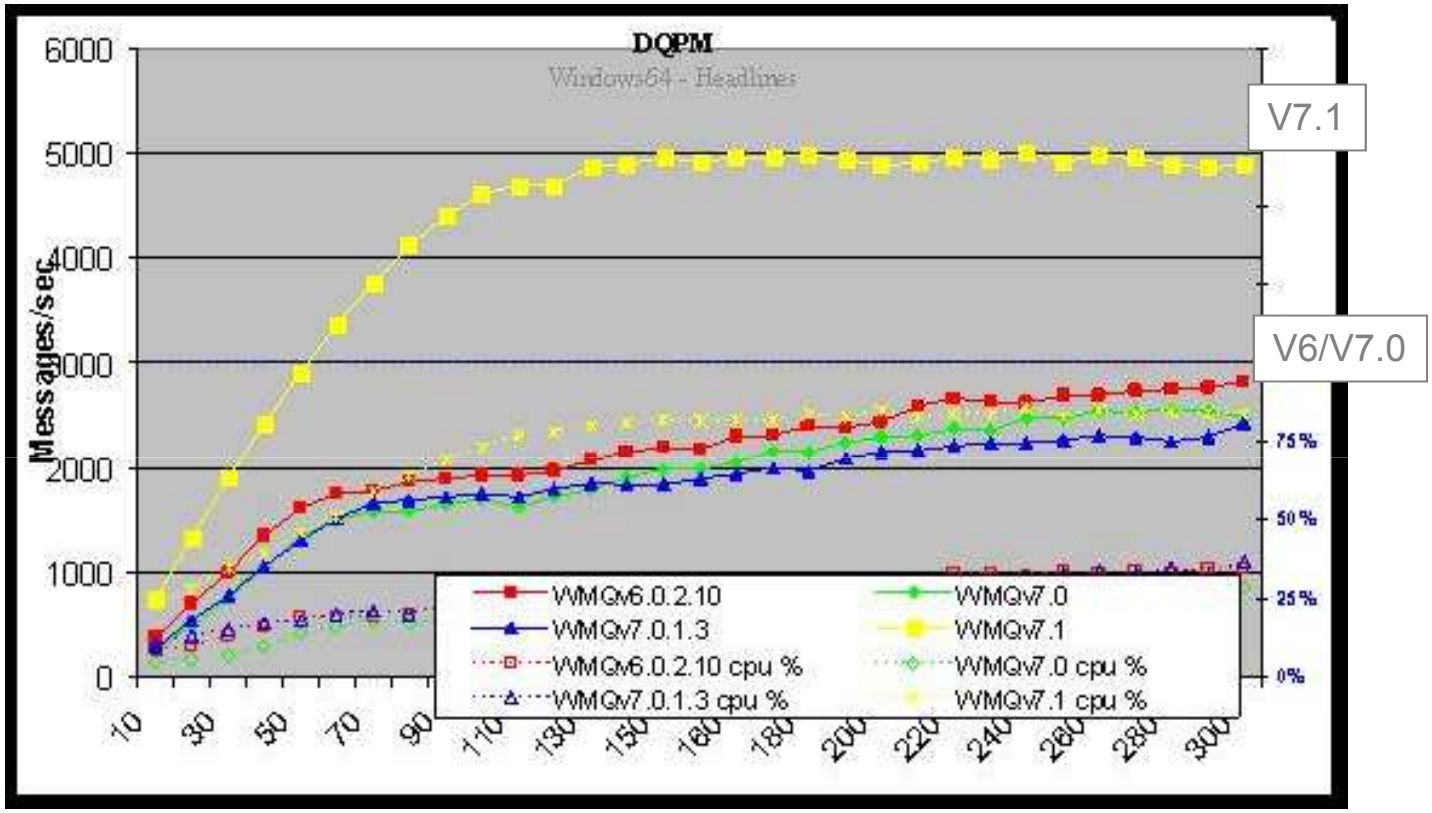
- Still significant improvements
  - In many cases 7.1 with SHARECNV (Shared conversations) faster than previous releases without it

Faster Persistent Messages (2K,Windows)



Client Persistent Messaging on Windows improved 120% compared to v7.0.1

Faster Persistent Messages (2K,Windows)



Distributed Queuing on Windows improved 137% compared to v7.0.1

Notes: Scaling Chart

N O T E S

- In the distributed queuing results, you can see that when network I/O is added to the picture, the contention on the single queues takes longer to become an issue.
- More threads are waiting on the network traffic, and overall there are more ‘moving parts’ in the system to keep the CPU busy

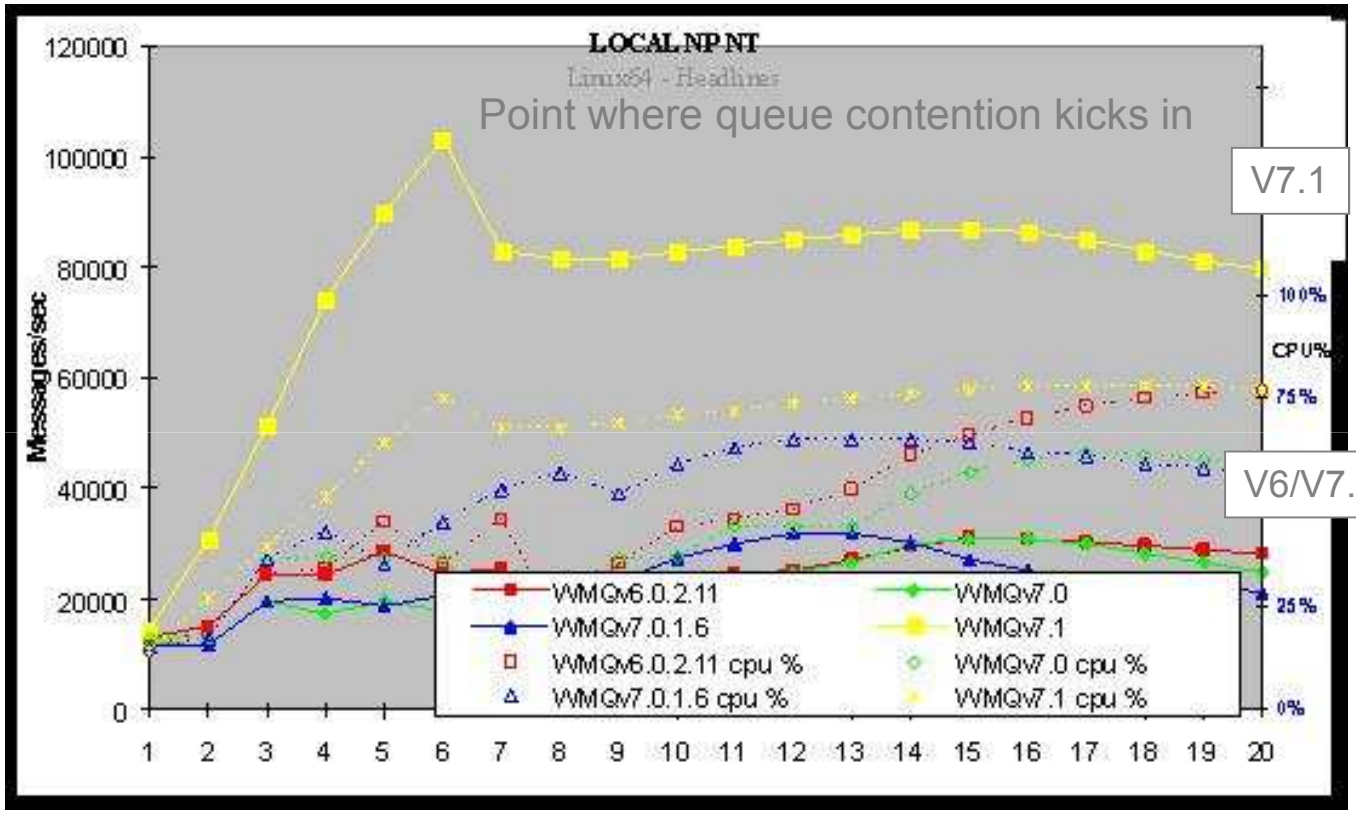
Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

Faster Non Persistent Messages

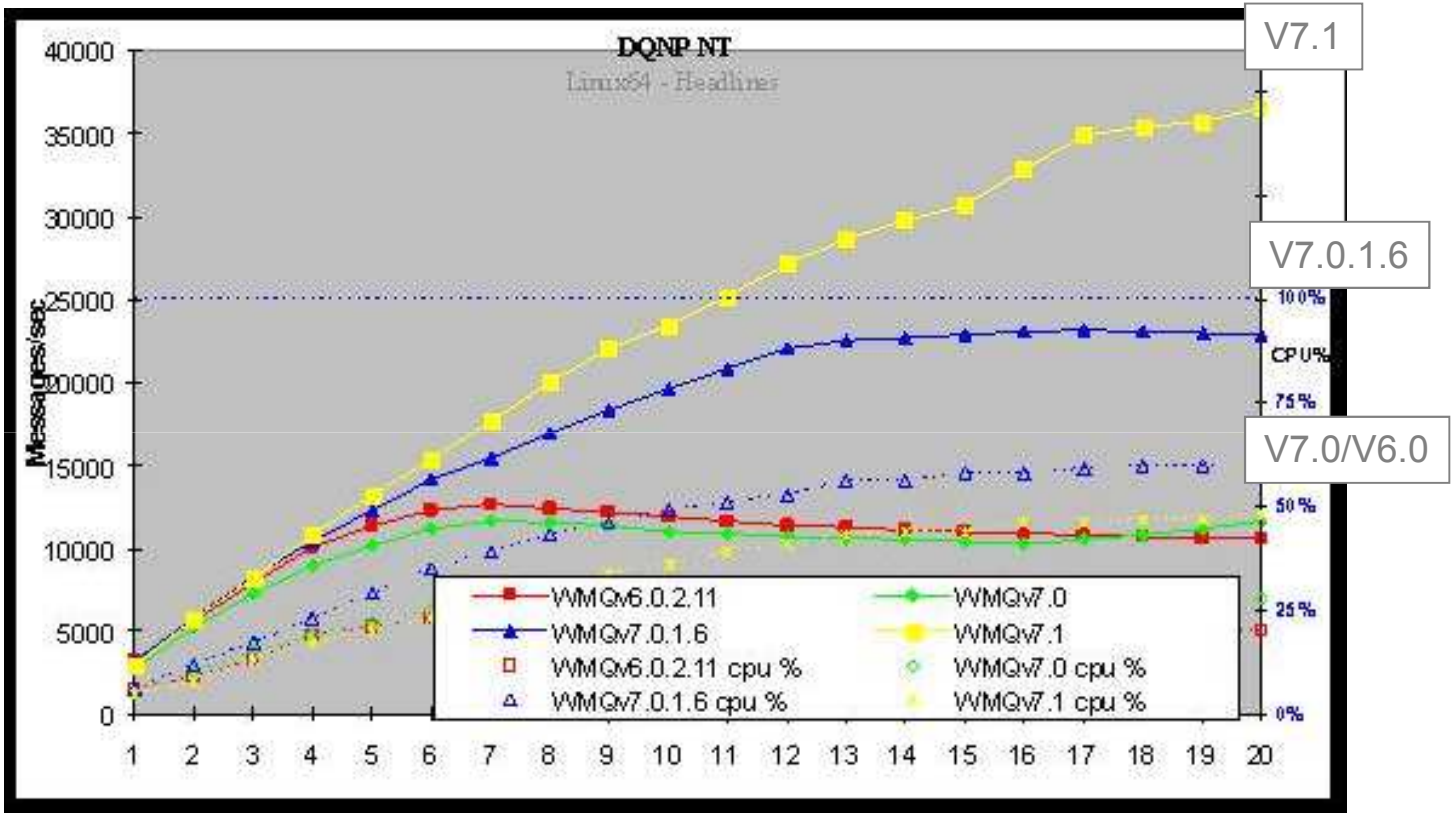
- Non Persistent messages do not get logged
  - Smaller messages can stay in in-memory buffers
  - No waiting on disk i/o
- The locking improvements which show the scalability on multi core machines really help with the performance here too
  - Once contention for resources kicks in
- There are other internal improvements such as avoiding message memory buffer copying
- The graphs tail off because of the single queue contention
  - With multiple queues, would go upwards further

Faster Non Persistent Messages (2K,Linux)



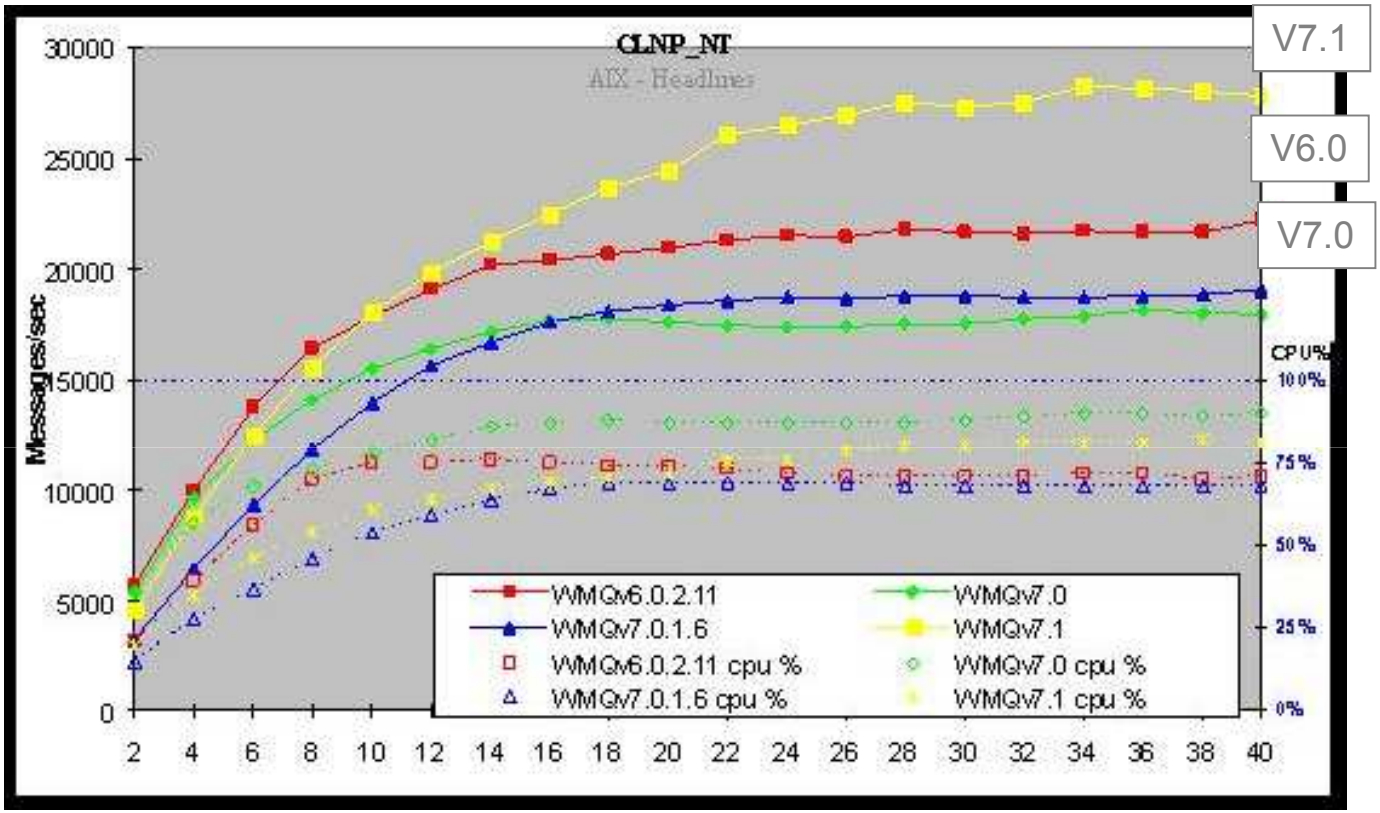
Non trusted local non persistent on Linux improved 231%
(Trusted applications around 50%)

Faster Non Persistent Messages (2K,Linux)



Distributed Queuing around 30% improved compared to v7.0.1.6

Faster Non Persistent Messages (2K,AIX)



Client non persistent on AIX improved 40% compared to 7.0.1.6

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

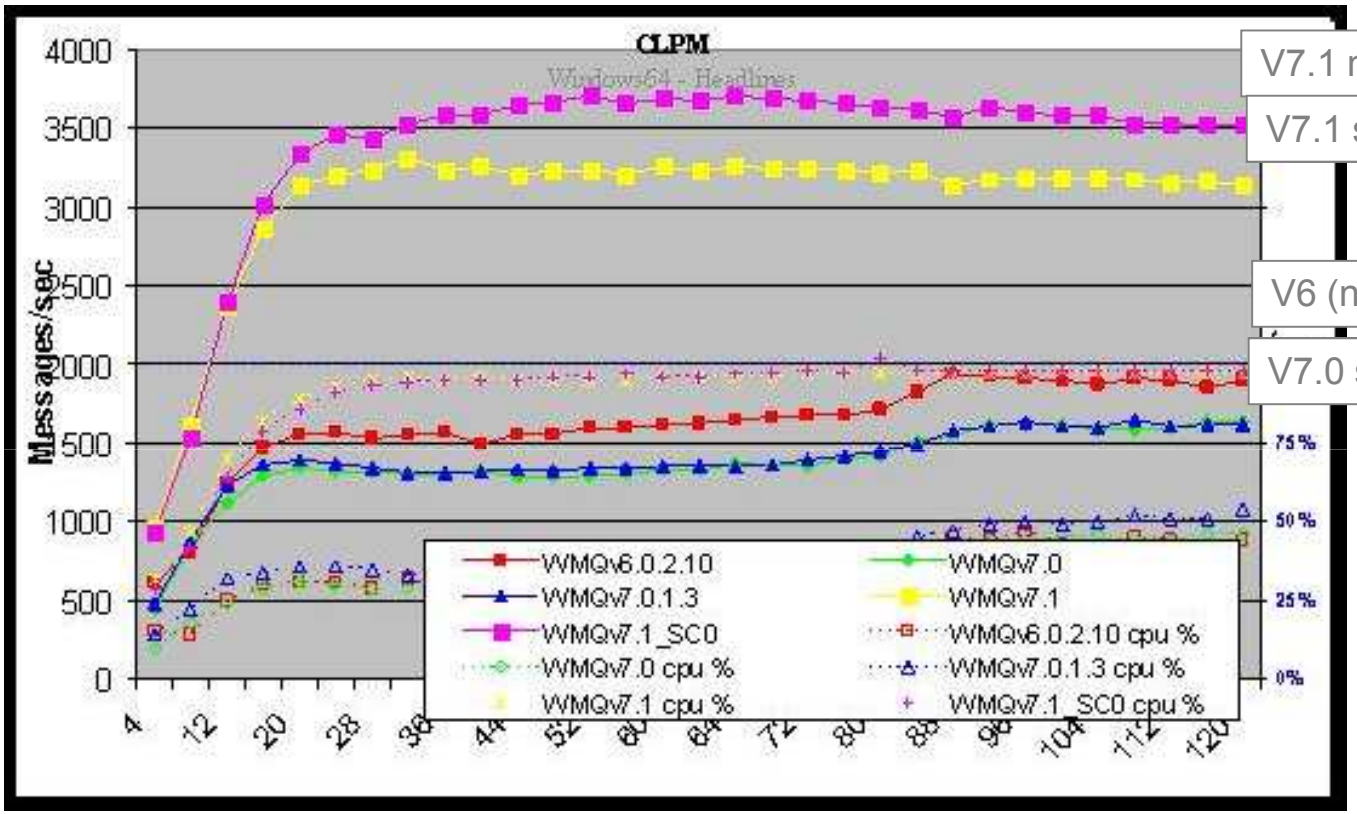
SHARECNV

- V7.0 SHARECNV brings benefits but at a performance cost
  - Client channels only
  - Bi-directional heartbeats, Aysnchronous put, Read ahead queueing
  - Multiple conversations have less setup costs

- V7.1 improvements mean even with SHARECNV set, performance is often better than at V6.0
  - Still a performance boost turning it off
  - Miss out on some of the benefits of having it though

- No attempt on the chart to utilize the potential performance benefits of SHARECNV

SHARECNV (P, 2K,Windows)



V7.1 no sharecnv

V7.1 sharecnv

V6 (no sharecnv)

V7.0 sharecnv

On Windows, SHARECNV faster than v6 and v7

Agenda

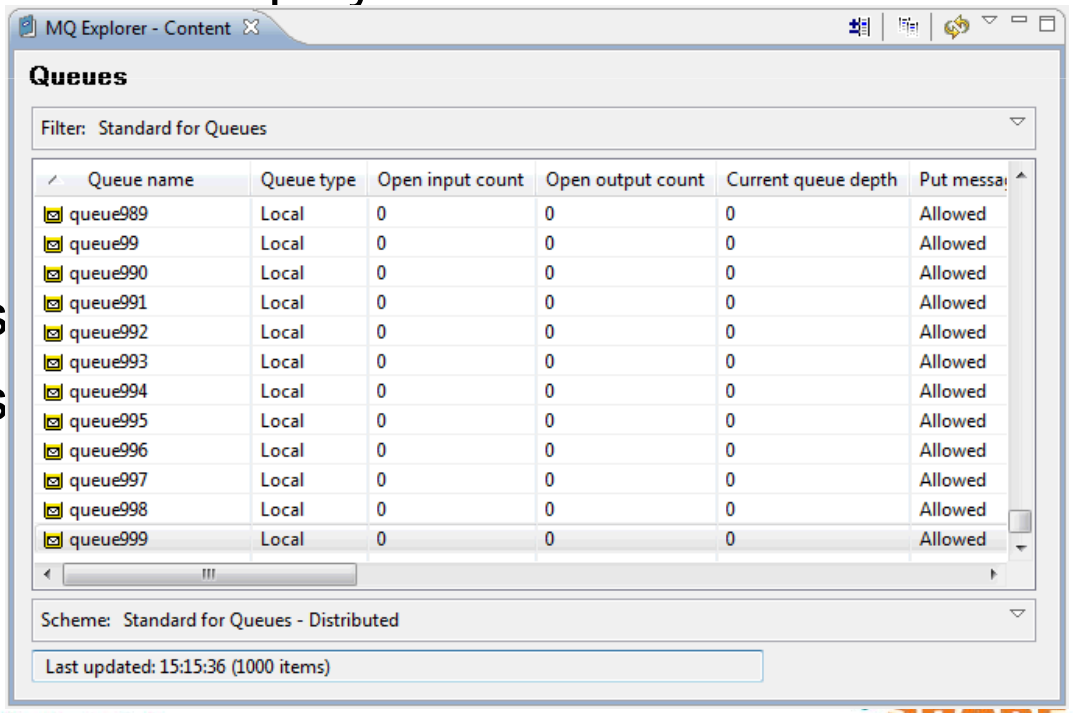
- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

Explorer (or MS0T) Scaling

- Pre-7.1, Explorer did not scale well
  - Slowed very quickly with
    - Large numbers of queue managers
    - Large numbers of objects (Queues, Channels)
  - Technote even talked about disabling a piece of functionality
    - http://www-01.ibm.com/support/docview.wss?uid=swg21452723
- Figures on subsequent foils are rough estimates on a laptop
  - Just as a comparative guide

Explorer Scaling

- Remote Queue Manager with 1000 queues
  - Clicking on the queues folder in the Navigator view will retrieve the list of queues and their attributes, sort them, then construct the table used to display them in the Content view
    - V7.0 about 8 seconds
    - V7.1 about 4 seconds



Explorer Scaling

- 100 remote Queue Managers
  - Refreshing the Navigator view is a common operation, it happens when the refresh button is pressed and whenever there are changes to be shown.
  - Time to connect to one of the 100 queue managers then disconnect.

    V7.0 about 60 seconds

    V7.1 about 4 seconds

Explorer – What changed?

- Significant internal rework to the navigator view results in ability to scale to multiple queue managers better
  - Almost no changes required to the sets plugin to achieve this

- Reducing the default amount of data queried
  - A lot of the time is spent building the table with all the attributes shown
  - In V7.1 the default schemes have been carefully pruned (for queues and channels) to show only the more frequently used attributes.

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- New capabilities of MQ 7.1
  - Multicast
  - Extended Reach (XR) support for MQTT
- Tips for improving performance

Lower QoS

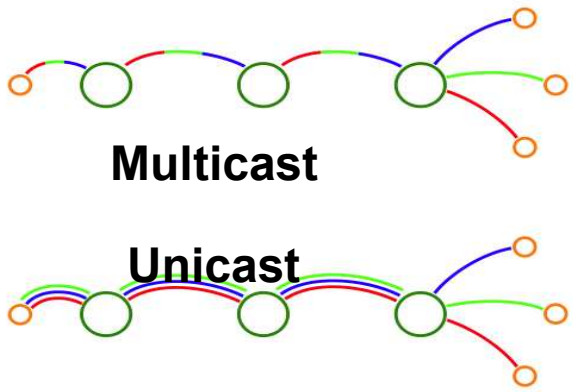
- If you can afford a lower quality of service MQ 7.1 has some new alternatives
  - Multicast
    - Publish once to all subscribers
    - Good for many consumers, small messages
    - No transactionality or persistence, No durable subscriptions
  - Extended Reach (XR)
    - Large numbers of concurrent connections
    - Good for small, infrequent messages

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- New capabilities of MQ 7.1
  - Multicast
  - Extended Reach (XR) support for MQTT
- Tips for improving performance

What is Multicast?

- Single message is duplicated in the network
  - Receivers register interest on specific IP addresses
  - Sender send datagrams to the multicast address
  - Network cards/ routers make copies of data and send to receivers who have registered for an address



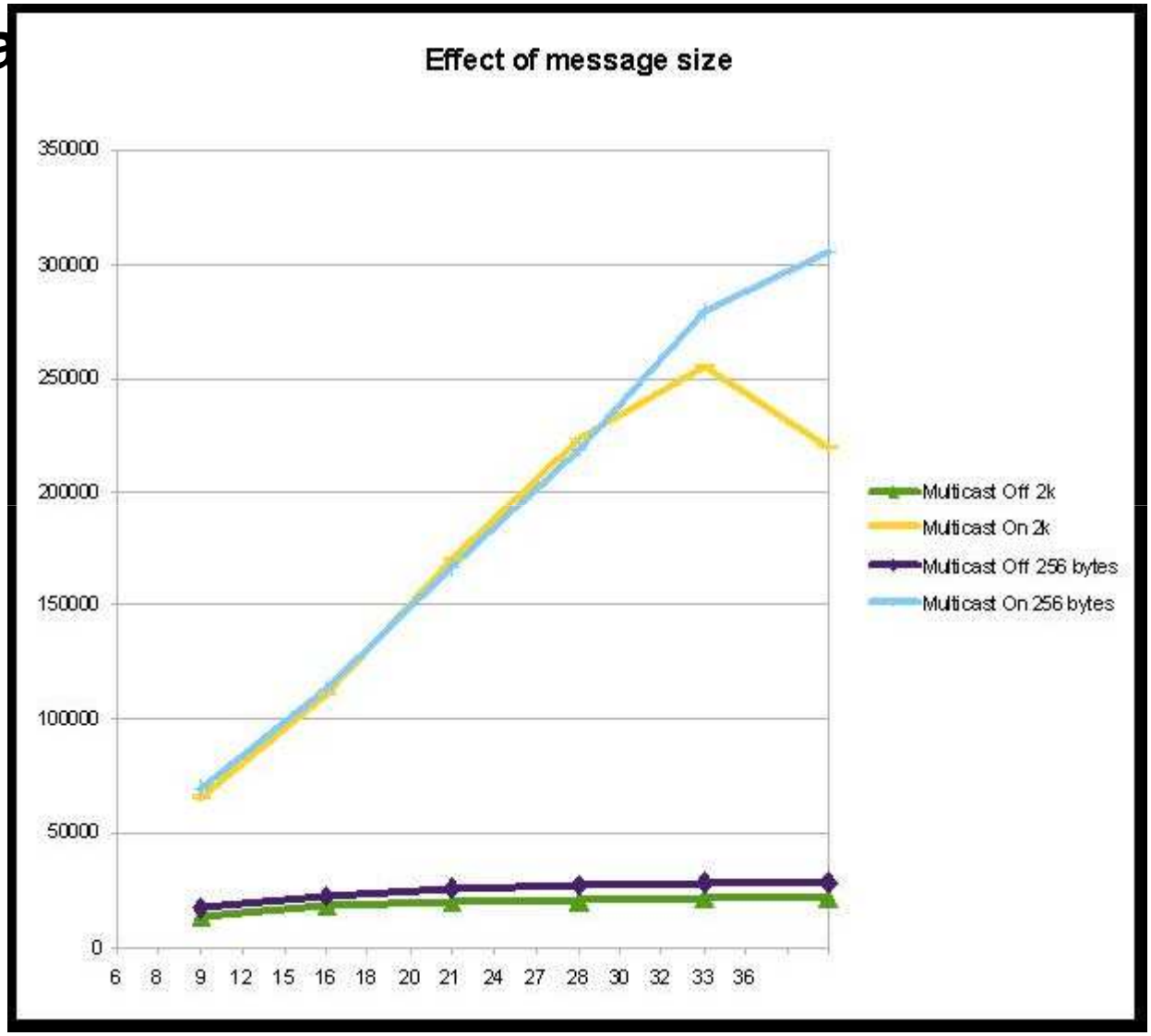
Multicast – One message cloned

Unicast – Multiple messages throughout

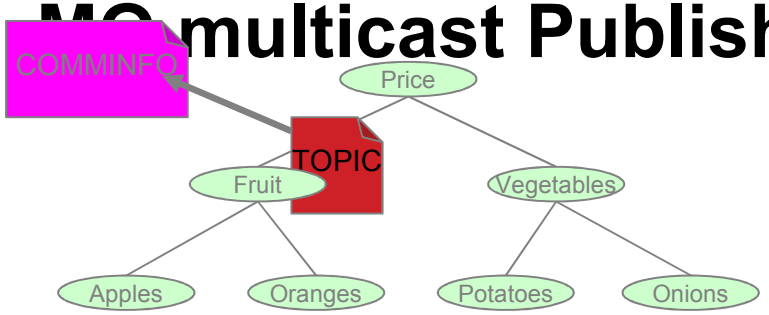
Multicast - What are the benefits

- Low latency
  - Much higher volumes than standard non-persistent messaging
  - Messages do not pass through qmgrs, and peer to peer communication
- High Scalability
  - Additional subscribers cause no slow down
  - Reduced network traffic
- ‘Fair delivery’ of data
  - Each subscriber ‘sees’ the data at the same time
  - Multicast offers near simultaneous delivery
- High availability
  - Multicast uses the network so no pub/sub engine to fan-out data
  - Reduces load on Queue managers servers

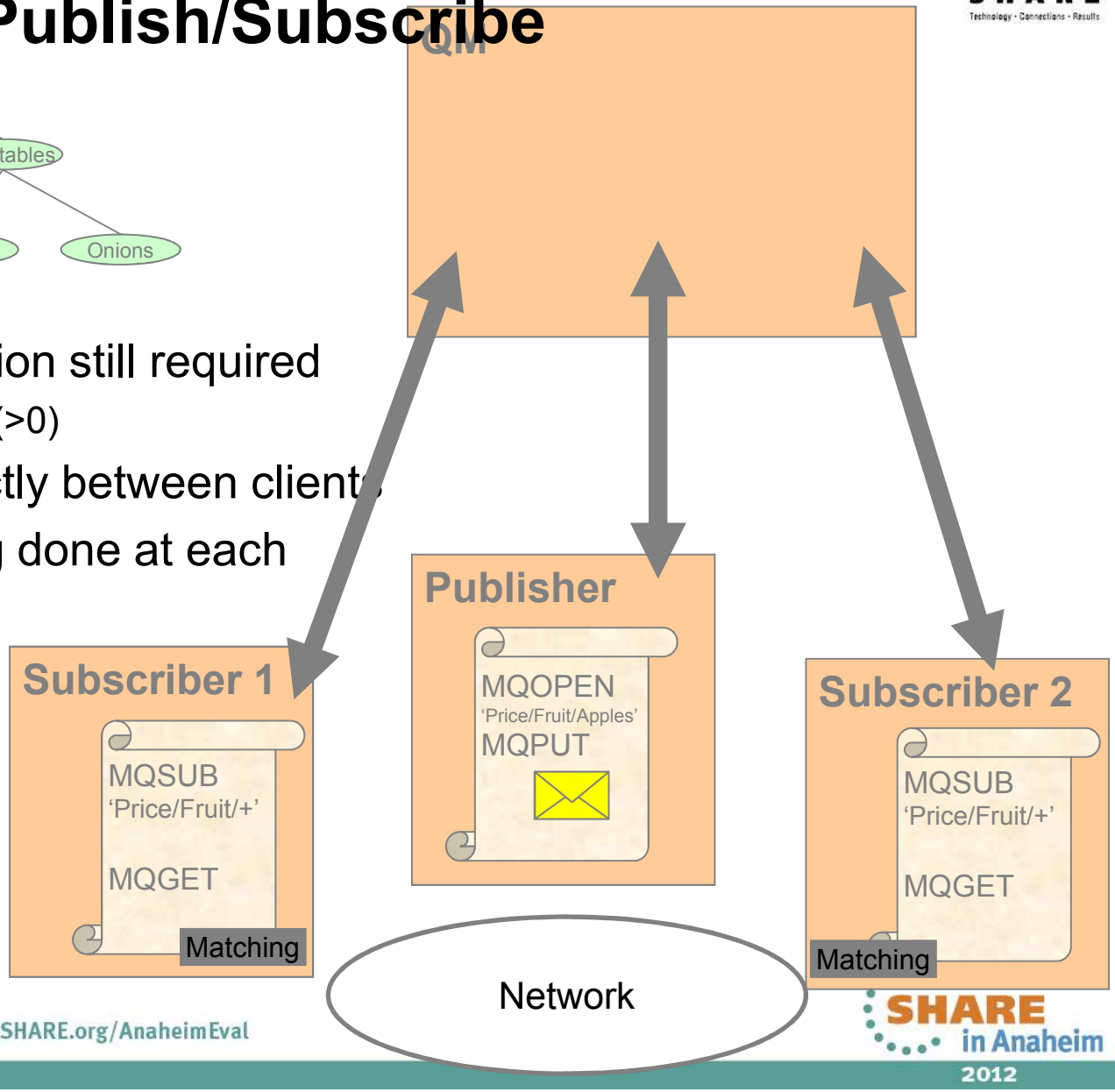
Multica


Effect of message size
350000
300000
250000
200000
150000
100000
50000
0
6 8 9 12 15 16 18 20 21 24 27 28 30 32 33 36
Multicast Off 2k
Multicast On 2k
Multicast Off 256 bytes
Multicast On 256 bytes

MQ multicast Publish/Subscribe



- Normal MQ connection still required
  - Must be SHARECNV(>0)
- Messages flow directly between clients
- Subscriber matching done at each client.



Multicast

- Multicast can be more efficient
  - But need to understand limitations around its QoS
- Multicast more complex to set up
  - Routers may need to be configured to pass traffic
- Uses normal MQI with some restrictions
  - No persistence nor transactionality
  - No durable subscribers
  - No message segmentation nor grouping
  - Pub/Sub only

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- New capabilities of MQ 7.1
  - Multicast
  - Extended Reach (XR) support for MQTT
- Tips for improving performance

Extended Reach (MQXR)

- Provides support for the MQTT protocol
  - Ideal for small or embedded devices
    - mobile devices, smart meters, set top boxes, remote telemetry units
  - Typically used for infrequent, small messages (256 bytes in charts)
  - Does not use the MQI
- Supports 3 Quality of Services
  - 0 - At most once (fast but unreliable)
  - 1 - At least once (duplicates possible)
  - 2 - Exactly once (slower but assured)
- Ideal for large numbers of connections with low message rates
  - Tested with up to 100,000 clients on Linux, 64,000 on Windows

MQXR – Single subscriber Multi

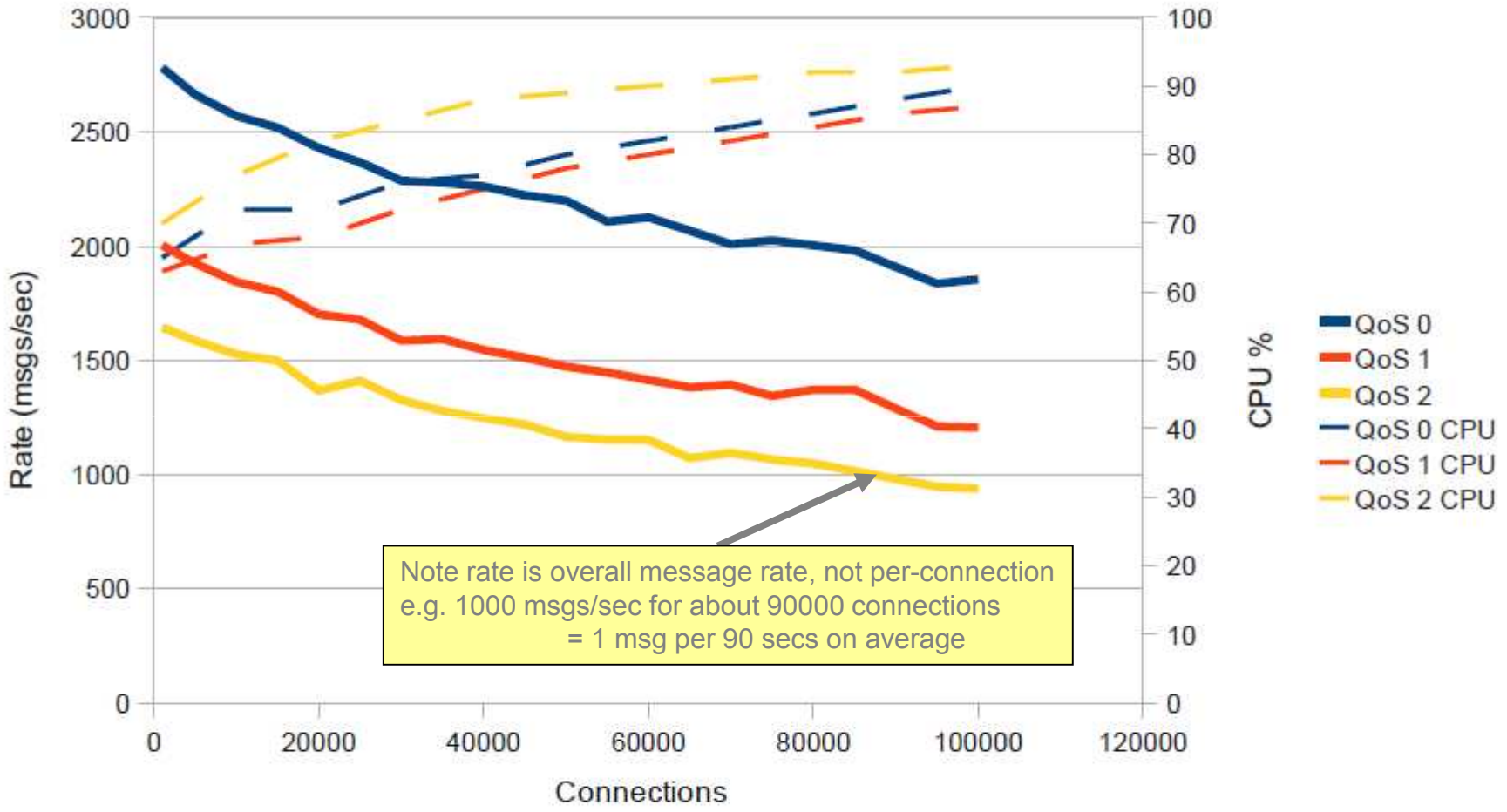


Figure 3: Linux multi-publisher, single-subscriber graph

Agenda

- Changes to MQ 7.1 with performance benefits
  - Better scaling with multiple CPUs
  - Faster persistent messages
  - Faster non-persistent messaging
  - SHARECNV channel performance
  - Explorer scaling
- Tips for improving performance

Performance Implications: Heavyweight MQI Calls

- MQCONN is a “heavy” operation
  - Don’t let your application do lots of them
  - Wrappers and OO interfaces can sometimes hide what’s really happening
  - Lots of MQCONNs can drop throughput 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
    - Loading pre-existing queue; creating dynamic queue
  - 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 (3x Network flows / IPC)

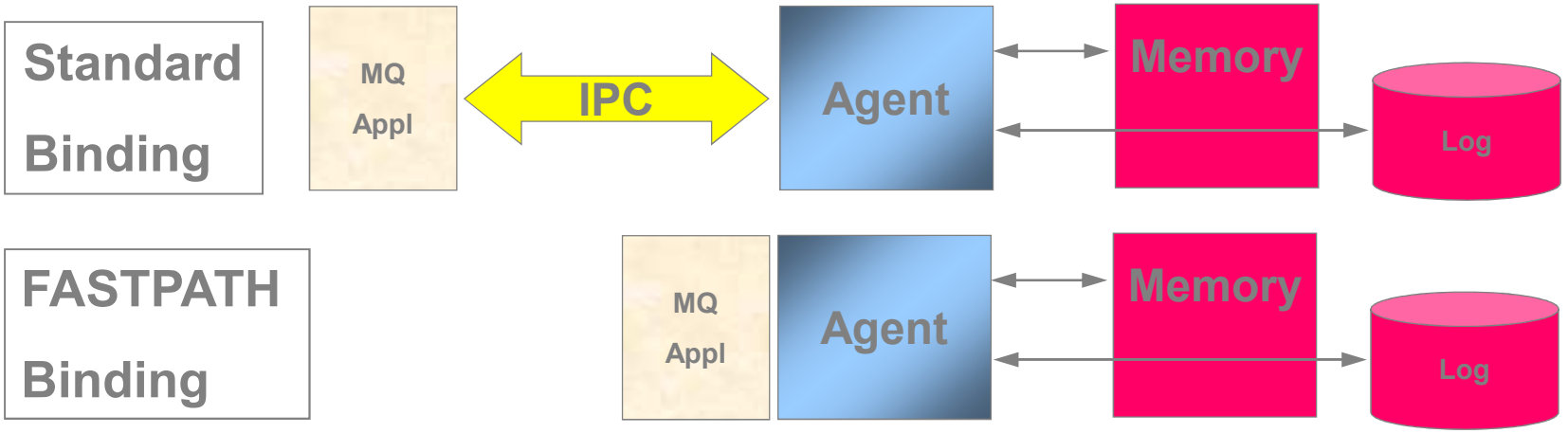
- Try to avoid exclusive access to the Queue
  - Makes it harder to scale the solution
    - For example adding more instances of application
  - Implies that reliance on message order is not required
    - Partition the data to allow parallel processing?

Performance Implications: Heavyweight MQI Calls

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 not 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.

Performance Implications: Connection Binding



- Fastpath binding removes inter-process communications
  - 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 Channels/MCAs, Broker especially if no user exits
    - - 30% CPU saving

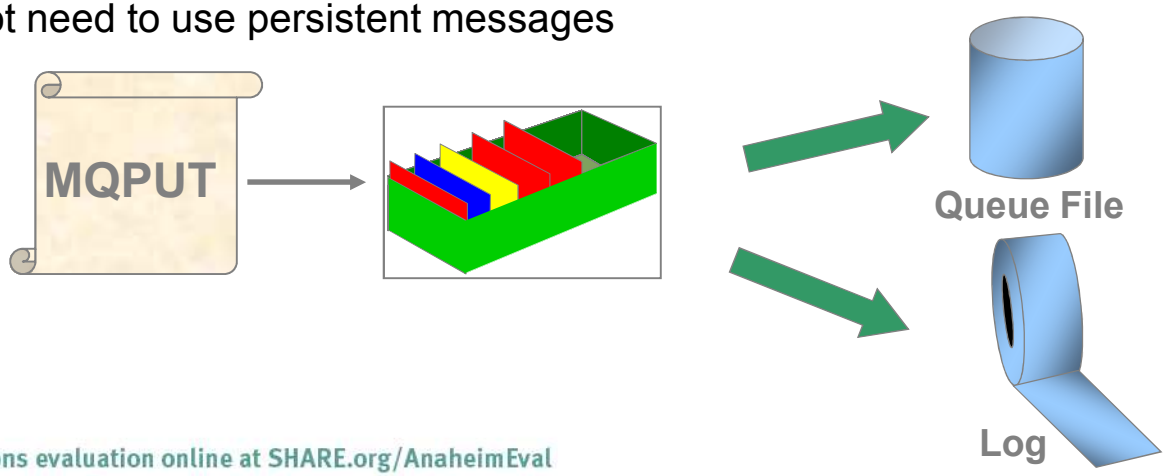
Performance Implications: Connection Binding

NOTES

- Two of the major overheads in the processing path for MQ 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 MQ 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

Performance Implications: Persistence

- Log bandwidth is going to restrict throughput
  - Put the log files on the fastest disks you have, separate from queue manager data
- Persistent messages are the main things requiring recovery after an outage
  - Can significantly affect restart times
- Why use persistence?
  - False assumption that persistence is for "important" data and non-persistent for when you don't care
  - The real reason for persistent messages is to reduce application complexity
  - With persistent, apps do not need logic to detect and deal with lost messages
  - If your app (or operational procedures) can detect and deal with lost messages, then you do not need to use persistent messages



Performance Implications: Persistence

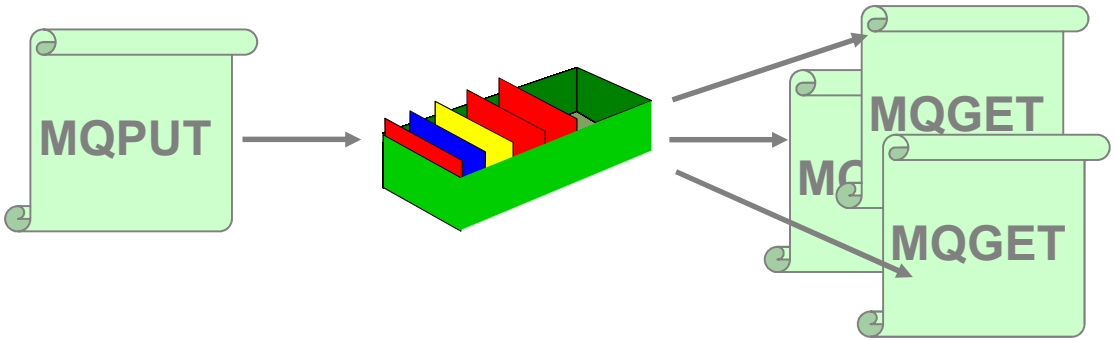
NOTES

- 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.
- 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.

Put to a waiting getter

- MQPUT most efficient if there is getting application waiting
  - Only for out of syncpoint messages
    - Both persistent and non-persistent
  - No queuing required
    - Removes a lot of processing of placing the message onto the queue

- Significantly reduces CPU cost and improves throughput
  - V7.1 improves memory buffer management in this scenario



Put to a waiting getter

NOTES

- "Put to a waiting getter" (aka I/O-avoidance) 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 be non-persistent). If these conditions are met then the message will be transferred from the putter's buffer into the getter's 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'
- You should not expect to see "even" workload distribution between applications when they are all getting from the same queue
- V7.1 can reduce the number of times it copies the message buffer when there is a waiting getter

Design tips for faster MQ applications

- Minimize disk i/o impact
  - Avoid writing persistent messages outside of syncpoint
    - Holds locks whilst waiting for data to flush to disk
  - Multiple messages in a single (short) UOW – lazy writes on put, commit flush is larger I/O
  - Consider tuning queue buffer sizes with Default[P]QBufferSize
- Avoid contention where possible
  - Thousands of applications all working on one queue can make that queue a bottleneck
- Read the performance reports carefully
  - See what they tuned

Useful Links

- Performance Reports available as SupportPacs
  - http://www.ibm.com/support/docview.wss?uid=swg27007197

  - Base product (by platform)
    - MP6Q,MP6R,MP6S,MP48,MP7J,MPL7,MPL8
  - MQ JMS, XMS
    - MP0B, MP7K

Questions?