This presentation is not a basic introduction to WebSphere MQ Queue Manager Clustering. It assumes you already have a basic knowledge of Queue Manager Clustering prior to using this material. To gain such a basic knowledge the following resources may be useful:-


In this presentation we will cover the following topics:-

- Workload Balancing
- Availability
- What’s New in MQ Clustering
Workload balancing - Bind Options

- **Bind on open**
  - Messages are bound to a destination chosen at MQOPEN
  - All messages put using open handle are bound to same destination

- **Bind not fixed**
  - Each message is bound to a destination at MQPUT
  - Workload balancing done on every message
  - Recommended - No affinities are created (SCTQ build up)

- **Application options**
  - MQOO_BIND_ON_OPEN
  - MQOO_BIND_NOT_FIXED
  - MQOO_BIND_AS_Q_DEF (Default)

- **DEFBIND Queue attribute**
  - OPEN (Default)
  - NOTFIXED

---

Workload balancing - Bind Options - Notes

- Affinity is the need to continue using the same instance of a service, with multiple messages being sent to the same application process. This is often also known as conversational style applications. Affinity is bad news for parallel applications since it means there is a single point of failure. It can also inhibit scalability (no longer shared nothing, long locks needed). Historically, affinities have caused many problems.
- When writing new applications, one of your design goals should be for no affinities.
- If you cannot get around the need for an affinity in your application, however, MQ will help you.
- If you require all the messages in a conversation to be targeted to the same queue manager, you can request that using the bind options, specifically the “Bind on open” option. Using this option, a destination is chosen at MQOPEN time and all messages put using that open handle are bound to same destination. To choose a new destination, perhaps for the next set of messages, close and reopen the destination in order to re-drive the workload balancing algorithm to choose a new queue manager.
- If you do not have any affinities in your applications, you can use the opposite option, the “Bind not fixed” option.
- This behaviour can either be coded explicitly in your application using the MQOO_BIND_* options or through the default attribute DEFBIND on a queue definition. This attribute defaults to “Bind on open” to ensure applications not written with parallelism in mind still work correctly in a cluster.
When does workload balancing occur?

- In a typical setup, where a message is put to a queue manager in the cluster, destined for a backend server queue manager also in the cluster, there are three places at which workload balancing may occur.
- The first of these is at either MQOPEN or MQPUT time depending on the bind options associated with message affinity. If bind_on_open is specified, then this means all messages will go to the same destination, so once the destination is chosen at MQOPEN time, no more workload balancing will occur on the message. If bind_not_fixed is specified, the messages can be sent to any of the available destinations. The decision where to send the message is made at MQPUT time, however this may change whilst the message is in transit.
- If a channel from a queue manager to a backend server goes into retry, then any bind_not_fixed messages waiting to be sent down that channel will go through workload balancing again to see if there is a different destination that is available.
- Once a message is sent down a channel, at the far end the channel code issues an MQPUT to put the message to the target queue and the workload algorithm will be called.
Workload management features

**QUEUES**
- Put allowed
  - PUT(ENABLED/DISABLED)
- Utilising remote destinations
  - CLWLUSEQ
- Queue rank
  - CLWLRANK
- Queue priority
  - CLWLPRTY

**CHANNELS**
- Channel status
  - INACTIVE, RUNNING
  - BINDING, INITIALIZING, STARTING, STOPPING
  - RETRYING
  - REQUESTING, PAUSED STOPPED
- Channel network priority
  - NETPRTY
- Channel rank
  - CLWLRANK
- Channel priority
  - CLWLPRTY
- Channel weighting
  - CLWLWGHT

**QUEUE MANAGER**
- Utilising remote destinations
  - CLWLUSEQ
- Availability status
  - SUSPEND/RESUME
- Most recently used
  - CLWLMRUC

**EXIT**
- A cluster workload exit can be used

---

**NOTES**

- There are a number of features in WebSphere MQ which affect the way the default workload balancing algorithm works. With all these attributes set to their default values, the workload balancing algorithm could be described as “round robin, excluding failed servers”. More control is required in some scenarios to allow more flexible interconnected cluster, or to allow greater scalability, and some of these attributes will then need to be altered to achieve the behaviour required.
- Only some attributes on a clustered queue or channel have an impact on the workload balancing algorithm. In other words, not all attributes are propagated around the cluster. Those that have an impact are shown on this page.
- If a queue is altered to be disabled for put, this change is propagated around the cluster and is used by other queue managers in the cluster when choosing which queues to send their messages to. A queue that is disabled for put is not a good choice!
- A queue manager may be suspended from the cluster. This is another way of removing queues on that queue manager from being chosen by the workload balancing algorithm. Note, that if all queue managers in a cluster are suspended, then they are all equally ‘good’ choices and the effects will be nullified.
**Workload management features**

- Channel status plays a part in the workload balancing algorithm, by indicating when a server has failed. ‘Bad’ channel status suggests a problem and makes that queue manager a worse choice. If the entire network is having a problem and all channels are in RETRY state, then all queue managers are equally ‘good’ choices and the effects again will be nullified – where-ever the messages are targeted, they will have to wait on the transmission queue for the network to come back up.

- Network priority provides a way of choosing between two routes to the same queue manager. For example, a TCP route and a SNA route, or an SSL route and a non-SSL route. The higher network priority route is always used when it’s channel is in a good state.

- All of the choices made by the queue manager using it’s default workload balancing algorithm can be over-ridden by the use of a cluster workload exit. This exit is provided with all the same information that the queue manager uses to make its choices, and is also told which choice the default algorithm made, and can then change that choice if it wishes. It can make decisions based on message content, or other information that the queue manager is not aware, such as business routing decisions.
**Workload Management**

- The workload balancing algorithm gives you a huge set of tools
  - Resist the urge to use all of them
  - Quickly get very complicated interactions with more than a few in play

- May not be immediately obvious how important channels are to the selection process
  - Multiple routes to a destination = multiple destinations (from WLM round robin point of view)
  - Stopped local receiver = local destination less favoured

- More often than not, problems hit by administrators are caused by the interactions between multiple WLM parameters.

- This is another area where a large number of overlapping clusters can cause issues – probably confusion in this case.

- On the other hand, remember that most balancing is per-channel. As well as the points on the main slide, this means that separating applications into separate channels (therefore clusters) will stop messages from one altering balancing for another.
### Utilising remote destinations

- **MQPUT CLUSQ**
  - SERVERA
  - SERVERB
  - SERVERC
  - SERVERD

- Allows remote queues to be chosen, when a local queue exists

- **DEFINE QL(CLUSQ) CLUSTER(GREEN) CLWLUSEQ( )**
  - `LOCAL` = If a local queue exists, choose it.
  - `ANY` = Choose either local or remote queues.
  - `QMGR` = Use the queue manager’s use-queue value.

- **ALTER QMGR CLWLUSEQ( )**
  - `LOCAL` = If the queue specifies CLWLUSEQ(QMGR) and a local queue exists, choose it.
  - `ANY` = If the queue specifies CLWLUSEQ(QMGR) choose either local or remote queues.

- Messages received by a cluster channel must be put to a local queue if one exists

---

### Utilising remote destinations

- Prior to WebSphere MQ V6, if a local instance of the named cluster queue existed, it was always utilised in favour of any remote instances. This behaviour can be over-ridden at V6 by means of the CLWLUSEQ attribute, which allows remote queues to be chosen, when a local queue exists.

- The CLWLUSEQ attribute is available on queue definitions, to allow only specific named queues to utilise this over-ride, or on the queue manager to over-ride this behaviour for all cluster queues.

- **DEFINE QL(CLUSQ) CLUSTER(GREEN) CLWLUSEQ( )**
  - `LOCAL` = If a local queue exists, choose it.
  - `ANY` = Choose either local or remote queues.
  - `QMGR` = Use the queue manager’s use-queue value.

- **ALTER QMGR CLWLUSEQ( )**
  - `LOCAL` = If the queue specifies CLWLUSEQ(QMGR) and a local queue exists, choose it.
  - `ANY` = If the queue specifies CLWLUSEQ(QMGR) choose either local or remote queues.

- Any messages received by a cluster channel will be put to a local queue if one exists in order to avoid looping round the cluster.
Rank allows greater control of the routing of messages through overlapping clusters.

Without using rank, messages put to CLUSQ by applications connected to SERVERA could bounce between the alias queues on SERVERB and SERVERC. Using rank solves this “bouncing” problem because once messages arrive on SERVERB or SERVERC, the CLUSQ instance on SERVERD is available to the cluster workload management algorithm. As CLUSQ on SERVERD is ranked higher it will be chosen.
Rank - basics

- Channels and queues with the highest rank are chosen preferentially over those with lower ranks.

- MQSC
  - DEFINE CHL(TO.ME) CHLTYPE(CLUSRCVR) CLUSTER(BLUE) … CLWLRRANK()
    - Range 0 – 9
    - Default 0
  - DEFINE QL(CLUSQ) CLUSTER(BLUE) CLWLRRANK()
    - Range 0 – 9
    - Default 0

- Channel rank checked before queue rank

- Alternative to put disabling queues
  - MQRC_NONE Vs MQRC_PUT_INHIBITED
**Channel Rank example**

- The channel rank for SERVER2 and SERVER3 are set higher than SERVER4. As the default channel rank is zero, SERVER1 has the lowest rank.

- Once the ranks for channels on SERVER2, SERVER3 and SERVER4 have been altered the messages are distributed equally between the highest ranked destinations (SERVER2 and SERVER3).
Queue Rank example

Once the ranks for queues on SERVER1 and SERVER2 have been changed the all messages are delivered to SERVER2. This is because the cluster workload management algorithm checks channel ranks before queue rank. The channel rank check leaves SERVER2 and SERVER3 as valid destinations and because the queue rank for DEMOQ on SERVER2 is higher than that on SERVER3 the messages are delivered to SERVER2. Note that as channel rank is more powerful than queue rank, the highest ranked queue (on SERVER1) is not chosen.

It is important to note that destinations with the highest rank will be chosen regardless of the channel status to that destination. This could lead to messages building up on the SYSTEM.CLUSTER.TRANSMIT.QUEUE.
Utilising processing power

- SERVERB is twice as powerful as SERVERC
- How can I send SERVERB twice as many messages?
- If more than one destination is valid, the round robin algorithm sends messages in numbers proportional to channel weights
- MQSC
  - DEFINE CHL(TO.ME) CHLTYPE(CLUSRCVR) CLUSTER(GREEN) … CLWLWGHT( )
    - Range 1 – 99
    - Default 50
Channel Weight example

In this example, the approximate percentage of messages distributed to each queue manager are as follows…

- SERVER1 20%
- SERVER2 50%
- SERVER3 20%
- SERVER4 10%

Weight enables the cluster workload management algorithm to favour more powerful machines.
Cluster Workload Algorithm

- Queue PUT(ENABLED/DISABLED)
- Local instance (CLWLUSEQ)
- Channel rank (CLWLRANK)
- Queue rank (CLWLRANK)
- Channel Status
  - INACTIVE, RUNNING
  - BINDING, INITIALIZING, STARTING, STOPPING
  - RETRYING
  - REQUESTING, PAUSED, STOPPED
- Channel net priority (NETPRTY)
- Channel priority (CLWLPRTY)
- Queue priority (CLWLPRTY)
- Most recently used (CLWLMRUC)
- Least recently used with channel weighting (CLWLWGHT)

The full algorithm (taken from the Queue Manager Clusters manual) is as follows…

1. If a queue name is specified, queues that are not PUT enabled are eliminated as possible destinations. Remote instances of queues that do not share a cluster with the local queue manager are then eliminated. Next, remote CLUSRCVR channels that are not in the same cluster as the queue are eliminated.
2. If a queue manager name is specified, queue manager aliases that are not PUT enabled are eliminated. Remote CLUSRCVR channels that are not in the same cluster as the local queue manager are then eliminated.
3. If the result above contains the local instance of the queue, and the use-queue attribute of the queue is set to local (CLWLUSEQ(LOCAL)), or the use-queue attribute of the queue is set to queue manager (CLWLUSEQ(QMGR)) and the use-queue attribute of the queue manager is set to local (CLWLUSEQ(LOCAL)), the queue is chosen; otherwise a local queue is chosen if the message was not put locally (that is, the message was received over a cluster channel). User exits are able to detect this using the MQWXP.Flags flag MQWXP_PUT_BY_CLUSTERS_CHL and MQWQR.QFlags flag MQQF_CLWL_USEQ_ANY not set.
4. If the message is a cluster PCF message, any queue manager to which a publication or subscription has already been sent is eliminated.
4a. All channels to queue managers or queue manager alias with a rank (CLWLRANK) less than the maximum rank of all remaining channels or queue manager aliases are eliminated.
4b. All queues (not queue manager aliases) with a rank (CLWLRANK) less than the maximum rank of all remaining queues are eliminated.
Cluster Workload Algorithm (cont)

- 5. If only remote instances of a queue remain, resumed queue managers are chosen in preference to suspended ones.
- 6. If more than one remote instance of a queue remains, all channels that are inactive (MQCHS_INACTIVE) or running (MQCHS_RUNNING) are included.
- 7. If no remote instance of a queue remains, all channels that are in binding, initializing, starting or stopping state (MQCHS_BINDING, MQCHS_INITIALIZING, MQCHS_STARTING, or MQCHS_STOPPING) are included.
- 8. If no remote instance of a queue remains, all channels in retrying state (MQCHS_RETRYING) are included.
- 9. If no remote instance of a queue remains, all channels in requesting, paused or stopped state (MQCHS_REQUESTING, MQCHS_PAUSED and MQCHS_STOPPED) are included.
- 10. If more than one remote instance of a queue remains and the message is a cluster PCF message, locally defined CLUSSDR channels are chosen.
- 11. If more than one remote instance of a queue remains to any queue manager, channels with the highest NETPRTY value for each queue manager are chosen.
- 11a. If a queue manager is being chosen: all remaining channels and queue manager aliases other than those with the highest priority (CLWLPRTY) are eliminated. If any queue manager aliases remain, channels to the queue manager are kept.
- 11b. If a queue is being chosen: all queues other than those with the highest priority (CLWLPRTY) are eliminated, and channels are kept.

Cluster Workload Algorithm (cont)

- 11c. All channels except a number of channels with the highest values in MQWDR.DestSeqNumber are eliminated. If this number is greater than the maximum allowed number of most-recenently-used channels (CLWLMRUC), the least recently used channels are eliminated until the number of remaining channels is no greater than CLWLMRUC.
- 12. If more than one remote instance of a queue remains, the least recently used channel is chosen (that is, the one with the lowest value in MQWDR.DestSeqFactor). If there is more than one with the lowest value, one of those with the lowest value in MQWDR.DestSeqNumber is chosen. The destination sequence factor of the choice is increased by the queue manager, by approximately 1000/(Channel weight) (CLWLWGT). The destination sequence factors of all destinations are reset to zero if the cluster workload attributes of available destinations are altered, or if new cluster destinations become available. Also, the destination sequence number of the choice is set to the destination sequence number of the previous choice plus one, by the queue manager.

Note that the distribution of user messages is not always exact, because administration and maintenance of the cluster causes messages to flow across channels. This can result in an apparent uneven distribution of user messages which can take some time to stabilize. Because of this, no reliance should be made on the exact distribution of messages during workload balancing.
Goals of Clustering

- Multiple Queues with single image
- Failure isolation
- Scalable throughput
- MQI applications to exploit clusters transparently
- Definition through usage (MQOPEN)
- MQGET always local

Consider a client using the black queue that is available in the cluster on three server queue managers. A message is MQPUT by the client and is delivered to "one" of the servers. It is processed there and a response message sent to a ReplyToQueue on the client queue manager.

In this system, if a server becomes unavailable, then it is not sent any further messages. If messages are not being processed quickly enough, then another server can be added to improve the processing rate.

It is important that both these behaviors are achieved by existing MQI applications, i.e. without change. It is also important that the administration of clients and servers is easy. It must be straightforward to add new servers and new clients to the server.

We see how a cluster can provide a highly available and scalable message processing system. The administration point in processing is MQOPEN as this is when a queue or queue manager is identified as being required by an application.

Note that only one message is sent to a server; it is not replicated three times, rather a specific server is chosen and the message sent there. Also note that MQGET processing is still local, we are not extending MQGET into the network.
Availability

- Clustering isn’t an HA solution
  - It can keep things up and running for you though used appropriately

- Loss of a destination queue means any messages there stranded – possibly gone for ever in worst case
  - One of the reasons avoiding ordering requirement at application level preferable

- 2 FRs means loss of one not critical to smooth running of cluster
  - 60 day grace period for existing knowledge if both down

- HA is having 2, DR is having them a long way apart
  - More seriously, HA is keeping things running, DR is recovering when HA has failed.

- WMQ, and particularly clustering, are good at getting you back up and running (dealing with new messages) rapidly when there is a failure. Having multiple destinations in a cluster (prioritized if preferred) gives you this almost for free!

- Software HA (Distributed) and Shared Queues (z/OS) are the best built in offering for keeping your existing messages available.
**Comparison of Technologies**

<table>
<thead>
<tr>
<th></th>
<th>Access to existing messages</th>
<th>Access for new messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Queues, HP NonStop Server</td>
<td>continuous</td>
<td>continuous</td>
</tr>
<tr>
<td>MQ Clusters</td>
<td>none</td>
<td>continuous</td>
</tr>
<tr>
<td>HA Clustering, Multi-instance</td>
<td>automatic</td>
<td>automatic</td>
</tr>
<tr>
<td>No special support</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

- This picture shows one view of the different capabilities. However you also need to consider factors such as total hardware/software price, the requirement for nonpersistent message availability (remember that they are discarded by a failover-restart), and the requirement for persistent message availability (not if you're using the shared queue support).
One sample implementation

- An account transfer request is initiated from channels and enter MQ gateway on AIX as a message.
- Request message is distributed (eg. evenly) to either MQ queue manager on z/OS.
  - MQ cluster workload balance.
- Request message arrives in shared queue,
  - Either having long running CICS transactions on multiple CICS regions to GET wait request messages from shared queue directly
    - for heavy workloads
  - Or, trigger to start transactions on multiple CICS regions to process request messages from shared queue
    - For less frequent workloads
- After transfer transaction is done, result is sent by CICS transaction back to the requesting client.
- Even if any node is down, business continuity is still guaranteed.
What’s New in WebSphere MQ Clustering

- **WebSphere MQ V7.1 enhancements**
  - Workload balancing by message group
  - Cluster queue monitoring/rebalancing sample
  - Pub sub cluster controls
  - Cluster Queue Security

- **WebSphere MQ V7.5 enhancements**
  - Split Cluster Transmit Queue
Workload balancing by message group

- **Before:**
  - To keep message groups together
  - Had to use BIND_ON_OPEN
  - MQOPEN/MQCLOSE around each group

- **Now:**
  - New MQOPEN option in the MQI
    - MQOO_BIND_ON_GROUP
  - Queue manager managed groups supported
    - Must specify MQPMO_LOGICAL_ORDER at put time
  - ‘Out of order/application managed groups not supported, so:
    - Must NOT manually specify group ID (leave as MQGI_NONE).
    - Will fall back to BIND_NOT_FIXED
  - Ungrouped messages
    - Will fall back to BIND_NOT_FIXED

---

Previously, when WebSphere MQ queue manager clusters have been used to balance messages across multiple instances of a queue hosted on different queue managers, two ‘binding options’ have been provided: BIND_ON_OPEN which ensures that all messages for the lifetime of the object handle are routed to the same instance, and BIND_NOT_FIXED which allows ‘spraying’ across multiple instances.

A third option is now available, BIND_ON_GROUP, to maintain integrity of groups of messages, while still exploiting workload balancing. It applies to queue manager managed groups, but not to manually specified groups of messages. In other words, you must use MQPMO_LOGICAL_ORDER.

Workload balancing algorithm unchanged, but will only be driven between complete groups when using the new option.

Allows exploitation of groups in usual manner on MQGET:
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_LOGICAL_ORDER

Note: As with all usage of groups on z/OS, must specify: INDXTYPE(GROUPID)
**Workload Balancing by Group: Example**

- `MQCONN QM1`
- `MQOPEN QX
  - MQOO_BIND_ON_GROUP`
- `MQPUT Msg1
  - MQMF_MSG_IN_GROUP`
- `MQPUT Msg2
  - MQMF_LAST_MSG_IN_GROUP`
- `MQPUT Msg3
  - MQMF_MSG_IN_GROUP`
- `MQPUT Msg4
  - MQMF_LAST_MSG_IN_GROUP`

In the diagram we see that all the messages in the first group go to the instance of queue QX hosted on QM2. A workload balancing decision is made when a new group is started, and then that chosen instance is used for all the remaining messages in the group, because MQOO_BIND_ON_GROUP has been used.

When a new group is started, the next message in a group following a message marked with MQMF_LAST_MSG_IN_GROUP, the we choose a new instance of queue QX, and this time the messages in the group all go to queue manager QM3.
**Cluster Monitoring and Rebalancing**

*Under normal running...*

Messages workload balanced across the clustered queue

*When things go wrong...*
Cluster Queue Monitoring Sample

- Tool amqsc1m + source
  - Cluster messages are directed to instances of cluster queues with applications consuming from them.
  - Messages on queues with no active consuming application are moved

- AMQSC1M Logic
  - The monitoring process polls the state of the queues on a defined interval.
  - If no consumers are attached:
    - CLWLPRTY of the queue is set to zero (if not already set).
    - The cluster is queried for any active (positive CLWLPRTY) queues.
    - If they exist, any queued messages on this queue are got/put to the same queue name. Cluster workload balancing will re-route the messages to the active instance(s) of the queue in the cluster.
  - If consumers are attached:
    - CLWLPRTY of the queue is set to one (if not already set).

- Availability
  - WMQ 7.0.1 fixpack 8
  - WMQ 7.1 (distributed platforms)
  - Or supported as any other user compiled application

A new tool amqsc1m is provided to ensure messages are directed towards the instances of clustered queues that have consuming applications currently attached. This allows all messages to be processed effectively even when a system is asymmetrical (i.e. consumers not attached everywhere).

- In addition it will move already queued messages from instances of the queue where no consumers are attached to instances of the queue with consumers. This removes the chance of long term marooned messages when consuming applications disconnect.
- The above allows for more versatility in the use of clustered queue topologies where applications are not under the direct control of the queue managers. It also gives a greater degree of high availability in the processing of messages.
- The tool provides a monitoring executable to run against each queue manager in the cluster hosting queues, monitoring the queues and reacting accordingly.
  - The tool is provided as source (amqsc1m.c sample) to allow the user to understand the mechanics of the tool and customise where needed.
- The AMQSC1M logic is based on the existing MQ cluster workload balancing mechanics and uses the cluster priority of individual queues. If all else is equal, preferring to send messages to instances of queues with the highest cluster priority (CLWLPRTY). Using CLWLPRTY always allows messages to be put to a queue instance, even when no consumers are attached to any instance. Changes to a queue’s cluster configuration are automatically propagated to all queue managers in the cluster that are workload balancing messages to that queue.

The monitoring process polls the state of the queues on a defined interval.

- If no consumers are attached:
  - CLWLPRTY of the queue is set to zero (if not already set).
  - The cluster is queried for any active (positive cluster priority) queues.
  - If they exist, any queued messages on this queue are got/put to the same queue. Cluster workload balancing will re-route the messages to the active instance(s) of the queue in the cluster.
- If consumers are attached:
  - CLWLPRTY of the queue is set to one (if not already set).
- Single executable, set to run against each queue manager with one or more cluster queues to be monitored. Defining the tool as an MQ service will ensure it is started with each queue manager
**When the queues are monitored...**

1. A consuming application stops
   - Consuming application

2. The stopped application is detected and the queue is deactivated
   - Queue manager

3. New messages are workload balanced to the remaining active queue
   - Queue manager

4. Queued messages are re-driven through the local queue manager’s workload balancer
   - Cluster queue

**Limitations**

- The monitoring tool is poll based.
  - The more queues to monitor the more overhead on the system - increased poll interval is recommended.
- Frequently connecting/disconnecting consuming applications will result in message churn and cluster repository overhead.
  - Tool is really suited to a set of long running consuming apps.
- Exclusive use of CLWLPRTY by the tool is required.
- If marooned messages are being relocated, any ‘BIND’ instructions are ignored.
- Monitoring is based purely on connected consuming applications, ‘slow’ consumers are not catered for.
- For simplicity the tool monitors queues within a single cluster.
Publish/Subscribe Cluster Controls

- **Pub/Sub Clusters – MQ V7.0**
- **Higher clustering overhead**
  - Rate of change of subscriptions
  - Existing interconnectivity
  - Publication rates
  - Number of channels supported / required.
- **Recommendations**
  - Create separate clusters for Pub/Sub
  - Don’t advertise topics in existing large queue clusters

- **How can we enforce this?**
- **Queue manager attribute**
  - PSCLUS (ENABLED/DISABLED)
- **Disables**
  - definition of cluster topic objects
  - sending/receiving of proxy subscriptions
- **Cannot be disabled if cluster topics already present somewhere in the cluster**
  - This is an up front ‘policy enforcement’ measure, not a quick fix!
- **Ideally set on every queue manager if no pub sub to be used.**
  - However configuring at least full repositories gives majority of protection. This also disables the ‘everybody learns about everybody’ aspect of a publish/subscribe cluster.

Publish Subscribe Cluster controls

- Publish Subscribe clusters introduced in WMQ Version 7 give a powerful way to combine the ‘topic space’ – pub/sub domain - across multiple queue managers
- However, publish subscribe is a much more dynamic environment than traditional point to point messaging, and this means in large or tightly stretched deployments the additional overhead can be undesirable
- Therefore for existing large point to point clusters the recommendation has been to avoid suddenly introducing clustered Topic objects (instead creating new cluster networks for any pub/sub activity), and many customers have taken this advice on board in their internal processes.
- However, until now there has been no way to enforce this…
Publish Subscribe Cluster controls

- Conference sessions previously have advised against using topics in ‘large’ clusters. 100 queue managers is a very approximate suggestion for ‘large’, but in reality what problems will be seen varies hugely depending on the usage and load on the system – 5 QMs may be ‘large’, or 500 may run acceptably.

- Things to consider
  - Rate of change of subscriptions – huge factor if lots of new subscriptions constantly trigger new proxy subs.
  - Existing interconnectivity (from p2p traffic). Full repositories probably already connected to everyone for example!
  - Publication rates
  - Number of channels supported / required. Any QM hosting subscriber will need sender channel to every other QM.

- Strong recommendation to create new pub sub infrastructure and scale gradually bearing all these considerations in mind.

---

New attribute at queue manager level controls whether or not this QM will participate in pub/sub clustering.

- PSCLUS (ENABLED/DISABLED)

Disables the definition of cluster topic objects, and the sending/receiving of proxy subscriptions.

Cannot be disabled if cluster topics already present (even defined elsewhere in the cluster).

- This is an up front ‘policy enforcement’ measure, not a quick fix!

Ideally set on every queue manager if no pub sub to be used.

- However configuring at least full repositories gives majority of protection. This also disables the ‘everybody learns about everybody’ aspect of a publish/subscribe cluster.
Cluster Queue Security in WebSphere MQ V7.1

- Cluster queue hosted locally
  - Access check on named queue profile
  - No change from before MQ V7.1

Cluster Queue Security in WebSphere MQ V7.1

- Cluster queue hosted locally
  - Access check on named queue profile
  - No change from before MQ V7.1

- Cluster queue hosted remotely
  - Access check on named queue profile
  - Location of hosted queue now makes no difference to access control
  - New in MQ V7.1 on Dist
  - Previously checked the SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Controlled in qm.ini using
    ClusterQueueAccessControl = Xmitq | QMName
Cluster Queue Security in WebSphere MQ V7.1

- Cluster queue hosted locally
  - Access check on named queue profile
  - No change from before MQ V7.1

- Cluster queue hosted remotely
  - Access check on named queue profile
  - **Location of hosted queue now makes no difference to access control**
  - New in MQ V7.1 on Dist
  - Previously checked the SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Controlled in qm.ini using
    
    ```
    ClusterQueueAccessControl = Xmitq | RQMName
    ```

- Fully qualified cluster queue
  - Access check on named queue manager profile
  - New in MQ V7.1 on Dist
  - Previously checked the SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Same qm.ini control (as above)
  - New profile type rqmname

```setAuthrec offspring(profile(QM2)
group(‘grp1’) authadd(put))
```

setmqaut -m QM1 -t rqmname -n QM2 -g grp1 +put

If your application is opening a cluster queue which is hosted remotely then on z/OS and on Distributed platforms from WebSphere MQ 7.1 and onwards, the named object is the profile that is checked. On distributed platforms prior to WebSphere MQ V7.1, the profile for the SYSTEM.CLUSTER.TRANSMIT.QUEUE was checked.

If your application is opening a cluster queue using the fully-qualified technique, then on z/OS and on Distributed platforms from WebSphere MQ 7.1 and onwards, a profile which represents the ‘ToQmgr’ is checked. On distributed platforms prior to WebSphere MQ V7.1, the profile for the SYSTEM.CLUSTER.TRANSMIT.QUEUE was checked.

Control or whether the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used or the new mode, is done by means of setting in the qm.ini file

```ClusterQueueAccessControl = Xmitq | RQMName```

The default in both cases is to continue to do the old way, checking the SYSTEM.CLUSTER.TRANSMIT.QUEUE, but once you are ready and have made the appropriate new profiles, this change can be made and the queue manager restarted in the new mode.

For the fully-qualified case, there is a new profile type called ‘rqmname’ which is used to set accesses for remote queue managers.
**Split cluster transmit queue**

- **Much requested feature for various reasons...**
- **Separation of Message Traffic**
  - With a single transmission queue there is potential for pending messages for cluster channel ‘A’ to interfere with messages pending for cluster channel ‘B’
- **Management of messages**
  - Use of queue concepts such as MAXDEPTH not useful when using a single transmission queue for more than one channel.
- **Monitoring**
  - Tracking the number of messages processed by a cluster channel currently difficult/impossible using queue monitoring (some information available via Channel Status).
- **Not about performance...**

---

**Split cluster xmit q background - Notes**

- This has been a very long standing requirement from a number of customers
- All the reasons on this slide are valid, but the number one reason often quoted in requirements was ‘performance’
  - In reality splitting out the transmit queue does not often buy much here, hence often other solutions (e.g. improving channel throughput) were really needed.
- Main reason for delivery now is to allow application separation
**Split cluster transmit queue - automatic**

- New Queue Manager attribute which effects all cluster-sdr channels on the queue manager
  - ALTER QMGR DEFCLXQ( SCTQ | CHANNEL )

- Queue manager will automatically define a PERMANENT-DYNAMIC queue for each CLUSSDR channel.
  - Dynamic queues based upon new model queue “SYSTEM.CLUSTER.TRANSMIT.MODEL”
  - Well known queue names: “SYSTEM.CLUSTER.TRANSMIT.<CHANNEL-NAME>”

- Authority checks at MQOPEN of a cluster queue will still be made against the SYSTEM.CLUSTER.TRANSMIT.QUEUE even if CLUSSDR is selected.

---

**Splitting out the S.C.T.Q. per channel**

[Diagram of splitting out the S.C.T.Q. per channel]
Split cluster transmit queue - manual

- Administrator manually defines a transmission queue and using a new queue attribute defines the CLUSSDR channel(s) which will use this queue as their transmission queue.
  - DEFINE QLOCAL(APPQMGR.CLUSTER1.XMITQ)
    CHLNAME(CLUSTER1.TO.APPQMGR)
    USAGE(XMITQ)

- The CHLNAME can include a wild-card at the start or end of to allow a single queue to be used for multiple channels. In this example, assuming a naming convention where channel names all start with the name of the cluster, all channels for CLUSTER1 use the transmission queue CLUSTER1.XMITQ.
  - DEFINE QLOCAL(CLUSTER1.XMITQ) CHLNAME(CLUSTER1.*) USAGE(XMITQ)
  - Multiple queues can be defined to cover all, or a subset of the cluster channels.

- Can also be combined with the automatic option
  - Manual queue definition takes precedence.

Splitting out by cluster (or application)

Cluster 1

QM_A

QM_B

Cluster 2

QM_A

QM_B

CLUSTER1.QM_B

CLUSTER2.QM_B

Q1

Q2