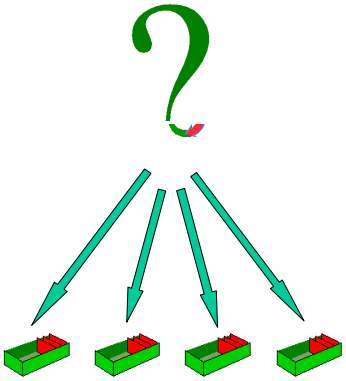
WebSphere MQ Queue Manager Clusters

This Slide Intentionally Left Blank

Agenda

- The purpose and goals of MQ clustering
- How Clustering Works
- Workload Management Considerations
- Full Repository Considerations
- Pub/Sub Clusters
- What's New in V7.1 and V7.5
- Summary



N

O

T

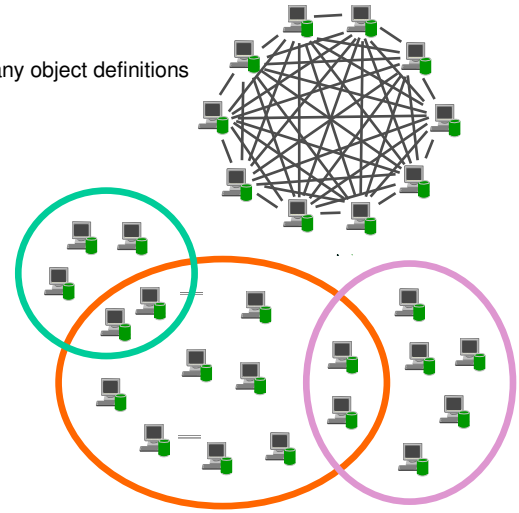
E

S

- The term "cluster" means different things to different people, as there are a large number of technologies that use the term and a large number of products that offer "clustering".
- Clustering is usually associated with parallel processing or high availability technologies.
- There exist many different types of parallel processing architectures. The amount of symmetry between the processing nodes varies between architectures.
- Although WMQ clustering does provide a level of high availability, its raison d'etre is as a WMQ parallel processing feature, NOT as a high availability solution.
- Systems like z/OS Parallel Sysplex consist of complexes of processors, each complex comprising numerous, but same type processors. The complex is connected together using a coupling facility, which, as well as allowing high speed communication, also allows efficient data sharing. Most parallel architectures do not have this type of data sharing capability.
- The most generalized parallel architectures involve processors of different types, running different operating systems connected together using different network protocols. This necessarily includes symmetric systems.
- A WebSphere MQ cluster is most similar to the most generalized parallel architecture. This is because WebSphere MQ exploits a wide variety of platforms and network protocols. This allows WebSphere MQ applications to naturally benefit from clustering.
- WebSphere MQ clusters are solve a requirement to group queue managers together (i.e. to increase processing power of the WMQ network) whilst minimising the administration costs associated with WMQ queue manager intercommunication.

The purpose of clustering

- **Simplified administration**
  - Large WMQ networks require many object definitions
    - Channels
    - Transmit queues
    - Remote queues
- **Workload Management**
  - Spread the load
  - Route around failures
- **Flexible connectivity**
  - Overlapping clusters
  - Gateway Queue managers
- **Pub/Sub Clusters**



N O T E S

- It would be nice if we could place all the queues in one place. We could then add processing capacity around this single Queue manager as required and start multiple servers on each of the processors. We would incrementally add processing capacity to satisfy increased demand. We could manage the system as a single entity. A client application would consider itself to be talking to a single Queue manager entity.
- Even though this is highly desirable, in practice it is almost impossible to achieve. Single machines cannot just have extra processors added indefinitely. Invalidation of processor caches becomes a limiting factor. Most systems do not have an architecture that allows data to be efficiently shared between an arbitrary number of processors. Very soon, locking becomes an issue that inhibits scalability of the number of processors on a single machine. These systems are known as "tightly coupled" because operations on one processor may have a large effect on other processors in the machine cluster.
- By contrast, "loosely coupled" clusters (e.g. the Internet) have processors that are more or less independent of each other. Data transferred to one processor is owned by it and is not affected by other processors. Such systems do not suffer from processor locking issues. In a cluster solution, there are multiple consumers of queues (client queue managers) and multiple providers of queues (server queue managers). In this model, for example, the black queue is available on multiple servers. Some clients use the black queue on both servers, other clients use the black queue on just one server.
- A cluster is a loosely coupled system. Messages flow from clients to servers and are processed and responses messages sent back to the client. Servers are selected by the client and are independent of each other. It is a good representation of how, in an organization, some servers provide many services, and how clients use services provided by multiple servers.
- The objective of WebSphere MQ clustering is to make this system as easy to administer and scale as the Single Queue Manager solution.

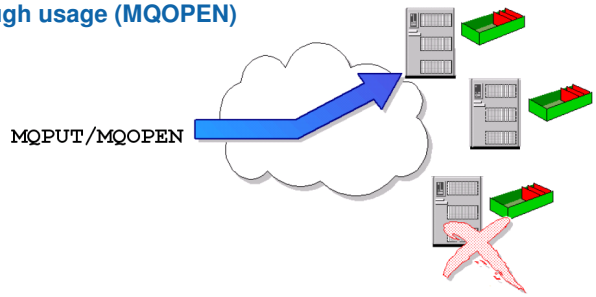
Capitalware's MQ Technical Conference v2.0.1.3

What Do We Mean By "Clustering"?

This Slide Intentionally Left Blank

Goals of Clustering

- Multiple Queues with single image
- Failure isolation
- Scalable throughput
- Applications to exploit clusters transparently
- Auto-definition through usage (MQOPEN)
- MQGET always local

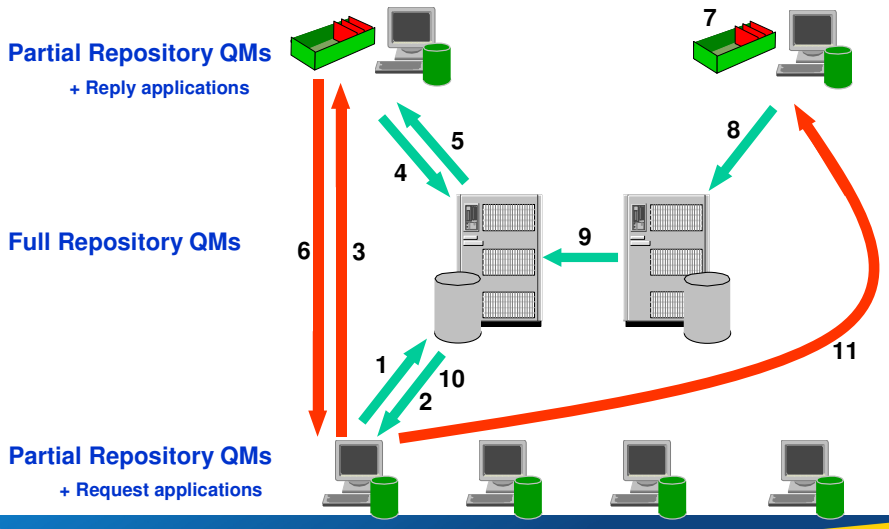


N O T E S

- Consider a client using the 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 straight forward 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.

Capitalware's MQ Technical Conference v2.0.1.3

WMQ Cluster Architecture



N

O

T

E

S

- Let us walk through the flow of a message from a request queue manager to a reply queue manager and the response message that is returned. We assume that the request app has never used the queue providing the service previously, and that the request app has not communicated with the request app previously.
- Clustering introduces a new architectural layer, the full repository and partial repository queue managers, purely for the sake of explanation. Full repository queue managers are not separate queue managers (contrast to DNS servers), and their role is to serve as a global directory of queues and queue managers in the cluster. There are a small number of these full repositories. Each request and reply queue manager have a partial repository. In practice, full repository queue managers often host applications too.
- At MQOPEN, the queue manager to which the application is connected detects that this queue has not been used by this queue manager previously. The queue manager sends an internal message (1) requesting the location of the servers for the green queue and channel definitions to get to these servers. The returned definitions (2) are installed on the request queue manager, a channel is automatically defined to the reply queue manager (3).
- Similar processing occurs at the request side, to locate the requestor's ReplyToQueue manager.
- The most frequent method used by a reply app to send a response message to a requestor is to use the routing information in the message descriptor, namely the ReplyToQ and ReplyToQMgr. The reply app's requirement is slightly different to the original request, since the originating application's ReplyToQ is normally private to its Queue manager, i.e. it is not visible to the whole cluster. In this case the server needs to be able to locate the ReplyToQMgr rather than the ReplyToQ.

N

O

T

E

S

- This happens as follows. When an MQPUT1 or MQOPEN request is made to send a message to a ReplyToQMgr for the first time, the queue manager sends an internal message (4) to the repository requesting the channel definition required to reach the client. The returned definition (5) is used to automatically define a channel to the request queue manager (6) to get the message back to the request queue manager where local queue resolution puts the message on the ReplyToQ.
- Finally, we see what happens when some attributes of a cluster queue or cluster Queue manager change. One interesting case is the creation of a new instance of a cluster queue manager holding a cluster queue being used by a request (7). This information is propagated from the reply queue manager (8). The full repository propagates the definition to the other full repository (9) and the full repository propagates it to any interested request QMs through the repository network (10), allowing this new instance to be used during an MQOPEN, MQPUT or MQPUT1 call (11).
- Note that channels are only automatically defined once on the first MQOPEN that resolves to a queue on the remote queue manager, not for every MQOPEN or MQPUT.

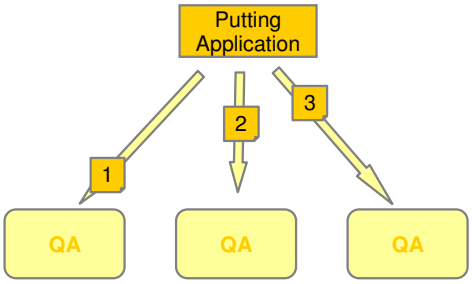
This Slide Intentionally Left Blank

Who Called it "Workload Balancing"?

This Slide Intentionally Left Blank

Workload Management

- MQ Clusters often implemented to do workload-balancing
- Workload Management algorithm used to accomplish this
  - Provides a means of distributing workload across set of queues
  - Default values intended to achieve an even distribution of messages
  - Distribution can be configured by users

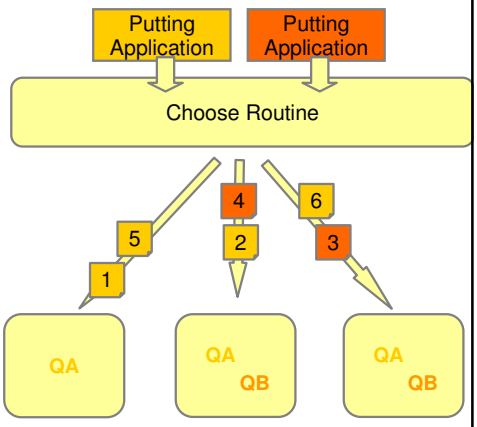


Why Isn't My Workload "Balanced"?

- Once implemented, distribution of messages may not be as expected
- Messages are being distributed across the queues, but not evenly
- May work fine in test, but in production a different behavior is observed
- Why is this?

Multiple applications sharing cluster channels

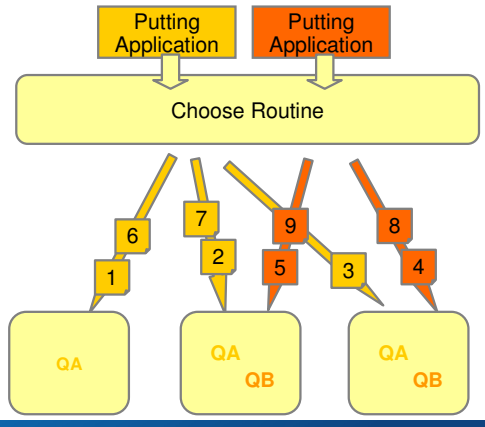
- Distribution of messages may not be as expected
- Seen when more than one putting application in the same queue manager is using cluster queues
- This can occur when frequency of puts between them follows a particular pattern
  - Occurs because the queue manager round-robin's messages based on the channel being used
    - Not the queue that the message is sent to
- Doesn't always occur; depends on the sequence of puts
- One way to understand this is…
  - Picture two queues, A and B, on two queue managers
  - If the puts always happen A, B, A, B, A, B, A, B, ...
    - Then all messages for A would go to one of the queue managers and all for B go to the other
  - This is due to the fact that MQ is round-robining across the two channels



What to do about it?

Using clusters to separate applications

- A new CLUSRCVR can be used to separate applications
- Queues A and B are hosted in different clusters to achieve this

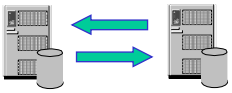


Considerations for Full Repositories

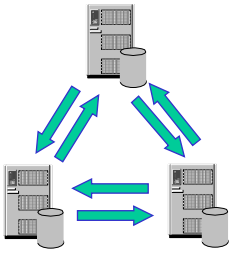
This Slide Intentionally Left Blank

Considerations for Full Repositories (FRs) - 1

- **FRs should be 'reasonably' available**
  - Avoid single point of failure
  - **Recommended to have exactly 2 unless you know of a very good reason to have more**
  - Put them on machines with good availability.
  - Multi-instance queue managers can be considered, but are probably not required



- **FRs must be fully inter-connected**
  - Using manually defined cluster sender channels



- **If at least one FR is not available or they are not fully connected**
  - Cluster definition changes via FRs will not flow
  - User messages between Partial Repositories over existing channels will flow

Why Two and Only Two Queue Managers?

N O T E S

- Full Repositories must be fully connected with each other using manually defined cluster sender channels.
- You should always have at least 2 Full Repositories in the cluster so that in the event of a failure of a Full Repository, the cluster can still operate. If you only have one Full Repository and it loses its information about the cluster, then manual intervention on all queue managers within the cluster will be required in order to get the cluster working again. If there are two or more Full Repositories, then because information is always published to and subscribed for from 2 Full Repositories, the failed Full Repository can be recovered with the minimum of effort.
- Full Repositories should be held on machines that are reliable and highly available. This said, if no Full Repositories are available in the cluster for a short period of time, this does not affect application messages which are being sent using the clustered queues and channels, however it does mean that the clustered queue managers will not find out about administrative changes in the cluster until the Full Repositories are active again.
- For most clusters, 2 Full Repositories is the best number to have. If this is the case, we know that each Partial Repository manager in the cluster will make its publications and subscriptions to both the Full Repositories.

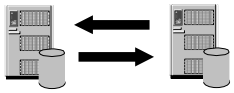
NOTES

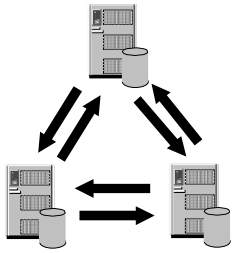
- It is possible to have more than 2 Full Repositories.
- The thing to bear in mind when using more than 2 Full Repositories is that queue managers within the cluster still only publish and subscribe to 2. This means that if the 2 Full Repositories to which a queue manager subscribed for a queue are both off-line, then that queue manager will not find out about administrative changes to the queue, even if there are other Full Repositories available. If the Full Repositories are taken off-line as part of scheduled maintenance, then this can be overcome by altering the Full Repositories to be Partial Repositories before taking them off-line, which will cause the queue managers within the cluster to remake their subscriptions elsewhere.
- If you want a Partial Repository to subscribe to a particular Full Repository queue manager, then manually defining a cluster sender channel to that queue manager will make the Partial Repository attempt to use it first, but if that Full Repository is unavailable, it will then use any other Full Repositories that it knows about.
- Once a cluster has been setup, the amount of messages that are sent to the Full Repositories from the Partial Repositories in the cluster is very small. Partial Repositories will re-subscribe for cluster queue and cluster queue manager information every 30 days at which point messages are sent. Other than this, messages are not sent between the Full and Partial Repositories unless a change occurs to a resource within the cluster, in which case the Full Repositories will notify the Partial Repositories that have subscribed for the information on the resource that is changing.
- As this workload is very low, there is usually no problem with hosting the Full Repositories on the server queue managers. This of course is based on the assumption that the server queue managers will be highly available within the cluster.
- This said, it may be that you prefer to keep the application workload separate from the administrative side of the cluster. This is a business decision.

This Slide Intentionally Left Blank

Considerations for Full Repositories (FRs) - 2

- **Should applications run on full repositories? (Should they host 'data' queues?)**
  - Best Practice hat on: No
  - Consider the risks (see Notes) and decide on what is appropriate given your environment
- **Where possible, dedicate FRs to a given cluster**
  - Particularly important for pub sub given that PSCLUS is now available
- **What if I need to take them down for maintenance?**
  - Use the fact that you have two!





N O T E S

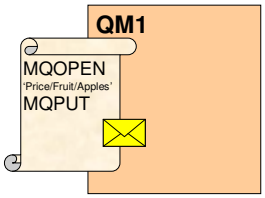
- The previous slide gave the 'standard' rules and reasons for working with full repository, but here are some tips based on the way people really tend to work with them and some common issues:
- There is no reason applications cannot happily run on a queue manager which is acting as a full repository, and certainly the original design for clustering assumes this will probably be the case. HOWEVER, many people actually prefer to keep FRs dedicated to just maintaining the cluster cache, for various reasons:
  - When any application in the cluster wants to use new features, can upgrade FRs without having to test ALL co-located applications
  - If for some reason you need to apply urgent maintenance to your full repositories they can be restarted or REFRESHed without touching applications
  - As clusters grow and demands on cache maintenance become heavier, there is no risk of this affecting application performance (through storage, CPU demands for example)
  - Full repositories don't actually need to be hugely powerful – a simple Unix server with a good expectation of availability is sufficient.
- Maintenance:
  - This is precisely the sort of reason you want 2 full repositories. The cluster will continue to function quite happily with one repository, so where possible bring them down and back up one at a time. Even if you experience an outage on the second, running applications should be completely unaffected for a minimum of three days
- Moving full repositories
  - Is a bit trickier than moving a regular queue manager. The migration foils look into this further

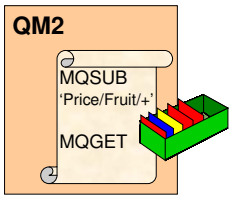
Publish/Subscribe Clusters

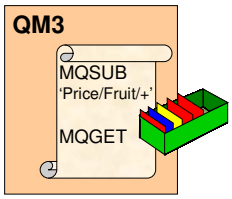
This Slide Intentionally Left Blank

Publish/Subscribe Topologies

- **Distributed Publish/Subscribe**
  - Application API calls remain the same
  - Administration changes have the effect
- **Topology Options**
  - Hierarchical
  - Cluster





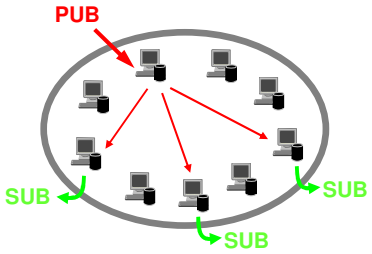


N O T E S

- Just as queuing can be on a single queue manager or can be moving messages between multiple queue managers – known as distributed queuing – so can publish/subscribe. We call this distributed publish/subscribe.
- This is the concept (and the features to implement it) that an application may be publishing to a topic on QM1 and other applications may be subscribing on that topic on others queue managers, here QM2 and QM3, and the publication message needs to flow to those other queue managers to satisfy those subscribers.
- The application code stays the same, you still call MQSUB or MQOPEN and MQPUT, the difference, as with distributed queuing is in the administrative set-up of your queue managers.
- We are going to look at the different ways you can set up your queue managers to publish messages to another queue manager.
- Two options are available for implementing a distributed publish/subscribe network – one is hierarchical, the other based on MQ Clusters. We will be exploring the latter in this presentation.

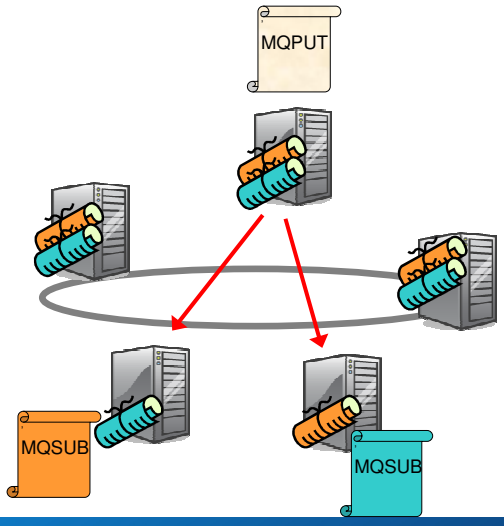
Pub/Sub Clusters

- **Similar to WebSphere Message Broker Collectives**
- **Availability – Direct links between qmgrs**
- **Benefits of traditional clusters**
  - Simplified administration
    - Useful for Pub/Sub Clusters
    - Non-disruptive add/remove of qmgrs
  - Workload management
    - NOT useful for Pub/Sub Clusters
- **Cluster channels**
  - Same CLUSSDR/CLUSRCVR model
  - Flexible connectivity (All to All connectivity)
  - Overlapping clusters
    - No Pub/Sub through gateway queue managers
      - Use Hierarchies for linking Pub/Sub Clusters
- **Cluster topics**
  - Required for publishing to remote Pub/Sub Cluster queue managers
- **Cluster queues**
  - Can be used in traditional way in a Pub/Sub Cluster



This Slide Intentionally Left Blank

Pub/Sub Clusters



TOPIC attributes

CLUSTER
SUBSCOPE
PUBSCOPE
PROXYSUB

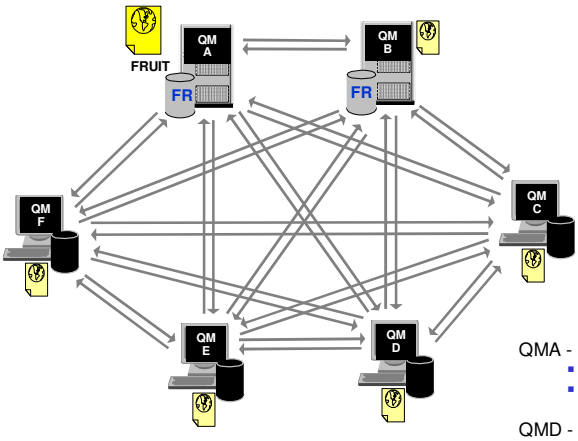
```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.

DEFINE TOPIC(APPLES)
       TOPICSTR('Price/Fruit/Apples')
       CLUSTER(DEMO)

DISPLAY SUB(*) SUBTYPE(PROXY) ALL
    1 : DISPLAY SUB(*) SUBTYPE(PROXY) ALL
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D312020202020202020204F57864820000F02)
   SUB(SYSTEM.PROXY.QM2 DEMO Price/Fruit/Apples)
   TOPICSTR(Price/Fruit/Apples)             TOPICOBJ( )
   DEST(SYSTEM.INTER.QMGR.PUBS)             DESTQMGR(QM2)
   DESTCLAS(PROVIDED)                       DURABLE(YES)
   SUBSCOPE(ALL)                            SUBTYPE(PROXY)
```


N O T E S

- A pub/sub cluster is a cluster of queue managers, with the usual CLUSRCVR and CLUSSDR definitions, but that also contains a TOPIC object that has been defined in the cluster.
- With a cluster you have “any-to-any” connectivity. There are direct links between all queue managers in the cluster. This provides good availability for the delivery of messages, if one route is unavailable, there may well be another route to deliver the messages to the target subscription.
- With a TOPIC object defined in the cluster, an application connected to one queue manager in the cluster can subscribe to that topic or any node in the topic tree below that topic and receive publications on that topic from other queue managers in the cluster.
- This is achieved by the creation of proxy subscriptions on the queue managers in the cluster, so that when a publication to the topic in question happens on their queue manager, they know to forward it to the appropriate other members of the cluster.
- You can view these proxy subscriptions through the same commands we saw earlier. By default proxy subscriptions are not shown to you because the default value for SUBTYPE is USER. If you use SUBTYPE(ALL) or SUBTYPE(PROXY) you will see these subscriptions.
- There are a few attributes that are specifically related to Distributed Publish/Subscribe. PUBSCOPE and SUBSCOPE determine whether this queue manager propagates publications to queue managers in the topology (pub/sub cluster or hierarchy) or restricts the scope to just its local queue manager. You can do the equivalent job programmatically using MQPMO_SCOPE_QMGR / MQSO_SCOPE_QMGR.
- PROXYSUB is an attribute that controls when proxy subscriptions are made. By default it has value FIRSTUSE and thus proxy subscriptions are only created when a user subscription is made to the topic. Alternatively you can have the value FORCE which means proxy subscriptions are made even when no local user subscriptions exist.

Pub/Sub Cluster Architecture 1

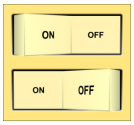
QMA - **DEFINE TOPIC(FRUIT) TOPICSTR(/global/price/fruit) CLUSTER(DEMO)**



QMA - **DIS TOPIC(FRUIT) TYPE(ALL)**
- Local topic
- Cluster topic

QMD - **DIS TOPIC(FRUIT) TYPE(ALL)**
- Cluster topic

The PSCLUS Queue Manager Property



- **New Queue Manager Property with WebSphere MQ V7.1**
- **Publish Subscribe clusters allow a topic space to span multiple queue managers**
- **Doing so in large or stretched deployments may result in unacceptable overhead**
- **For existing large point-to-point clusters the recommendation has been to avoid suddenly introducing clustered Topic objects**
  - But prior to V7.1 there was no way to enforce this – often leading to problems
- **So in V7.1 a new queue manager attribute controls whether or not the QM will participate in pub/sub clustering**
  - PSCLUS (ENABLED/DISABLED)
- **Disables the definition of cluster topic objects, and the sending/receiving of proxy subscriptions.**
- Ideally **set on every queue manager if no pub sub to be used**
  - However, configuring at least full repositories gives majority of protection
    - Disables the 'everybody learns about everybody' aspect of a Pub/Sub cluster.

The PSCLUS Queue Manager Property (continued)

- **When to set PSCLUS to DISABLED?**
  - When it is known (or suspected) that a cluster could not accommodate the overheads of publish/subscribe

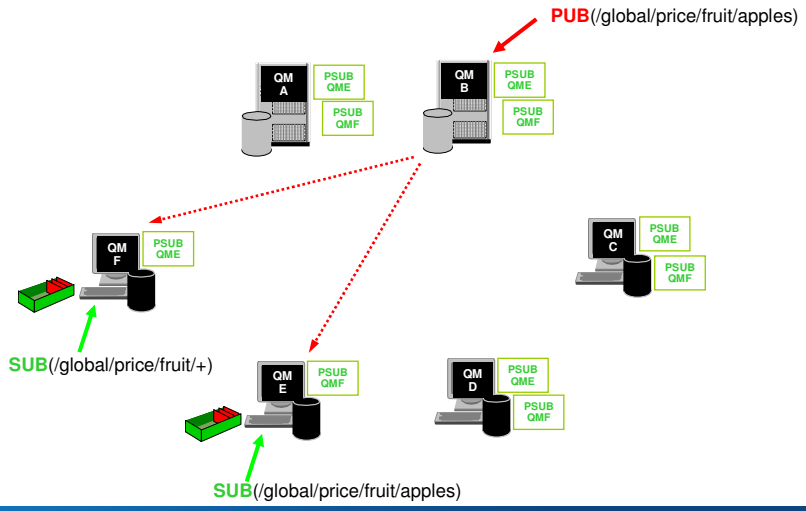
- **Setting PSCLUS to DISABLED modifies three aspects of queue manager functionality:**
  - An administrator of this queue manager is no longer able to define a Topic object as clustered.
  - Incoming topic definitions or proxy subscriptions from other queue managers are rejected (a warning message is logged to inform the administrator of incorrect configuration).
  - Full repositories no longer share information about every queue manager with all other partial repositories automatically when they receive a topic definition.

Publish Subscribe Cluster controls

N O T E S

- 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.

Pub/Sub Cluster Architecture 2



Pub/Sub Cluster Architecture 2

N O T E S

- We start this slide assuming that Pub/sub Cluster with cluster topic with topic string /global/sports already exists.
- The proxy-subscriptions are fanned out from the queue manager to which the subscriber is connected to all other queue managers.
- Notice that…
  - the cluster topic string is /global/price/fruit.
  - the subscribers' topic strings are /global/price/fruit/+ and /global/price/fruit/apples
    - Proxy-subscriptions are sent to other queue managers in the Pub/Sub cluster for any topic string below the cluster topic string.
  - the publisher's topic string is /global/price/fruit/apples
    - Publications are sent based on local proxy-subscriptions

Cluster Topics

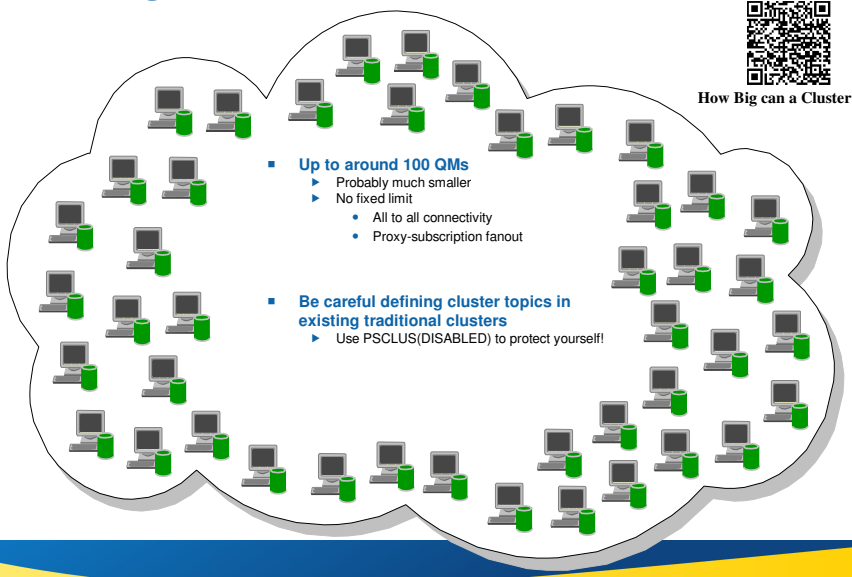
- **Required for publishing to remote Pub/Sub Cluster QMs**
- **Pub/Sub Cluster exists when there are one or more cluster topics**
  - Cluster topic is pushed to ALL queue managers in the cluster
  - Channels are automatically defined between ALL queue managers in the cluster
- **Not to be confused with a subscriber**
- **Define**
  - **DEFINE TOPIC(FRUIT) TOPICSTR(/global/price/fruit) CLUSTER(DEMO)**
- **Display**
  - **DISPLAY TOPIC(FRUIT)**
  - **DISPLAY TOPIC(FRUIT) TYPE(LOCAL)**
  - **DISPLAY TOPIC(FRUIT) TYPE(ALL)**
    - Local objects only
  - **DISPLAY TOPIC(FRUIT) TYPE(CLUSTER)**
  - **DISPLAY TOPIC(FRUIT) CLUSINFO**
  - **DISPLAY TCLUSTER(FRUIT)**
    - Cluster objects only
  - **DISPLAY TOPIC(FRUIT) TYPE(ALL) CLUSINFO**
    - Both local and cluster objects

Cluster Topics

N O T E S

- Cluster TOPICS are regular TOPIC objects that are advertised to the cluster. When a cluster TOPIC is defined, the cluster in which it is defined becomes a Pub/sub Cluster.
- Like traditional clusters, Pub/sub clusters are designed for many-many queue manager connectivity. In traditional clusters, cluster objects are automatically defined based on usage (e.g. put to a queue). The usage model is based on a putter using a set of cluster queues (not necessarily defined on all queue managers in the cluster). Therefore in traditional clusters it is unlikely that all queue managers will actually be connected to all other queue managers by auto-defined channels.
- In Pub/sub clusters, cluster objects are automatically defined before usage, at the time the first cluster TOPIC is defined in the cluster. This is because the usage model is different to that of traditional clusters. Channels are required from any queue manager to which a subscriber (for a cluster topic) is connected to all other queue managers so that a proxy-subscription can be fanned out to all the queue managers. Channels are also required back from any queue manager where a publisher (for a cluster topic) is connected to those queue managers which have a connected subscriber. Therefore in Pub/sub Clusters it is much more likely that all queue managers will actually be connected to all other queue managers by auto-defined channels. It is for this reason that, in Pub/sub clusters, cluster objects are automatically defined before usage.
- To define a cluster TOPIC, you must provide a topic object name, topic string, and cluster name. The queue manager on which the topic is defined should be a member of the specified cluster.
- When displaying cluster topics, there are two types:
  - Local: Directly administrable objects.
  - Cluster: Cluster cache records, based upon local objects.

How large can a Pub/Sub Cluster be?

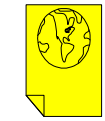


How Big can a Cluster Be?

- **Up to around 100 QMs**
  - Probably much smaller
  - No fixed limit
    - All to all connectivity
    - Proxy-subscription fanout
- **Be careful defining cluster topics in existing traditional clusters**
  - Use PSCLUS(DISABLED) to protect yourself!

Further Administration

- **Displays status of the match space**
  - **DISPLAY TPSTATUS('#')**
- **Display proxy-subscriptions**
  - **DIS SUB(*) SUBTYPE(PROXY)**
- **Display Pub/Sub Local and Hierarchy status**
  - **DIS PUBSUB TYPE(LOCAL / CHILD / PARENT / ALL)**
    - Displays QMNAME + STATUS
      - LOCAL -> ACTIVE / COMPAT / ERROR / INACTIVE / STARTING / STOPPING
      - CHILD -> ACTIVE / ERROR / REFUSED / STARTING / STOPPING
      - PARENT -> ACTIVE / ERROR / STARTING / STOPPING
      - ALL
- **Display Cluster status**
  - **DIS CLUSQMGR(*)**
- **Other cluster command**
  - **RESET CLUSTER** - Topics always reset
- **Monitor health of channels and health of transmit and subscriber queues**
  - **DIS CHSTATUS(*)**
  - **DIS QL(*) CURDEPTH**



Recommendations and Reference

- **Start small**
  - Use PSCLUS(DISABLED) until ready to implement
    - At least at the FRs, preferably at all queue managers
- **Don't put the root node (/) into a cluster!**
  - Make global topics obvious (e.g. /global or /cluster)
- **Monitor depth of transmit queues!!!**
  - SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Do you need durable subscribers?
  - Did you stop subscribers before stopping queue manager
- **Be careful mixing traditional clusters**
  - Large clusters and cluster topic objects leads to large number of channels
  - Admin controls on topic objects
- **Further Information in the Infocenter**
  - http://www.ibm.com/software/integration/wmq/library

Publish Subscribe

N O T E S

- Clustered Topics make extending your pub sub domain between queue managers easier than it's ever been, but…
- Start small
- Don't put the root node (/) into a cluster
  - Make global topics obvious (e.g. /global or /cluster)
  - Unless migrating…
- Be careful mixing traditional clusters
  - Large clusters and cluster topic objects leads to large number of channels
  - Admin controls on topic objects

What's New with Clustering in MQ V7.1?

This Slide Intentionally Left Blank

Reliability, Maintainability, Administration

- **Significant focus on improving the 'user experience' of clustering.**
  - Handling errors smoothly
  - Scaling reliably, maintaining performance
  - Clearly informing administrator of potential problems.
- **Some examples you will see in WMQ 7.1 and beyond**
  - More error and information messages in the logs and on the command line
    - e.g. restoring repository fails at queue manager start up – immediate message
  - Timely failure and diagnostic capture rather than 'limping on' and potentially hiding problems
    - 5 day warning on critical errors before queue manager shut down.
  - Major full repository performance enhancements.

Notes: Reliability improvements etc.

N O T E S

- One of the big shifts to note here is from allowing the queue manager to continue without a running repository manager. When the repos manager encounters a critical error it will now go into 'retry' for a few days, issuing warnings with a countdown timer, before shutting down the queue manager.
  - This is important because without a repository manager the cluster cache will grow 'stale' and applications may suddenly experience outages sometime in the future, when the root cause may be near impossible to diagnose.
  - On z/OS the behaviour is slightly different – because of MSTR/CHIN separation, and as historically here 'noticing' that the repos mgr has failed has not been a problem we allow the qmgr to continue running with warning messages in the logs.
- Get-Inhibiting the SYSTEM.CLUSTER.COMMAND.QUEUE suspends this process allowing the administrator to go away and resolve the issue (e.g. fix underlying hardware problem, contact IBM service).
- If REALLY desired tuning parameter TolerateRepositoryFailure can be used to allow queue manager to run without a repository manager process. This is not recommended!

Capitalware's MQ Technical Conference v2.0.1.3

Changes to Workload Management

- **Prior to MQ V7.1, when MQ queue manager clusters were used to distribute messages across multiple cluster queue instances, two "bind" options were available**
  - BIND_ON_OPEN which ensures that all messages for the lifetime of the object handle are routed to the same cluster queue instance
  - BIND_NOT_FIXED which allows 'spraying' across multiple instances.
- **These dictate when the workload management algorithm is invoked**
  - One, when the object is opened
  - Continually, whne each message is put
- **"All or Nothing" options did not allow for "in-between" situations, message groups being the primary example**
  - Handling these situations requires application programming involvement

Workload Management With MQ V7.1

- **MQ V7.1 introduces a third option into the mix:**
  - BIND_ON_GROUP allows workload distribution while maintaining the integrity of groups of messages
- **Note that the Workload mamagement algorithm itself is unchanged**
  - What is changed is when the workload management algorithm is invoked
  - With BIND_ON_GROUP, the algorithm will only be driven between complete groups
- **Include the following to take advantage of this option:**
  - Include open option MQOO_BIND_ON_GROUP on MQOPEN of putting application
    - Or, leave MQOO_AS_Q_DEF, and specify DEFBIND(GROUP) on the queue definition
  - Include put option MQPMO_LOGICAL_ORDER on MQPut
  - Include the following in flags within the MD of messages:
    - Last message in group - MQMF_LAST_MSG_IN_GROUP)
    - All other messages in group - MQMF_MSG_IN_GROUP

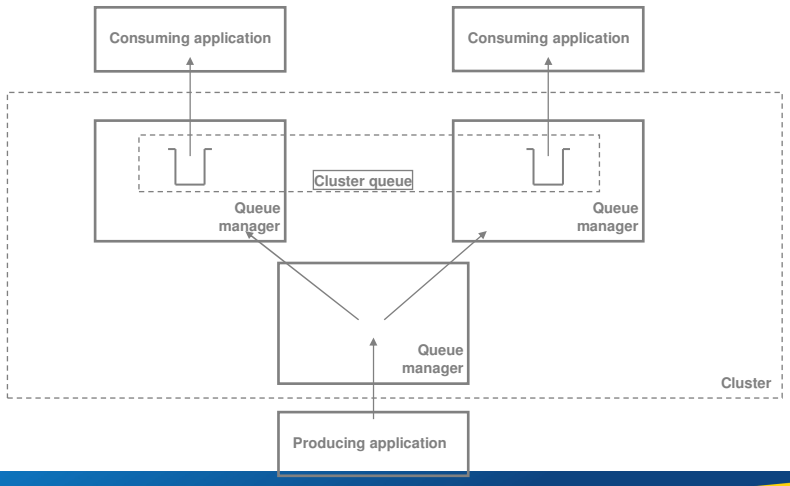
Additional Considerations and Restrictions

- **Does not support 'application managed' groups!**
  - Where attempting this would give an error today (e.g. MQRC_INCOMPLETE_GROUP) this will still be the case
  - In other situations, will fall back to BIND_NOT_FIXED behaviour
- **Do NOT manually specify group ID (set to MQGI_NONE)**
  - Specify MQPMO_LOGICAL_ORDER
  - Queue manager provides GroupId
- **If channel state changes etc, bound groups will NOT be considered eligible for reallocation**
  - This is to protect the integrity of a partially transmitted group of messages

This Slide Intentionally Left Blank

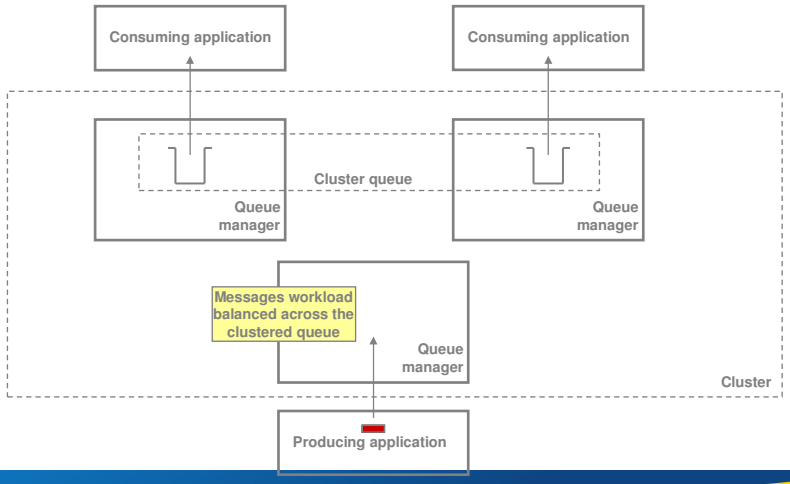
Cluster Workload Rebalancer

NEW: Cluster Monitoring and Rebalancing
A typical workload balancing scenario

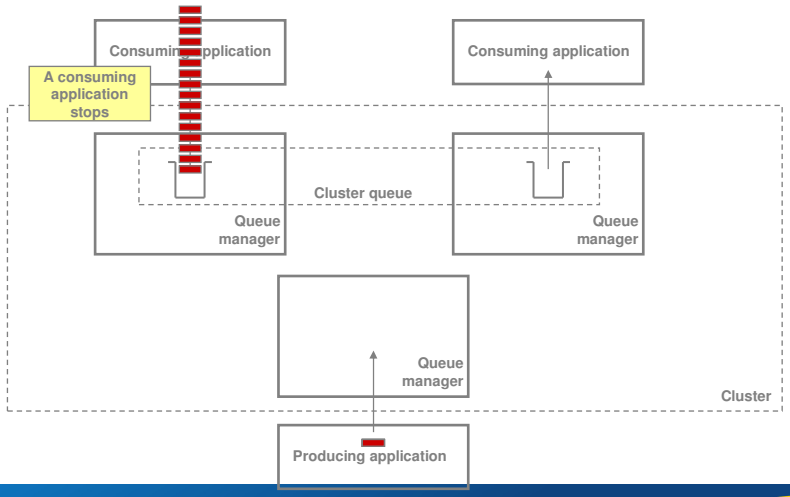


Capitalware's MQ Technical Conference v2.0.1.3

Under normal running…



When things go wrong…



AMQSCLM - Cluster Queue Monitoring Sample

- **Provided to ensure messages are directed towards the instances of clustered queues that have consuming applications currently attached**
  - Allows all messages to be processed effectively even when consumers not attached everywhere
- **In addition, it will move already queued messages to instances of the queue with consumers**
  - This removes the chance of long term marooned messages when consuming applications disconnect
- **Objective is to allow for more versatility in the use of clustered queues 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 (amqsclm.c sample) to allow the user to understand the mechanics of the tool and customise where needed.

Cluster Queue Monitoring Sample

N O T E S

- A new tool **amqsclm** 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 (amqsclm.c sample) to allow the user to understand the mechanics of the tool and customise where needed.
- This sample was introduced in WMQ 7.1 (distributed platforms), but has been backported to WMQ 7.0.1 fixpack 8.

Capitalware's MQ Technical Conference v2.0.1.3

AMQSCLM – How Does It Work?

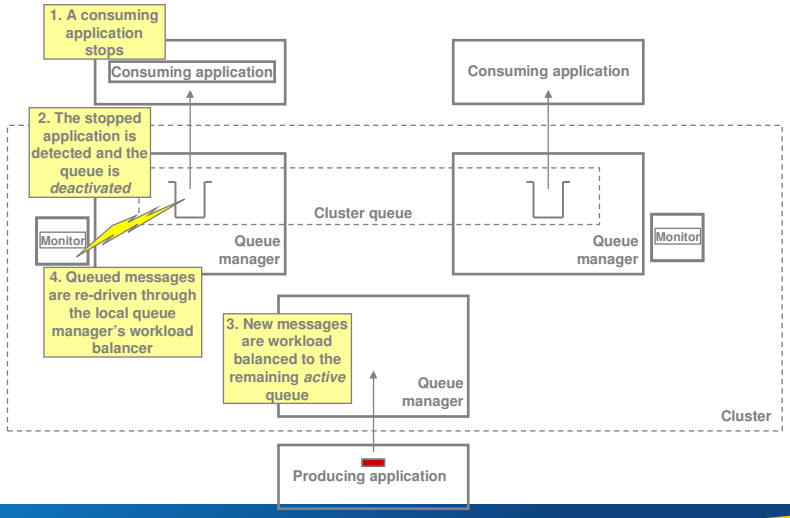
- **Based on the existing MQ cluster workload balancing mechanics:**
  - Uses cluster priority of individual queues
    - All else being equal, tries to send messages to instances of queues with the highest CLWLPRTY
    - Using CLWLPRTY always allows messages to be put to a queue instance, even when 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.
- **Single executable, set to run against each queue manager with one or more cluster queues to be monitored.**
- **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).
- **Defining the tool as an MQ service will ensure it is started with each queue manager**

AMQSCLM Logic

N O T E S

- Based on the existing MQ cluster workload distribution mechanics:
  - Uses cluster priority of individual queues – all else being 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 distributing workload to that queue.
- Single executable, set to run against each queue manager with one or more cluster queues to be monitored.
- 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 management 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).
- Defining the tool as an MQ service will ensure it is started with each queue manager

When the queues are monitored…



Limitations

N

O

T

E

S

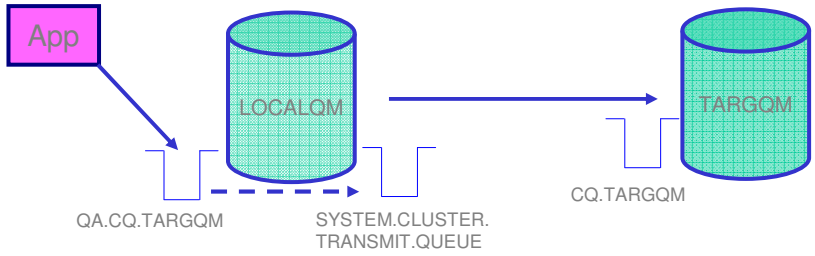
- The monitoring tool is poll based.
  - The more queues to monitor the more overhead on the system - increased poll interval is recommended.
- Frequently connecting/disconnecting consuming applications will result in message churn and cluster repository overhead.
  - Tool is really suited to a set of 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.

Security

This Slide Intentionally Left Blank

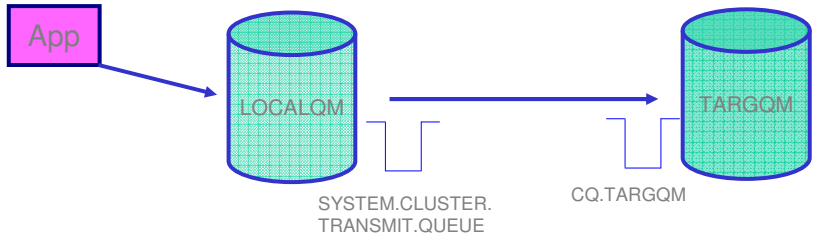
Using the new security model

- **Pre WMQ 7.1:**
  - Define local queue alias QA.CQ.TARGQM (targeting remote cluster queue CQ.TARGQM)...
    - **DEF QALIAS(QA.CQ.TARGQM) TARGQ(CQ.TARGQM)**
  - Give application permission to put to the local alias
    - **setmqaut -m LOCALQM -n QA.CQ.TARGQM -t queue -p myuser +put**
  - No access required to CQ.DUBLIN or the cluster transmit queue



Using the new security model

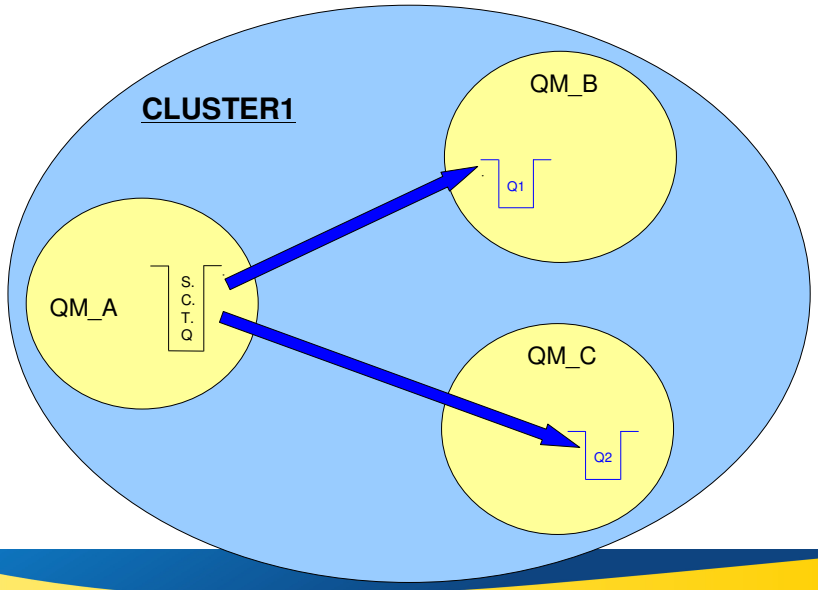
- **Now - using WMQ 7.1:**
  - Give application permission to put to the real target queue name
    - **setmqaut -m LOCALQM -n CQ.TARGQM -t queue -p myuser +put**
  - No access required to the cluster transmit queue



Split Cluster Transmit Queue

This Slide Intentionally Left Blank

The SYSTEM.CLUSTER.TRANSMIT.QUEUE



This Slide Intentionally Left Blank

NEW: 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...**

N O T E S

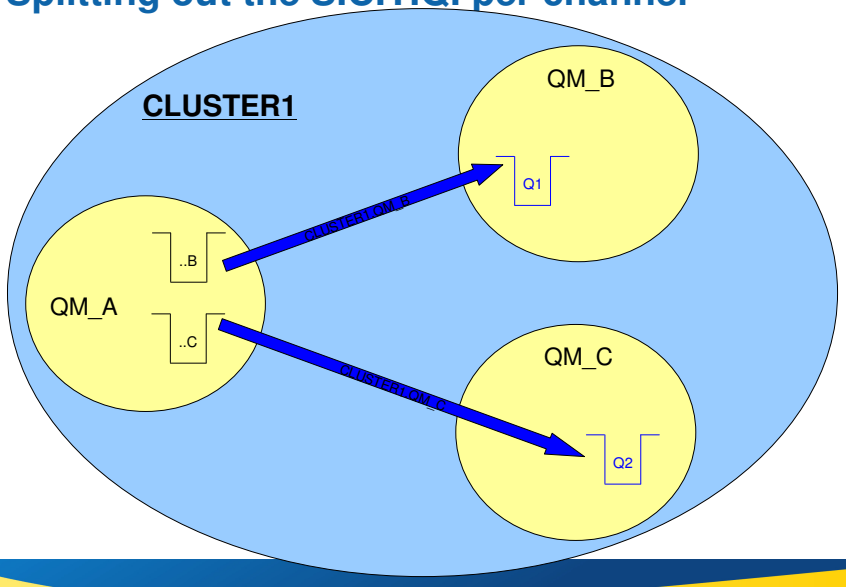
- This has been a very long standing requirement from a number of customers
- All the reasons on this slide are valid, but the number one reason often quoted in requirements was 'performance'
  - In reality splitting out the transmit queue does not often buy much here, hence often other solutions (e.g.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

Capitalware's MQ Technical Conference v2.0.1.3

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



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.

Questions?

Summary

Summary - WebSphere MQ Clusters

- **Goals of MQ Clustering**
- **How MQ Clustering Works**
- **Workload Management**
  - Why things don't always "Balance"
- **Full Repositories**
  - Considerations
  - How many?
- **Pub/Sub Clustering**
  - Considerations
  - PSCLUS
- **What's New in V7.1 and V7.5**
  - General Improvements to:
    - Reliability
    - Maintainability
    - Administration
  - BIND_ON_GROUP
  - Cluster Workload Rebalancer
  - Security
  - Split Cluster XMIT Queue

Capitalware's MQ Technical Conference v2.0.1.3