Les for Land Market popular John Market

Application
Resilience with
IBM MQ Shared
Queues



Agenda

- Application Resilience and Availability
- Why are shared queues so popular?
- Identify applications that are good candidates
- Common Pitfalls & Mitigation Techniques
- Uncommon Pitfalls
- How do I tell the application group theirs is not a good candidate?

Application Resilience and Availability

- "It is better to look good than to feel good"
- The goal is to mask both planned and unplanned outages from end users
 - The outages can be infrastructure or application related
- Application resilience is built on top of infrastructure and data availability
 - Without infrastructure and data availability, the all the application availability features won't matter
 - Without application availability and resilience, the most available infrastructure does not matter
- For IBM MQ the highest level of availability is provided by MQ Shared Queues
 - Only on z/OS
 - Unlike distributed platform, this solution is not based on disk replication and queue manager restarts – but on a shared resource known as the Coupling Facility

Application Resilience and Availability

- What about MQ Clustering?
 - MQ Clustering can provide 'New' message availability
 - 'New Messages' Messages that are flowing into the cluster from a putting or publishing application.
 - 'Old Messages' Messages that are committed to a queue instance are locked to that instance and are unavailable if the hosting queue manager or its supporting platform fail.

Why are shared queues so popular?

- From an application perspective, 'free' continuous availability.
 - Well behaved applications often require no changes at all.
- From an administration perspective, 'almost free' continuous availability.
 - Well behaved applications in a stable sysplex computing environment often require a limited number of administrative changes.
- From a hardware perspective, 'expensive' continuous availability but a reliable consistent technique that is used by every subsystem in the 'plex.
 - CF is not free
- From an infrastructure perspective
 - Closely integrated with other z/OS Sysplex aware systems
 - CICS
 - IMS
 - DB2

What is the biggest problem with Shared Queues?

- Storage!
 - Private queues can hold up to 48GB of messages
 - Shared queues are often sharing a smaller space
 - There are administrative techniques to relieve the issues more on that later
 - BUT this issue of space is often the critical factor

What Applications are good candidates?

- Zero or few affinities
 - Ideally no serialization requirements
- Quick turn around
 - Queue depth is consistently low
 - Messages do not remain on the queue for an extended period of time
- Parallel processing
- Robust error checking/handling
- Small messages
 - Unless you are using Shared Message Data Sets
- Frequent commits

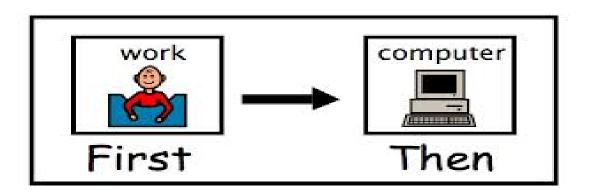
Message affinities – Loose or Tight?

- Tight affinities
 - Messages must be processed in strict FIFO order
 - Messages are processed by groups in a specific order
 - Groups can be very large
 - Message grouping or other application techniques to guarantee order have not been implemented
- Loose affinities
 - Message still must be processed in a specific order
 - Affinity is the exception, not the rule
 - Limited number of messages in the group often only one



Tight Affinities - Examples

- An unlimited number of messages associated with a new order.
 - PO header
 - 1-n PO line items
 - PO trailer
- All requests have to be handled in strict sequence across the enterprise
 - Demand deposits
 - Stock purchases
 - Inventory requests



Loose Affinities - Examples

- Typically the message affinities are the exception, not the rule. Examples can include things like:
 - New order with cancellation
 - New customer with change request
- Often can be addressed with a simple application change:
 - Application may have to rollback or re-MQPUT change request if initial input has not been processed

Quick Message Throughput

- How long do messages remain on queues?
 - If the answer is I don't know, you might be in trouble.
 - Look at queue accounting data SMF116 queue records
 - "+cpf START TRACE(A) CLASS(3)"
 - Evaluate Periodic 'DISPLAY QSTATUS' commands
 - Once is NOT enough!
 - Use application information to determine rates
 - Application logs
 - Database logs
 - Batch jobs need to be carefully evaluated
- Message size and throughput are contributing factors to CF structure size needed

Quick Message Throughput – DISPLAY QSTATUS Example

- /BWF0 DISPLAY QSTATUS('CICSTSTD*') all
- Result:
 - QSTATUS(CICSTSTD.BRIDGE.QUEUE)
 - TYPE(QUEUE)
 - OPPROCS(0)
 - IPPROCS(1)
 - CURDEPTH(3)
 - UNCOM(NO)
 - MONQ(HIGH)
 - QTIME(590,553)
 - MSGAGE(13318)
 - LPUTDATE(2007-02-01)
 - LPUTTIME(16.40.26)
 - LGETDATE(2007-02-01)
 - LGETTIME(16.40.26)
 - QSGDISP(QMGR)
 - END QSTATUS DETAILS

Oldest message has been on queue for > 3 hours!

Quick Message Throughput – DISPLAY QSTATUS Example

- /BWF0 DISPLAY QSTATUS('CICSTSTD*') all
- Result:
 - QSTATUS(SYSTEM.IP13.INOUT)
 - TYPE(QUEUE)
 - OPPROCS(0)
 - IPPROCS(0)
 - CURDEPTH(1)
 - UNCOM(NO)
 - MONQ(HIGH)
 - QTIME(11122,12368)
 - MSGAGE(6)
 - LPUTDATE(2007-02-02)
 - LPUTTIME(14.26.23)
 - LGETDATE(2007-02-02)
 - LGETTIME(14.26.23)
 - QSGDISP(QMGR)
 - END QSTATUS DETAILS

Oldest message has been on queue for 6 seconds.

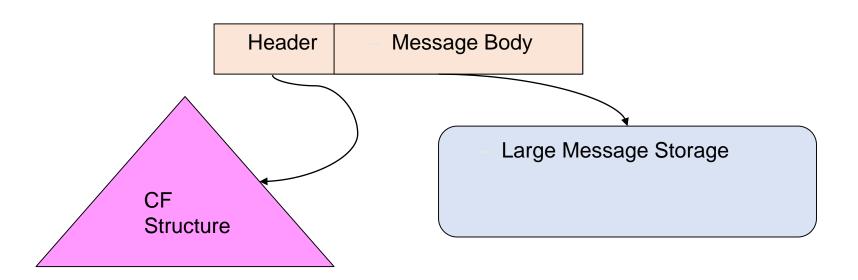
Parallel Processing

- Can server application be run in parallel?
 - If not, why not?
 - If the answer is yes, is it currently running that way?
- Message (data) Serialization is the most common issue
 - Targeted serialization techniques
 - Identify message relationships and target the messages to the same queue
 - Can achieve parallel processing while maintaining serialization
 - May require application changes
 - Products like Message Broker can help



Small messages

- Message size matters!
- Messages greater than 63K are always stored in two parts:
 - Message control information is stored on the CF structure
 - This is one element and two entries
 - Rounded to 2K for CF sizing estimates
 - These messages take more CPU
 - The message body storage depends MQ definition



Small messages

- MQ V7.0.1 Large message storage:
 - Message control information is stored on the CF structure
 - Message body is always stored on in Db2 BLOBs
 - MQ CFSTRUCT definition has CFLEVEL(4)
- For MQ V7.1 and above:
 - Message control information is stored on the CF structure
 - MQ CFSTRUCT definition has CFLEVEL(5)
 - Each Structure can identify an OFFLOAD location
 - DB2 Higher CPU cost, lower throughput
 - Shared Message Data Sets (SMDS) Lower cost, higher throughput
 - Each structure can have three offload rules
 - Two attributes per rule:
 - CF structure full percentage
 - Maximum message body size to store on CF

Robust Exception handling

- Most common problem is running out of physical storage or queue getting full
 - 2192 MQRC_STORAGE_MEDIUM_FULL
 - 2053 MQRC_Q_FULL
- How does application behave for the other CF Return Codes?
 - 2345 MQRC CF NOT AVAILABLE
 - 2348 MQRC CF STRUC AUTH FAILED
 - 2349 MQRC CF STRUC ERROR
 - 2373 MQRC_CF_STRUC_FAILED
 - 2346 MQRC_CF_STRUC_IN_USE
 - 2347 MQRC CF STRUC LIST HDR IN USE
- How does the getting application behave when it encounters a poisoned message?
- How often are messages rolled back?

Message availability - Frequent commits

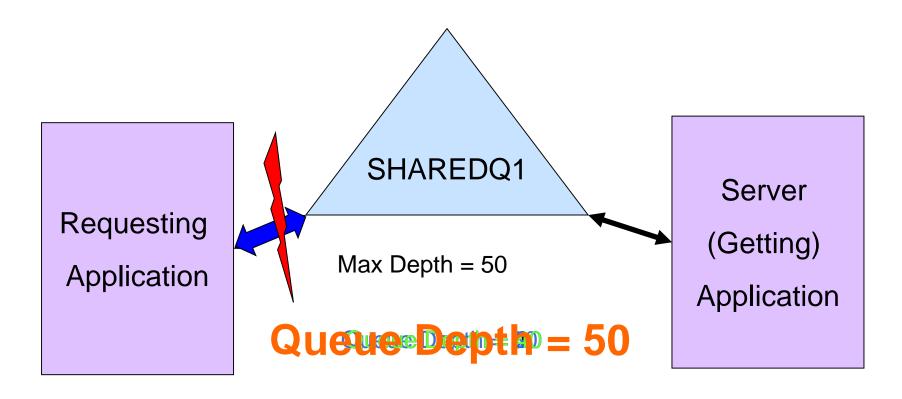
- Pulling applications cannot actually get the messages until they are committed
- If commit counts are high (>100), CF storage might become constrained
 - In addition VS in the QMGR address space might become constrained, but that's another issue
- If this is a tunable parameter, how often is it evaluated?



Common Pitfalls and Mitigation Techniques

- Slow Servers
- Media Full
 - A new experience for some applications
 - A new opportunity for 'sympathy sickness'
- Poisoned messages

What happens when the server application is slow?



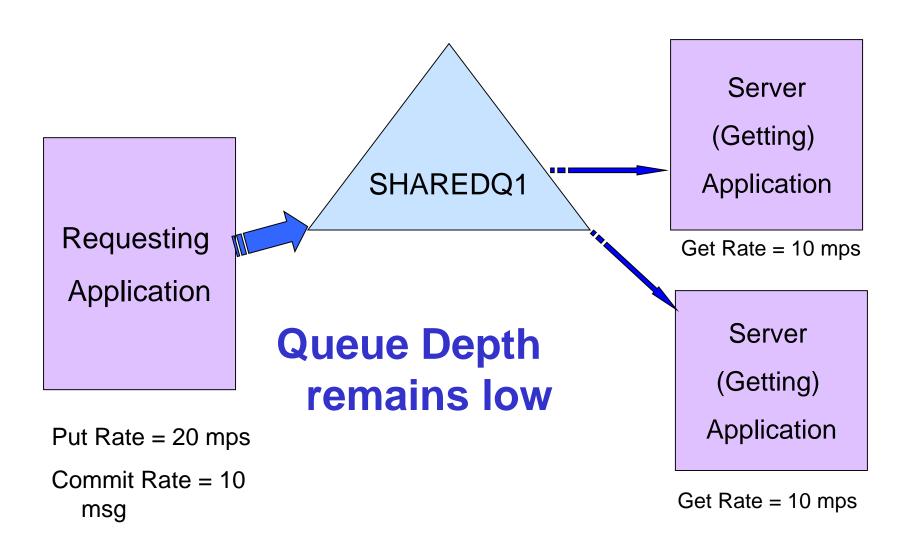
Put Rate = 20 mps
Commit Rate = 10
msg

MQPUT FAILS – QUEUE FULL

Get Rate = 10 mps

© IBM Corporation 2019

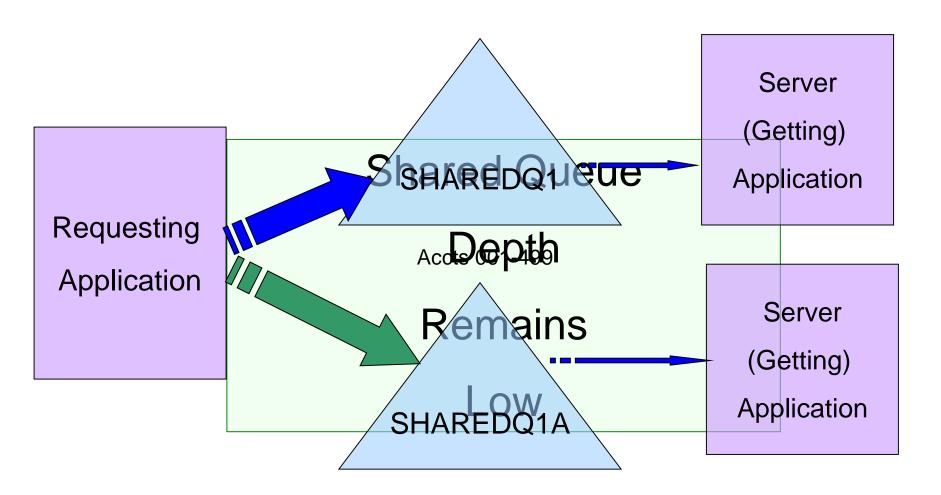
Slow Server Mitigation – What happens when we add a server application instance?



Slow Server Mitigation - Targeted Serialization

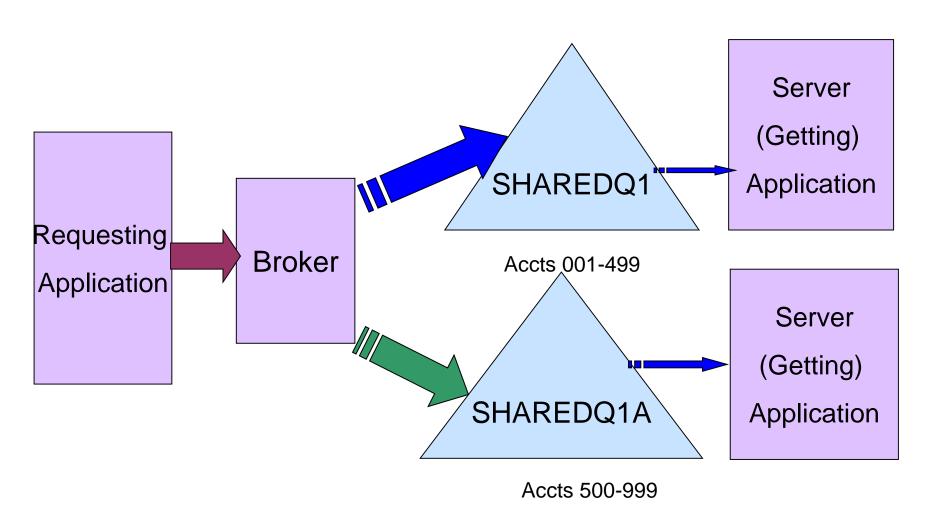
- Targeted serialization:
 - Divide the messages into multiple queues based on identifiable information within the message itself
 - This technique preserves the order of the data, allowing a parallel process to handle each queue.
- Can be used when the distribution of data is known or can be determined.
- Common examples:
 - Customer account numbers
 - Item numbers
 - Geographical location

Slow Server Mitigation – Simple targeted serialization



Accts 500-999

Slow Server Mitigation – Using a broker for targeted serialization



Media Full – New opportunities for unexpected return codes and sympathy sickness

- The CF Beach Front property, protected from hurricanes:
 - Typical CF is 32-64G, with 4-6G allocated to MQ for all the structures
 - Minimum 3 structures for MQ (2 admin and 1 application)
 - Maximum private queue size is
 - Often the CF Structures allocated to MQ are a fraction of the current maximum queue size
- The CF is common to all
- Multiple queues defined on the same structure will compete
 - No different from multiple queues on the same pageset but the available storage is usually a lot smaller
 - Careful positioning and monitoring of the queues is needed
 - One application 'running amok' can impact every other application using the same structure

Media Full – What to do when there is no room?

- Applications may be set up to 'throttle' message puts
 - Much like the message retry parameter on a receiver channel
 - This only works if the message is being put locally
- Putting applications may stop or abend
 - If there is a UOW in progress, it should be committed or rolled back
 - · Rolling back can free up some space
 - ARM or scheduling software may be used to restart the application
 - Be aware of possible loops
- Put inhibit the shared queue
 - Often done by automated processes using QDEPTHHI and QDEPTHLO events

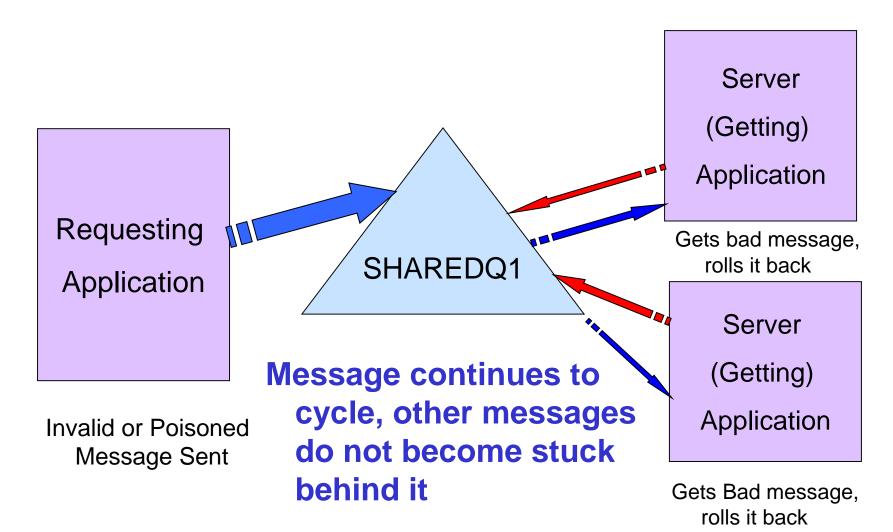
Problem Avoidance Techniques – What to do when there is no room?

- Make sure ALLOWAUTOALT is set to YES on structure definition even if you do not allow the structure size to expand.
 - Effects of letting the system set the entry/element ratio are described in the V7.1 and 7.5 Redbook
- Run multiple instances of the getting program
 - Monitor queue depths and time on queue to determine when you need more instances
- Resize the queue
 - May be done when structure is underutilized
- Resize the CF Structure

Problem Avoidance Techniques – What to do when there is no room?

- Putting application can put to a secondary of back-up queue
 - Usually a private queue
 - Must move messages to the primary queue when the situation has been resolved
 - Out of sequence issues possible

Message Backouts – A different facet with shared queues



Problem Avoidance Technique – Message Backouts

- When messages are rolled back, the back out count is incremented
- Getting application should check this, and if it exceeds a predetermined value put the message on a 'backout queue', a file, or discard the message completely
 - Not recommended to put 'poison' messages on the DLQ
- If poison messages are not handled properly, a bad message can become a 'Politician message' it impacts your processing, but never does anything

Uncommon Pitfalls - Triggering

- Triggering Behavior Differences:
 - For Private Queues trigger events take place when a message is put to the triggered queue
 - Messages can be put, a trigger message generated, but the original message is not available until the message is committed
 - If the original message is backed out, the triggered application will never retrieve a message
 - Applications are normally written to tolerate a '2033' reason code
- For Shared Queues trigger events take place when the message is committed
 - No application changes are usually required
 - There can be multiple trigger messages generated , one per queue manager that has registered an interest
 - This can cause high CPU consumption if there are many triggered instances and a low message arrival rate.

Uncommon Pitfalls – Queue Indexing

- On z/OS queues private queues may be indexed for more efficient processing
 - MQGETs with a match option on message ID or correlation ID benefit greatly
- z/OS Shared Queues must be indexed if the serving applications use a match option on the MQGET

Uncommon Pitfalls – Hard coded Names

- If a queue manager name is hard coded into a batch application program, that must be changed to use the queue sharing group name.
 - CICS regions can connect to a QSG rather than an individual queue manager using the normal MQCONN Resource Definition
- IMS programs can connect to queue managers that are part of a QSG, but cannot connect to a QSG at this time
- If existing queues will not be migrated to shared queues, but new queues with new names added, hard coded queue names within any application may have to be changed.
 - A QALIAS may help under some circumstances

Uncommon Pitfalls – Expired Messages

- Some instances have been reported of expired message build up on shared queues after they have been migrated from private queues
 - If the queue was first indexed after it was converted:
 - Then the application was causing the queue manager to scan the queue looking for a matching message
 - During the scan expired messages will be discarded
 - When messages are retrieved via the index, the queue is not scanned so expired messages are not discarded
 - The EXPRYINT on queue managers defaults to OFF,
 - The expired message scavenger task will never run unless set to an integer value on at least one queue manager in the queue sharing group

Uncommon Pitfalls – Queue Scanning

- Until recently, moving to shared queues reduced or eliminated most queue scanning.
 - Most scanning was a result of an MQGET with a match option which cannot happen on shared queues
- Applications using Message Properties can cause scanning as well
 - JMS applications use message properties as a natural part of their implementation
 - Message properties cannot be indexed, so the queue manager will scan
- Only solution keep message depths low!

How can an application group tell that theirs is not a good candidate?

- Are there external constraints that may prevent the adoption?
 - Batch window
 - CPU constraints
- Historical analysis of private queue usage.
 - Is queue depth frequently higher than the CF structure allocation will allow?
 - Are there frequent rollbacks?
 - Does the application only commit when Atlanta freezes?
- Historical analysis of application outages.
 - If planned or unplanned outages regularly cause very high queue depths, this might not be a good candidate.
 - Analyzing this may not be politically palatable, but may be necessary.