

Shared Queues – Beyond the Hype

Lyn Elkins, IBM WSC
elkinsc@us.ibm.com

Mitch Johnson, IBM WSC
mitchj@us.ibm.com

First A Story

- First exposure to Sysplex enablement
- Project stopped and restarted about 6 times
- Company became enamored of 'code names'
- I suggested.....

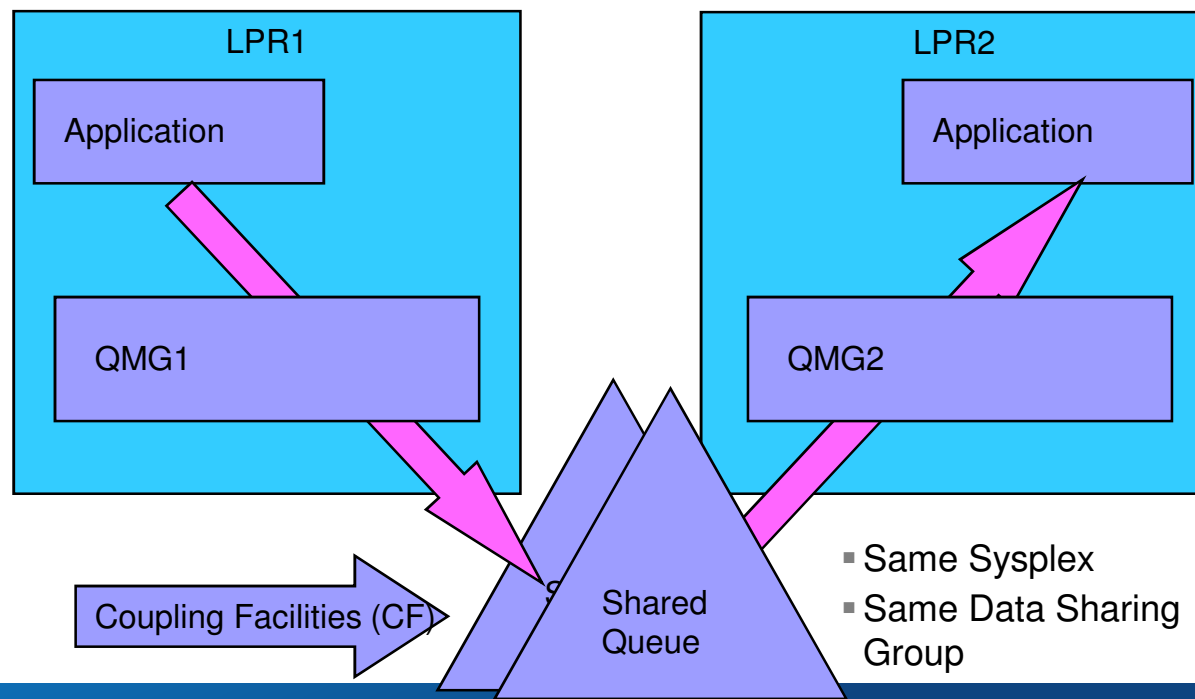


What is a queue sharing group?

- **The Sysplex enablement for MQ**
 - ▶ Sometimes called a Queue 'Plex

- **Group of queue managers connected by**
 - ▶ LPAR is connected to the Sysplex
 - ▶ Coupling Facility Structures
 - Two (2) MQ admin structures
 - One or more application structures
 - ▶ DB2 Data Sharing
 - Yes it's still required
 - Tables hold information about queues

Queue Sharing Group



Queue Sharing group - notes

- **Chart shows a simple two member QSG**
 - ▶ Application on LPR1 puts to shared target queue
 - ▶ Application on LPR2 gets from shared queue
- **The remote application can put a message to the reply-to queue using the same method**
- **No Channel initiator activity**
 - ▶ Can be much faster, and at time less costly than 'normal channels
- **Note that applications connected to any queue manager with access to the shared queue can get the message. To access the same shared queues, queue managers must be:**
 - ▶ In the same z/OS sysplex
 - ▶ In the same queue-sharing group (QSG)
 - ▶ Registered to the same DB2 Data Sharing group

The Hype - Why are shared queues so popular?

■ It depends on your point of view:

- ▶ Applications see shared queues as 'free' continuous availability.
 - Well behaved applications often require no changes at all.
- ▶ MQ Administrators see shared queues as 'almost free' continuous availability.
 - Well behaved applications in a stable sysplex computing environment often require a limited number of administrative changes.
- ▶ Overall Infrastructure view
 - 'Almost Free' availability
 - And – very closely integrated with other z/OS Sysplex aware systems
 - CICS
 - IMS
 - DB2

The Reality - Why are shared queues so popular?

- **First of all, the continuous availability is not hype**
 - ▶ Customers have had continuous availability of messages for years
 - Using rolling outages for maintenance
 - And the occasional real emergency
 - In most cases users did not 'see' an outage



The Hype – applications can take advantage instantly

- There is a myth that applications require no changes
 - But that may not always be true

The reality - 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**

The Reality - 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

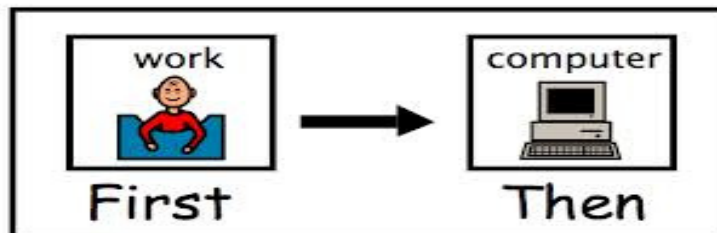


■ Loose affinities

- ▶ 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 deposit
 - ▶ Stock purchases
 - ▶ Inventory reques



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

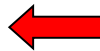
Quick Message Throughput – DISPLAY QSTATUS Example

- /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

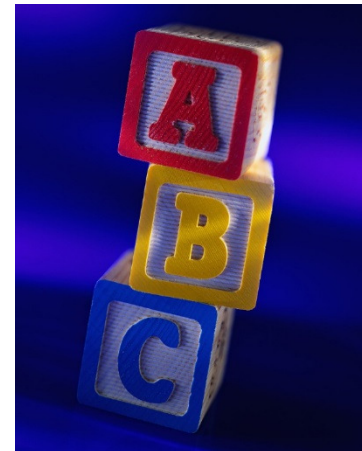
**Oldest message has
been on queue for
6 seconds.**



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



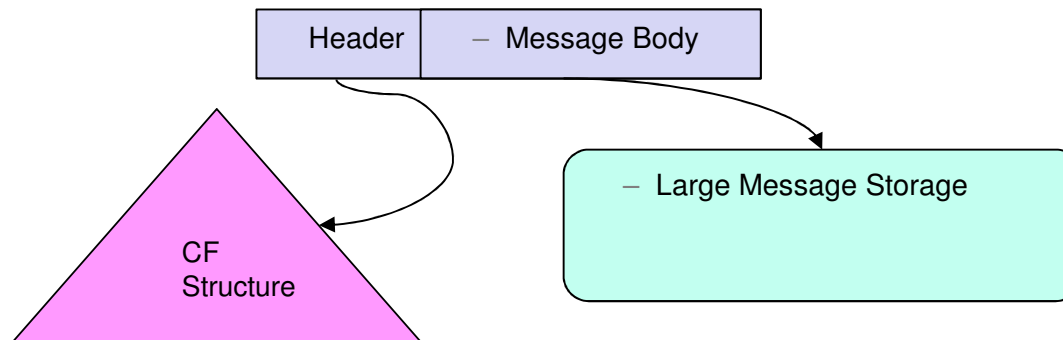
Parallel Processing - Notes

- **Can server application be run in parallel?**
 - ▶ If not, why not?
 - ▶ If the answer is yes, is it currently running that way?
 - In practice, there are times when no one believes there are serialization questions no one has ever actually tested the theory. Between theory and practice, reality can rear a very ugly head. It is a good idea to do a stress/load test that includes lots of application scenarios running in parallel to see if parallel processing is successful.

 - **Message (data) Serialization is the most common issue**
 - ▶ Targeted serialization techniques
 - Identify message relationships and target the messages to those queues
 - Account number, part number, or other identifier is often used
 - Can achieve parallel processing while maintaining serialization
 - Instance A processes accounts 0001-0200, instance B processed 0201-0400
- Can requires application changes

Small messages

- **Message size matters!**
- **Messages greater than 63K or those that are offloaded are always stored in two parts:**
 - Message control information is stored on the CF structure
 - This is one element and two entries
 - Rounded to 1K for CF sizing estimates
 - ▶ These messages take more CPU
 - ▶ The message body storage depends on the offload rules!



Small messages

- **For MQ V7.1 and above:**
 - Message control information is stored on the CF structure
 - 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?**

Robust Exception handling - Notes

- **New reason codes were added in V7.1 for Shared message data sets.**
- **How does the getting application behave when it encounters a poisoned message?**
 - ▶ This is both an administration and application question.
 - ▶ Admin – are queues defined with:
 - A Backout threshold
 - Harden backout counter turned on
 - A backout re-queue queue, sometimes called an application DLQ
 - Are the re-queue queues monitored?
 - ▶ App:
 - Does the application check the backout counter?
 - Does the application put the message to the re-queue queue and remove it from the input?
- **How often are messages rolled back?**
 - ▶ Can tell this from the MQ SMF data.
 - ▶ If the answer is 'a lot', the queue is probably not a good candidate for sharing.

Message availability - Frequent commits

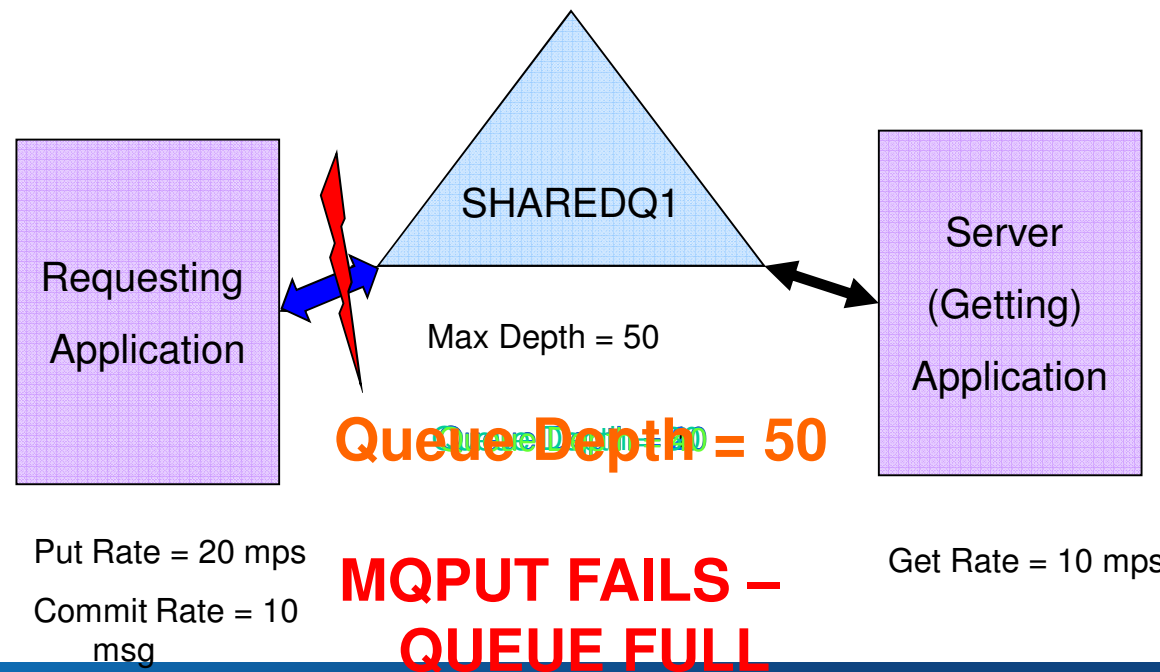
- Pulling applications cannot actually get the messages until they are committed
- If commit counts are high (>100), CF storage might become constrained
 - ▶ In addition VS in the QMGR address space might become constrained, but that's another issue
- If this is a tunable parameter, how often is it evaluated?



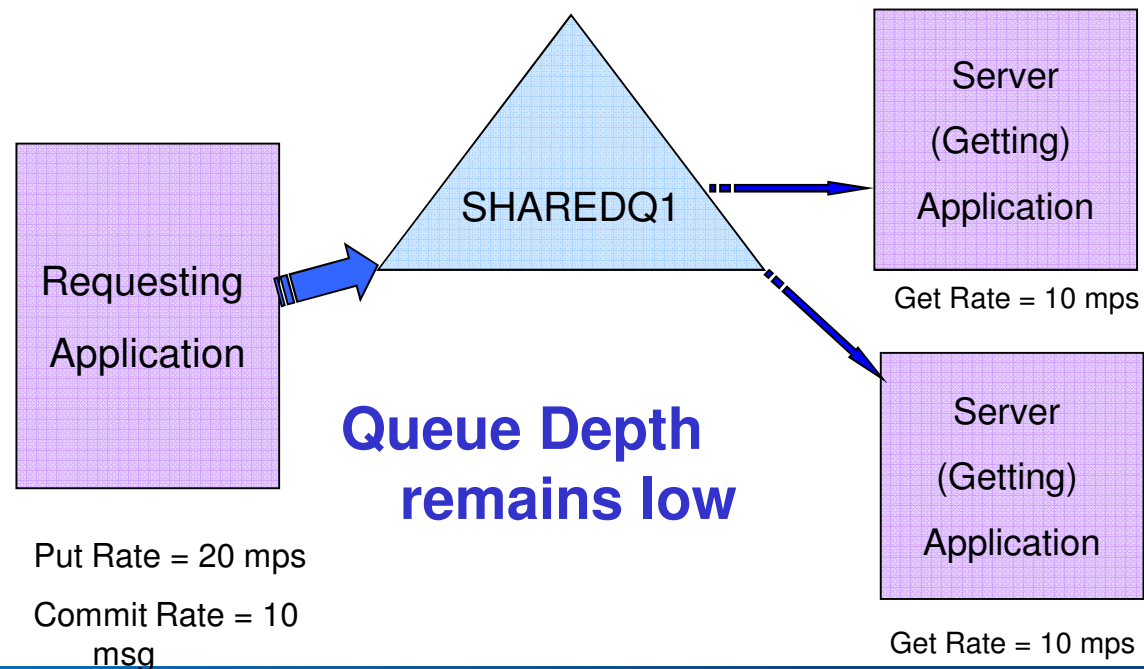
Common Pitfalls and Mitigation Techniques

- **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?



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



Slow Server Mitigation - Targeted Serialization

- **Targeted serialization:**

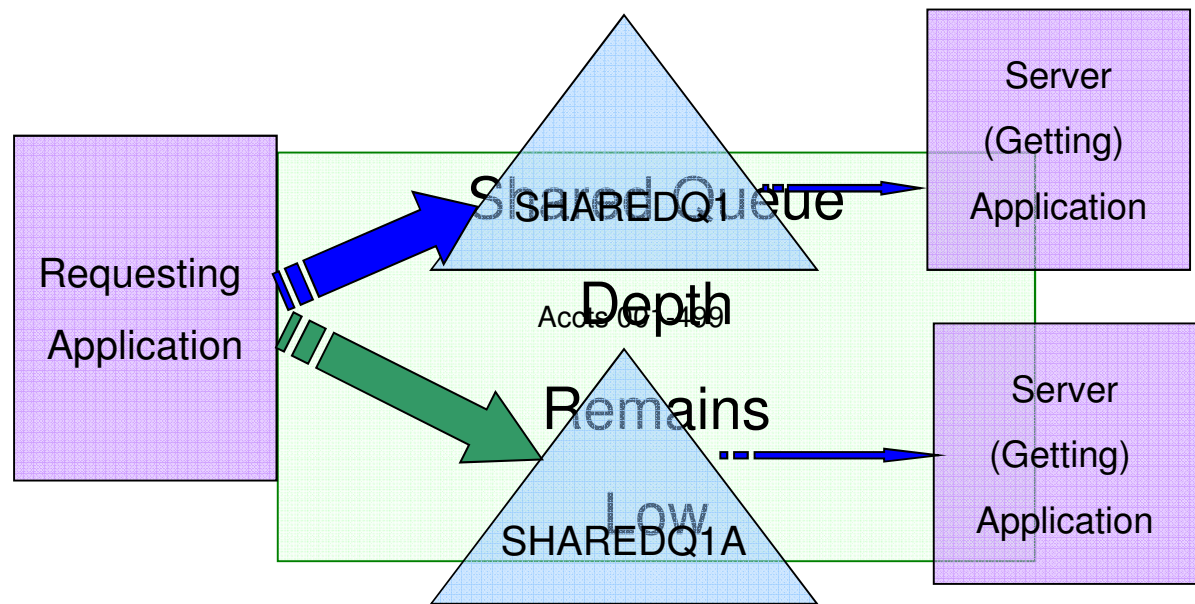
- ▶ Divide the messages into multiple queues based on identifiable information within the message itself
- ▶ This technique preserves the order of the data, allowing a parallel process to handle each queue.

- **Can be used when the distribution of data is known or can be determined.**

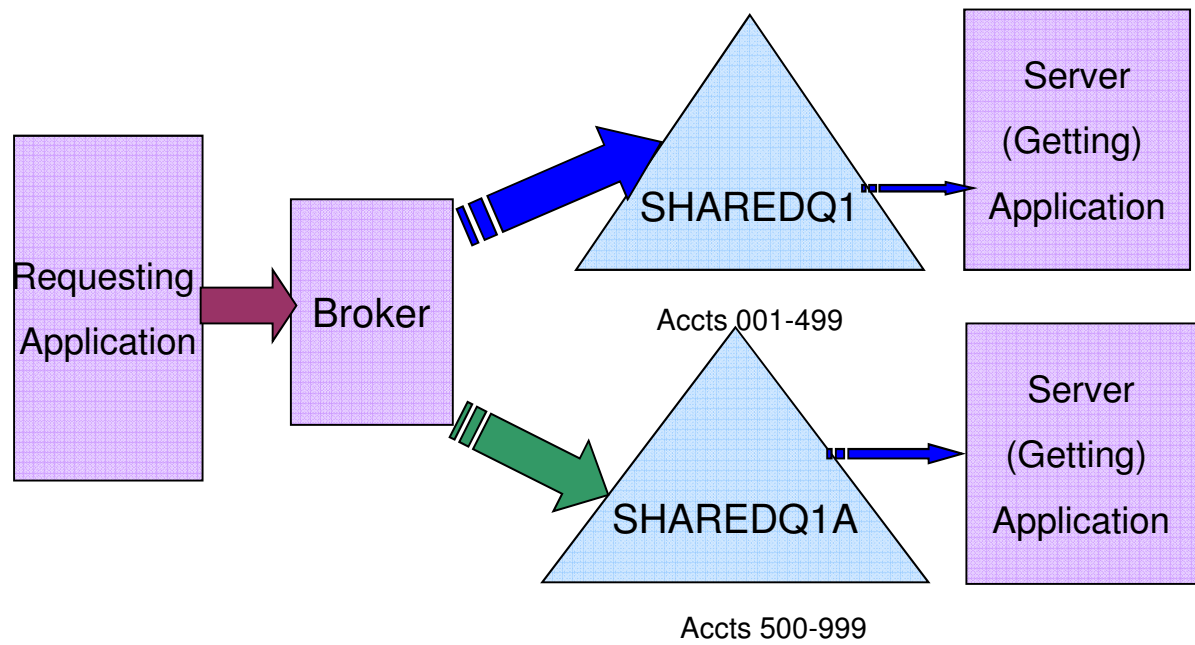
- **Common examples:**

- ▶ Customer account numbers
- ▶ Item numbers
- ▶ Geographical location

Slow Server Mitigation – Simple targeted serialization



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 2 structures for MQ (admin and application)
 - ▶ Maximum private queue size is
 - ▶ So, 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

The Hype – Shared queues provide ‘Pull’ workload balance

- Technically True
- The more powerful, more available Queue Manager will pull more messages



The Reality – Shared queues can cause workload skewing

- **Can be a surprise**
 - ▶ Connection skewing
 - ▶ Asymmetrical sysplex



The Hype – Shared queues are always more expensive

- **Most Often true**

- ▶ Especially when using offloading
- ▶ But not always, and not for all requests!

The Reality - Costs

- Sometimes there can be surprises
- But overall they tend to be more expensive

Queue Name	Queue Manager	Queue Type	Average CPU for Valid MQGET	Average CPU for Valid MQPUT
ELKINSC.SHARED.QUEUE	QML1	SHARED	128.77	0
ELKINSC.SHARED.QUEUE	QML2	SHARED	245.67	0
ELKINSC.SHARED.QUEUE	QML3	SHARED	0	18.42
ELKINSC.SHARED.QUEUE	QML4	SHARED	0	30.83
Sum FOR SHARED QUEUE			176.01	24.6
SYSTEM.CLUSTER.TRANSMIT.QUEUE	QML1	PRIVATE	21.15	51.76
SYSTEM.CLUSTER.TRANSMIT.QUEUE	QML2	PRIVATE	13.13	45.85
SYSTEM.CLUSTER.TRANSMIT.QUEUE	QML3	PRIVATE		
SYSTEM.CLUSTER.TRANSMIT.QUEUE	QML4	PRIVATE		
SUM FOR CLUSTER TRANSMIT QUEUE			17.91	49.37

Reality costs notes

■ The summary is taken from a real stress test

- ▶ Names have been changed
- ▶ This also shows the impact of an asymmetric Sysplex – the per MQPUT and MQGET costs are higher on QML2 and QML4, and the service times were much slower due to the physical location of the CF used to host the queue
 - ICF links between the LPAR hosting QML1 and QML3

Full Comparison

Queue Manager	Queue Type	Sum of all MQGETs	Sum of MQGET CPU	Sum of Valid MQGETs	Average CPU for Valid MQGET	Total Bytes MQGET	Sum of all MQPUTs	Sum of MQPUT CPU	Sum of all MQPUTs	Sum of MQPUT CPU	Sum of Valid MQPUTs	Average CPU for Valid MQPUT	Total Bytes MQPUT
QML1	SHARED	204808121	1181322809	9173709	128.77	15415705192	0	0	0	0	0	0	0
QML2	SHARED	194172851	1528149684	6220216	245.67	8506605051	0	0	0	0	0	0	0
QML3	SHARED	0	0	0	0	0	7719465	142176323	0	0	7719465	18.42	11997684629
QML4	SHARED	0	0	0	0	0	7674460	236588225	0	0	7674460	30.83	11924625614
		398980972	2709472493	15393925	176.01	23922310243	15393925	378764548	0	0	15393925	24.6	23922310243
QML1	PRIVATE	21136626	194002645	9173709	21.15	19342052644	9173709	474841640	0	0	9173709	51.76	15415705192
QML2	PRIVATE	16973980	81683915	6220216	13.13	11168857499	6220216	285192488	0	0	6220216	45.85	8506605051
QML3	PRIVATE	0	0	0	0	0	0	0	0	0	0	0	0
QML4	PRIVATE	0	0	0	0	0	0	0	0	0	0	0	0
		38110606	275686560	15393925	17.91	30510910143	15393925	760034128	0	0	15393925	49.37	23922310243

Summary

- **There can be a lot of hype around the use of shared queues**
 - ▶ But they do consistently fulfill high to continuous availability requirements
 - ▶ Using them may require application changes
 - ▶ They may not be as costly as you have heard
 - Very dependent on physical configuration
 - ▶ They may cause downstream workload skewing
 - But some of that can be mitigated

Trying Something New

MQ Labs Experiment

Booth next to Aloeswood

MQ for z/OS Images

MQ for Distributed Image

Monday & Tuesday Morning –
drop in





Any questions?