Shared Queues – where is my workload running
Agenda

- What are shared queues – briefly

- What is workload skewing and why is it a problem?
  - What are the symptoms and causes
    - Asymmetrical Sysplex
    - Connection Skewing
    - Put to Waiting Getter
  - ‘Local’ favoritism

- Mitigation Techniques:
  - Queue Manager Clustering
  - Gateway queue managers
  - CICS CPSM options
Briefly – What is a Shared Queue

- MPX1
  - QML1
- MPX2
  - QML2
- MPX3
  - QML3
- MPX4
  - QML4

APPQUEUE

MQGET/MQPUT

MQGET/MQPUT

MQGET/MQPUT

MQGET/MQPUT
What is a shared queue?

- **Unique to z/OS**
  - Requires a coupling facility to host the queues
  - DB2 data sharing for the queue definitions
  - The gold standard for message availability

- **Treats a shared queues as local to each queue manager in the QSG**
  - Applications can PUT and GET

- **Messages are available as long as one QMGR in the QSG can access the Coupling Facility Structure.**

- **Nonpersistent messages are only ‘lost’ if the structure or CF itself are lost.**
What is MQ Workload Skewing?

- Workload skewing is detected when MQ driven work, typically transactions, is not close to being evenly distributed across the queue managers.
Why is MQ Workload Skewing a problem?

- This is often less a technical problem, more of a pricing problem
  - If the MLC ‘rolling average’ is taken from the LPAR that is heavily favored, usage pricing is not going to reflect reality
  - Technical solutions to this problem may prove to be less efficient overall - lower throughput, slower response
  - Using a VUE version of MQ can eliminate this issue
Why is MQ Workload Skewing a problem?

- Can cause increased capacity demands in downstream workload
  - Known to produce responsiveness problems
    - Overloading the processing programs
  - Again this can contort MLC charges
MQ Workload Skewing Causes

- Workload skewing in a QSG is often a result of the efficiencies of working locally
  - z/OS, and all subsystems try to process requests locally to take advantage of CPU efficiency
MQ Workload Skewing Causes - Hardware

- Asymmetric Sysplex
  - When the LPARs in the Sysplex are not equally weighted
    - Examples include:
      - One LPAR is on an EC12, the others on older hardware
      - Two LPARs have 12 dedicated engines, two have 12 shared
MQ Workload Skewing Causes - Hardware

- Asymmetric Sysplex
  - Most common example - One LPAR is co-located with the primary coupling facility, the others are on different CPCs
  - ICF links give much better service times than CBP
We (the WSC) tend to use the CF Activity report rather than the MQ Statistics when looking at shared queue usage.

In the example shown above it is easy to see that the MPX2 LPAR is getting a much longer service time (almost 4 times!) than the MPX1 LPAR and that MPX2 is making many more requests.

In this particular case, this exposed some internal workload skewing that was not apparent to the customer - except that they were missing SLAs consistently!
MQ Workload Skewing Causes - Hardware

- **Location of the Coupling Facility**
  - When the coupling facility is internal, LPARs on the same CEC tend to get faster response
  - When the coupling facility is external and one LPAR has more, faster, or less heavily used links it will get faster service

![Diagram showing MQ Workload Skewing Causes - Hardware](image-url)
Connection Skewing

- Connection skewing may be historical
  - Hard-coded connections to specific queue managers

- Connection skewing may be the result of a queue manager outage
  - Connections to a QSG are routed to available queue managers
‘Downstream’ consequences

- We’ve talked about the MLC impact

- Resource use
  - Not every queue manager is sized to absorb the entire workload
  - Log impact of skewing has been seen
    - Rapid Log switches due to heavier workload – increasing I/O and CPU costs
  - Bufferpool/Pageset impact
    - Filling the bufferpool, forced into I/O
  - SMDS impact
    - One queue manager in QSG gets all offloaded messages
MQ Workload Skewing Causes

- **Put to waiting getter**
  - In V6 a performance feature was added called ‘put to waiting getter’
  - If a local put, from an application or message channel agent, is done and there is a getting application waiting the message is moved directly to the getting applications buffer
    - There is no posting to a shared queue
    - There is no notification to other available waiting applications
    - The CPU savings can be substantial
    - This works with connection skewing, and can maximize the effect
Put to Waiting Getter – SMF

- This shows messages flowing across a channel taking advantage of P2WG

<table>
<thead>
<tr>
<th>Base_Name</th>
<th>CF Struct</th>
<th>Total_Valid_Gets</th>
<th>Total_Bytes_Put</th>
<th>Total_Valid_Puts</th>
<th>Total_Put2_Waiting_Getter</th>
<th>Puts not to Waiting Getter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.QSG.CHANNEL.SYNCOQ</td>
<td>CSQSYSAP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHARED.INPUT.QUEUE</td>
<td>APP1</td>
<td>0</td>
<td>4501092223</td>
<td>2095814</td>
<td>2012394</td>
<td>83420</td>
</tr>
</tbody>
</table>

- The CPU comparison shows why it can be a good thing!

<table>
<thead>
<tr>
<th>BASE_NAME</th>
<th>VALID_PUTS</th>
<th>PUT_ELAPSED_TIME</th>
<th>PUT_CPU_TIME</th>
<th>PUT2_WAITING_GETTER</th>
<th>Average_PUT_ET</th>
<th>Average_PUT_CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLOCAL.PUT2WG</td>
<td>14879</td>
<td>127753</td>
<td>117956</td>
<td>14793</td>
<td>8.59</td>
<td>7.93</td>
</tr>
<tr>
<td>QLOCAL.NO.PUT2WG</td>
<td>41547</td>
<td>1025028</td>
<td>1010038</td>
<td>0</td>
<td>24.67</td>
<td>24.31</td>
</tr>
</tbody>
</table>

- The CPU costs can be 3 times as high!
MQ Workload Skewing Causes

- Local Favoritism
  - When a message is posted to a shared queue, the queue manager where the message is put is typically notified FIRST about the availability.
  - Normal processing by XCF, taking advantage of the efficiency of local processing.
Skewing Mitigation Techniques

- **Queue Manager Clusters**
  - Clusters provide workload balancing across queue managers
  - Works with shared queues to distribute message ‘puts’ across queue managers in the QSG

- **Connection skewing mitigation**
  - Gateway queue managers
  - Re-driving connections

- **CPSM mitigation**
Queue Manager Clustering

- When messages are not bound to a specific queue manager (‘bind not fixed’), the messages are routed evenly across the receiving queue managers
  - Black arrows show the first message put to the clustered queue
  - Green arrows show the second message
Connection Skewing Mitigation

- The slides that follow outline two mitigation techniques for connection skewing:
  - Gateway queue managers
  - Re-driving connections
Connection Skewing – No Gateway queue managers

- When external queue managers or clients are passing work directly to application hosting queue managers, every attempt is made to process the work locally.

- Environments that use gateway queue managers into the Queue Sharing group often eliminate connection skewing.
Gateway queue managers – the mitigation

QM_WIN1 → QM_WIN2
QM_WIN3
QM_WIN4
EC12
QM_G1

QM_AIX1
EC12
QM_GA

QM_WIN5

CICS5
CICS6
CICS7
CICS8
QC

CICSD
CICSE
CICSF
CICSG
Re-driving Connections

- When a queue manager is unavailable, inbound connections can get skewed to the other queue manager(s) in the group.
  - This is normal availability processing!
  - Once a connection is live and active, no attempt is made to balance the connections once all the queue managers are available.
CICS – CPSM Mitigation

The slides that follow outline a CPSM solution to the skewing problem based on the interaction between MQ triggering (CKTI) and CICS
Configuration for inbound WMQ work using triggering (schematic)

- Each CICS region acts as an independent consumer from the shared queues
- Unbalanced workload distribution
Trigger monitor (CKTI)

Business transaction

Start

MQGET trigger message

EXEC CICS START

MQGET request message

Business logic

MQPUT response message

End

Trigger-every (schematic)

- Each business transaction processes few (~1) request messages
- Fastest CKTIs take lion's share of work
Preferred configuration for trigger-every (schematic)

- “TORs” run CKTI
- Each acts as an independent consumer from the shared init queue
- Each distributes STARTs across all AORs
- “Balanced” workload distribution

EXEC CICS START
Trigger-first/depth (schematic)

- Each business transaction processes many (~all) request messages
- Fastest CKTI takes lion's share of work
- Corresponding business transaction takes lion's share of the work
Trigger monitor (CKTI)

Trigger-first/depth staged (schematic)
- Staging transaction processes all request messages

Staging transaction
- Start
  - MQGET
  - MQGET request message
  - EXEC CICS LINK
  - MQPUT
    - MQPUT response message

Business logic
- Start
  - Business logic
  - Return

- Business transaction processes one request message
Preferred configuration for trigger-first/depth staged (schematic)

- "QORs" run CKTI and staging transaction
- Each acts as an independent consumer from the shared init queue and shared application queue
- Each distributes LINKs across all AORs
- "Balanced" workload distribution

### CICS AORs

- CEC1: z10
- CEC2: z196
- CEC3: z196

### Queue manager

- Application queue
- CF (shared queues)
- Initiation queue

### Diagram Elements

- EXEC CICS LINK
- Request message
- Trigger message
Highlights

- Solution uses proven technology for CPSM routing:
  - Each TOR/QOR uses link-neutral goal algorithm
    - Selects target AOR based on AOR load and health
    - Does not “prefer” local (= same LPAR) AORs
    - Even distribution across AORs, but …
    - … responds to transient load/health variation
  - XCF MRO for “remote” STARTs or LINKs
    - High-performance System z sysplex technology
    - Uses coupling facility (CF) instead of TCP/IP stack
  - Sysplex-optimised workload routing
    - Highly responsive to transient variations
    - Uses CF to maintain current status for AORs

- Continuous operation and high availability through WMQ shared queues:
  - “Glitchless” recovery from region/LPAR/CEC outage
  - “Instant” redistribution of workload
  - In-flight messages backed-out, restart in another CICS region

- High throughput:
  - Exploits all available capacity
  - Highly responsive to transient spare capacity
MQ Workload Balance Summary

- **MQ is a message delivery system, it does not try to balance workload**

- **Balancing the workload is attempting a technical solution for what is often a pricing problem**
  - Beware spending a lot of effort for a solution to a temporary problem as well!
  - Turning off performance improvements like put to waiting getter will impact all applications, not just the skewed ones

- **There are some mitigation techniques that can help the overall environment**
  - Clustering!
  - Gateway queue managers
  - Using CPSM to make appropriate routing decisions
Additional Resources

- The following links are to additional information about WMQ
  - Queue Sharing Groups:  
  - Clustering:  
  - Intercommunication  
  - Redbooks:
    - IBM WebSphere MQ V7.1 and V7.5 Features and Enhancements  
    - High Availability in WebSphere Messaging Solutions  
    - WebSphere MQ Queue Sharing Group in a Parallel Sysplex environment (dated, but still good basic information)  
  - Lyn’s first YouTube video:  
    http://www.youtube.com/playlist?list=PL9N7JP2yU3T8JycrCOvEPM8c-0UdE97VT
MQ Workload Balance - thanks

- Many thanks to
  - Steve Hobson for the CICS/CPSM expertise and the wonderful graphics
  - Mark Taylor for their patience and guidance on the rest of the foils
  - Mark Taylor for providing the excellent editing and recording studio for the YouTube version of this pitch