Miscellenous MQ for z/OS changes

Lyn Elkins – elkinsc@us.ibm.com
Mitch Johnson – mitchj@us.ibm.com

WebSphere software

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

© IBM Corporation 2014

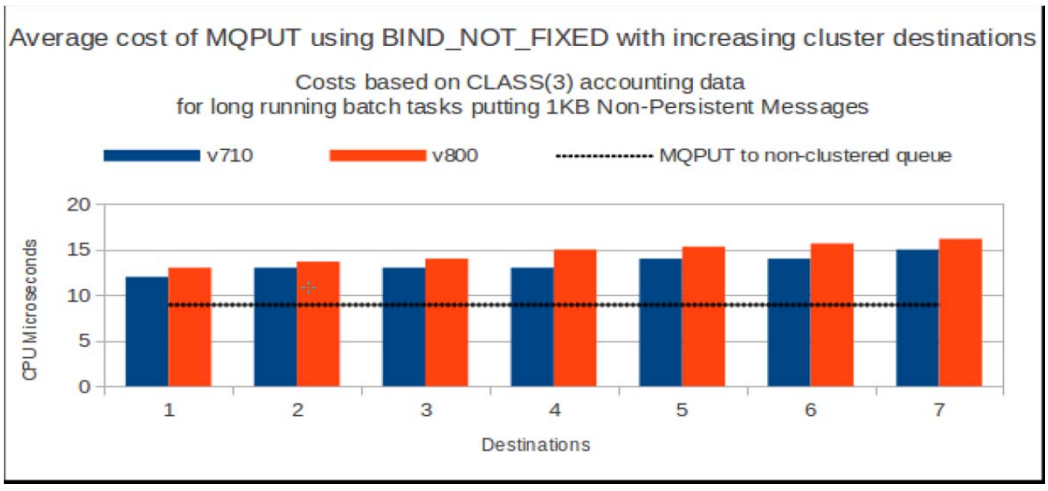
Agenda

- Multiple Cluster Transmission queues
- Message Suppression
- Logging Changes – including more logs!
- Other changes
- Capped Expiry
- Z Hardware and Software Exploitation

Multiple Cluster Transmission Queues

- Harmonization with WMQ V7.5 Distributed
- Advantages:
  - Isolation – Isolated transmission queues can be specified for particular cluster sender channels
  - Providing separate service classes
    - A high priority transmission queue can be located on an above the bar bufferpool (large) and get better service
- Disadvantage:
  - Slightly increases the cost of the MQPUTs to a cluster queue
    - The overhead of supporting multiple cluster transmit queues is between 1 and 2 CPU microseconds. This does not change with message size or persistence.

Cost of MQPUT to clustered queue



Message Suppression

- This includes separate messages for client channels
  - CSQX511I and CSQX512I for client connection to a SVRCONN
  - CSQX500I and CSQX501I report other channel start/stop events
- Message suppression
  - Messages that are suppressed are not written to the JES log, so can reduce CPI
  - Implemented via a new ZPARM, EXCLMSG
  - Can be altered via a SET SYSTEM command
  - Up to 16 messages can be suppressed
  - Significant performance improvements have been observed noted when the SVRCONN messages are suppressed
    - Up to a 20% CPU reduction seen for poorly behaved clients

IBM MQ for z/OS Wildfire Workshop

Message Suppression

- Care should be taken when selecting messages to suppress.
- A poorly behaved client is defined as one following this general pattern:
  - connect
  - open
  - put or get a single message
  - close
  - Disconnect
- Client applications that issue more MQ API requests see less CPU recovery from the message suppression
  - The big costs are in the MQCONN, MQCONNX, and MQDISC!

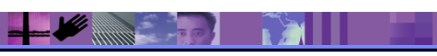
8 byte log RBA: The Problem

- WebSphere MQ for z/OS V7.1 (or prior):
  - Implements a 6 byte Log RBA (Relative Byte Address)
  - The log RBA range is 0 to x'FFFFFFFFFFFF'
  - If the Log RBA range exceeds the max (i.e. “wrap”):
    - Queue Manager terminates and requires a cold start – a disruptive outage !
    - Potential for loss of persistent data
- To avoid an outage:
  - Run CSQUTIL RESETPAGE, at regular planned intervals, to RESET the LOG RBA
- At 100MB/sec, log would be full in 1 month
  - Very high volume queue managers are getting close to this
  - Many customers are reporting reset page process required annually

IBM MQ for z/OS Wildfire Workshop

8 byte log RBA: The Problem

- For versions of WMQ 7.1 or earlier, a 6 byte log RBA (Relative Byte Address) was used. This gave a maximum log range of 0 to x'FFFFFFFFFFFF' which is 255TB. Some customer were reaching this limit within about 12 to 18 months. When the end of the log is reached, a cold start of the queue manager is required, causing a disruptive outage and potential loss of persistent data. To avoid the disruptive outage, it is necessary to have a planned outage to run the CSQUTIL RESETPAGE function to RESET the LOG RBA.
- With APAR PM48299, WMQ V7.0.1 (and above) Queue Managers will issue more message to warn of the log being exhausted.
- CSQI045I/CSQI046E/CSQI047E to warn if RBA is high (>= x'700000000000')
  - CSQI045I, when RBA is x'700000000000', x'7100..', x'7200..' and x'7300..'
  - CSQI046E when RBA is x'740000000000', x'7500..', x'7600..' and x'7700..'
  - CSQI047E when RBA is x'780000000000', x'7900..', x'nn00..' and x'FF00..'
- CSQJ031D on restart to confirm Queue Manager start-up even though log RBA has passed x'FF8000000000'
- Terminates, to prevent loss of data, if log RBA reaches x'FFF800000000'. The LOG RBA will need to be RESET to allow the Queue Manager to continue to operate



8 byte log RBA: The Solution

- Upper limit on logical log will be 64K times bigger
  - At 100MB/sec this will now take about 5578 years to fill!
- BSDS and log records changed to handle 8 byte RBAs and URIDs
  - Utilities or applications that read the following are impacted:
    - The BSDS
    - The Logs
    - Console messages that contain the log RBA or URID
- Queue Manager will use 6 byte RBA until 8 byte RBA is enabled
  - BSDS conversion utility (same model as used by DB2) to migrate to 8 byte RBA

IBM MQ for z/OS Wildfire Workshop

8 byte log RBA: The Solution

- Using an 8 byte RBA will increase the maximum log RBA by 65536 times, compared to using a 6 byte RBA. This means that it would now take about 5578 years to use the full range is writing constantly at 100MB/sec.
- The BSDS and log records have been changed to handle 8 byte RBAs and URIDs, so any application or utility that reads these will need to be updated to handle them. Console messages and MQ utilities have also been updated to output 8 byte RBAs, so if these are processed by other applications they may need to be modified to handle the extra data.
- The queue manager will continue to use 6 byte RBAs until 8 byte RBAs are enabled by using the BSDS conversion utility to migrate. This is the same model as used by DB2.

8 byte log RBA: Migration

- Get MQ V8 running in NEWFUNC mode
  - You CANNOT fall back to V7 or COMPAT mode!!!
- In QSG, ALL queue managers MUST BE in V8 NEWFUNC
- STOP queue manager
- Run the new CSQJUCNV utility
  - Utility will check that entire QSG is at correct level
  - Copies data from old primary / secondary BSDS pair to new pair
  - Old data is NOT changed for fallback
- Restart queue manager with new 8 byte format BSDS
  - Queue manager will ONLY start if in NEWFUNC mode
  - New message CSQJ034I issued indicating 8 byte RBA mode
  - All subsequent log data in new format

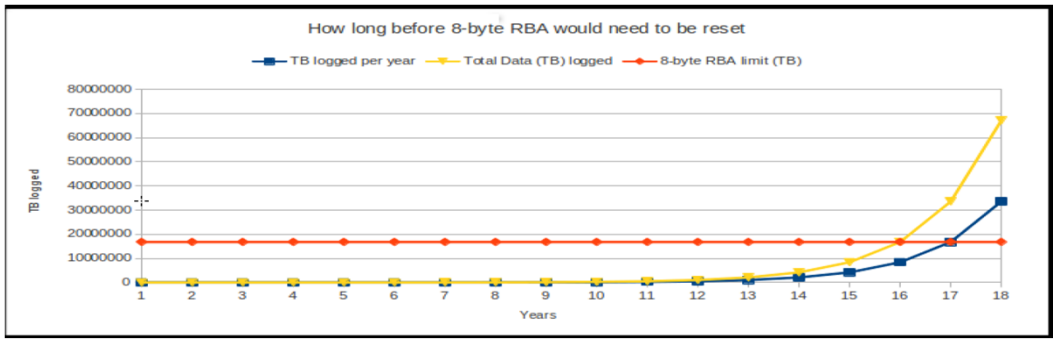
IBM MQ for z/OS Wildfire Workshop

8 byte log RBA: Migration

- To be able to migrate to using 8 byte RBA, you first need to have your queue manager running in V8 NEWFUNC mode. If in a QSG then all of the other queue managers in the QSG also need to be in NEWFUNC mode. This is because until that point you would be able to migrate the queue manager back to a previous release, and only V8 understands how to handle 8 byte RBA records.
- To convert a queue manager to 8 byte RBAs you need to run the CSQJUCNV utility. The utility will check that the entire QSG is at the correct level, if successful then it will copy the data from the primary and secondary BSDS pair into a new set enabled for 8 byte RBAs. The original BSDS pair are un-modified in-case fallback is necessary. Once the utility has been run, the queue manager can be restarted with the new 8 byte format BSDS. The queue manager will ONLY start if in NEWFUNC mode. A new message, CSQJ034I, will indicate that the queue manager is running in 8 byte RBA mode.

Logging Changes

- Log RBA constraint relief
  - Already improved messages to warn of approaching RBA
  - Now widening RBA field from 6 to 8 bytes
  - At 100MB/sec this will now take about 5578 years to fill



Log capacity improvement

- Log access changed to BSAM
  - Archive logs can now hold 4G of data when written to DASD, active logs can be defined as 4G
  - Note recommendations from KnowledgeCenter:
    - The maximum log size is 4 GB. You need to take care when defining a log of 4 GB, as the system might round up the number of records specified to a value that results in a log being greater than 4 GB.

      When the queue manager reads such a log, the log is seen as being much smaller than it actually is, giving undesirable results; for example, very frequent log switches.

      When defining a large log, you should check the HI-A-RBA value of the log using IDCAMS LISTCAT, to ensure that the log is strictly less than 4 GB (4 294 967 296 bytes).

Increasing Capacity of Active Logs

- **Active Logs today**
  - Active log datasets are limited to 4GB
  - In V7 and below, active logs limited to 31 datasets
  - If there is site problem with archiving media
    - Active logs can fill an a very short time – potentially crippling the queue manager ability to process persistent messages
- **Increasing Active log capacity**
  - Increase size of active logs datasets above 4GB
  - Increase number of active logs from 31
  - Or do both …….
- **Initial solution**
  - Delivered as part of continuous delivery POC
    - Increased number of active logs to max of 310 datasets
    - Each dataset still limited to 4GB

- Support for additional logs was added via PI46853
- For additional information, please see: https://www.ibm.com/developerworks/community/blogs/messaging/entry/Additional_active_logs_on_MQ_V8_for_z_OS?lang=en
- http://www-01.ibm.com/support/docview.wss?uid=swg1PI46853

Capped Expiry

- Message Expiration is traditionally a per-message setting
  - Set in TENTHs of seconds, it allows messages that have a temporary useful life span to be removed from the queues without application intervention
    - Like pruning a tree, removing replies that will never be retrieved can improve the health of your queues
- CAPEXPRY (note – no 'I' here!) provides an administrative solution
  - Implemented on z/OS via APAR **PI50761**
  - Implemented on distributed via FixPac 8.0.0.4
  - On z/OS it is enabled in two parts:
    - Using the 'RECOVER QMGR (TUNE CAPEXPRY ON) command on the QMGR
    - Setting the CUSTOM (CAPEXPRY(XXXX)) value on the queue

IBM MQ for z/OS Wildfire Workshop

Capped Expiry

- It applies to both Queues and Topics to enforce a maximum lifetime for messages
- Admin control of application behaviour
  - Use QALIAS for fine-grained, per application, control
  - Minimum value of all objects on resolution path used, eg QALIAS->QREMOTE->XMITQ
- Operational notes:
  - Messages already on the queue are not altered, only new messages
  - If messages have a message expiration value:
    - The SHORTER of the message expiration value or the CAPEXPRY value is used
  - If messages have the message expiration report option set, that is honoured

Capped Expiry

- Some message sources not able to provide expiry, eg MQTT
- Some applications haven't been coded with it, and can't be re-engineered
- Can lead to operational issues when getters not available, huge backlog of messages to process
- CAPEXPRY provides a way to impose a maximum expiry interval on messages
  - If no expiry specified, the CAPEXPRY value is used at MQPUT time
  - If application provides too long an expiry interval, CAPEXPRY overrides to shorter
  - But cannot lengthen expiry if the application provided value is too short
- Implemented on both z/OS and distributed platforms within CUSTOM field, eg
  ALTER QLOCAL(MY.QUEUE) CUSTOM(CAPEXPRY(50))
  imposes a 5 second lifetime on messages

Capped Expiry - control

- On z/OS each queue manager has the EXPRYINT attribute
  - Set with the ALTER QMGR command
    - Default is EXPRYINT(OFF)
  - Set the number of seconds (not tenths!) between executions of the Expiry scavenger task
- For more information on distributed, please see:
  https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/when_are_expired_websphere_mq_messages_removed_from_a_queue?lang=en

Capped Expiry - control

- On z/OS the queue manager keeps information about which queues have messages with an expiry set
- When the specified interval expires, a scavenger task kicks off to remove any expired messages automatically.
- Changes to the value take place when the current interval expires
  - Immediately if the value was 'OFF'
- QSG notes:
  - All queue managers in a QSG must have the same interval
  - Only one queue manager in the QSG will run the scavenger task
    - Typically it is the queue manager that was altered first
- For distributed platforms, please open a PMR for information

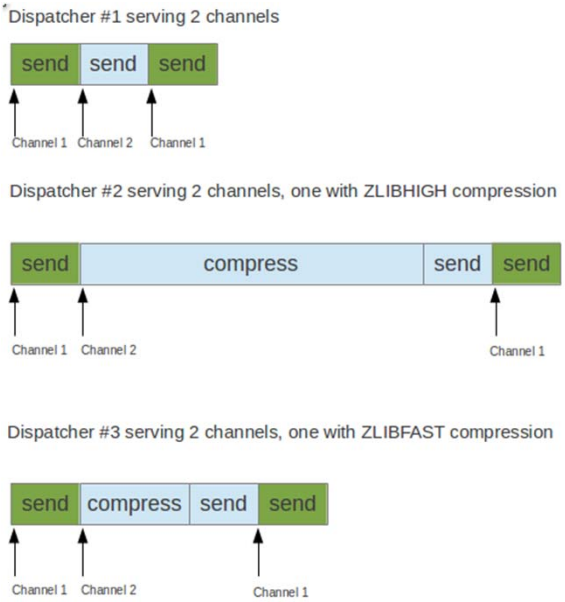
MQ for z/OS: Taking advantage of the hardware

- Exploitation of zEDC compression accelerator
  - This is a combination of hardware and software, and is a separately chargeable feature
  - SMF
    - We are awaiting performance numbers on this, but initial results looks promising!
    - Lower costs for the prolific MQ SMF 116 class 3 and 4 data.
  - Channel compression (zlibfast option) can be useful when using SSL
    - In benchmark tests zEDC hardware compression can reduce the message costs by 80% vs software compression
    - Note that this can impact the dispatcher tasks, please see MP1J for additional information

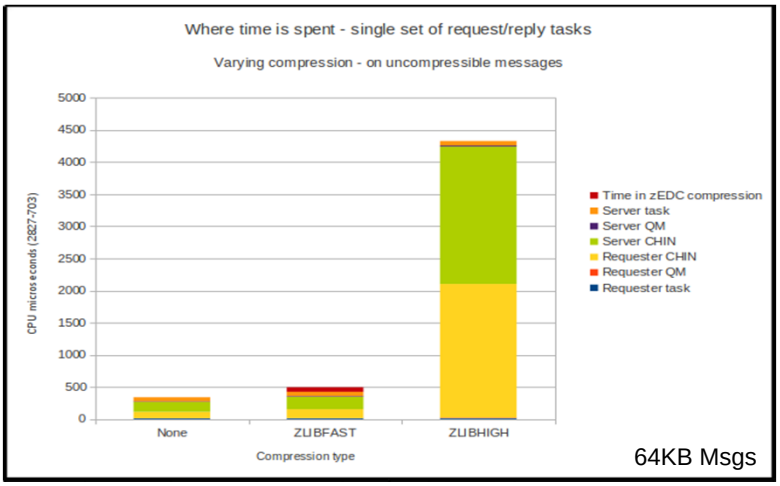
Channel Compression using zEDC hardware

- Channel Compression
  - Typically used on **high latency/low-bandwidth** networks
  - Reduces amount of data being flowed
  - Reduces CPU cost per message
  - Can yield higher reduction in CPU costs for SSL channels (dispatcher compresses before encrypting)
  - Channel attributes:
  - **COMPHDRS(ZLIBFAST)**
  - **COMPMSG(ZLIBFAST)**
  - Performed by **Dispatcher tasks** in the Channel Initiator address space

Channel Compression - Impact on Dispatcher Task

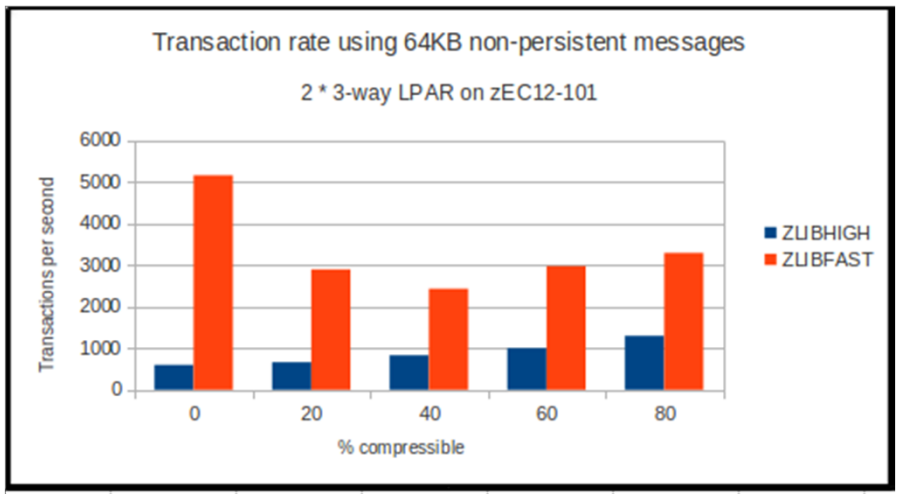


Channel Compression – Uncompressible messages

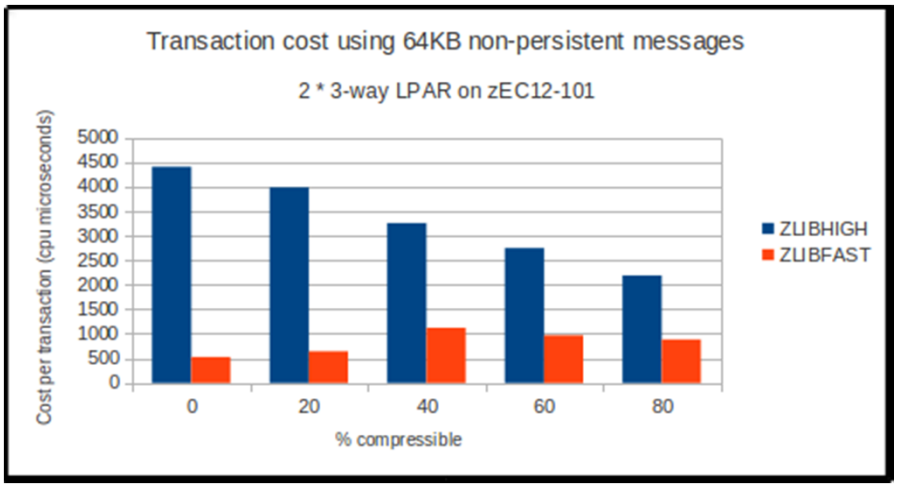


ZLIBFAST Costs Roughly 50% more Than NONE !!

Channel Compression – Compressible messages

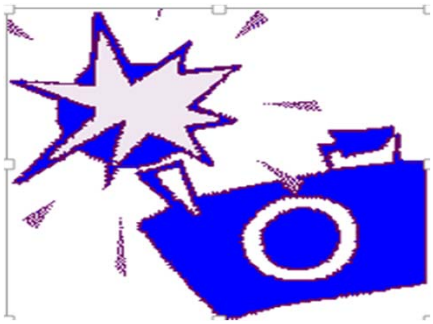


Channel Compression – Compressible messages



MQ for z/OS: Taking Advantage of SCM

- Support for Storage Class Memory (SCM) (Flash)
- Improves resiliency of Coupling Facility with cost-effective standby capacity to handle overflow of shared queues
  - Messages <63K fully held in Flash
  - Messages >63K have pointers in Flash, body in SMDS just as for traditional CF structures



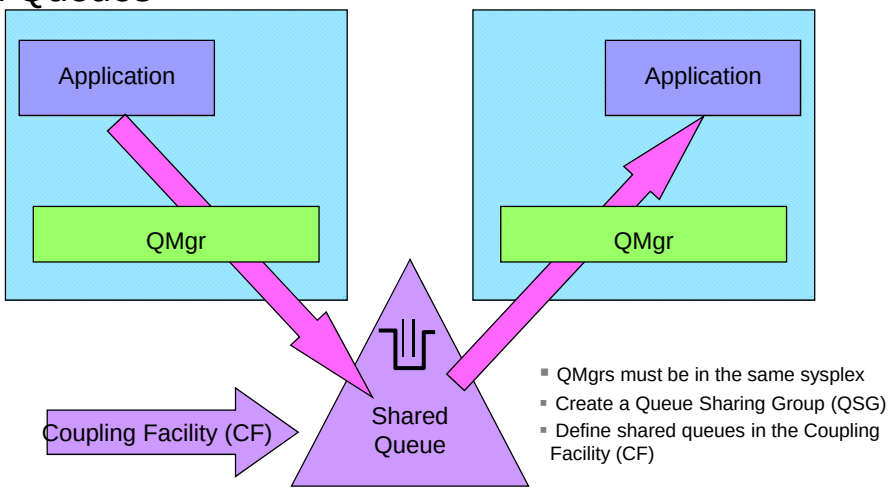
Storage Class Memory (SCM) (Flash)

•This is a change that the Coupling Facility (CF) development team have implemented to help a Queue Sharing Group (QSG) store messages. The cost of storing messages in the CF is reduced but some augmented storage (used from the CF) is required to achieve this gain.

•This is not a true version 8 feature because all the configuration is done at the CF level, with the right hardware (zEC12 or zBC12 with Flash Express cards (Solid State Drives) installed) and software. This means that this CF flash memory can be used with MQ version 8 as well as prior versions. Each Flash Express card has a capacity of 1.6 TBs and up to 4 cards can be installed giving a total of 6.4 Tbs.

With the z/OS V1R13 RSM Enablement Offering web deliverable (FMID JBB778H) for z/OS V1R13, z/OS exploits Flash through a new tier of memory called Storage Class Memory (SCM) for paging and SVC dump processing. This function is expected to provide faster paging and dump processing because flash storage is faster compared to hard disk storage. In addition to support for the existing large (1 MB) pages and frames, zEC12 supports pageable large pages when SCM is configured and allocated to z/OS.

Shared Queues

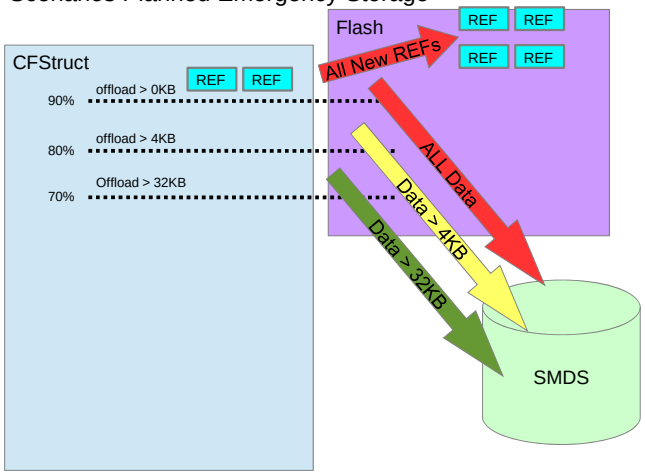


- QMgrs must be in the same sysplex
- Create a Queue Sharing Group (QSG)
- Define shared queues in the Coupling Facility (CF)

Shared Queues

- A Parallel Sysplex is a Sysplex that uses one or more coupling facilities (CFs), which provide high-speed caching, list processing, and lock processing for any applications on the Sysplex.
- MQ exploits the CF to implement Shared Queues.
- Queue Managers configured in a Queue Sharing Group (QSG) connect to the CF to process messages on shared queues. This provides for high availability.
- One of the 'problems' with shared queues is the cost of CF memory:
  - Some situations require a little storage for 'normal' processing, because messages are put and retrieved quickly, but in an emergency require significantly more storage.
    - Offloading to SMDS or DB2 does not offload the message control information (about 1K) that can contribute to filling the structure
    - Called 'emergency storage scenario'
  - Some situations want the fastest processing, and message offloading to SMDS is comparatively slow
    - Called the 'speed scenario'

CF Flash: Scenarios Planned Emergency Storage



CFSTRUCT OFFLOAD rules cause progressively smaller messages to be written to SMDS as the structure starts to fill. Leaving only the control information in the CF structure.

Once 90% threshold is reached, the queue manager stores the minimum data per message (control information) to squeeze as many messages as possible onto the shared queue.

Once at 90% threshold, CF Flash pre-staging algorithm also starts to move message control information for new messages arriving into the CF structure into SCM (assume msgs are of the same priority). Older messages, which are likely to be got first are kept in the faster CF storage.

Note: Assume all msgs < 63KB

CF Flash: Scenarios Planned Emergency Storage

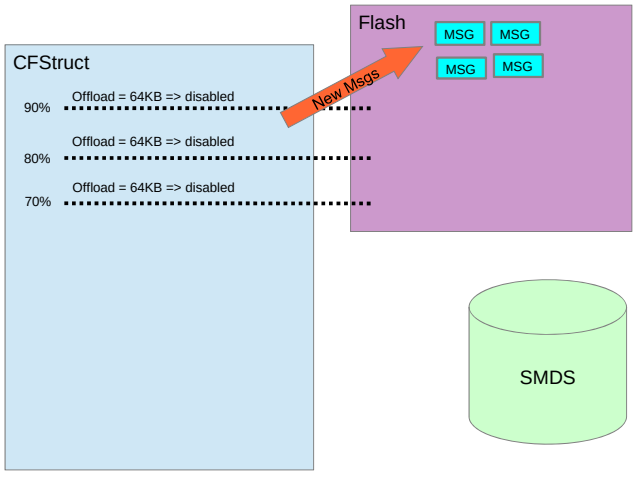
•This first slide shows a planned emergency storage scenario.

•The CF is being used mainly for messages but, Shared Message Data Sets (SMDS) are configured to hold messages once some the CF structure offload (CFSTRUCT OFFLOAD) rules come into play. The rules cause progressively smaller messages to be written to SMDS as the structure starts to fill up.

•Once the 90% threshold is reached, the queue manager stores the minimum data per message in the CF to squeeze as many message references as possible into the remaining storage in the CF structure.

•The CF development team have used queuing theory to develop the Flash algorithm. Messages that have been put most recently and have the same (i.e. all messages on the queue have the same) or lowest priority are the most unlikely to be got next, so the CF also starts moving theses new reference messages out to flash storage, keeping the faster CF storage for messages most likely to be gotten next.

CF Flash: Scenarios Maximum Speed



We want to keep high performance messages in the CF for most rapid access.

CFSTRUCT OFFLOAD are configured with special value '64k' to turn them off.

Once 90% threshold is reached, the CF Flash algorithm starts moving new messages to flash storage, keeping the faster 'real' storage for messages most likely to be gotten next.

As messages are got and deleted, the CF flash algorithm attempts to pre-stage the next messages from flash into the CFSTRUCT so they are rapidly available for MQGET.

In this scenario the flash storage acts like an extension to 'real' CFSTRUCT storage. However it will be consumed more rapidly since all message data is stored in it. Though, you could define a threshold to offload >16KB messages to SMDS if the CF structure is say 40% full. This would mean that only messages <=16KB ever get moved to flash storage.

Note: Assume all msgs < 63KB

CF Flash: Scenarios Maximum Speed

•In this scenario, we want to keep high performance messages in the CF for most rapid access.

•The CFSTRUCT OFFLOAD rules are configured with special value 64K to disable off loading.

•Once the 90% threshold is reached, the CF Flash algorithm starts moving new messages out to flash storage, keeping the faster - real storage for messages most likely to be gotten next.

•As messages are got and deleted, the CF flash algorithm attempts to pre-stage the next messages from flash into the CF structure so they are rapidly available for MQGET.

•In this scenario the flash storage acts like an extension to, real CF structure storage. However, it will be consumed more rapidly since all message data is stored in it. However, you may choose to use one rule to alter the large data threshold to indicate that all messages >16KB should be off-loaded to SMDS if the CF structure is 40% full say. This would mean that only messages <=16KB get moved to SCM.

CF Flash: Storage

Scenario	Offload Rule	Msg Size	Total Msgs	# in 'real'	SMDS space	# in 200 GB flash	Augmented (limit 30GB)
No SMDS No Flash		1kB	3M	3M			
		4kB	900,000	900,000			
		16kB	250,000	250,000			
SMDS No Flash	MQ 90%	1kB	3.2M	3.2M	800MB		
	MQ 80%	4kB	1.8M	1.8M	5GB		
	MQ 80%	16kB	1.3M	1.3M	20GB		
"Emergency" Scenario	MQ 90%	1kB	190M	2M	270GB	188M	30GB
	MQ 80%	4kB	190M	600,000	850GB	189M	30GB
	MQ 80%	16kB	190M	150,000	3TB	189M	30GB
"Speed" Scenario	CF 90%	1kB	150M	2M		148M	26GB
	CF 90%	4kB	48M	600,000		47M	8GB
	CF 90%	16kB	12M	150,000		11M	2GB

CF Structure size = 4GB

CF Flash: Storage

•In the table, a CFSTRUCT with SIZE 4 gigabytes is defined. There is a maximum of 200 GB of flash memory and an additional 30 GB of augmented storage available.

•The table addresses 4 scenarios, and shows the effect of message size in each, the amount in real, estimates how many MQ messages in CF real storage.

•First scenario has no flash nor SMDS. Entire structure is available to store messages. A total of 250,000 16 kilobyte messages can be stored.

•Second scenario introduces SMDS off-load with default rules. The 1k messages don't get to SMDS till 90% full, but the 4k and 16k cases both start off-loading at 80%. The CF space released by off-loading data can hold more message pointers, so the 1k case doesn't increase number of messages greatly, but 4k number doubles, and 16k gets 5 times as many to 1.3 million.

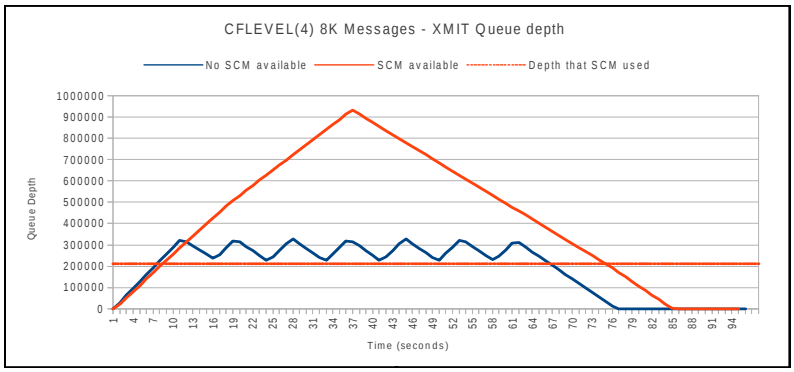
•The third scenario is our “Emergency Flash”. Because flash is configured, there is less 'real' storage for storing data, I've assumed 1 gigabytes less for 200 gigabytes flash. Here flash only holds the message references. This means the message size in flash is small. Our experiments show about 175 bytes of Augmented storage per message in flash. We have chosen to limit Augmented storage to 30 gigabytes, so message numbers are limited by augmented storage. The SMDS holds the actual data. In this scenario 190 messages can be stored.

•The last scenario is entitled “Max Speed”. All message sizes are below SMDS threshold, so SMDS not used. Limit is now how many messages will fit in 200 gigabytes flash. We show approximate size of augmented storage needed to support these volumes of messages in flash. This gives us space for 12 million messages.

•The numbers are estimates only, based on our limited test scenarios, and CFSIZER data.

CFLEVEL(4) using 8KB Messages

- Saw-tooth effect
  - Occurs when putting task goes into retry mode due to MQRC_STORAGE_MEDIUM_FULL
  - Results in 5 sec pauses
- Non-SCM workload still completes in 90% of the time of SCM workload
- CPU cost of non-SCM v's SCM workload in MVS differs by less than 2% (hence, insignificant)
- Get rate once the putting task has completed:
  - Non-SCM: 21100 x 8K messages / second ~ 164MB/sec
  - SCM: 19000 x 8K messages / second ~ 148MB/sec



CFLEVEL(4) using 8KB Messages

Chart demonstrates that put and get rates do not significantly alter as CF 'real' storage overflows and data is offloaded to, or read back from, CF flash. This is demonstrated by comparing the slopes of the red and blue lines and noticing no significant kink in red line as we overflow CF 'real' 90% threshold.

NOTE: SCM = Storage Class Memory, alternative terminology for flash storage.

The scenario being tested uses CFLEVEL(4) so SMDS config was not required. However, it corresponds identically with our “Max Speed” scenario above.

The test case has a message generator task putting 8kB messages on to a transmission queue. A channel is getting these in FIFO order. The message generator pauses for a few seconds when it hits storage media full leading to the saw-tooth shape of the blue line where no Flash memory is available.

The red dotted line indicates the threshold at which messages in the red line test, started to be written to flash storage. Notice that it is significantly lower than 90% of the 'media full' blue peaks, because some of the structure storage has gone to control flash, and the threshold is 90% of the remainder.

Final point is that cpu costs are not significantly different whether flash is being used or not.

From performance perspective, using sequential queues, flash memory behaves like CF real.

IBM MQ for z/OS Wildfire Workshop

Questions????