Morag Hughson -
hughson@uk.ibm.com
IBM UK

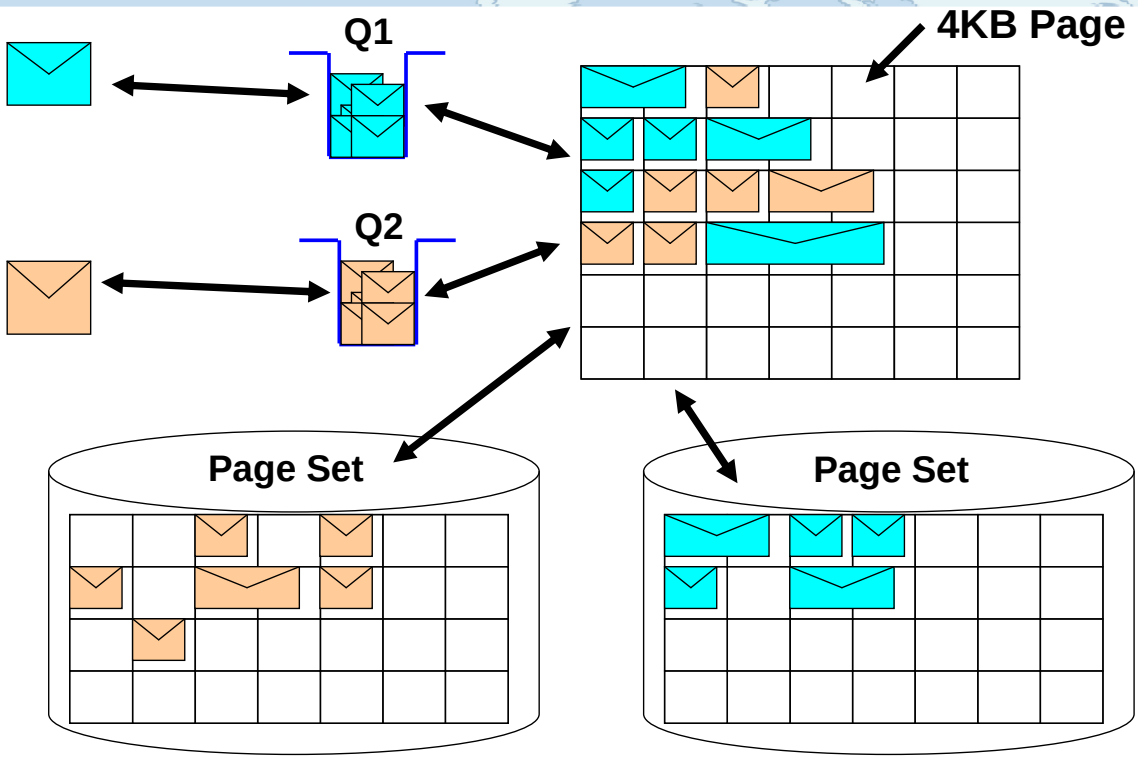
THINK GLOBAL – ACT LOCAL

IBM WebSphere MQ for z/OS Latest Features Deep Dive

IBM MQ V8 delivering best in class enterprise messaging

Platforms & Standards	*Security*	*Scalability*	*System z exploitation*
64-bit for all platforms	Userid authentication via OS & LDAP	Multiplexed client performance	64-bit buffer pools in MQ for z/OS means less paging, more performance
Support for JMS 2.0	User-based authorisation for Unix	Queue manager vertical scaling	Performance and capacity
Improved support for .Net and WCF	AMS for IBM i & z/OS	Publish/Subscribe improvements	Performance enhancements for IBM Information Replicator (QRep)
Changes to runmqsc	DNS Hostnames in CHLAUTH records	Routed publish/subscribe	Exploit zEDC compression accelerator
SHA-2 for z, i & NSS	Multiple certificates per queue manager	Multiple Cluster Transmit Queue on all platforms	SMF and shared queue enhancements

Buffer Pools: What We Have Today

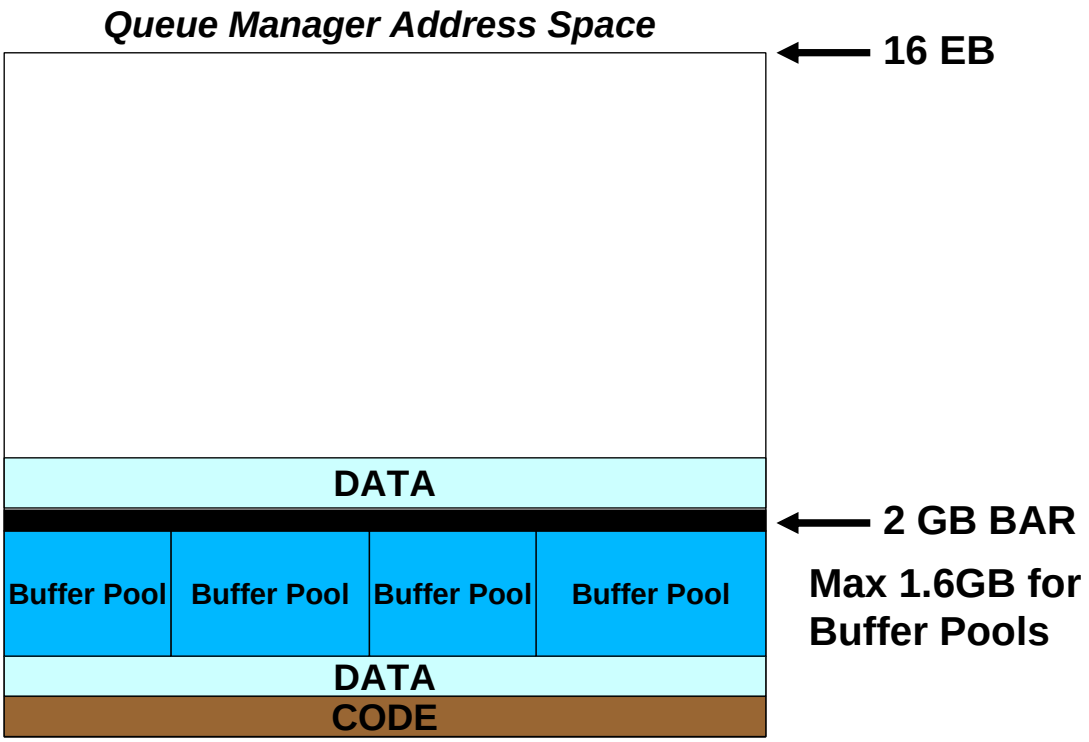


Buffer Pools: What We Have Today

N O T E S

- The diagram on the previous slide shows a representation of the current use of buffer pools with queues and page sets.
- A queue is configured to use a specific page set for the storage of messages. One of more page sets can be configured to use a particular buffer pool to “buffer” the messages.
- When a message is written to a queue, it is stored as one or more 4KB pages. The data is initially written to the buffer pool, and may at some later stage be written out to the page set. The pages may be written out to the page set if there aren’t enough pages in the buffer pool to contain all the messages on the queues. When a message is got, the data is retrieved from the buffer pool, if the necessary pages aren’t in the buffer pool, then they must first be retrieved from the page set, before the message can be returned.
- In the diagram on the previous slide, there are two queues containing messages, which are using two different page sets, however, these two page sets are both using the same buffer pool.

Buffer Pools: What We Have Today



Buffer Pools: What We Have Today

NOTES

- The diagram on the previous slide shows a representation of the queue manager address space.
- All of the queue manager code resides below the bar, in 31 bit storage.
- There is various queue manager data which resides below the bar. In previous releases, some things were moved into 64 bit storage, like locks, security data, IGQ buffer etc. Also, when new features were added, they would often exploit 64 bit storage, for example pub/sub.
- Prior to IBM® WebSphere® MQ v8.0 the buffer pools have had to be in 31 bit storage. The means that when taking into account the code and data requirements already described, there would be up to a maximum of about 1.6GB of available space for buffer pools (dependant on the common storage usage on the system).

Buffer Pools: The Problems

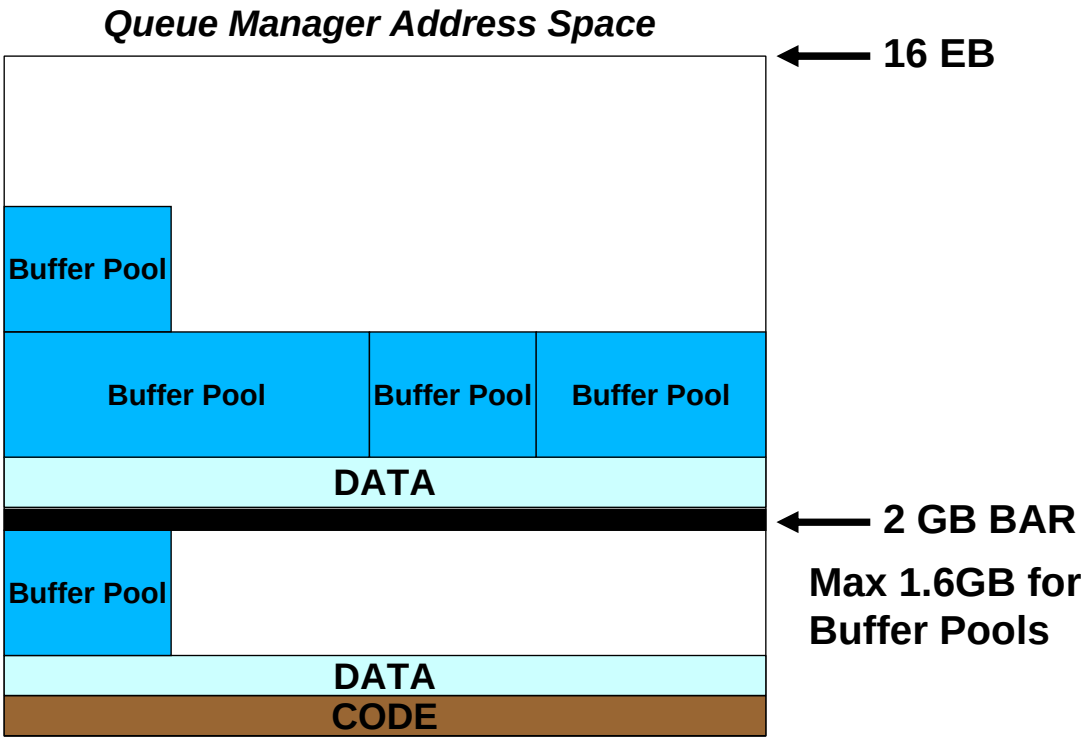
- **Not much space below the bar for buffer pools**
  - Maximum 1.6GB, depending on common area
- **Put/Get with bufferpool = 'memory' speed**
- **Put/Get from page set = 'disk' speed**
- **Can spend a lot of time moving data around**
  - Getting pages from page set into buffer pool
  - Putting pages from buffer pool into page set
  - This is detrimental for performance
- **A maximum of 16 buffer pools**
  - Although up to 100 page sets are supported
- **Lots of time spent performing tuning**
  - buffer pool sizes and queue/buffer pool/page set mappings

Buffer Pools: The Problems

NOTES

- There is not much space below the bar for buffer pools once queue manager code and data is taken into account. There is generally a maximum of 1.6GB available for buffer pools depending on common storage usage.
- Putting/getting messages into the buffer pool works at 'memory' speed where as putting/getting messages from the page set works at 'disk' speed.
- For scenarios where several applications read and/or write large numbers of messages to the same buffer pool, a lot of time is spent getting pages from page set into buffer pool and putting pages from buffer pool into page set. This is detrimental for performance.
- A maximum of 16 buffer pools are supported while up to 100 page sets are supported, meaning that if you have more than 16 page sets, you need to share the same buffer pool for some of the page sets.
- Because of these reasons, a lot of time and effort can be spent in tuning the buffer pool sizes, and the mapping of queue to page sets and page sets to buffer pools.

Buffer Pools: Using 64 bit storage



Buffer Pools: Using 64 bit storage

NOTES

- The diagram on the previous slide shows a representation of the queue manager address space, similar to the previous diagram, but this time using 64 bit storage for the buffer pools.
- Other storage usage has remained the same, with queue manager code, and some of the data remaining in 31 bit storage. However, being able to support 64 bit storage for buffer pools means that they may be moved out of 31 bit storage, relieving the constraint for other uses of the 31 bit storage. The diagram shows that buffer pools may continue to use 31 bit storage, providing a migration path to using 64 bit storage. The diagram also shows that because of the greater availability of storage above the bar, the sizes of the buffer pools may be increased, not being constrained by the 1.6GB overall storage availability.

64 Bit Buffer Pools: The Solution

- **Allow buffer pools to be above the bar.**
  - Buffer pools can (theoretically) make use of up to 16 EB of storage
- **Increase maximum size of buffer pool**
  - if above the bar
- **Allow up to 100 buffer pools**
  - Can have a 1-1 page set to buffer pool mapping

64 Bit Buffer Pools: The Solution

N O T E S

- WebSphere MQ v8.0 introduces the capability of having buffer pools above the bar. This means that buffer pools can (theoretically) make use of up to 16 EB of storage.
- The maximum size of an individual buffer pool has also increased to 999,999,999 buffers (4KB page) if the buffer pool is above the bar. The previous limit was 500,000 buffers, which remains in force if the buffer pool is located below the bar.
- The maximum number of buffer pools has been increased from 16 to 100. This enables a 1 to 1 mapping of page sets to buffer pools.
- Note: Increased storage capacity of 64-bit buffer pools is 8 billion times bigger. Ratio is equivalent to an extra thin cigarette paper (20 microns) compared to 500 Eiffel Towers one of top the other!

64 Bit Buffer Pools: What Has Changed?

- **LOCATION/LOCs attribute specifies where the buffer pool is located**
  - BELOW: The default. Buffer pool is located below the bar in 31 bit storage
  - ABOVE: Buffer pool is located above the bar in 64 bit storage
  - This can be altered dynamically
- **BUFFERS attribute has a valid range of up to 999,999,999**
  - if LOCATION(ABOVE) set
- **Buffer pool ID can be in range of 0 – 99**
- **PAGECLAS attribute enables permanent backing by real storage for maximum performance**
  - 4KB – The default, uses page-able 4KB buffers
  - FIXED4KB – Uses permanently page fixed 4KB buffers. Only supported if LOCATION(ABOVE) is specified

64 Bit Buffer Pools: What Has Changed?

NOTES

- A new attribute called LOCATION (LOC for short) has been added to the buffer pool definition. This enables the location, relative to the bar to be specified. A value of “BELOW” indicates that the buffer pool should be below the bar in 31 bit storage, this is the default, and matches what was available in WebSphere MQ v7.1 and earlier releases. A value of “ABOVE” indicates that the buffer pool should be located above the bar in 64 bit storage. The LOCATION value can be altered dynamically, and the queue manager will then dynamically move the buffer pool to the new location.
- The buffer pool BUFFERS attribute now has an extended valid range, being able to accept a value up to 999,999,999 if LOCATION(ABOVE) has been set.
- The buffer pool ID attribute can now have a valid range of 0 – 99.
- A new attribute called PAGECLAS has been added to the buffer pool definition. This attribute enables 64 bit buffer pools to be configured to be permanently backed by real storage for maximum performance. The default value of 4KB means that the buffer pool will be page-able. Using a value of FIXED4KB means that WebSphere MQ does not have to pagefix/pageunfix buffers when doing I/O. This can give significant performance benefits if the buffer pool is under stress, and therefore doing lots of reads from/writes to page set.
- *WARNING* storage will be page fixed for the life of the queue manager. Ensure you have sufficient real storage otherwise other address spaces might be impacted.

64 Bit Buffer Pools: Migration

- **To use this function OPMODE(NEWFUNC,800) must be set**
  - Otherwise behaviour is same as in version 7
  - Although LOCATION(BELOW) is valid regardless of OPMODE
- **Some messages have changed regardless of the value of OPMODE**

64 Bit Buffer Pools: Migration

N O T E S

- To exploit 64 bit storage for the buffer pools, the queue manager must be running in v8.0 NEWFUNC mode. However, LOCATION(BELOW) can be specified when running in COMPAT mode.
- Some console messages have changed, regardless of OPMODE. For example to display the location of the buffer pool when using the DISPLAY USAGE PSID command.

64 Bit Buffer Pools: Configuration

- **CSQINP1**
  - DEFINE BUFFPOOL(22) LOCATION(ABOVE) BUFFERS(1024000) REPLACE
  - DEFINE BUFFPOOL(88) BUFFERS(12000) REPLACE
- **CSQINP1 or dynamically with dsn**
  - DEFINE PSID(22) BUFFPOOL(22) REPLACE
- **CSQINP2 or dynamically**
  - ALTER BUFFPOOL(88) LOC(ABOVE)

```
CSQP024I !MQ21 Request initiated for buffer pool 88
CSQ9022I !MQ21 CSQPALTB ' ALTER BUFFPOOL' NORMAL COMPLETION
CSQP023I !MQ21 Request completed for buffer pool 88,  now has 12000 buffers
CSQP054I !MQ21 Buffer pool 88 is now located above the bar
```


64 Bit Buffer Pools: Configuration

- **CSQINP2 or dynamically**
  - DISPLAY USAGE PSID(*)

```
CSQI010I !MQ21 Page set usage …
<REMOVED>
End of page set report
CSQI065I !MQ21 Buffer pool attributes ...
      Buffer   Available   Stealable   Stealable   Page       Location
        pool     buffers     buffers  percentage   class
_          0        1024        1000          99   4KB        BELOW
_         22     1024000      234561          23   FIXED4KB   ABOVE
_         88       12000        1200          10   4KB        ABOVE
      End of buffer pool attributes
```

64 Bit Buffer Pools: Performance Summary

Single Requester per Queue:

Test	Transaction Rate (per second)	Transaction Cost (cpu microseconds)	LPAR %Busy	Channel Path %Busy
31-bit	232762	35.92	54%	56%
64-bit	235217	37.48	57%	57.4%
64-bit (enough buffers)	324213	38.12	83%	0.07%
64-bit (4GB per buffer pool)	341412	38.23	83%	0.08%

2 Requesters per Queue:

Test	Transaction Rate (per second)	Transaction Cost (cpu microseconds)	LPAR %Busy	Channel Path %Busy
31-bit	149140	42.3	42%	75.4%
64-bit	145623	44.84	43.5%	75.9%
64-bit (enough buffers)	384062	40.65	99.59%	0.08%
64-bit (4GB per buffer pool)	370546	52.15	99.69%	0.07%

- 16 CP LPAR
- Each transaction puts and gets a random message from a pre loaded queue. Second test requires a doubling in buffer pool size

64 Bit Buffer Pools: Performance Summary

NOTES

- The previous slide shows two tables comparing the performance of 31 bit buffer pools and 64 bit buffer pools.
- The first table shows the results when running tests using a single requester on a queue. There is a small increase in transaction cost when using 64 bit buffer pools vs 31 bit buffer pools, with the CPU microseconds increasing from 35.92 to 37.48. However, when we increase the number of buffers in use in the 64 bit case, the channel path %busy drops to nearly 0, indicating that we are no longer needing to access the page set, and all requests are satisfied from the buffer pool. The transaction rate has also increased by about 40%.
- The second table shows that when using two requesters against the queue, there is a high channel path %busy rate, of about 75%, for both the 31 bit and 64 bit buffer pool case. However, when extra buffers are added in the 64 bit case, this channel path busy drops to nearly 0 and the transaction rate more than doubles. The LPAR busy % also increases from about 43% to very close to 100%, showing that we are now driving the system as hard as possible.
- Being able to provide more buffers by using 64 bit storage means that we can drive the system much more efficiently for a slight increase in per transaction cost.

8 byte log RBA: The Problem

- **WebSphere MQ for z/OS V7.1 (or earlier):**
  - Implements a 6 byte Log RBA (Relative Byte Address)
  - This give an RBA range of 0 to x'FFFFFFFFFFFF' (= 255TB)
  - Some customers reach this limit in 12 to 18 months
  - At 100MB/sec, log would be full in 1 month
- **If we reach the end of the Log RBA range (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

8 byte log RBA: The Problem

N O T E S

- For versions of WebSphere MQ 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, WebSphere MQ V7.0.1 (and above) queue managers will issue more messages 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

8 byte log RBA: The Solution

N O T E S

- 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 WebSphere MQ utilities have been updated to handle both 6 byte and 8 byte RBA logs and will always 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 WebSphere MQ v8.0 running in NEWFUNC mode (you promise not to fall back to V7)**
- **In QSG, entire group must be v8.0 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

8 byte log RBA: Migration

NOTES

- To be able to migrate to using 8 byte RBA, you first need to have your queue manager running in v8.0 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 v8.0 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 unmodified 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.

Chinit SMF: The Problem

- Prior to WebSphere MQ v8.0 no SMF data for Chinit address space or channel activity
- Many customers have had to create their own ‘monitoring’ jobs with periodic DISPLAY CHSTATUS
- Difficult to manage historical data, perform capacity planning and investigate performance issues

Chinit SMF: The Solution

- Additional SMF data for Chinit address space and channel activity to enable:
  - Monitoring
  - Capacity planning
  - Tuning
- Separate controls from queue manager SMF allows 'opt in'
- Updated MP1B SupportPac will format the data and introduces rule based reporting

Chinit SMF: The Solution

- **You can START Chinit STAT trace by:**

```
!MQ08 START TRACE(STAT) CLASS(4)
CSQW130I !MQ08 'STAT' TRACE STARTED, ASSIGNED TRACE NUMBER 05
CSQ9022I !MQ08 CSQWVCM1 ' START TRACE' NORMAL COMPLETION
```

- **You can START Chinit ACCTG trace by:**

```
!MQ08 START TRACE(ACCTG) CLASS(4)
CSQW130I !MQ08 'ACCTG' TRACE STARTED, ASSIGNED TRACE NUMBER 06
CSQ9022I !MQ08 CSQWVCM1 ' START TRACE' NORMAL COMPLETION
```

- **You can display trace by:**

```
!MQ08 DISPLAY TRACE(*)
CSQW127I !MQ08 CURRENT TRACE ACTIVITY IS -
TNO     TYPE    CLASS   DEST        USERID RMID
02      STAT    01      SMF         *      *
05      STAT    04      SMF         *      *
06      ACCTG   04      SMF         *      *
END OF TRACE REPORT
```

- **ALTER and STOP TRACE commands have also been updated**

Chinit SMF: Controls

- **SMF data collected on SMF interval of QMGR**
  - Can be same as z/OS SMF interval
- **Chinit STAT trace allows high level view of activity**
- **Chinit ACCTG trace allows detailed view at channel level**
  - STATCHL attribute on channel to control granularity
  - Data collected is a superset of that collected on distributed with STATCHL message

Chinit SMF: New record subtypes and DSECTs

- **SMF 115**
  - Subtype 231 (0xE7='X') for CHINIT control information
    - e.g. adapter and dispatcher CPU time etc. to help with tuning numbers of tasks configured
    - DNS resolution times
- **SMF 116**
  - Subtype 10 for per channel 'accounting' data
    - Bytes sent, achieved batch size etc.

- **New DSECTs**
  - **CSQDQWSX (QWSX)** : Self defining section for subtype 231 which consists of:
    - **CSQDQWHS (QWHS)** : Standard header
    - **CSQDQCCT (QCCT)** : Definition for CHINIT control information
  - **CSQDQWS5 (QWS5)** : Self defining section for subtype 10 which consists of:
    - **CSQDQWHS (QWHS)** : Standard header
    - **CSQDQCST (QCST)** : Definition for channel status data

CHINIT Summary

```
MV45,MQ20,2014/04/08,20:43:57,VRM:800,
From 2014/04/08,20:41:54.984681 to 2014/04/08,20:43:57.237939
duration 122.253258 seconds
Number of current channels..................... 20
Number of active channels .... ................ 20
MAXCHL. Max allowed current channels.......... 602
ACTCHL. Max allowed active channels........... 602
TCPCHL. Max allowed TCP/IP channels........... 602
LU62CHL. Max allowed LU62 channels............ 602
Storage used by Chinit......................... 22MB
```

Dispatchers

```
MV45,MQ20,2014/04/08,20:43:57,VRM:800,
From 2014/04/08,20:41:54.984681 to 2014/04/08,20:43:57.237939
duration 122.253258 seconds
Task,Type,Requests,Busy %,     CPU used, CPU %,"avg CPU","avg ET"
   ,   ,        ,     ,      Seconds,      , uSeconds,uSeconds
   0,DISP,      46,   0.0,   0.000000,  0.0,        12,       14
   1,DISP,  168912,   1.4,   0.028218,  0.0,         8,       10
   2,DISP,  168656,   1.3,   0.025142,  0.0,         7,       10
   3,DISP,       0,   0.0,   0.000000,  0.0,         0,        0
   4,DISP,       0,   0.0,   0.000000,  0.0,         0,        0
Summ,DISP,  337614,   0.6,   0.053360,  0.0,         9,       10

   0,DISP, number of channels on this TCB,    0
   1,DISP, number of channels on this TCB,   10
   2,DISP, number of channels on this TCB,   10
   3,DISP, number of channels on this TCB,    0
   4,DISP, number of channels on this TCB,    0
Summ,DISP, number of channels on all TCBs,   20
```


Adapters

```
MV45,MQ20,2014/04/08,20:43:57,VRM:800,
From 2014/04/08,20:41:54.984681 to 2014/04/08,20:43:57.237939
duration 122.253258 seconds
Task,Type,Requests,Busy %,     CPU used, CPU %,"avg CPU","avg ET"
   ,   ,        ,     ,      Seconds,      , uSeconds,uSeconds
   0,ADAP,  127599,  16.5,   0.953615,  0.8,         7,      158
   1,ADAP,   46790,   7.6,   0.309678,  0.3,         7,      199
   2,ADAP,   13702,   3.2,   0.065380,  0.1,         5,      284
   3,ADAP,    2909,   0.7,   0.029541,  0.0,        10,      279
   4,ADAP,     395,   0.1,   0.003179,  0.0,         8,      392
   5,ADAP,      37,   0.0,   0.000241,  0.0,         7,      149
   6,ADAP,      10,   0.0,   0.000175,  0.0,        17,      111
   7,ADAP,       0,   0.0,   0.000000,  0.0,         0,        0
Summ,ADAP,  191442,   3.5,   1.361809,  0.1,         7,      179
```

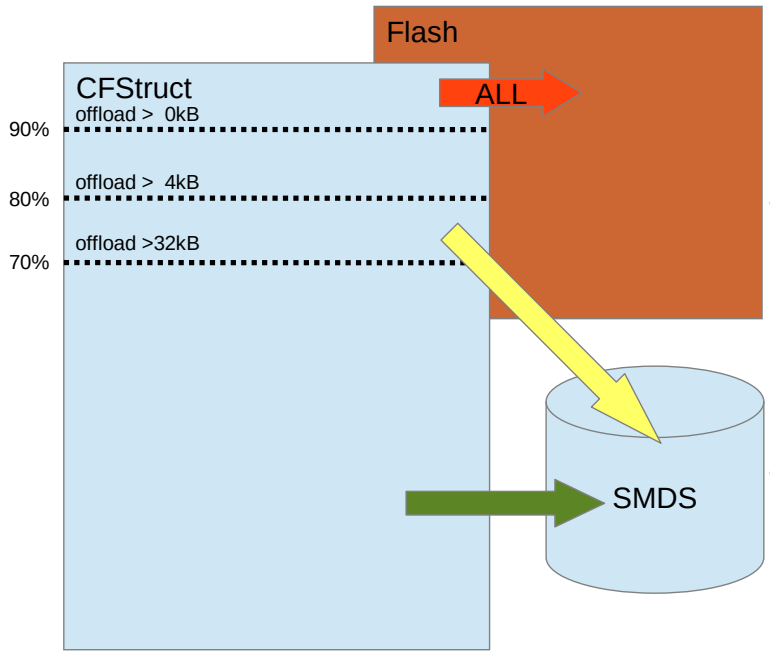
Channel Information (1)

127.0.0.1	MQ89_1	Connection name	127.0.0.1
127.0.0.1	MQ89_1	Remote qmgr/app	MQ89
127.0.0.1	MQ89_1	Channel disp	PRIVATE
127.0.0.1	MQ89_1	Channel type	SENDER
127.0.0.1	MQ89_1	Channel status	RUNNING
127.0.0.1	MQ89_1	Channel STATCHL	HIGH
127.0.0.1	MQ89_1	Channel started date & time	2014/04/08,19:41:48
127.0.0.1	MQ89_1	Channel stopped time	
127.0.0.1	MQ89_1	Channel status collect time	2014/04/08,19:43:57
127.0.0.1	MQ89_1	Last msg time	2014/04/08,19:43:52
127.0.0.1	MQ89_1	Active for	122 seconds
127.0.0.1	MQ89_1	Batch size	50
127.0.0.1	MQ89_1	Messages/batch	38.9
127.0.0.1	MQ89_1	Number of messages	2,998
127.0.0.1	MQ89_1	Number of persistent messages	1,506
127.0.0.1	MQ89_1	Number of batches	77
127.0.0.1	MQ89_1	Number of full batches	42
127.0.0.1	MQ89_1	Number of partial batches	35
127.0.0.1	MQ89_1	Buffers sent	3,319
127.0.0.1	MQ89_1	Buffers received	109
127.0.0.1	MQ89_1	Xmitq empty count	13

Channel Information (2)

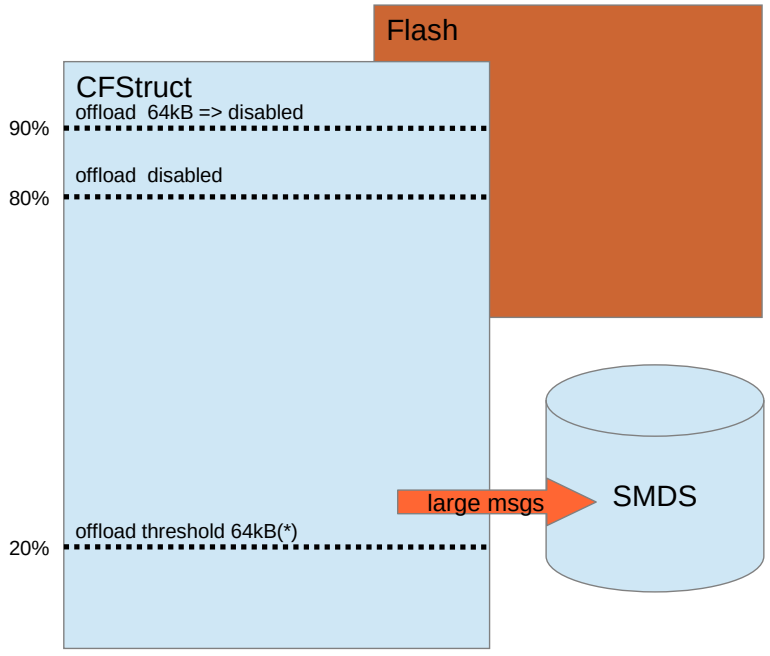
127.0.0.1	MQ89_1	Message data	17,198,653	16 MB
127.0.0.1	MQ89_1	Persistent message data	4,251,780	4 MB
127.0.0.1	MQ89_1	Non persistent message data	12,946,873	12 MB
127.0.0.1	MQ89_1	Total bytes sent	17,200,221	16 MB
127.0.0.1	MQ89_1	Total bytes received	3,052	2 KB
127.0.0.1	MQ89_1	Bytes received/Batch	39	39 B
127.0.0.1	MQ89_1	Bytes sent/Batch	223,379	218 KB
127.0.0.1	MQ89_1	Batches/Second	0	
127.0.0.1	MQ89_1	Bytes received/message	1	1 B
127.0.0.1	MQ89_1	Bytes sent/message	5,737	5 KB
127.0.0.1	MQ89_1	Bytes received/second	25	25 B/sec
127.0.0.1	MQ89_1	Bytes sent/second	140,985	137 KB/sec
127.0.0.1	MQ89_1	Compression rate	0	
127.0.0.1	MQ89_1	Exit time average	0 uSec	
127.0.0.1	MQ89_1	DNS resolution time	0 uSec	
127.0.0.1	MQ89_1	Net time average	312 uSec	
127.0.0.1	MQ89_1	Net time min	43 uSec	
127.0.0.1	MQ89_1	Net time max	4,998 uSec	
127.0.0.1	MQ89_1	Net time max date&time	2014/04/08,19:43:52	

CF Flash: Scenarios
Planned Emergency Storage



- CFSTRUCT OFFLOAD rules cause progressively smaller messages to be written to SMDS as the structure starts to fill.
- Once 90% threshold is reached, the queue manager is storing the minimum data per message to squeeze as many message references as possible into the remaining storage.
- CF Flash algorithm also starts moving the middle of the queue out to flash storage, keeping the faster 'real' 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.
  - you might choose to use one rule to alter the 'large data' threshold from 63kB down
- Once 90% threshold is reached, the CF Flash algorithm starts moving the middle of the queue 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 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 small message data is stored in it.

CF Flash: Storage (4GB Structure)

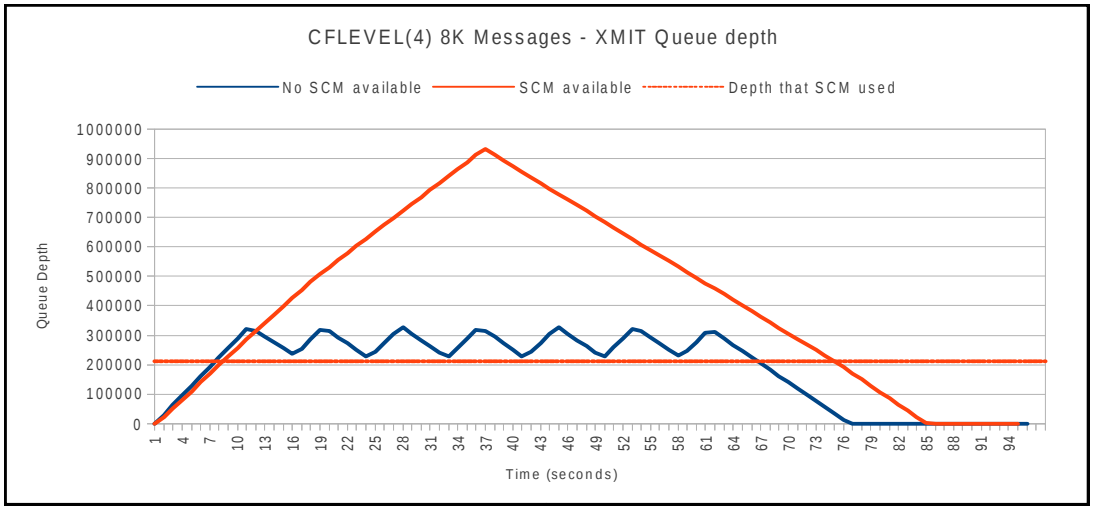
Scenario	Msg Size	Total Msgs	# in 'real'	SMDS space	# in 200 GB flash	Augmented (limit 30GB)
No SMDS No Flash	**1kB**	**3M**	**3M**			
	4kB	**900,000**	**900,000**			
	16kB	**250,000**	**250,000**			
SMDS No Flash	**1kB**	**3.2M**	**3.2M**	**800MB**		
	4kB	**1.8M**	**1.8M**	**5GB**		
	16kB	**1.3M**	**1.3M**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

CF Flash: Storage

NOTES

- Example: we define a CFSTRUCT with SIZE 4GB. We have a maximum of 200GB flash memory available, but we decide not to use more than an additional 30GB of augmented storage.
- Table Considers 4 scenarios, and shows the effect of message size in each.
  # in real: estimates how many in CF real storage
- First scenario has no flash nor SMDS. Entire structure is available to store messages.
- Second scenario introduces SMDS offload with default rules. The 1k messages don't got to SMDS til 90% full, but the 4k and 16k cases both start offloading at 80%. The CF space released by offloading data can hold more message pointers, so the 1k case doesn't increase number of messages greatly, but 4k number doubles, and 16k gets 5x as many.
- Third scenario is our chart#1 "Emergency Flash".
  Because flash is configured, there is less 'real' storage for storing data, I've assumed 1GB less for 200GB 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 30GB, so message numbers are limited by augmented storage. SMDS holds data.
- Fourth scenario is our chart#2 "Max Speed".
  All message sizes are below SMDS threshold, so SMDS not used.
  Limit is now how many messages will fit in 200GB flash. We show approximate size of augmented storage needed to support these volumes of messages in flash.
- CAVEAT: Estimates only, based on our limited test scenarios, and CFSIZER data.

CFLEVEL(4) using 8KB Messages



- **Saw-tooth effect occurs when capture task goes into retry mode due to "storage medium full" reason code.**
- **Even with these 5 second pauses, the non-SCM capable workload completes in 90% of the time of the SCM capable workload.**
- **Cost of workload in MVS differs by less than 2%.**
- **Get rate once the capture task has completed:**
  No SCM: 21100 messages / second ~ 164MB/sec
  SCM: 19000 messages / second ~ 148MB/sec

CFLEVEL(4) using 8KB Messages

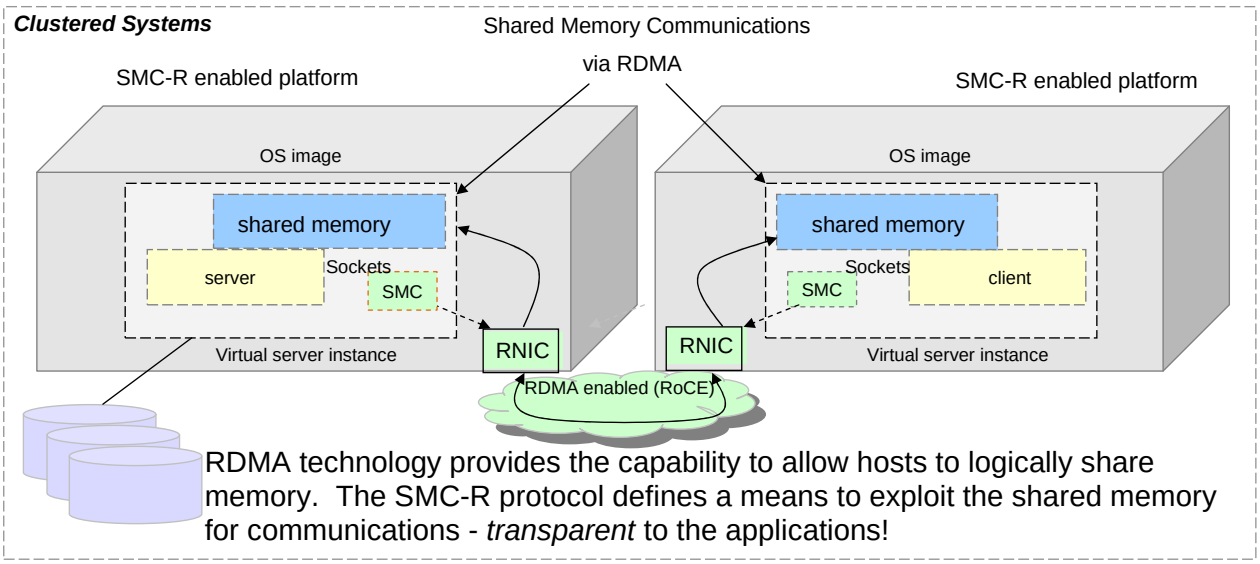
N O T E S

- 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.
- 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.

THINK GLOBAL – ACT LOCAL

Shared Memory Communications over RDMA

"Shared Memory Communications over RDMA" concepts



RDMA technology provides the capability to allow hosts to logically share memory. The SMC-R protocol defines a means to exploit the shared memory for communications - *transparent* to the applications!

This solution is referred to as *SMC-R* (Shared Memory Communications over RDMA). SMC-R is an *open* sockets over RDMA (Remote Data Memory Access) protocol that provides transparent exploitation of RDMA (for TCP based applications) while preserving key functions and qualities of service from the TCP/IP ecosystem that enterprise level servers/network depend on!

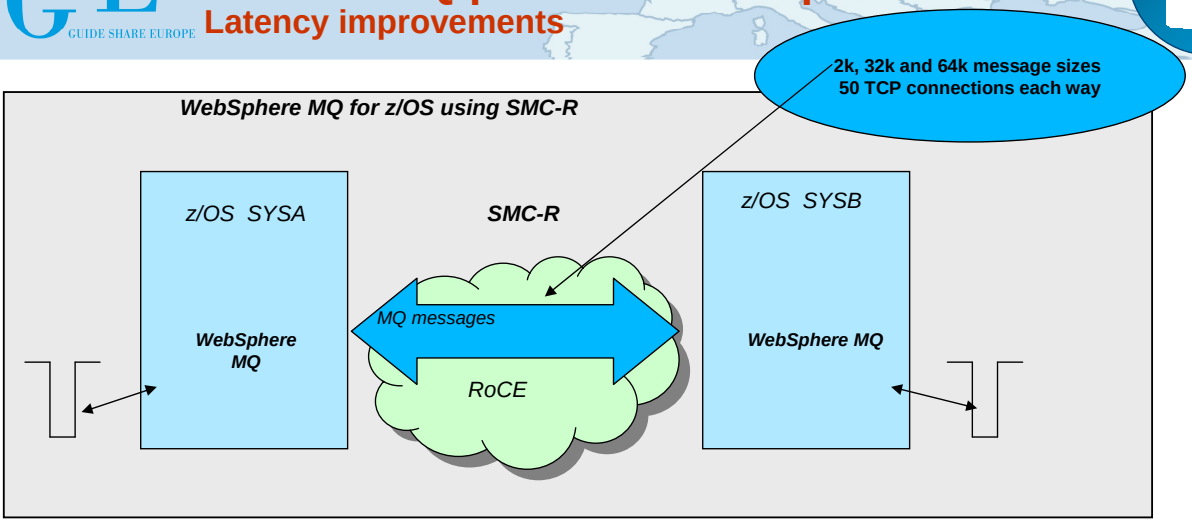
Draft IETF (Internet Enginnering Task Force) RFC for SMC-R: http://tools.ietf.org/html/draft-fox-tcpm-shared-memory-rdma-03

SMC-R - MQ performance improvement

- **Latency improvements**
- **Workload**
  - Measurements using WebSphere MQ for z/OS V7.1.0
  - MQ between 2 LPARs on zEC12 machine (10 processors each)
  - On each LPAR, a queue manager was started and configured with 50 outbound sender channels and 50 inbound receiver channels, with default options for the channel definitions (100 TCP connections)
  - Each configuration was run with message sizes of 2KB, 32KB and 64KB where all messages were non-persistent
    - Results were consistent across all three message sizes

SMC-R - MQ performance improvement

Latency improvements



•WebSphere MQ for z/OS realizes *up to a 3x increase* in messages per second it can deliver across z/OS systems when using SMC-R vs standard TCP/IP *

Based on internal IBM benchmarks using a modeled WebSphere MQ for z/OS workload driving non-persistent messages across z/OS systems in a request/response pattern. The benchmarks included various data sizes and number of channel pairs. The actual throughput and CPU savings users will experience may vary based on the user workload and configuration.

THINK GLOBAL – ACT LOCAL

Other z/OS Items

64bit application support

- **64 bit application support for C language**
  - no 64bit COBOL
  - LP64 compile option
  - supported by cmqc.h
- **restricted environments**
  - batch, TSO, USS
  - CICS® and IMS® do not support 64bit apps
  - WebSphere Application Server already 64bit
- **must use sidedeck & DLL, not stubs:**
  - csqbmq2x (uncoordinated batch & USS)
  - csqbrr2x (RRS coordinated, srrcmit())
  - csqbri2x (RRS coordinated, MQCMIT)

Clients

- **Client Attachment Feature no longer exists in MQ v8.0**
  - Client capability available by default
    - Use CHLAUTH rules to protect QMGR if you didn't previously use clients
  - Client attachment feature also now non-chargeable on previous releases
    - APAR (PI13429) also available to enable functionality without installing CAF

- **CSQUTIL MAKECLNT stabilised**
  - New V8 attributes on CLNTCONN not added on z/OS
  - MAKECLNT on V8 reports that you shouldn't use it
  - Use **`runmqsc -n`** instead on the client machine

Clients

NOTES

- The Client Attachment Feature that has been required on previous versions of WebSphere MQ to connect client applications into a z/OS queue manager no longer exists in WebSphere MQ v8.0. The client capability exists by default on a v8.0 queue manager. If you didn't previously use client applications into a z/OS queue manager, you consider using CHLAUTH rules to protect the queue manager.

- The CSQUTIL MAKECLNT feature has been stabilised at pre-V8 level. This means you cannot use it to make a V8 level CCDT. You can still use it to make a V7.1 level CCDT, but it will report it is stabilised and that you should instead use the new V8 runmqsc feature to create and edit CCDTs. This is done with the **`runmqsc -n`** flag.

Other z/OS Items

- **Message suppression EXCLMSG**
  - formalized a service parm which suppressed Client channel start & stop messages
  - extended to be generalized and applicable for most MSTR and CHIN messages
- **DNS reverse lookup inhibit REVDNS**
  - response to customer requirement for workaround when DNS infrastructure impacted
- **zEDC compression hardware exploitation for COMPMSG(ZLIBFAST)**
  - need zEC12 GA2 + zEDC card
  - can yield higher throughput & reduced CPU for SSL channels

Other z/OS Items

N O T E S

Message suppression EXCLMSG

- EXCLMSG is a zParm value, but can also be specified using the SET SYSTEM command. Specify a list of message identifiers to be excluded from being written to any log. Messages in this list are not sent to the z/OS console and hardcopy log. As a result using the EXCLMSG parameter to exclude messages is more efficient from a CPU perspective than using z/OS mechanisms such as the message processing facility list and should be used instead where possible. The default value is an empty list ( ).
- Message identifiers are supplied without the CSQ prefix and without the action code suffix (I-D-E-A). For example, to exclude message CSQX500I, add X500 to this list. This list can contain a maximum of 16 message identifiers.

DNS reverse lookup inhibit REVDNS

- The queue manager REVDNS attribute can now be used to prevent the use of DNS for host name lookup. In earlier releases a serviceparm was provided to some customers so that channel hangs could be avoided in situations where a DNS infrastructure becomes non-responsive.

zEDC compression hardware exploitation

- When using COMPMSG(ZLIBFAST) on channels to message compression, this will exploit the zEDC card if also using the zEC12 GA2. This can yield higher throughputs and reduced CPU cost for SSL channels.