MQ for z/OS – Performance report MP16 update!

Tony Sharkey
Published on 13/03/2019

The latest version of MP16 – the capacity planning and tuning guide for MQ for z/OS is now available.

This version of MP16 and other performance documents can now be found on the mqperf github repository.

MP16 Updates

This latest update (March 2019) includes a number of changes that have been driven as a result of customer-driven questions and we have taken the opportunity to take some of the guidance from archived performance reports and re-assess on more modern hardware.

The following items are discussed in detail in MP16 but I will mention them here and share the highlights below:

1. Backup CF structure limitations.
2. Tuning shared message data sets (SMDS) for best performance.

Backup CF structure limitations

MQ has implementation limits as to how much data can be recovered. The key attributes to check are:

- Size of active logs
- Number of active logs
- MAXARCH – which is the maximum number of archive logs that can be recorded in the BSDS. Once this number is exceeded, recording begins again at the start of the BSDS.

Using this data you can calculate the maximum amount of data that your queue manager is able to recover the from a “BACKUP CFSTRUCT” command.

For example, a queue manager with 10 active logs of 4GB and a MAXARCH of 100, has the capacity to recover:

Active (4 * 10) + Archive: (4 * 100) = Total 440GB

Should you attempt to recover data that exceeds this total, or where the start point of the recovery is outside the scope of the data in the active and archive logs, you will not be able to recover the data.
In terms of limitations, the maximum amount of data that a queue manager could backup and successfully recover is 5.1TB, based on 1000 archive logs and 310 active logs – however a single SMDS has a maximum capacity of 16TB.

**Tuning shared message data sets (SMDS) for best performance.**

We have taken the guidance from the [MQ V7.1.0 performance report](#) (now archived) that discusses how to tune your SMDS for best performance, remeasured on the latest hardware and refreshed the data.

The section discusses tuning DSBUFS and DSBLOCKS to minimize delays for I/O and demonstrates that with a little bit of tuning, you can improve the performance up to 5 times!

**Shared queues with INDXTYPE(GROUPID) – Optimizing match options for best performance.**

Message grouping is a useful way to allow logical groups of messages to be associated together.

Consideration should be given to the size of the message groups as groups consisting of 1 message are not processed as efficiently as the equivalent get using CORRELID.

When selecting groups of messages from shared queues, there are some ways that are more efficient than others – particularly when the queue depth increases. This is largely as some MQGMO option configurations will use get-specific to locate the message group, and others use get-next to scan messages until the desired group is located.

In terms of best performance with grouped messages on shared queues:

1. Specify MQGMO’s MatchOptions to include MQMO_MATCH_GROUP_ID plus MQGMO Option to include MQMO_ALL_MSGS_AVAILABLE and MQGMO_LOGICAL_ORDER
2. If MQGMO_LOGICAL_ORDER is not appropriate, specify MQGMO’s MatchOptions to include MQMO_MATCH_GROUP_ID plus MQMO_MATCH_SEQ_NUMBER and specify the MQMDE MsgSeqNumber (default is 1).

The chart below shows the difference in performance when randomly selecting a message using a GroupID from a shared queue with 100,000 messages.
There is a significant difference in the rate and cost when the queue manager has to rely on get-next processing, particularly on deep queues.

MQ’s class(3) accounting data will indicate if the MQGET is using get-specific or get-next.

Note that a JMS application using GroupID will use get-next with shared queue, and performance will be impacted by the depth of the queue.

Also note that private queues are able to use the GroupID index even when the MQMO_MATCH_MSG_SEQ_NUMBER is not specified, ensuring get-specific processing, thus avoiding a performance impact from deep queues.

**Using message selectors.**

Using message selectors to identify messages with specific properties can be an expensive way to identify particular messages. This is particularly true when there are many messages on the queue.

For both private and shared queue, MQ must retrieve each message and attempt to match the properties in the message against the desired selection criteria.

For private queue, you may see high instances of messages skipped in the class(3) accounting data relative to the number of MQGETs completed.

For shared queue, you may see high instances of IXLLSTM calls relative to the number of successful MQGETs. You may also see high CF CPU utilisation when MQ attempts to locate the desired message.
In both instances, a high ratio of skips or IXLLSTM calls per successful MQGET will see increased MQGET cost and may indicate that many messages have to be scanned for a matching selection criteria.

For best performance with message selectors, use ‘select by message ID’ or ‘select by correlation ID’ as these are able to use the index to locate the message rather than scanning the queue.

When using message selectors from client applications, you may see high CPU usage in the channel initiator address space. MQ’s class(4) accounting data can show the utilisation of the adaptor tasks. Ensure that there is at least 1 idle adaptor task even at peak times to ensure there is no waiting for the adaptor task.