MP1B: Monitoring your WebSphere MQ for z/OS system using accounting and statistics data

Colin Paice
March 2010

Document Number MP1BV7Using

This document is for MQ system programmers on z/OS to help mange your systems by showing you how to use the WebSphere MQ statistics and accounting to monitor your queue manager to avoid performance problems with it.

Take Note!
Before using this User's Guide and the product it supports, be sure to read the general information under "Notices".

V7 Edition March 2010. Printed 3/18/2010 7:51 AM

If you wish to send comments to IBM about this document please email them to IDRCF@HURSLEY.IBM.COM

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you. You may continue to use the information that you supply.

Notices

This report is intended to give guidance on the use and interpretation of the statistics and accounting in WebSphere MQ for z/OS Version 7.0.1. The information in this report is not intended as the specification of any programming interfaces that are provided by z/OS or WebSphere MQ.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed "as is". The use of this information, and the implementation of any of the techniques, is the responsibility of the customer, and depends on the customer's ability to evaluate and integrate them into their operational environment.

The following terms, used in this document, are trademarks of the IBM Corporation in the United States or other countries or both:

CICS

IMS

RMF

WebSphere MQ

Z/OS

Summary of Amendments

Date	Change
March 2010	Original document

Introduction

You need to monitor your system and may need to change the configuration or applications to maintain good performance.

When and what should you monitor?

What you monitor and the frequency of monitoring depends on the usage of your system.
The more important performance and availability of your system is important to you, the more closely and frequently the system should be monitored.

Analogy: using a car

Consider a car.
If you only use it to go a few miles to the store once a week, then just an annual service may be ok. You might do no more checks than watching the temperature gauge on the dashboard – and only do something when it goes into the red.
A more cautious person might check the oil, the water, and the tires once a week.
If you do a lot of driving then you check more frequently, and you may take other actions, such as removing heavy items that you do not need to keep in the car, because carrying around unnecessary weight increases the fuel consumption. Additionally you might monitor your fuel consumption (miles per gallon) and if it differs significantly from the norm, you might investigate why.
If you use your car for racing, then you will check the car daily and will do things to give you a competitive edge, such as using different fuel and different tires.
During a race your car may be monitored remotely to make sure it is running at peak performance, and to identify areas for improvement.

Another factor is how important your car is to you. In the example, where the car was only used once a week, if it wasn't available for a day then it is merely inconvenient, but if your racing car is not available when needed then this is very serious.

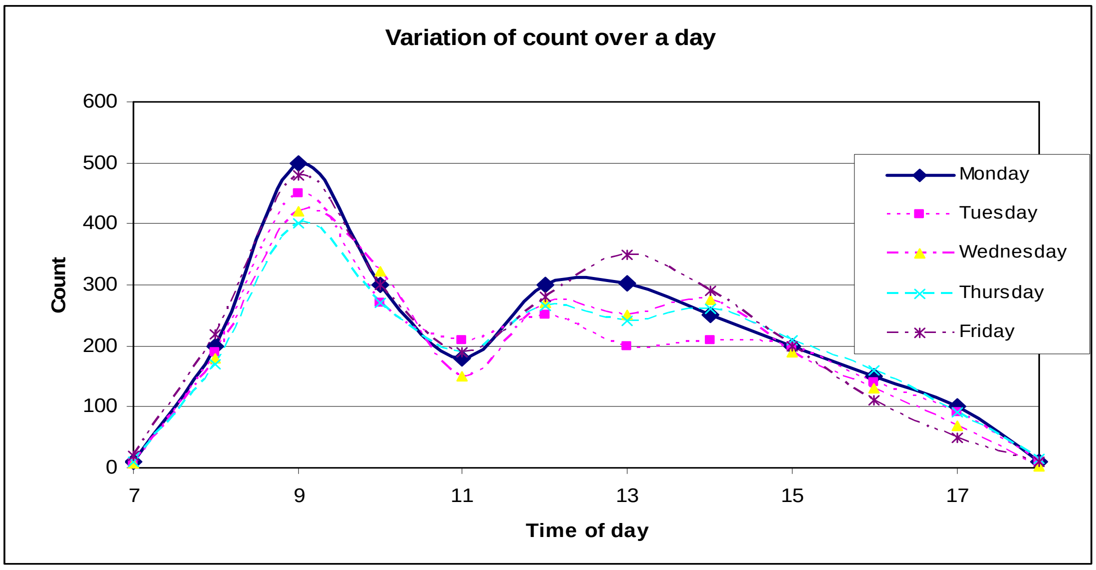
The z/OS queue manager provides statistics on its internal processing. These are produced typically hourly or more frequently and give you a high level overview of the status of the queue manager. You should collect statistics and regularly review them.

Accounting information is provided for applications and channels to show where time and resources are used by the application. You might only use accounting to monitor changes in an application, or for charging users for MQ usage.

As you monitor the statistics over time you will observe peak times and quiet times corresponding to peaks and troughs in your business.

For example in Figure 1 below, on Monday morning between 0800 and 0900 there is a peak in workload, On other weekdays there is also a peak between 0800 and 0900, but the peak is smaller than on Mondays. At the peak time of 0900 the expected range of the count values is between 400 and 500. A value outside of this range at this time would be out-of-line and should be investigated. A value of 400 at 1300, even though it is less than the value at 0900, should also be considered out-of- line, and should be investigated

Figure 1 Variation of count over time



Statistics guidelines

Using the guidelines

The guidelines below are broken down into

- Low MQ use systems
- Medium MQ use systems
- High MQ use systems

You should review the sections and take the appropriate actions.
The more important guidelines are at the top, but a lower item may have a bigger impact than items above it, if it is significantly outside of normal values.

A guideline may have a prefix; these have the following meanings

- **Serious** The guideline can have a major impact on your system's performance or availability
- **Action** You need to do something
- **Warning** The guideline indicates a possible out of line situation.
- **Information** You do not need to take any action.

Some guidelines are easy to monitor, for example 'If the value is greater than 0…. Other guidelines have a non zero value or range which depends on the environment, and it is harder to give values which will apply to all environments. These values, such as "high logging rate" are given as an illustration and should not be taken as absolute numbers, as in different environments, different values may apply. For example the value of "high logging rate" is given as 25MB/second. In some environments the queue manager may easily be able to sustain 50MB/second, whereas in another environment 20MB/second may be the absolute maximum.

You should collect information when your system is running well, so if you experience a problem you can compare data from the bad day with a good day, and identify differences between the days**.** For example with **Figure 1** on page 5, you have a picture of how the value of count changes over time for different days of the week.

If your environment changes, for example by adding new applications, increased workload, or change of system configuration you may need to review the items to see if any have changed the amount of focus needed. You should also update your 'normal' usage system information.

Using the following sections

The next section gives some characteristics of low-use, medium-use, and high-use queue managers. They give examples of what to monitor, with a reference to a section later in this document. You should read the chapter the reference is in, as this provides more information and context. For example there is a reference to Action: Some write-to-log-buffers requests delayed, on page 12. You should read the whole of the section that this reference is in, to understand what log buffers are.

In the later sections there are references to fields like **QJST->qjstciwr.** These are described in the companion document MP1B in this SupportPac.

Characteristics of low use queue managers

1. This typically logs less than 5MB a second
2. This system may process 10s of messages a second
3. You expect the system to run itself; you haven't had to do much to keep it running.
4. You check the system if you start to get application problems which include MQ as a component
5. You do not have critical work that has sub-second response time criteria
6. If there is a peak in workload then it is not an issue if this workload is delayed slightly.

You should review the criteria in the Medium use systems every six months, or if the environment changes, to see if you have a medium-use queue manager.

What to monitor

You should review the following typically weekly or if your environment changes.

- Information: Logging rate is below 5MB a second, on page 9
- Serious: Buffer Pool short on storage, on page 13
- Overall Message rate, on page 16
- Serious: The number of structure full conditions is > 0, on page 17

Characteristics of medium use queue managers

1. This typically logs between 5MB/second and 25MB/second
2. This system may process 100s of messages a second
3. You monitor the system and look for ways to make it more efficient and available
4. You have critical work that has sub second response time criteria
5. If there is a peak in workload then it is not an issue if this workload is delayed slightly
6. You regularly monitor it and change the configuration to remove bottlenecks
7. You aim to have enough capacity so it can handle expected peaks in workload
8. You have multiple systems for availability

What to monitor

You should review the following typically weekly or if your environment changes

- Action: Logging rate is between 5MB and 25MB a second, on page 9
- Serious: Number of log records read from Archive logs is > 0, on page 10
- Action: Some write-to-log-buffers requests delayed, on page 12
- Serious: Buffer Pool short on storage, on page 13
- Warning: Buffer pool filled many times, on page 13
- Warning: Many buffer pool pages read from disk, on page 14
- Serious: The number of structure full conditions is > 0, on page 17
- Overall Message rate, on page 16
- Monitor the Maximum entries and Maximum elements for each structure, on page 17
- Monitor the Coupling facility response time as seen by MQ, on page 17.

Characteristics of high use queue managers

1. You are logging more than 25MB a second across your queue managers
2. You monitor it daily and identify any out of line conditions
3. You have business critical work which is time critical and which cannot tolerate delays
4. You have capacity available to you to handle unexpected workloads
5. You have multiple systems with redundancy for availability.

What to monitor

You should review the following typically weekly or if your environment changes

- Serious: Logging rate is > 25MB , on page 9
- Serious: Number of log records read from Archive logs is > 0, on page 10
- Serious: High proportion of write-to-log-buffer requests delayed, on page 12
- Serious: Buffer pool buffers written immediately to page set, on page 13
- Serious: Buffer Pool short on storage, on page 13
- Action: Number of log buffers paged in is > 0, on page 11
- Action: More than 10 checkpoints an hour, on page 11
- Serious: The number of structure full conditions is > 0, on page 17
- Monitor the Maximum entries and Maximum elements for each structure, on page 17
- Monitor the Coupling facility response time as seen by MQ, on page 17.

Monitoring Logging

When persistent messages are used, information about them is written to log data set. Non persistent messages do not get written to the logs.

You should consider if your messages need to be persistent on not. Inquiry requests are typically non persistent. There are applications which use non persistent messages for their financial requests. If the application has not received a response within a certain time period, then a persistent message is sent saying backout the previous requests if you have seen it. This way the majority of the messages are non persistent.

In MQ V7 there is log compression which compresses data in the buffers and reduces the amount of data logged – but at a higher CPU cost.

Logging rate

The logging rate is determined by the number of control intervals (CIs) written in the SMF record interval. Each CI is 4KB, so the logging rate in MB per interval is
(QJST->qjstciwr * 4 * 1024) /(1024 * 1024) / SMF_interval =
(QJST->qjstciwr / 256) / SMF_interval

Serious: Logging rate is > 25MB a second

You need to ensure your logs are neither being constrained by DASD nor by the I/O subsystem. You may need to monitor this daily or weekly.
Using Striped DASD can improve the rate at which data can be logged, see below.
Using zHPF can significantly improve I/O throughput.
Remote or mirrored DASD provides additional availability, but as the DASD is further away, it can have a big impact on your throughput.

Action: Logging rate is between 5MB and 25MB a second

Log placement is not so critical compared to higher logging rates and if you are using RAID DASD then selecting particular volumes may not be so important.
You should check the I/O configuration, perhaps weekly, and check you are not constrained.

Information: Logging rate is below 5MB a second

You should use appropriate (e.g. not high use) volumes when you define your logs, but review the DASD and I/O subsystem performance perhaps monthly, by talking to the storage management people in your organization.

Comments

You can use RMF, or an equivalent product, to display information about the disk subsystem, including channels to the DASD, the cache, and the DASD volumes themselves.

Using Striped DASD

It is good practice to have striped DASD so you are configured for growth.

Using Striped DASD, where a data set is spread across multiple volumes, can improve the I/O rate as, for example, you do four I/Os writing one page to each of 4 volumes, compared to one I/O writing 4 pages to one volume.
You should speak to your data management team who can set up an SMS DATACLAS for the data sets you want striped.

To see if your data sets are stripped, you can either use the

LISTCAT ENTRIES(++HLQ++.LOGCOPYn.DSnn) ALLOCATION[1] command or use ISPF option 3.4.
If you use ISPF 3.4 to display the dataset, if the volume has a + after it, it means it spans multiple volumes. (The dataset may be spread across multiple volumes, and not striped)
Type I in the prefix area of the data set, and press enter. This displays the *List Catalog Information* panel. Select ALLOCation and press enter, type EXECute and press enter to execute the LISTCAT statement.

In the output of the LISTCAT command, the data component will have the volumes used, and the stripe number such as STRIPE-NUMBER----------3

Using zHPF can significantly improve I/O throughput. See http://www-01.ibm.com/support/docview.wss?uid=swg27015698

Log buffers read

When performing recoverable persistent requests, data is logged, and is written to log buffers. A log buffer may have data from more than one application. The log buffers are written to the active logs when an application issues a commit request. This causes all of the pages up to and including the page containing the commit request to be written to the active log dataset(s). Pages can be written to the active log if the value of *WRTHRSH* pages are filled. Once written to the active log datasets, the buffers are made available again. Most applications only write to log buffers. If an application performs roll back activity, the log has to be read to undo changes made by the application. The records may be in the log buffers, or may have been written out to the active log, or may have been copied from active logs to archive logs.

If you have long running transactions or a very busy system, then the buffers may have been written to the active log and the memory copy overwritten before the transaction ends.

- Records read from the in-memory buffers indicate that a transaction using persistent messages has rolled back. This is OK, but if you have a lot of backout activity you may want to understand why.
- Records read from active logs indicate a long running activity, or a short transaction in a busy system. This value should be zero or close to zero. Increasing OUTBUFF may help.
- The number of records read from archive logs should be zero, because this indicates a very long running unit of work or page set recovery. At shut down, or system restart after an abnormal shutdown the archive logs would be read, which will significantly increase the start or shut down time. If you have restored a page set from a backup and restarted the queue manager, the archive log may be read to recover the page set.
  Note. During normal running the queue manager will shunt log records from the archive into the active logs and so you may not see any reads from archive logs. Shunting occurs every three checkpoints for a unit of work, so you need four logs in the ring to be sure that shunting will work, and the active log is not overwriting the records it needs to copy.

Serious: Number of log records read from Archive logs is > 0

If **QJST->qjstrarh > 0** then you have records being read from archive logs. You should investigate which transactions are causing this and prevent them from being long running. See below on how to do this.

Action: Number of log records read from Active logs is > 0

If **QJST-> qjstract > 0** then you have records being read from active logs. You should investigate which transactions are causing this and understand why they rollbacks occur. See below on how to do this.

[1] For example: "LISTCAT ENTRIES(SCENDATA.MQZZ.LOGCOPY1.DS01) ALLOCATION"

Warning: Number of log records read from the Memory log is > 0

If **QJST-> qjstrbuf > 0** then you have records being read from memory. This may be expected behavior, but if the number of records is large you should investigate this.

Which transactions were backed out?

The accounting class(3) will report which transactions performed roll back activities.

You can use the EXTRACT function of the CSQ1LOGP utility to display the log records for transactions that have been backed out.

Identifying long running transactions

You can use the command
+cpf DIS CONN(*)TYPE(CONN) UOWLOGTI UOWLOGDA
WHERE(UOWSTATE,EQ,ACTIVE)
to display the start time and date of units of work on your queue manager, and so identify long running transactions.

The queue manager displays message CSQJ160I and CSQR026I if it identifies long running transactions, though these messages may not be produced if the queue manager can use log shunt to move the old buffers into the active log.

Number of checkpoints taken per hour

The interval between checkpoints should be about 10-15 minutes, though on a very busy system this may be every 5 minutes. The field **QJST->qjstllcp** gives the number of checkpoints in the SMF interval due to LOGLOAD log records being written to the active logs.

Action: More than 10 checkpoints an hour

This may be caused by logs being too small, or the LOGLOAD value being too small.
A checkpoint causes the buffer pools to be scanned, and any old pages to be moved to the page set. Thus too frequent checkpoints can cause a lot of page set activity in a busy system.
The log data sets may be too small or OUTBUFF may be too small. You can use the +cpf DISPLAY LOG command to display the value of OUTBUFF. If the value is not 4000 then you should change this value to 4000 in the **CSQ6LOGP** macro and reassemble it.

Number of log buffers paged in

If your system is constrained for real storage, you may get a lot of paging. If the log buffers have been paged in this indicates a real storage issue in your system

Action: Number of log buffers paged in is > 0

The field **QJST->qjstbpag >0**. If this value is non zero you should consult your z/OS systems programmer as this indicates a real storage or capacity issue.

Delays due to tape

When rollbacks occur then the log has to be read. If logs are archived to tape then there will be a significant delay while tapes are mounted and read.
In normal use, there should be no tape read activity, so the fields below should all be zero.

Serious: Number of tape requests greater than 0.

If **QJST->qjstwur > 0, QJST->qjstlama > 0, qjstlams > 0 or QJST->qjsttvc > 0**, then there have been tape requests.

- **QJST->qjstwur** is the number of times a read log was delayed.
- **QJST->qjstlama** is the number of tape look aheads attempted
- **QJST->qjstlams** is the number of tape look aheads that were successful
- **QJST-> qjsttvc** is the number of read accesses delayed to tape volume contention.

See Which transactions were backed out? on page 11 to find how to identify long running units of work.

Delays due to insufficient log buffers

In a busy system all of the in-memory log buffers may become full, and so applications are delayed while pages are written to the active log and the buffers freed up.
This can occur

- if the number of pages allocated to the log buffer (OUTBUFF) is too small
- in a lightly loaded system where the I/O performance is very slow,
- or in a busy system where the I/O performance is good – the delays are just due to the high volume of work.

You can use the DISPLAY LOG command to display your current OUTBUFF value. Normally the OUTBUFF value should be the default, maximum value of 4000. If value is not 4000 then you should change this in the **CSQ6LOGP** macro and reassemble it.

Serious: High proportion of write-to-log-buffer requests delayed

If the percentage of delayed request is > 5%,
100* QJST->qjstwtb/ (QJST->qjstwrw+QJST->qjstwrnw+QJST->qjstwrf) > 5%,
then you should check your I/O performance and consider increasing OUTBUFF up to the maximum value of 4000 KB

If OUTBUFF is at the maximum, you should review your expected messaging growth and consider creating a new queue manager if you expect more message traffic.

Action: Some write-to-log-buffers requests delayed

If you have some requests delayed (**QJST->qjstwtb**) then you should check your I/O performance and consider increasing OUTBUFF up to the maximum value of 4000 KB. If **QJST->qjstwtb** is often > 0 and the percentage of delayed requests is less than 5% you should review your expected messaging growth and consider creating a new queue manager if you expect more message traffic.

Buffer pool activity for local queues

Buffer pool usage

Short lived messages

Short lived messages usually exist for a few seconds, or occasionally minutes. For performance reasons you should keep these in the buffer pool and avoid having them written to disk. If the buffer pool is too small you will get messages spilled to disk, and have disk I/O when getting them.
When a buffer pool gets to 85% full of modified pages an internal task is started to write the older pages out to the page set. So to prevent the buffer pool getting to 85% full, make sure the buffer pool is large enough

Long lived messages

Usually long lived messages will be written to the page set, either as a result of the buffer pool filling up, or when a checkpoint occurs, and the page is older than 2 checkpoints.
Having a small buffer pool will mean that the buffer pool will reach 85% more frequently, and so messages are written to the page set more frequently.
Getting long lived messages usually requires I/O to the page set, though there is a read ahead capability which can read messages from the page set before they are needed.

Serious: Buffer pool buffers written immediately to page set

The value of the number of pages written immediately to disk, **QPST->qpstimw,** should be close to zero (10's) per hour.

Data is written to one or more pages in the buffer pool.

- If the buffer pool is 95% full or more, the page is written immediately to the page set.
- If the page was in use during a checkpoint activity, then when the page is unlocked the page may be written synchronously to the page set.

If the buffer pool is greater than 95% full you should make the buffer pool larger, reallocate page sets to different buffer pools, or move queues to different page sets.

Serious: Buffer Pool short on storage

The value of the number of times Short on Storage was detected, **QPST->qpstsos,** should be zero.

If there are no free buffers that can be used, (so the buffer pool is Short Of Storage), then a change to a page will be written synchronously to the page set. The application will be delayed while this write to the page set occurs. At the same time the queue manager will be writing pages to disk to create free space, so there will be a lot of contention resulting in a long delay.

You should make the buffer pool larger, reallocate page sets to different buffer pools, or move queues to different page sets, to minimise interaction between applications.

Warning: Buffer pool filled many times

The value of the number of times the internal task was started at 85% full, **QPST->qpstdwt,** should be monitored
When a buffer pool gets to 85% full it starts a task to write changed pages out to the page set. This behavior is typical of an application which writes messages to a queue for deferred processing – such as overnight.

If this occurs for a buffer pool where the messages are expected to be short lived (lasting a few seconds) the buffer pool could be too small,

- you have a mix of messages, perhaps from different applications, some long lived and some short lived,
- you have many short lived messages, perhaps because the applications or channels are not processing the messages fast enough.

You should make the buffer pool larger, or reallocate queues with long lived messages to be on a page set in a 'long lived messages' buffer pool.

Warning: Many buffer pool pages read from disk

The value of the number of pages read in from disk, **QPST->qpstrio,** should be zero for buffer pools for short lived messages.

Short lived messages should reside in the buffer pool, and should not be read from disk except after starting the queue manager. If a buffer pool with short lived messages had a lot of reads from disk, the buffer pool may be too small, or you are processing large messages.

Long lived messages usually get moved out to the page set, and read in when needed. In this case having pages read from disk is expected behavior.

If a queue is not indexed then searching it for a message with a particular msgid or correlid can result in many pages being processed – which can cause a lot of read from disk if the buffer pool is too small.

Storage management

Serious: Critical short of storage within in the queue manager

The value of the count of times the queue manager detected a critical shortage of storage, **QSST->qsstdata.qsstcrit**, should be 0.

- If an obtain storage request within the queue manager fails, then the queue manager compresses its storage pools. The storage request is then reissued. If the compression fails to release enough space then this indicates a critical shortage of storage within the queue manager.
- The easiest change is to reduce the size of the buffer pools to free up more space. You should review if you need an additional queue manager. Check your queue manager is not constrained by the REGION value on the EXEC PGM=CSQYASCP, or the installation IEFUSI exit.

Action: The number of storage contractions is greater than zero.

The value of the count of times the queue manager contracted its internal storage, **QSST->qsstdata.qsstcont,** should be 0.

If a storage request within the queue manager fails, then the queue manager compresses its storage pools.
If the number of contraction is above 10 requests an hour, then this indicates you should monitor this. Reducing the size of your buffer pools is the easiest way to free up more space. You should also review if you need an additional queue manager.

Information about Messages request rates

Overall Message rate

You should monitor the rate of messages put and got and how these numbers change over time. You might monitor total messages per day, per week, or peak rate over a week.

This information is useful to indicate changes in your messaging environment, such as new applications, or increasing workload. You can use the information to understand your growth and help you understand your future growth.

The values are

QMST -> QMSTGET/ SMF_interval
QMST -> QMSTPUT/ SMF_interval
QMST -> QMSTPUT1/ SMF_interval

An increase in put or put1 requests usually indicates an increase in workload. An increase in gets can indicate an increase in workload, but also that the environment has changed, for example some applications may have more unsuccessful gets.

Coupling Facility Information

The coupling facility is used for shared queue messages.

The MVS command D XCF,STR,STRNAME=… displays information about the structure, including the maximum number of entries and elements.

Note the maximum number of entries and elements can change if the CF AUTOALTER facility is used.

Serious: The number of structure full conditions is > 0

The field **QEST->qeststuc[].qestsful** gives the number of structure full conditions detected. Review the size of your structure, and consider moving queues to a different structure.

The structure full condition may have occurred as a result of an application or channel not being operational, and so messages accumulated. In this case a structure full condition may be expected.

Monitor the Maximum entries and Maximum elements for each structure

You should monitor the maximum number of entries **QEST->qeststuc[].qestmnus** used and the maximum number of elements **QEST->qeststuc[].qestmlus** used and see if there is an upward trend, and compare with the structure maximum – see above.

Monitor the Coupling facility response time as seen by MQ

You can use the RMF CFACT command to display the rates and response times for requests to each structure. It reports for each Coupling Facility if the structure is duplexed.

You can also use fields in the QEST to report the average response times for CF requests as seen by MQ.

QEST. qeststuc[].qestsstc/ QEST. qeststuc[].qestcsec gives you the average response time for single requests. This value needs to be converted to microseconds from a STCK value,
QEST. qeststuc[].qestmstc/ QEST. qeststuc[].qestcmec gives you the average response time for multiple requests per cf requests. This value needs to be converted to microseconds from a STCK value.
For example, for an application structure the value of of qestmstc was 0000000F 09E4E080 and the number of requests, qestcsec was 927598. The time in microseconds is 0000000F 09E4E080/4096 = 0000000F 09E4E. The average time is this value, converted to decimal, and divided by 927598 is 17 microseconds.

Planning for application performance

Why read this section?

This section tells you what data to collect to help you manage application performance so you are prepared if you have a performance problem.

MQ on z/OS provides information in the Accounting class(3) SMF records on MQ verbs, resources and times spent in the MQ verbs.

Using Accounting class(3)

The amount of time you spend working with the accounting class(3) records will depend on the message volumes and impact to your business.
Changing the application to save 5 milliseconds of CPU in MQ may be justified if you are doing 1000 messages a second, but not if you are only doing 1 message a second.

What you should monitor

You should have processes in place to monitor your system, so if you think you have a problem, you have a 'normal' baseline to compare with.

Typical data to monitor over a three month period includes

For key transactions, or top usage transactions
- Peak number of transactions per hour,
- Total CPU used from MQ perspective,
- Elapsed time within MQ

For top activity queues
- Peak message rate per hour
- Total CPU used by applications from an MQ perspective
- Elapsed time within MQ from applications
- Volume of data passing through queue (MB/Day)
- Typical time on queue between put and get of a message – for queues with time critical data.

To find your high use queues and applications see What are my top used queues and applications? on page 32.

From this data you should be able to determine usage over time, and take action before problems occur.

For a peak day you should collect more detailed information, including information from other products and the operating system, so you can compare good days and bad days. You may have good performance even though some parts are '*running hot'.* If one day you get bad performance you might focus on a hot area, even though this is normal, and so miss the real problem area.

Z/OS data

Use RMF, or similar products, to collect and report data at the operating system level. Data you may want to collect includes
- Processor usage data – what percent utilised was the LPAR
- CF processor usage
- CF path usage
- Structure information, rate and average duration of synchronous, and asynchronous requests
- I/O subsystem usage

- DASD usage
- Storage utilisation
- WLM and transaction data.

Data from MQ

You need to collect data to see if MQ is the problem or not. Data to collect from a peak hour includes

- Peak number of transactions per hour for important transactions
- Total CPU used from MQ perspective for important transactions
- Elapsed time within MQ for important transactions
- Peak message rate per hour
- Volume of data passing through queue (MB/day)
- Typical time on queue between put and get of a message – for queues with time critical data

To help with problem determination, you should be able to answer

- Where are any increases in delays? Is it due to logging, reading messages from page sets, internal locking, or CF delays
- If there is an increase in CPU in MQ, where is this – which MQ verb.
- Is there more MQ application activity than on a good day.

Deployment design guidelines

This section gives you some application best practices from a performance perspective. You can use information from the class 3 accounting records to check your applications.

The accounting class 3 records provide a lot of information about the MQ requests used by an application. This data is provided in 3 sections

- **WTID** Task information, including such information as channel name, or CICS transaction name
- **WTAS** Task related information, such as number of commits
- **WQ** a section for each queue used

Do as much work as possible in a transaction instance

There is a CPU overhead when an application connects to MQ. You can save CPU if a transaction processes many messages while connected to MQ rather than connect to MQ for every message processed.

For example, there is a CICS transaction which processes messages. It gets a message from a queue does some processing and returns.
At a low message rate having the queue with trigger-every may be an efficient way of initiating the transaction.
As the message rate increases, and the transaction is running often so there is often a transaction instance running, it may be more efficient to have it triggered first, and the transaction instance loops around doing a get-with-wait for a message, processing the message, and committing before looping for the next message. If no messages arrive in a short period the transaction ends.
This would eliminate many trigger messages, CICS transaction initiating and terminating costs, (implicit) MQ connect processing, and MQ open and close of the queue, and security checks.

At a high message rate you may want to have many instances of the transaction running concurrently. Your transaction could monitor the queue depth, and if it exceeds a value, and the number of open handles is below a value, then start a new transaction instance. If the application instance received a no-message-returned condition, and there are many other transaction instances with the queue open, then the application instance should end.

Fields to review

For the application queue **WQ-> getn** is the number of get request for a queue in the interval. If this has a value of 1, then there may be one message per transaction.
For the trigger queue review how many trigger messages were produced.

For an IMS MPP you will typically see one or two queues used, and few messages processed. For each IMS MPP transaction there will be MQ Connect /Disconnect costs. See SupportPac MP16 for information on how to reduce these costs.

Reduce the number of gets which do not retrieve a message

If you have many gets, but only a small number of these successfully retrieve a message, then this can be an expensive waste of CPU.

WQ-> GETN is the number of get requests for a queue in the interval. **WQ -> validget** is the number of gets which returned a message. If only a small percentage of the requests are successful, you may wish to review the applications

There are several application scenarios which can cause this:

Polling a queue

Rather than have the application have a get with wait, the application does a get, which may not retrieve a message, and then waits for a short period before reissuing the get request. This technique in inefficient because it uses CPU for these unsuccessful requests, and when a message does arrive, it has to wait until the application reissues the get request, before it can get the message.

Getting a message with a zero length buffer supplied, to get the size of the message

The application does a get with a zero length buffer, to determine the size of the message, then gets a buffer of the appropriate size and reissues the get request.
With multiple concurrent getting applications, the message which is retrieved may not be the same one as was originally retrieved. If your expected message size is relatively small (say under 20KB) using a buffer big enough for most message will mean the messages can be retrieved in one MQ get request. If you are getting many messages, you may be able to reuse the same buffer.

There are multiple applications in a get-with-wait on a queue

If there are multiple applications getting messages from a queue, then this is good from an availability perspective, but can be inefficient. When a message arrives on a queue, those applications which are waiting for such a message are woken up and reissue the get request. Only one application will get the message. If this is a shared queue being used on multiple queue managers, there will be activity to signal each queue manager that there is a message to process.

Where will my work run?

Local queues

Which application instance gets a message depends on many factors including the priority of the application. You may wish to monitor which applications are getting messages, and use z/OS Workload Manager (WLM) facilities to change application priorities. If put-to-waiting-getter[2] is used, then just one application instance will get the message. If not, then all applications are posted, and the first to get the message retrieves it, the other application instances will be unsuccessful.

Shared queues

If put-to-waiting-getter is used, then one application instance on the same queue manager will get notified. If not, then all applications are posted, and the first to get the message retrieves it.

Applications on the same queue manager as the putting application may be notified before other queue managers

Queue managers on the same LPAR as the putting application may get notification before queue managers on a different LPAR, because it is a cross memory notification rather than a cross system notification.

The application instance which gets a message depends on many factors including
- the priority of the application,

[2]Put-to-waiting-getter: When a non persistent message is put out of syncpoint, put-to-waiting-getter may be used. The put request puts the message directly to a waiting application's buffer on the same queue manager without the message being put onto the queue. If put-to-waiting-getter is not used, for example at that moment there is no application in a out-of-syncpoint-get-wait for that message, then the message will be put to the queue, which costs more than a put to waiting getter

- which queue manager the application is on,
- which LPAR the queue managers are on from a connectivity perspective
- the links between LPARs,
- how busy the LPARs are

It is not a simple task to modify which application instances get the messages. You would need to talk to your z/OS systems programmer.

Reducing Shared Queue costs

Using an application to keep a shared queue open

Reduce first-open, last-closed effects. The first open of a shared queue on a queue manager requires CF access to inquire and update status. The last close of a shared queue on a queue manager requires CF access to update status. Other open and close requests do not usually require CF accesses.
For a high use shared queue which is frequently opened and closed, having an application with the queue open for input and output, doing no other work, can avoid first-open and last-closed effects. A put1 request counts as an open, a put, and a close.

However if you have this application with the queue open on each queue manager this may give you some performance benefit, but you may not be able to alter the queue, because it will be in use continuously.

Fields to review

The field wq->***openn*** is the number of times the queue was opened in the interval. In V701 there is a field wq->***opencf0*** which is the number of times an open request did not require any CF accesses.
The field wq->***closen*** is the number of times the queue was closed in the interval. In V701 there is a field ***closecf0*** which is the number of times a close request did not require any CF accesses.

If you sum wq->**openn, wq->opencf0, wq->closen** and wq->**closecf0** for a queue, across all applications over the interval, you will be able to see if you have a large number of open and close requests. If you have a large percentage these requests with CF activity **100* (openn-opencf0)/open,** and **100 * (closen – closecf0) /closen**, consider having an application which keeps the queue open, but with no other activity.

Getting from a high use shared reply-to-queue on one queue manager

For very high use shared reply-to queues, consider having a reply-to-queue per CICS region or batch job rather than a common queue used by all applications. If a queue is shared between queue managers, there is more potential serialization misses, where the CF request has to be reissued. After a queue manager has updated information about the queue, any other queue manager accessing the structure will experience a serialization miss, and need to reissue the CF request. If only one queue manager is updating queue information it does not experience serialization misses.
A typical scenario will be server applications putting to the reply queue from multiple queue managers, the original application will get from this queue. If these original applications are spread across multiple queue managers you will have a higher chance of serialization miss.
Having a reply-to-queue with the CICS region name as part of the reply to queue means you can benefit from shared queue, and minimize the interaction between different application queue managers.

Fields to review

In V701 there are fields which give the number of calls to a CF routine, and the total number of retries. See Detailed information about CF request on page 27.

I think I have a performance problem

Why read this section?

If you think you have a resource or performance problem with an application which uses MQ, this chapter will give you some help to identify the resources used by MQ, and reasons for delays.

You may be investigating
- Application CPU used by MQ is too high
- Response time of MQ requests is too long
- It take a long time to process messages

The amount of time you spent working on the problem depends on the message volumes and impact to your business. It is worth repeating: changing the application to save 5 milliseconds of CPU in MQ may be justified if you are doing 1000 messages a second, but not if you are only doing 1 message a second.

Ideally you have statistics and accounting information from a good day so you can compare the data with a problem day with to see any differences.

Application CPU used by MQ is too high

An increase in CPU used could be cause by
- More requests are being issued
- The same number of requests is being issued, but they are using more CPU

More requests are being issued

More MQPUT and MQPUT1 requests are being issued

From the QMST statistics records the number of MQPUTS and MQPUT1s is larger than usual.
This indicates more messages are being processed.
This could be due to
1. An increase in application messages
2. More messages are flowing through the system to and from other queue managers
3. More trigger messages are being generated
4. More events, COA, COD or similar messages are being generated
5. Channels were not sending messages, or applications were not processing messages, and there is now a peak in workload while the system recovers.

You may wish to reset your baseline if this increase is not temporary.

More MQGET requests are being issued

From the QMST statistics records more MQGETs are being issued. This does not necessarily mean that there is an overall increase in throughput. It may be that the environment has changed.

An increase in the number of MQGETs could be due to the following reasons
1. There is an increase in work – there will be a matching increase in MQPUT and MQPUT1 above
2. Applications are not successfully getting a message. There may be many applications waiting for a message, only one will succeed. You may have more applications getting from the queue
3. When put-to-waiting-getter occurs, just the recipient application is notified. If put-to-waiting-getter does not occur, then all applications waiting for the message get notified, one of which will succeed.

4. In a shared queue environment messages are arriving on a different queue manager than usual, and so now applications on a different queue manager are succeeding in getting the message, and applications on this queue manager are not successful.
5. Cooperative browse (an application option which is used by the CICS bridge and MDBs is being used, and now messages are not being processed as fast, so the monitor task is browsing messages on the queue more often.

Local queues - A similar number of requests are being issued, but they are using more CPU

ReplyTo queues are not indexed

In a request-reply type of application, a get of the reply message by CorrelId or MsgId can use more CPU time as the queue depth increases.
If a queue is not indexed, and a get is issued specifying message id or CorrelId then the queue is scanned sequentially for the message. As the queue depth increases, there are more potential messages to scan. More messages scanned lead to an increase in CPU cost.

After the request has been put, the application does a get for the reply. Typically the reply message is not yet available and if the queue is not indexed, the whole queue is scanned. When the reply message arrives, the queue is scanned again to retrieve the message.

If the queue is indexed then the id can be looked up in an index, and retrieved much faster.

- The field ***WQ->indxtype*** tells you how the queue is indexed, if at all.
- The fields ***WQ->getbrws*** and ***WQ->gets*** are the count of browse specific and destructive get of a specific message. If these values are greater than 0 you should consider indexing the queue.
- The field ***WQ->getsmsg*** is the number of messages skipped in order to find the specific message. This value should typically be less than 10 messages skipped per message returned. If this value is larger than 10 check the index type of the queue matches how the application retrieves messages. The more messages skipped, the higher the cpu cost.
- The field ***WQ->getepage*** is the number of empty pages scanned in order to find the specific message. This value should typically be less than 2 pages skipped per message returned. If this value is larger than 2 check the index type of the queue matches how the application retrieves messages. The more empty pages processed, the higher the cost

Many server type applications just get the <u>first</u> message from a queue and process it, and use MQPUT or MQPUT1 to put the message to the ReplyTo queue. In this case the cost of getting a message is almost independent of the number of messages on the queue, and there is no benefit of indexing the queue if the requests are to get the first message on the queue. If the server browses the first message, then either gets the message or starts another task to get the specified message, the application is not a simple get-first-message, and so the queue should be indexed. See above for the fields to monitor.

- The field ***WQ->qetbrwa*** is the browse of the first or next message from the queue. This is used by the CICS bridge, and WAS to locate a message, and a transaction is started and then the message locator is passed to the transaction.
- The field **WQ->** ***geta*** is the count of destructive get of the first (next) message from the queue.

Expired messages are being processed

- The field ***WQ-> getexmsg*** contains the number of messages processed which had expired. Processing expired message may generate additional messages to notify the originator of the expired message. The more expired messaged processed, the higher the cost.

You should investigate any increase in the number of expired messages, as this can indicate a problem with applications or channels not processing messages, or the value for the expiry interval which is too small.

If the number of expired messages is to be expected, you could use Queue manager expiry interval (EXPRYINT) to drive the queue-managers expiry processor, or increase the frequency with which it runs

Page set I/O adds to the cost and slows down get requests

For long lived messages, they will typically be on page sets rather than in a buffer pool. Short lived messages should be in the buffer pool and not be read from the page set. The field ***WQ-> getpsn*** contains how many messages were read from the page set. If the queue is used for short lived messages this field should be 0, and if it is often greater than 0 may indicate a set up problem with the buffer pools, or other queues in the buffer pool. If the queue is not indexed, then many messages may have to be scanned to locate the required message, and each of these messages can be on the page set, and so requiring the whole queue to be read from the page set, rather than just the required pages.

When the queue manager is shut down, any messages in the buffer pool will be moved to a page set. When the queue manager is restarted and these messages processed, they will be read from disk, and so you will get a value of ***WQ -> getpsn*** > 0 when these messages are got.

The page set number is in the field WQ->***nps***, and the buffer pool number in WQ->***nbuffpool****.* These values are set when messages have been put or got from the queue. So do not use these values if WQ->getn, WQ->putn, and WQ->put1n are zero.

A buffer pool filling up can be caused by

- Bigger messages – in this case making the buffer pool bigger may help.
- The applications are not able to keep up with the arrival rate. Consider having more application instances.
- A sudden increase in the number of messages in the buffer pool. This may be due to a getting application or channel not processing messages, or not processing messages fast enough. Check to see if you had a problem in with your channels, or applications. You might need to have more applications processing the messages to keep the queue depth small.
- You have a mix of long lived and short lived messages in the buffer pool, and the long lived messages are impacting short lived messages. Split the long lived and short lived messages to different page sets and buffer pools.

Put to waiting getter not being used

When a non persistent message is put out of syncpoint, put-to-waiting-getter may be used. The put request puts the message directly to a waiting application's buffer on the same queue manager without the message being put onto the queue. If put-to-waiting-getter is not used, for example at that moment there is no application in a out-of-syncpoint-get-wait for that message, then the message will be put to the queue, which costs more than a put to waiting getter.

The field ***WQ - > PUTPWG*** tells you how many of the puts were to a waiting getter, and (***WQ-putn + WQ ->put1n) – WQ- >PUTWSG*** tells you how many were not put to a waiting getter.

Long running units of work

Long running units of work, where messages have been put and got within syncpoint,but the commit has not happened, can cause an increase in CPU. This is because application try to get the messages, but find they are in use, so the next message is tried, Instead of getting the first message multiple messages are tried. This increases the CPU cost of the request.

Shared queues - A similar number of requests are being issued, but they are using more CPU

There are several areas which can cause an increase in application CPU

- Messages are being processed on different queue managers
- Open and closes requests are accessing the CF
- The way requests to the CF are issued has changed

Put to waiting getter not being used

See above under local queue. With shared queue you will also get the cost of notifying the other queue managers

Messages are being processed on different queue managers

When applications are waiting for a message and a suitable message is put, those applications waiting for the message are notified that a message has arrived.

Applications on the same queue manager where the message was put may be notified before applications on other queue managers. Only one application in the QSG will get the message; the other applications will fail to get the message.

At low message rates an application instance running on the same queue manager as the putting applications may be able to process the messages with no delays. As the message rates increase then application instances on other queue managers may get and process the messages. This behavior is typical of IMS messages using OTMA.

How can I tell?
You can see which queue managers are processing messages by using the **reset qstats** command.   This will show you how many messages have been put and got on each queue manager.

When a message is put to a queue, there could be additional processing depending on the queue manager name specified in the MQMD.

- If the queue manager name is blank, the message is put to the local queue manager.
- If the queue manager name is the name of the local queue manager, the message is put to the local queue manager
- If the queue manager name is not blank and not the local queue manager name, the message has to be routed to the destination queue manager. The message can be routed via the mover, or if the queue is a shared queue, by the Intra-Group Queuing (IGQ)

So if the application is putting a message to a queue and specifying the remote queue manager name there will be additional costs if the message is targeted for a different queue manager name. If the environment has changed, so instead of messages coming from the same queue manager as the server, the messages are now coming from a different queue manager in the QSG, there will be additional costs during the put of the reply.

What can I do about it?

1. Check you have SQQMNAME(IGNORE) specified.
2. Check you have a queue manager alias XmitQ specified. So if you application is running on queue manager MQPA and the reply message is being put on queue manager MQPB then you need an XMITQ defined on MQPB called MQPA.   This can be a shared queue. This queue needs to exist, but may not be used.

Open and close requests accessing the CF

You can get first-open and last-closed effects and serialization misses which cause additional CF accesses. These additional CF access use CPU.
If the queue manager’s use count of a queue goes from 0 to 1, or from 1 to 0, (ie the first open or last close of the queue on the queue manager) the queue manager has to update the CF with the change of status of the queue.

- If the queue was already open on the queue manager then will not be any first-open effects. The number of times an open required no CF access is in ***WQ->opencf0***. The number of times CF access was required is **WQ->openn – WQ-> opencf0**.
- If the queue is open by multiple applications on the queue manager, and a close request is made, there is no last-closed effect. The number of times a close required no CF access is in ***WQ -> closecf0***, so the number of close requests which required CF access is **WQ -> closen – WQ -> closecf0**.

The type of CF requests issued or duration of the requests has changed

Requests to the CF can be

- synchronous – where the requests looks like a single long instruction to the CF
- asynchronous where a request is made to the CF and the task then suspends. When the CF request has finished, it notifies the operating system, which resumes the task

Usually synchronous request to the CF are the fastest way of accessing data in the CF. As the CF gets busier the duration, and hence the CPU cost, of the request increases until the operating system decides that it is overall more efficient to use an asynchronous request.
If the request is asynchronous, the task is suspended for the duration of the request – and the CPU time required for the suspend/resume may be more than the cost of a synchronous CF access.

For example, comparing similar workload on a lightly loaded, and a busier CF:

	Lightly loaded CF	Busier CF
CF utilization	10%	29%
Average Synchronous response time for the structure from RMF	9 uS	12 uS
Average CPU time used by a MQGET from this queue	44 uS	71 uS
Of this, the amount of time spent in synchronous CF calls	26 uS	42 uS

Because the CF was busier, the synchronous requests took longer, and therefore the CPU used when the get request issued CF requests has increased. This may appear a large increase but it is small when considering the cost of the business transaction as a whole which may be measured in many thousands of microseconds.

The operating system decides on making the request synchronous or asynchronous depending on many factors. These include

- Processor utilization
- Type of CF – including duplexed, non duplexed.
- Links to the CF
- CF utilization
- How busy the paths are to the CF
- Expected cost of a synchronous CF request compared to an asynchronous request

How do I tell?
You can get information about the response time of CF requests from RMF or similar products. RMF reports the synchronous and asynchronous rate and their response times on for each structure.

Detailed information about CF request

In MQ V701 there are fields in the WQ & WTAS blocks which give information about the detailed CF calls, how long they took, and if the requests were synchronous or asynchronous.

There are different types of request to the coupling facility, such as

- NEW, - to create a new message,,

- Delete – to remove a message
- Signal other queue managers

Within the Queue block (**wq**) and the task block (**wtas**) are an array of structures containing information about these different requests.
The structure has the following fields

- CFCount – the number of times in this routine
- CFSyncN – the number of synchronous requests
- CFSyncET – the total elapsed time in microseconds for this request
- CFAsyncN – the number of asynchronous requests
- CFAsyncET – the total elapsed time in microseconds for this request.

The logic is typically as follow

```
Module entry
Increment CFCOUNT
Do until successful
 Issue CF request
 If synchronous then update synchronous fields
 Else update the asynchronous fields
End
Module exit
```

The CF request may need to be reissued because of a 'serialisation-miss' due to other queue managers using the structure.

Below are some numbers to illustrate this

	Value	Row
Number of puts	14717	1
Average elapsed time of puts is uS	49	2
Average CPU time used for puts in uS	46	3
CF requests		4
New – Synchronous count	14685	5
New – average synchronous elapsed time	16	6
New - asynchronous count	32	7
New – average asynchronous elapsed time	51	8

- From rows 1,5 and 7, the total number of CF requests 14685 + 32 = 14717 is the same as the number of puts, so there were no serialisation misses, and each CF request was successful
- From lines 5 and 7,The majority of the CF requests were synchronous (14685 :32)
- From lines 3 and 6, of the 46 microseconds of CPU, 16 microsecond were in the CF request

What can I do about it?
The operating system determines whether to use synchronous or asynchronous requests, depending on many factors. The MQ workload can be identical on different days, but the request type may change. A CF may be used by CICS, DB2 and MQ, and a change in the usage of the CF by CICS or DB2 may impact the usage from MQ.

You should discuss this with your z/OS systems programmer.

Response time of MQ requests is too long

In an ideal environment, the elapsed time of an MQ request processing a non persistent message is the same as the amount of CPU used. MQ requests can take longer for the following reasons

Shortage of CPU

If there is a shortage of CPU in the LPAR then the application can be delayed for CPU. Check that the application is not constrained by CPU by using products like RMF.

The accounting data reports the total amount of CPU used by the MQ verb, such as ***WQ-> openct.*** To get the average CPU time used, divide the total cpu time by the number of requests, such as ***WQ-> openct/WQ-> openn*.**
You may see an increase in response time of the MQ calls, but the suspend time and CPU used within MQ has not changed significantly.

Queue not indexed

If the application is getting from a queue, and the queue is not indexed, then it will take longer to scan deep queues. See Queue not indexed , on page29.

Internal latching

To maintain integrity with multiple concurrent requests, the queue manager may serialize on a resource. Typically this interrupts processing for a few microseconds, though if the system is constrained, this duration can be longer. If the application is holding this latch and has to perform disk access or other long requests, then the latch will be held across this activity.
The amount of suspend time is reported at the queue level, but the information about which latch classes are used is recorded in the task block.
The data is in an array ***WTAS->WTASLWET*** of durations in STCK format. The number of times a latch of a given class was held is in the array ***WTAS->WTASLWN****.*

The fields ***WTAS-> getsuset*** and ***WTAS-> getsusn*** are the duration and count of the suspensions of MQ get request. This includes delays for latching, log I/O and page set I/O, so the delay due to latching can be calculated from these figures. There are fields ***WTAS-> setsuset, setsusn, putsuset, putsusm, put1suset, put1susn, getsuset****,* and ***getsusn***. In MQ V701 there are also ***WTAS-> opensuet, opensun,closesuet****,* an*d* ***closesun***. The elapsed times are in STCK format.

Common latch classes include

- Latch class 11 (e.g. DMCSEGAL) is a often held across an API request to a queue to ensure serialization
- Latch class 15 (e.g. CMXL1) is held across an open and close of a shared queue where a CF access is required. This is the first-open last-closed effect. You can reduce this by having an application keep the queue open to avoid the first- open, last-closed effects.
- Latch class 16 (e.g. BMXL2) is used for serializing access to a page set
- Latch class 19 (e.g. BMXL3) is used for serializing access to a buffer
- Latch class 21(e.g. RLMLWRT) is used when updating log buffers
- Latch class 23 (e.g.RLMBSDS) is used when serializing access to the BSDS
- Latch class 24 (e.g. LMXL1) is used when serializing access to a chain of locks, and serializing access to a queue by get-waiters
- Latch class 25 (e.g. LMXL2) is used when serializing access to a particular lock, and when writing Accounting records to SMF
- Latch class 30 (e.g. ASMSAGT) is used when task switching, and serialization to security resources
- Latch class 31(e.g. DPSLTCH) is used scanning a queue for messages especially when the queue is not indexed, and serialising access to security resources associated with a userid. Also used when allocating/freeing storage and contraction.
- Latch class 32 (e.g. SMFPHB) is used to serialise requests when allocating/freeing storage

Logging of persistent data

For out of syncpoint puts, out of syncpoint gets, or commits, the application has to wait until data has been written to the I/O subsystem. This delay is usually of the order of milliseconds. It can take longer if the DASD is mirrored to a remote site.

Data is written to log buffers using an internal log write request. For a syncpoint requests an internal log force request is issued. This suspends the application until the data has been written out to disk. It may be that the data has already been written to disk, and so the log force request does not delay the application. The fields are ***WTAS-> putjwet, putjwn, put1jwet , put1jwn, getjwet,getwjn, setjwet*** and ***setjwn*** for the elapsed time and count of writing to the log buffers, and ***WTAS-> putjcet, putjcn, put1jcet , put1jcn, getjcet,getcjn, setjcet*** and ***setjcn*** for the time and count of waiting for the log data to be written (forced) to the log data sets. These times are in STCK format

Page set access

Accessing messages on local queues when they are not in the buffer pool means that a page set access is required. This can take milliseconds per 4KB page and therefore can take longer for a big message spread across multiple pages. If the data is not in the DASD cache it can take additional time to read the physical disks.
The fields ***WQ-> getpset*** and ***WQ-> getpsn*** is the elapsed time in STCK format and count of the time the application was suspended when getting a message from a page set. During normal operation these fields should be zero for short lived messages. When long lived messages are got, or there are messages on the queue when the queue manager is restarted, then these values are likely to be non zero. You can estimate the average time to get from the page set by converting *getpset* to microsecond and calculating WQ-> ***getpset/ WQ-> getpsn***. The elapsed time is in STCK format.

Coupling facility access

It can take time to access the CF. If the CF is constrained or not local to the LPAR then it can take milliseconds for each CF access, and it may take several CF accesses to retrieve one message.
Usually when all of the requests to the CF are synchronous then this is the fastest way of accessing data in the CF. If the request is asynchronous, the task is suspended for the duration of the request – and the CPU time required for the suspend resume may be more than a synchronous CF access. The operating system decides on making the request synchronous or asynchronous depending on many factors. These include

- Processor utilisation
- Type of CF – including duplexed, non duplexed.
- CF utilisation
- How busy the paths are to the CF
- Expected cost of a synchronous CF request compared to an asynchronous request

It takes a long time to process messages

When messages are got, information about the time on queue is stored in the **WQ** block. This is the time between the message being put and the message being got.

If the put is in syncpoint, then the message cannot be got until a commit has occurred, so the minimum times may be milliseconds or longer.
The fields are

- ***WQ->Maxlatnt*** the maximum time on queue time
- ***WQ->Minlatnt*** the minimum time on queue time
- ***WQ->Totlatnt*** the total time on queue for all messages got, in STCK format. This value divided by the number of valid destructive gets (wq->***validget***) gives you the average time on queue.

Note for large number of long duration messages, this field may overflow.

If the messages were put to waiting getter these fields are not updated. The number of messages put to waiting getter is wq->***putpwg + wq->put1pwg***.

You can use the reset qstats command to monitor the depth and message rate of a queue. For a shared queue, messages can be processed on other queue managers to where the messages were put.

What are my top used queues and applications?

Why read this section?

This section will give you information on using the Accounting class(3) records to identify the top used queues and transactions.

What are my top queues and applications?

This question can have several meanings see

- Which queues have most messages pass through them? on page 32
- Which queues have the most data pass through them? on page 32
- Which queues are business critical – even though they do not process the most messages? on page 32
- Which applications process the most messages – they may use many queues? on page 32
- Which applications process most messages through a single queue? on page 32
- What applications use business critical queues? on page 33
- What are the high use channels by number of messages? on page 33
- What are the high use channels by amount of data? on page 33
- What are the channels to my critical destination? on page 33

What is in the accounting class(3) data?

Which queues have most messages pass through them

Which queues have most messages pass through them?

Use the base queue name, number of valid messages put and got to identify the hot queues. The fields are wq->***basename, wq->validput, wq->validget***.

Which queues have the most data pass through them?

Use the base queue name, number of bytes put to the queue and the number of bytes got from the hot queues. The fields are wq->***basename, wq->putbytes, wq->getbytes***.

Which queues are business critical – even though they do not process the most messages?

You need to work with your applications architects and programmers to determine this information.

Which applications process the most messages – they may use many queues?

Use the information in the task data to identify the transaction, CICS region, connection name or jobname and the number of valid messages put and got. The fields are wq->***validput, wq->validget***.

Which applications process most messages through a single queue?

Use the information in the task data to identify the transaction, CICS region, connection name or jobname and the number of valid messages put and got to identify the hot queues. The fields are wq->***basename, wq->validput, wq->validget***.

What applications use business critical queues?

You need to work with your applications architects and programmers to determine this information.

What are the high use channels by number of messages?

Use the information in the task data to channel name and address, and the number of valid messages put and got. The fields are wq->***wtidchl, wq->wtidchlc*****, wq->*****validput, wq->validget***.

What are the high use channels by amount of data?

Use the information in the task data to channel name and address, and the number of valid messages put and got for each channel used. The fields are wq->***wtidchl, wq->wtidchlc, wq->putbytes, wq->getbytes.***

What are the channels to my critical destination?

You can use the fields are wq->***wtidchl, wq->wtidchlc*** to identify the channel name(wq->***wtidchl*****)** from the address (wq->***wtidchlc****).*

Number of messages on a queue and amount of space needed on a page set for local queues

The fields wq->***putbytes***, and wq->***getbyte*****s** are the number of bytes of message data processed during put/put1 and get processing. This is for requests which ended in completion code 0, or completion code 1 and reason MQRC_truncated_msg_accepted. So you can calculate the average message size processed.
There is also a field wq->***maxqdpth*** which is the maximum depth of the queue during put or get processing.

There are also fields giving the smallest, maximum and total time of messages on the queue. These fields are wq->***minlatnt, wq->maxlatnt****,* and ***wq->totlatnt****.*

To get an indication of the space used by the messages, you can calculate the maximum depth (for all applications using this queue) * average message size. This will underestimate the space used. A more realistic, but possibly over estimate is to calculate how many 4KB pages will be needed to hold the message, so for a thousand 1KB messages, for each message allocate 1 4KB page, so for 1000 messages, this would use about 1000 pages. These are only estimates because when messages are got from a queue the space is not immediately released and reused, and it may take 10s of seconds before empty pages are released, and if there were two messages on a page and one has been got, the page is still used by the other message, until after this message has been got and committed.

Collecting Accounting class(3) information

Using Accounting class(3) data

MQ on z/OS provides information in the Accounting class(3) SMF records on MQ verbs, resources and times spent in the MQ verbs.

To start collecting this accounting data you use the
+cpf start trace(A)class(3)
command.
To stop collecting this data you use the
+cpf stop trace(a)class(3)

By default the data goes to SMF.

Accounting data will be recorded for any queues opened after the start command has been issued. If the queue was opened before the start command as issued, accounting data will not be recorded for the queue.
For a particular queue you can suppress the collection of accounting data by the queue attribute ACCTQ;

When are the SMF records produced?

The accounting data is written to SMF when the application ends. If this is a long running application, the data is written out on the statistics broadcast. This is controlled with the STATIME field. If STATIME is 0 then the SMF statistics broadcast is used to signal collection of long running accounting data. You can thus synchronize the data for a particular time, such as on the hour; If this value is non zero it is the number of minutes between collecting the data.
You can use the DISPLAY SYSTEM command to display the current value of STATIME.

Time values

In some fields a time value is given as a number of microseconds. Other values are STCK (Store Clock) format. This is an 8 character field which can be converted to a time interval by dividing by 4096. This gives a value in microseconds.