#SHAREorg

SHARE
Technology · Connections · Results

Diagnosing Problems for MQ z/OS

Neil Johnston - neilj@uk.ibm.com
WebSphere MQ z/OS L3 – IBM Hursley

February 8th, 2013
Session #12625

SHARE in San Francisco 2013

Agenda

- “My application failed”.
  - Gathering available information.
  - Creating additional diagnostic data.
- “My message is missing”.
  - Message tracking techniques.
    - Locating a message in a simple system.
  - Advanced message tracking.
    - Identifying message delivery routes.
    - Delayed messages.
- How to avoid problems.

“My application failed”

Ask the user and application owner

- What were they doing?
  - Which application, queue manager and queue?
  - Was this normal processing, or something unusual?
- What went wrong?
  - Get specific details.
  - Any error messages?
- What was the expected result?
- When did it happen?
  - Only once
  - Repeatedly over a period
  - Still occurring

Ask the user and application owner

- In many cases the users of the system will be the first to notice when something isn't behaving the way it should, so they would generally be the source of primary information about the issue to guide further investigation.
- An initial problem report often only includes a basic description of what the issue is. There is likely to be more useful information available, and it is important to get as many details as possible at the start of your investigations.
- Asking the right questions at the start can save a lot of hard work later on.
- Get a detailed description of what the problem is and what applications, queue managers and queues are being used. If the users won't have that level of detail then ask the application owners
- Find out about the scope of the problem. How many users and applications are impacted? Is there an ongoing issue or was it just during a specific period?

Application symptoms

- MQ provides details about failures to the application
  - Specific reason codes
- Check application error logs
  - Detailed error reports are a big help
    - “Application failed” - Unhelpful
    - “Error opening queue with completion code 2” – Slightly better
    - “MQOPEN failed with reason 2059 for APP1.REPLY” – Good
- Applications can have multiple components
  - Web page – servlet – EJB – JMS – MQ API
  - Errors may be reported in several places

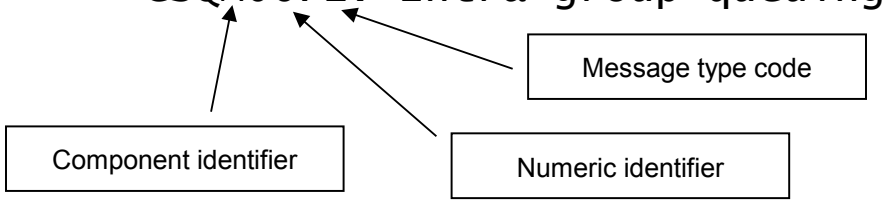
Application symptoms

- When an MQ API call fails, MQ notifies the application via the completion code and reason code fields.
  - Completion code gives an indication of the severity of the issue.
  - Reason code identifies the cause. These are a specific as possible to make it easier to identify and correct the problem.
- For most cases, these are the only indication of the error that MQ gives. It is expected that applications will take appropriate action for failures reported through the API.
- The quality of the information captured by the application can have a significant impact on how easy it is to identify the cause of the problem. It is worth producing detailed error messages for MQ API failures.
- Many applications have a complex architecture with a number of components involved between the user interface and MQ. Since any or all of the components may capture useful diagnostic information, it is important to understand the places where each part reports errors.

MQ error reporting

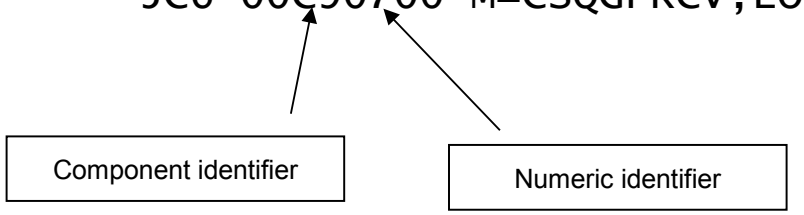
- MQ MSTR and CHIN tasks provide diagnostics for errors
  - Messages in joblog

```
CSQM067E: Intra-group queuing agent ended abnormally.
```



  - Task abends
    - Abend code x'5C6' or x'6C6'
    - Reason code identifies cause

```
5C6-00C90700 M=CSQGFRCV,LOC=CSQILPLM.CSQILCUR+00000302
```



MQ error reporting

- Errors detected by MQ are reported via messages and abends.
- Messages are written to the queue manager or chinit joblog to provide information about the state of the system, or to request system operator action.
- All MQ messages have an eight character identifier:
  - Characters 1-3 – MQ Message prefix. Always “CSQ” for MQ on z/OS.
  - Character 4 – Identifier for the component of MQ which produced the message.
  - Characters 5-7 – Numeric value to uniquely identify the message within the component.
  - Character 8 – Message type code.
    - A = Immediate action
    - D = Immediate decision
    - E = Eventual action
    - I = Information only
- MQ issues abends when it detects errors which the task cannot resolve.
- Two system abend codes are used by MQ.
  - X’5C6’ – Error during normal operation. Internal task or user task is abended.
  - X’6C6’ – Severe error which could result in loss of data integrity. MQ subsystem abended.
- MQ abends are accompanied by a four byte reason code which identifies the cause of the failure.
  - Byte 1 – Always x’00’
  - Byte 2 – Hexadecimal identifier of the MQ component.
  - Bytes 3-4 – Unique identifier identifying the specific abend within the component.
- The infocenter provides detailed descriptions about the cause of each message or abend, and identifies actions which should be taken when a particular error occurs.

GTF trace

- MQ uses z/OS GTF trace facility for diagnostic trace.
- API trace and internal trace
  - 5E9 – API entry
  - 5EA – API exit
  - 5EE – Internal trace
- Trace data written to wrapping dataset
- IPCS formatting required to produce readable output.

GTF trace

- MQ has the ability to produce detailed diagnostic trace for both user parameters and internal queue manager processing.
- MQ uses the Generalized Trace Facility (GTF) on z/OS to record its trace information.
- The GTF component collects trace records from MQ, along with other z/OS system components and products, and records them in binary format either to an in-storage trace table or to a dataset.
- After collection, the IPCS tool can be used to process the binary GTF trace into a readable format. The z/OS components and products which use GTF supply formatters to interpret their GTF trace records and produce appropriate output.

GTF trace cont'd

- Start GTF

```
START GTF.DB
   £HASP100 GTF.DB ON STCINRDR
   £HASP373 GTF.DB STARTED
  *01 AHL100A SPECIFY TRACE OPTIONS
R 01,TRACE=JOBNAMEP,USRP
   TRACE=JOBNAMEP,USRP
   IEE600I REPLY TO 01 IS;TRACE=JOBNAMEP,USRP
  *02 ALH101A SPECIFY TRACE EVENT KEYWORDS - JOBNAME=,USR=
R 02,JOBNAME=(MQ11MSTR,MQAPP1),USR=(5E9,5EA)
   JOBNAME=(MQ11MSTR,MQAPP1),USR=(5E9,5EA)
   IEE600I REPLY TO 02 IS;JOBNAME=(MQ11MSTR,MQAPP1),USR=(5E9,5EA)
  *03 ALH102A CONTINUE TRACE DEFINITION OR REPLY END
R 03,END
   END
   IEE600I REPLY TO 03 IS;END
   AHL103I TRACE OPTIONS SELECTED-USR=(5E9,5EA)
   AHL103I JOBNAME=(MQ11MSTR,MQAPP1)
  *04 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
R 04,U
   U
   IEE600I REPLY TO 04 IS;U
   AHL031I GTF INITIALIZATION COMPLETE
```

GTF trace cont'd

- A GTF task must be started to collect the trace data produced by MQ.
- GTF uses a wrapping trace model. When the available dataset or in storage table are full, the oldest trace entries are overwritten.
- When tracing to a dataset GTF only uses the dataset's primary allocation. Although GTF can support tracing to multiple datasets, the start procedure used in most cases only has a single dataset specified. The primary allocation size for the dataset needs to be sufficient to hold a period of tracing which will allow capture of trace of the problem.
- The parameters supplied on the start command can be used to limit the trace collected by specifying trace ids and jobnames. Restricting the trace to only those jobs and trace types of interest increases the time period which can be covered by the trace dataset.
- For MQ, USR trace entries with event identifiers 5E9 and 5EA are used for user parameter trace on entry and exit of MQ API calls.
- 5EE provides detailed internal trace. This is not normally useful for user problem diagnosis and is only likely to be needed for problem diagnosis by IBM support.

GTF trace cont'd

- Start MQ Trace

```
+MQ11 START TRACE(G)CLASS(3) DEST(GTF)
```

  - All Entry and Exit

```
+MQ11 START TRACE(G)CLASS(2) DEST(GTF)
```

  - Only when exit reason is not MQRC_NONE

- Other MQ trace control

```
+MQ11 DISPLAY TRACE ...
+MQ11 ALTER TRACE ...
+MQ11 STOP TRACE ...
```

GTF trace cont'd

- MQ global trace needs to be enabled with the START TRACE command.
- The destination needs to specify that GTF tracing is required.
- There are several constraint parameters which can be specified to restrict the type of trace recorded. For user trace, the class parameter should be specified to indicate the type of events which should be traced.
  - CLASS(2) traces API exit when the reason code for the API call is not MQRC_NONE.
  - CLASS(3) traces API all entry and exit calls.
- The DISPLAY TRACE command lists the currently active traces.
- ALTER TRACE and STOP TRACE can be used to make changes to a specific active trace.

GTF trace cont'd

- Example output

```
USRD9 5EA ASCB 00F87E80            JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
PUTONE
    Thread..  004C2B10  Userid... CICSUSER  pObjDesc. 106B2010
    pMsgDesc. 106B20B8  pPMO..... 106B2200  BufferL.. 00000064
    pBuffer.. 106A0578  RSV1..... 00000000  RSV2..... 00000000
    RSV3..... 116BC830  CompCode. 00000002  Reason... 000007FB
    C9E8C1E8  C5C3C9C3  AA8E8583  76270484  | IYAYECIC..ec...d |
    D4D8E3E3  0000048C  00000000  00000000  | MQTT............ |
    00000000  1910C7C2  C9C2D4C9  E8C14BC9  | ......GBIBMIYA.I |
    C7C3E2F2  F0F48E85  83762979  00010000  | GCS204.ec..`.... |
MQRC_OBJECT_TYPE_ERROR

              GMT-01/30/11 14:42:08.412678    LOC-01/30/11 14:42:08.412678

USRD9 5EA ASCB 00F87E80            JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D6C44040  00000001  00000000  C2404040  | OD  ........B    |
 ...
 +00A0  00000000  00000000                      | ........         |
```

GTF trace cont'd

- After formatting, the GTF trace gives a detailed report of the parameters at entry and exit for each API call.
- The first section reports details of the values passed on the API call and some information about the caller.
  - Thread identifier and userid.
  - Pointer values for the parameters passed on the call.
  - Completion code and reason code on exit.
- Text identifiers are included for the API call and the reason code if it is non-zero.
- Additional sections produce hexadecimal and eyecatcher output for each of the parameter blocks for the call. Message data is also written out, limited to 256 bytes to prevent large messages from filling the trace.

Capturing a dump

- z/OS system dumps are an important tool for capturing system state at the time of an error.
- Dump may have already been captured.
  - MQ 5C6 abends
  - Application requested dump
  - Other z/OS components
- Several methods to generate a dump for a failure
  - Console DUMP command
  - SLIP trap
  - RECOVER QMGR(MQRD,2051,1)
- MQ Dump formatters CSQWDPRD and CSQXDPRD

Capturing a dump

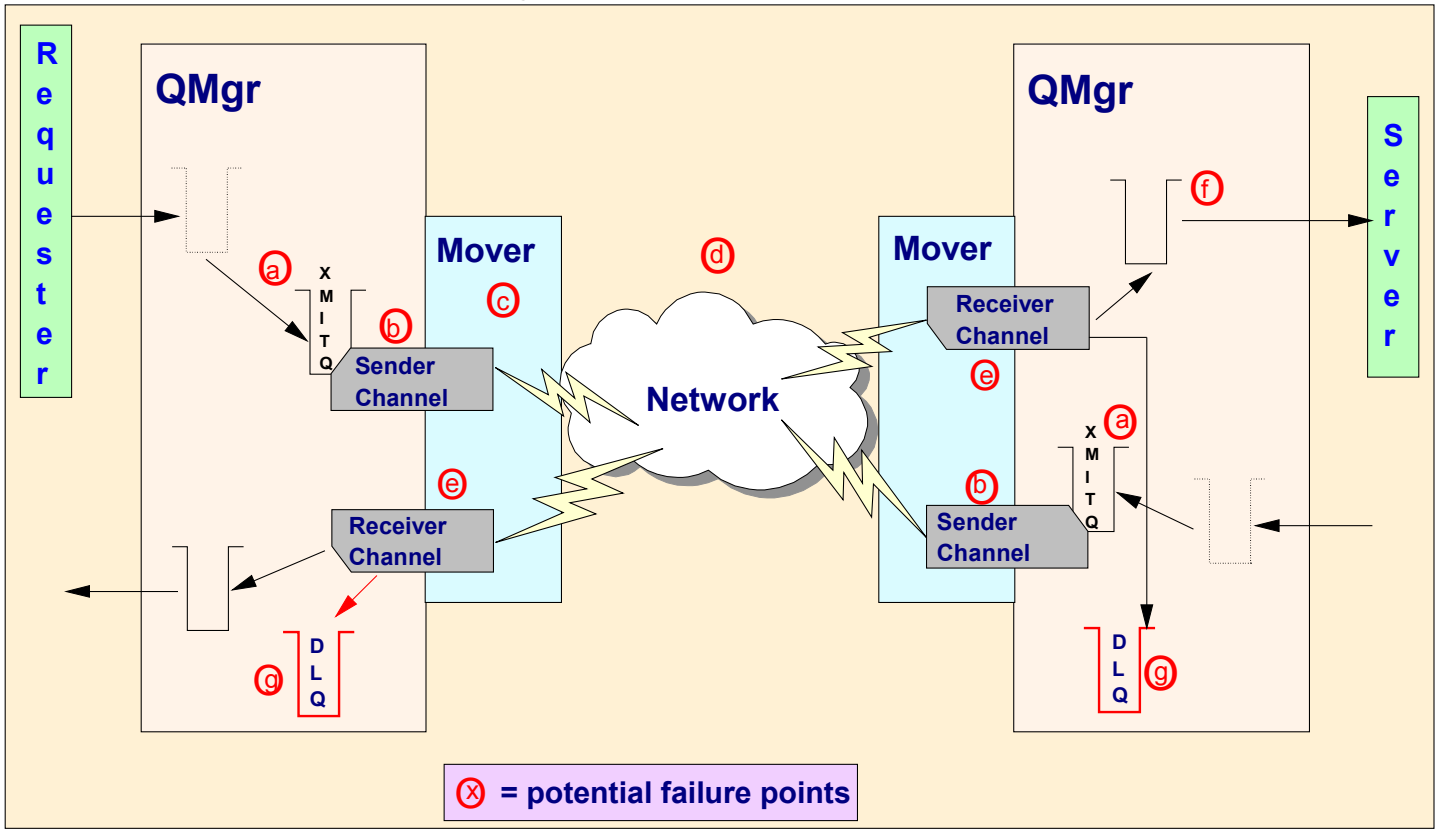
- z/OS SVC dumps are very useful for problem analysis as they capture the state of the relevant address spaces at the time of a failure.
- Depending on the type of error being encountered, one or more dumps may have been captured by MQ, a z/OS component or another application in response to the failure.
- Check the system log for indications that a dump has been taken.
- If no dump was taken, there are several different ways to request a dump on a subsequent incident. The best method to choose depends on the nature of the problem.
- A dump can be requested from the z/OS console using the dump command. This gives good control over what is dumped. As it is requested manually, however, it makes it very difficult to capture the dump at the right time. Useful for an ongoing issue such as a hang or a loop.
- A slip trap can be used to create a dump when an event occurs. The event can be a message being issued, an abend, execution of a particular piece of code or based on storage values or alterations.
- MQ V7 provides an undocumented diagnostic command to capture a dump of the application and qmgr address spaces when a particular reason code occurs for an API call.
  - RECOVER QMGR(MQRD,xxxx,n)
    - xxxx is the reason code
    - n is the number of times it will trigger a dump before disabling.
  - RECOVER QMGR(MQRD,9999) will disable the monitoring from a previous command.
- MQ provides two dump formatting utilities
  - CSQWDPRD for the queue manager
  - CSQXDPRD for the chinit

“My message is missing”

Message tracking techniques

Where might it have gone wrong?

- A simple request/reply application



Where might it have gone wrong?

- Even with a fairly simple application message path, there are plenty of places where messages could go astray.
  - (a) XMITQ
    - PUT/GET disabled
    - Not defined with USAGE(XMITQ)
  - (b) Sender Channel
    - Not running
    - May be running but nothing is actually happening
  - (c) Mover
    - Not running
  - (d) Network
    - Down or very slow
  - (e) Receiver Channel
    - Not running
  - (f) Target queue PUT disabled
  - (g) Dead Letter Queue
    - The Receiver channel may be putting messages onto it

Should the message still be in MQ?

- There are valid reasons why a message could be removed from MQ.
  - Was the MQPUT successful?
  - Did the application commit?
  - Is the message non-persistent?
    - Queue manager restart
    - Channel failure
    - Read ahead
  - Did the message have expiry set?

Should the message still be in MQ?

- Before trying to track down a missing message in MQ, it is worth determining if if would be expected to still be there. Some messages are not intended to stay in MQ indefinitely.
- If the MQPUT call did not complete successfully, then the message will never have been put to a queue. The data will have been left in the application's buffer.
- For messages put in syncpoint, the application must commit the unit of work for the message to be successfully put to the queue. A backout of the unit of work results in the message being removed from the system.
- Non-persistent messages can be discarded by MQ when a queue manager restarts or a failure occurs on a channel. The read ahead option for clients getting messages from a queue can also cause non-persistent messages to be discarded if the client ends with messages still on its in-storage read ahead queue.
- Message expiry causes messages with a time-limited value to be discarded if they remain in the queue manager longer than the specified time.

MQ Commands

- Command interfaces to inquire on MQ object status
  - MQSC – Text format commands
    - CSQUTIL COMMAND
  - PCF – Programmable format, useful for monitoring applications
  - Information also obtainable via tools
    - MQExplorer
    - MQ Operations and Control ISPF panels
- Display object commands show object attributes
    - E.g. DISPLAY QUEUE(APP1.INPUT) MAXDEPTH
- Display status commands show current state information
    - E.g. DISPLAY QSTATUS(APP1.INPUT) CURDEPTH

MQ Commands

- The primary method for obtaining information about the state of an active queue manager is though the MQ command interfaces (or through tools which provide user interfaces to the data supplied by those commands).
- MQ can supply the information about objects using two different command formats.
  - MQSC uses readable text for requests and replies, making it the most appropriate choice for direct user interactions or for scripted automated requests.
  - Programmable Command Format (PCF) is a structured data format where both requests and responses are constructed from data fields in a format which is suitable for programmatic interpretation.
- MQ supplies user interfaces which take the command output and present it in a format suitable for the environment.
  - MQ Explorer. An eclipse based graphical user interface for MQ administration. Available on Windows and Linux with the MQ server, or as SupportPac MS0T.
  - MQ Operations and Control Panels. ISPF panel interface provided with MQ on z/OS.
- Interrogating the queue manager for problem diagnosis will mostly use the DISPLAY command. There are two types of display command for most MQ objects:
  - Object display (e.g. DISPLAY QUEUE) shows the configurable attributes for the object.
  - Status display (e.g. DISPLAY QSTATUS) shows current state information for the object. Some status commands allow further qualification to format different aspects of the status (e.g. TYPE(QUEUE) or TYPE(HANDLE)).

MQ Commands cont'd

- DISPLAY QSTATUS

```
+MQ11 DISPLAY QSTATUS(APP1.INPUT) ALL
CSQM293I +MQ11 CSQMDRTC 1 QSTATUS FOUND MATCHING REQUEST CRITERIA
CSQM201I +MQ11 CSQMDRTC  DISPLAY QSTATUS DETAILS
QSTATUS(APP1.INPUT)
TYPE(QUEUE)
OPPROCS(1)
IPPROCS(0)
CURDEPTH(4)
UNCOM(NO)
MONQ(HIGH)
QTIME(6639576,9403795)
MSGAGE(7)
LPUTDATE(2011-07-30)
LPUTTIME(21.15.57)
LGETDATE(2011-07-30)
LGETTIME(21.16.00)
QSGDISP(QMGR)
 END QSTATUS DETAILS
CSQ9022I +MQ11 CSQMDRTC ' DISPLAY QSTATUS' NORMAL COMPLETION
```

MQ Commands cont'd

- The display qstatus command gives details about the current state of the queue.
- When trying to find a lost message, the following fields are particularly relevant:
  - OPPROCS – how many handles have the queue open for put
  - IPPROCS – how many handles have the queue open for get
  - CURDEPTH – are there any messages being left on the queue? Is there a backlog?
  - UNCOM – does the queue have uncommitted activity?
    - If UNCOM is YES, then there have been puts of gets against the queue which have not yet been committed.
    - Put messages will show up in the curdepth, but won't be available for a getting application to process.
    - Got messages are no-longer counted in the curdepth. They are still logically in place on the queue, but are locked until the application commits. If backout is requested then the message would return to the queue.
    - For shared queues this only tells about applications connected to THIS queue-manager so use CMDSCOPE(*).

MQ Commands cont'd

- DISPLAY CHSTATUS

```
+MQ11 DISPLAY CHSTATUS(MQ12.TO.MQ11) ALL
CSQM293I +MQ11 CSQMDRTC 1 CHSTATUS FOUND MATCHING REQUEST CRITERIA
CSQM201I +MQ11 CSQMDRTC  DISPLAY CHSTATUS DETAILS
CHSTATUS(MQ12.TO.MQ11)                                  CHSTATI(21.25.35)
CHLDISP(PRIVATE)                                        CHSTADA(2011-07-30)
CONNAME(::ffff:192.168.1.100)                           BUFSSENT(20)
CURRENT                                                 BUFSRCVD(32)
CHLTYPE(RCVR)                                           MONCHL(HIGH)
STATUS(RUNNING)                                         EXITTIME(0,0)
SUBSTATE(RECEIVE)                                       XBATCHSZ(1,1)
INDOUBT(NO)                                             COMPTIME(0,0)
LSTSEQNO(20)                                            COMPRATE(0,0)
LSTLUWID(AB68344E10000112)                              STOPREQ(NO)
CURMSGS(0)                                              KAINT(360)
CURSEQNO(20)                                            QMNAME(MQ11)
CURLUWID(AB68344E10000112)                              RQMNAME(MQ12)
LSTMSGTI(21.30.14)                                      MCAUSER(MQMTASK)
LSTMSGDA(2011-07-30)                                    LOCLADDR( )
MSGS(20)                                                BATCHSZ(50)
BYTSSENT(976)                                           MAXMSGL(4194304)
BYTSRCVD(10346)                                         HBINT(300)
BATCHES(18)                                             NPMSPEED(FAST)
 END CHSTATUS DETAILS
CSQ9022I +MQ11 CSQMDRTC ' DISPLAY CHSTATUS' NORMAL COMPLETION
```

MQ Commands cont'd

- The display chstatus command gives details about the current state of the channel.
- For a channel the priority is to confirm that it is active and sending/receiving messages.
- The following fields provide this information:
  - STATUS – gives an overall indication of the health of the channel. For some status values there may be additional useful information in the substate field.
  - MSGS – the number of messages sent or received since the channel was started.
  - CURMSGS – how many messages have been sent or received for the current batch.

MQ Log Data Sets

- MQ Log Data Sets record
  - Persistent messages
  - MQ object changes
- CSQ1LOGP utility to format logs
  - EXTRACT function provides a report record for each event
    - Persistent puts and gets
    - Commit and backout
    - Object changes
  - Extracted messages can be replayed to queues
  - DATA keyword to find specific data e.g. MsgId or CICS taskid (for example 0123456C for taskid 123456)
  - URID keyword for a specific unit of recovery

MQ Log Data Sets

- MQ records information relating to persistent messages and queue manager objects in Log data sets. These logs contain sufficient information to restore the state of queues and messages in the event of a queue manager failure. They can also be used to rebuild queue state from backups in the event of pageset media failure or CF failure.
- The information contained in the logs provides a very powerful resource for determining the course of events when diagnosing an MQ issue.
- The CSQ1LOGP utility is provided with MQ to allow offline formatting of the MQ log data sets. The data can be formatted in several ways, but the output which is easiest to interpret for diagnosis purposes is produced by the EXTRACT option.
- The EXTRACT option produces data records showing relevant information for log entries relating to persistent message puts and gets, transaction commit and backout, and alterations to object attributes.
  - Record format mapped in C header file CSQ4LOGD.
  - Records are sufficient to “replay” the processing.
    - Sample application CSQ4LOGS provided to do this.

MQ Log Data Sets cont'd

- CSQ1LOGP EXTRACT output

Time	UR identifier	Userid	App type	Job	Data length	Queue Name	Message key	Verb	MD and body
15:08:40.319	00000B2EEE82	JSMITH	BATCH	APP1	0155	APP.INPUT	00009101	MQPUT	D4C44040....
15:08:40.319	00000B2EEE82	JSMITH	BATCH	APP1	0000			PHASE1	
15:08:40.319	00000B2EEE82	JSMITH	BATCH	APP1	0000			PHASE2	
15:08:43.151	00000B2EF3FA	DJONES	BATCH	APP2	0000	APP.INPUT	00009101	MQGET	
15:08:43.151	00000B2EF3FA	DJONES	BATCH	APP2	0000			PHASE1	
15:08:43.151	00000B2EF3FA	DJONES	BATCH	APP2	0000			PHASE2	

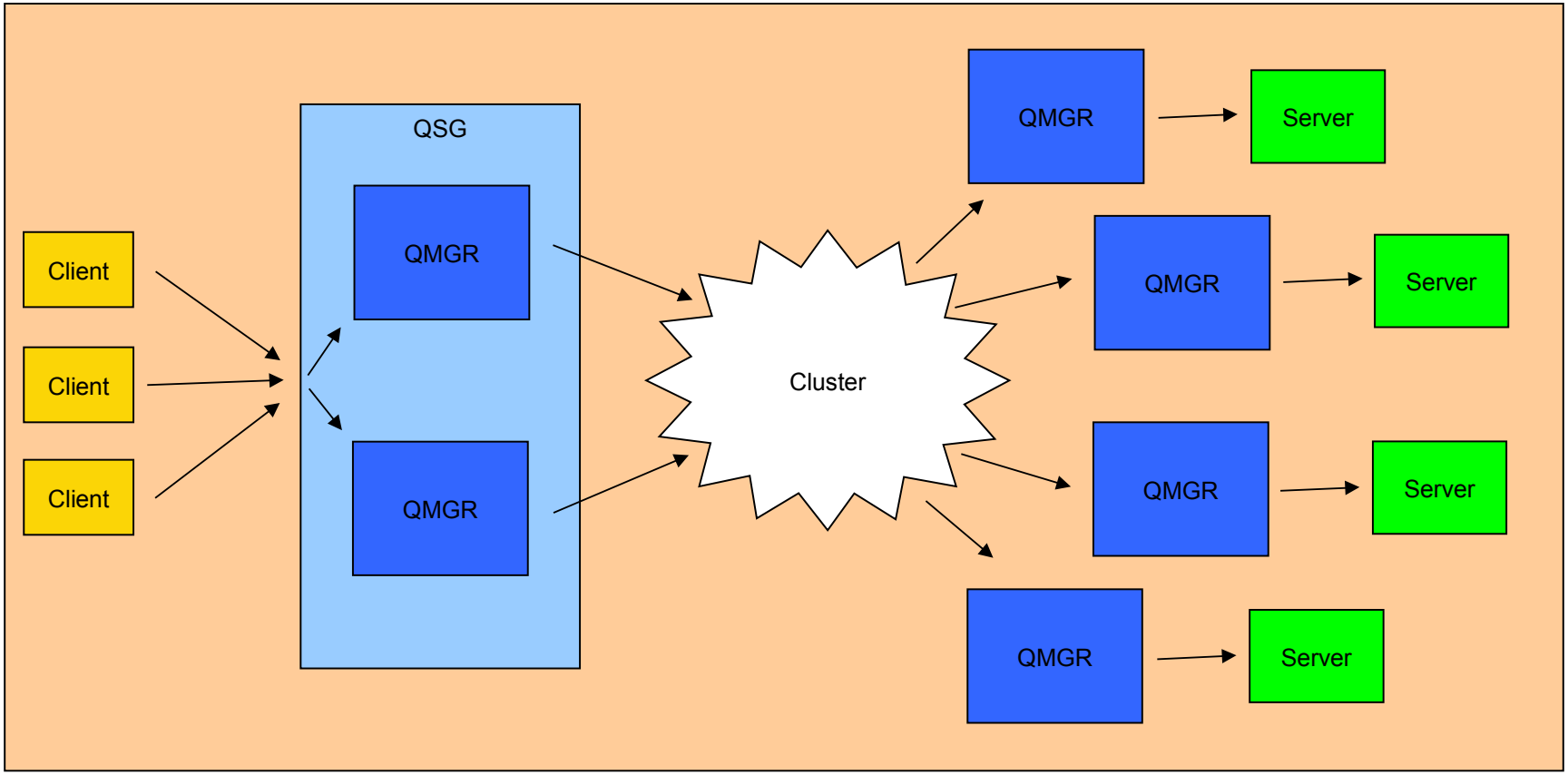
MQ Log Data Sets

- The example data shows a message being put and then got by different applications.
- Use the UR identifier to match all the records for a given unit of recovery.
  - The data shows 2 URs with 3 records each.
- The Verb field shows the task that the record is performing.
  - The first UR does an MQPUT followed by commit (phase 1 and 2).
  - The second UR does an MQGET and commits.
- Put records contain the MQMD and message body. Specific messages can be found by searching for unique values in the MQMD or body (e.g. msgid, correlid or some recognisable portion of the data).
- Get records do not contain any message data.
  - Match up with the message key field from the MQPUT to find the get or a particular message.
  - Note that the message keys are reused for local messages, so you need to ensure that you match a put with the get for the next occurrence of the key.

“My message is missing”

Advanced message tracking

Variable message routes



Variable message routes

- More complex configurations make it much more difficult to determine where a missing message might have gone.
- Messages may pass through multiple queue managers to reach their destination.
- Some queue manager network configurations introduce multiple possible routes and destinations for a message.
  - QSG group listener allows client and messaging channels to connect to any one of the queue managers in the QSG that is listening.
  - Clusters allow remote queue resolution to one of a collection of queue managers hosting the requested queue.
- The set of available routes / destinations can change over time
  - Queue manager or channel failures.
  - Object changes (e.g. setting a clustered queue to put disabled or stopping a channel).

Identifying message routes

- Activity recording
  - Activity reports generated by applications which perform actions on a message
    - Queue Manager and Chinit
    - User applications
- Can be requested for application messages
- Trace-route messages provide more flexibility
  - dspmqrte tool
    - Generates trace-route requests
    - Collects and displays results

Identifying message routes

- MQ messages can request that activity records are generated to record the processing that applications perform on those messages.
- When activity reporting is enabled, the queue manager and chinit produce activity reports whenever they process a message which requests them.
- User applications are able to produce their own activity reports.
- The MQRO_ACTIVITY report option can be set on application messages to request activity reporting for that message.
- Trace-route messages are diagnostic messages which can be used to collect activity reports. They provide additional flexibility on how the activity reporting is gathered, and can record additional information about the route taken by the message.
- The dspmqtre tool, provided with distributed MQ, can be used to generate trace-route messages and format the returned data.

dspmqrte tool

- Test application for submitting trace-route requests and processing responses
- Not available on z/OS, but can connect to z/OS queue manager in client mode

Summary output:

```
C:\>SET MQSERVER=SYSTEM.DEF.SVRCONN/TCP/192.168.1.100(1999)

C:\>dspmqrte -c -q WINQMGR1.APP1.QUEUE -o

AMQ8653: DSPMQRTE command started with options '-c -q WINQMGR1.APP1.QUEUE -o'.
AMQ8659: DSPMQRTE command successfully put a message on queue
'WINQMGR1.APP1.QUEUE', queue manager 'MQ11'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'WINQMGR1.APP1.QUEUE' on queue manager 'MQ11'.
AMQ8666: Queue 'MQ12.TO.WINQMGR1' on queue manager 'MQ12'.
AMQ8666: Queue 'APP1.QUEUE' on queue manager 'WINQMGR1'.
AMQ8652: DSPMQRTE command has finished.
```

dspmqrte tool

- The MQ display route application (dspmqrte) provides the ability to produce trace-route request messages and interpret the generated activity responses.
- The basic functionality involves connecting to a queue manager and putting a trace-route message to the queue name or topic string of interest. The application then waits for the responses to be returned and produces an output report containing the results.
- There are a significant number of options available to alter the attributes of the trace-route message, how the results are recorded and returned, and the content of the output report.
- The summary output gives a series of messages indicating which queues and queue managers the message traversed.

dspmqrte tool cont'd

Detailed output:

```
C:\>dspmqrte -c -q WINQMGR1.APP1.QUEUE –o –v outline

-------------------------------------------------------------------------------
Activity:
 ApplName: 'ebSphere MQ\bin\dspmqrte.exe'
 Operation:
  OperationType: Put
  QMgrName: 'MQ11                                                '
  QName: 'WINQMGR1.APP1.QUEUE                                   '
  RemoteQName: 'WINQMGR1.APP1.QUEUE                                    '
  RemoteQMgrName: 'MQ12                                                 '
-------------------------------------------------------------------------------
Activity:
 ApplName: 'MQ11CHINCSQXRCTL1464FA50    '
 Operation:
  OperationType: Get
  QMgrName: 'MQ11                                                '
  QName: 'SYSTEM.CLUSTER.TRANSMIT.QUEUE                         '
  ResolvedQName: 'SYSTEM.CLUSTER.TRANSMIT.QUEUE                          '
 Operation:
  OperationType: Send
  QMgrName: 'MQ11                                                '
  RemoteQMgrName: 'MQ12                                                 '
  ChannelName: 'TO.MQ12             '
  ChannelType: ClusSdr
  XmitQName: 'SYSTEM.CLUSTER.TRANSMIT.QUEUE                      '
-------------------------------------------------------------------------------
```

dspmqrte tool cont'd

- When the summary report does not give enough detail to determine exactly what happened to the message, the –v option can be used to request additional levels of detail.
- The outline report shown gives key fields for each of the activities and operations reported for the test message.
- In the example output, it can be seen that the WINQMGR1.APP1.QUEUE queue actually resolved to a remote queue, and that the SYSTEM.CLUSTER.TRANSMIT.QUEUE was used.

Delayed messages

- “Missing” messages may just have been delayed
  - Application sees MQRC_NO_MSG_AVAILABLE
  - Message is found on target queue
- Finding processing delays for problem messages
  - CSQ1LOGP
  - Activity reports
- Identifying queue manager components with backlogs
  - Status commands
  - Statistics and accounting data

Delayed messages

- Many request / response style applications will only wait for their response for a specified time period before giving up and failing the request. With this processing model, the application has no way of knowing whether the request or reply has been lost, or has just taken a long time to get through the system. In both cases, the MQGET would receive MQRC_NO_MSG_AVAILABLE.
- If it is found that the message did get to the target queue, then the next line of investigation would be to determine how long it took to get there, and where it got slowed down.
- CSQ1LOGP and activity reports both provide timestamps for each processing step of a message, so they can provide good indications of where in the system the message spent most time.
- Performance problems often impact all the processing for an affected queue, application or queue manager. Statistical monitoring of the state of the queue manager can highlight problem components.

Real-time monitoring

- DISPLAY QSTATUS

```
+MQ11 DISPLAY QSTATUS(APP1.INPUT) ALL
CSQM293I +MQ11 CSQMDRTC 1 QSTATUS FOUND MATCHING REQUEST CRITERIA
CSQM201I +MQ11 CSQMDRTC  DISPLAY QSTATUS DETAILS
QSTATUS(APP1.INPUT)
TYPE(QUEUE)
OPPROCS(1)
IPPROCS(0)
CURDEPTH(4)
UNCOM(NO)
MONQ(HIGH)
QTIME(6639576,9403795)
MSGAGE(7)
LPUTDATE(2012-07-30)
LPUTTIME(21.15.57)
LGETDATE(2012-07-30)
LGETTIME(21.16.00)
QSGDISP(QMGR)
 END QSTATUS DETAILS
CSQ9022I +MQ11 CSQMDRTC ' DISPLAY QSTATUS' NORMAL COMPLETION
```

Real-time monitoring

- Queue and channel objects can be enabled to keep in-storage monitoring information reflecting the ongoing performance of the object.
- Some of the fields record a weighted average of a value for recently processed messages. For these fields, two values are supplied.
  - Short term indicator – new values receive a high weighting when calculating the average, so the value reflects the recent trend.
  - Long term indicator – new values receive a lower weighting, resulting in a value which shows the trend over a longer period.

  As well as being able to examine the absolute size of the values, comparing the value pairs provides an indication of how the performance is changing over time.
- The monitoring fields for qstatus show:
  - MSGAGE – Age of the oldest message on the queue (in seconds).
  - LPUTDATE & LPUTTIME – The time that the last message was put to the queue.
  - LGETDATE & LGETTIME – The time that the last message was got from the queue.
  - QTIME – The interval (in microseconds) between a message being put to the queue and being destructively got.

Real-time monitoring cont'd

- DISPLAY CHSTATUS

```
+MQ11 DISPLAY CHSTATUS(MQ11.TO.MQ12) ALL
CSQM293I +MQ11 CSQMDRTC 1 CHSTATUS FOUND MATCHING REQUEST CRITERIA
CSQM201I +MQ11 CSQMDRTC  DISPLAY CHSTATUS DETAILS
CHSTATUS(MQ11.TO.MQ12)                          CHSTATI(09.19.04)
CHLDISP(PRIVATE)                                CHSTADA(2011-08-04)
XMITQ(MQ11.TO.MQ12)                             BUFSSENT(22)
CONNAME(192.168.1.100)                          BUFSRCVD(13)
CURRENT                                         LONGRTS(999999999)
CHLTYPE(SDR)                                    SHORTRTS(10)
STATUS(RUNNING)                                 MONCHL(HIGH)
SUBSTATE(MQGET)                                 XQTIME(229,167)
INDOUBT(NO)                                     NETTIME(2896,3059)
LSTSEQNO(11)                                    EXITTIME(0,0)
LSTLUWID(C82B9F203F851910)                      XBATCHSZ(1,1)
CURMSGS(0)                                      COMPTIME(0,0)
CURSEQNO(11)                                    COMPRATE(0,0)
CURLUWID(C82B9F21F04E1D5E)                      STOPREQ(NO)
LSTMSGTI(09.21.02)                              KAINT(360)
LSTMSGDA(2011-08-04)                            QMNAME(MQ11)
MSGS(11)                                        RQMNAME(MQ12)
BYTSSENT(6022)                                  LOCLADDR(192.168.1.99(4330))
BYTSRCVD(780)                                   BATCHSZ(50)
BATCHES(11)
 END CHSTATUS DETAILS
CSQ9022I +MQ11 CSQMDRTC ' DISPLAY CHSTATUS' NORMAL COMPLETION
```

Real-time monitoring cont'd

- For channels, the monitoring values shown by the display chstatus command are:
  - XQTIME – The length of time which messages spent waiting on the transmission queue before being retrieved by the channel (in microseconds).
  - NETTIME – The time taken to send a request to the remote end of the channel and receive a response (in microseconds).
  - EXITTIME – The time spent processing user exits per message (in microseconds).
  - XBATCHSZ – The size of batches transmitted over the channel.
  - COMPTIME – The amount of time spent doing compression or decompression (in microseconds)
  - COMPRATE – The compression rate achieved, expressed as a percentage.

Statistics and accounting

- MQ can record statistics and accounting data in SMF
- Performance statistics
  - Record type 115
  - Component related
  - Written at statistics interval
- Accounting data
  - Record type 116
  - Task related
  - Written when task disconnects, or at statistics interval for long-running tasks

Statistics and accounting

- MQ is able to collect a wide range of statistical data showing the performance and resource usage of the queue manager.
- Statistics data is recorded using z/OS System Management Facilities (SMF).
- MQ collects two types of statistical data:
  - Performance statistics and record at regular collection intervals. Each of the MQ components records information relevant to its function. This data is recorded in the SMF record type 115.
  - Accounting data is information about each task's use of MQ and system resource, and its primary function is to allow usage-based billing for MQ services. The data is recorded when the task disconnects from MQ, and uses record type 116.
- SMF writes the data in binary records which need to be interpreted by appropriate analysis tools to produce usable reports.
  - User written tools
  - Commercial system monitoring tools
- MQ provides record format information and a sample application to show how the SMF data can be processed.

How to avoid problems

Detect problems early

- Know what the normal state is for your system
  - MQ joblog messages
  - DISPLAY QSTATUS and CHSTATUS
  - Dspmqrte
  - SMF 115 statistics
- Configure instrumentation events
  - Queue manager events
  - Performance events
  - Channel events
  - Configuration events
  - Command events

Detect problems early

- An important aspect of both early problem detection and subsequent diagnosis of issues is the ability to identify which parts of the system are showing abnormal behaviour. Picking out the abnormal results from all the information available is far easier if you know what the system looks like under normal running conditions.
- Running the diagnostic tools when there isn't a problem will give a feel for the normal state of the system and will make sure you know how to use them when needed.
  - Understand what messages MQ issues to the joblog.
  - Use the DISPLAY commands discussed previously. Does a particular queue normally stay empty, or does it always have hundreds of messages?
  - Run dspmqrte to identify the expected routes of messages through complex systems.
- MQ provides configurable instrumentation events which will put an event message to a queue when the required conditions are met. These can provide an automated alerting mechanism for abnormal conditions in the queue manager and chinit and can also provide an audit trail for system changes.
  - Queue manager events – An application tries to open a queue which is not defined or is put disabled.
  - Channel events – A channels starts/stops. A channel fails to establish an SSL connection.
  - Performance events – Queue depth reached. Queue service interval exceeded.
  - Configuration events – Objects created, deleted or changed.
  - Command events – Record the origin, context and content of commands.

Detect problems early cont'd

- System resource monitoring
  - CPU usage
  - I/O
  - Storage
  - Paging
- External monitoring tools
  - Track MQ supplied data (SMF, RMF, events, messages)
  - Show history of data
  - Configure more sophisticated alerts

Detect problems early cont'd

- The z/OS system itself provides information on the usage of system resources such as CPU, I/O and storage. Understanding the requirements of both the queue manager and applications accessing MQ helps to manage the system resources appropriately and avoid issues caused by insufficient resource availability.
- In addition, other z/OS components used by MQ can provide information on the use of their resources. For example:
  - Resource issues in the CF or DB2 could affect queue managers in a QSG.
  - TCP/IP and network resource issues could impact the chinit.
- The volume of data produced by MQ and the z/OS system can be huge, making manual system monitoring difficult. A variety of monitoring tools are available which use the statistical and event data to provide automated system monitoring.
- These can provide real-time and historical status and can be used to provide alerts when action is required based on more sophisticated criteria.

Thank-you

Any questions?

This was session 12625 - The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday	
08:00					Are you running too many queue managers or brokers?	
09:30		What's New in WebSphere Message Broker			Diagnosing Problems for MQ	CICS and WMQ - The Resurrection of Useful
11:00		Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud	WMQ - Introduction to Dump Reading and SMF Analysis - Hands-on Lab	BIG Data Sharing with the cloud - WebSphere eXtreme Scale and WebSphere Message Broker integration	Getting the best availability from MQ on z/OS by using Shared Queues	
12:15						
01:30	Introduction to MQ	MQ on z/OS – Vivisection	Migration and maintenance, the necessary evil	The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation		
03:00	First Steps With WebSphere Message Broker: Application Integration for the Messy	BIG Connectivity with WebSphere MQ and WebSphere Message Broker	WebSphere MQ CHINIT Internals	Using IBM WebSphere Application Server and IBM WebSphere MQ Together		
04:30	WebSphere MQ application design, the good, the bad and the ugly	What's New in the WebSphere MQ Product Family	MQ & DB2 – MQ Verbs in DB2 & Q-Replication	WebSphere MQ Channel Authentication Records		
06:00			Clustering - The Easier Way to Connect Your Queue Managers			

#SHAREorg

SHARE
Technology · Connections · Results

SHARE in San Francisco 2013