2

MQ update

August 1999

In this issue

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: +44 1635 550955
e-mail: HarryLewis@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: +1 940 455 7050

Contributions

Articles published in *MQ Update* are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code. For more information about contributing an article, please check Xephon's Web site, where you can download *Notes for Contributors*.

MQ Update on-line

Code from *MQ Update* is available from Xephon's Web site at www.xephon.com (you'll need the user-id shown on your address label to access it). If you've a problem with your user-id or password call Xephon's subscription department on +44 1635 33886.

Editor

Harry Lewis

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

MQSeries in an OS/390 client/server environment

INTRODUCTION

MQSeries applications are typically designed to function asynchronously. The programs that comprise an MQSeries application might be running on different computers, on different operating systems, and at different locations and times. However, many installations now wish to use MQSeries for their on-line OS/390 client/server applications. This article explores the implications of implementing this architecture and recommends a number of options to make the implementation successful.

MQSeries' original design objective was to provide assured (but eventual) message delivery in asynchronous environments. In many cases, messages were expected to be stored and forwarded when a connection was later made or when the application that was to process the message became available. An example might be a retailing system that queues restocking requests to be transmitted to a warehouse system at the end of the day.

To many people the idea of using MQSeries in a client/server application, where a near-synchronous response is required, seems wrong. However, in spite of MQSeries' apparent unsuitability for this role, it's nevertheless very attractive to installations to standardize on a single communications programming interface across all applications and platforms. Also, even with client/server, it's possible to overlap front-end processing with back-end responses, so that the asynchronous nature of MQSeries can still be exploited.

PROBLEMS WITH MQSERIES IN CLIENT/SERVER APPLICATIONS

A client/server application in this context might be a Java GUI program running on a user's Windows NT workstation that sends requests and receives replies from a CICS/DB2 application running on an OS/390 mainframe. The two communicate over a LAN or WAN using MQSeries messages.

While client/server applications require real-time communication of a largely synchronous nature, programmers could still overlap the GUI processing, thus partially exploiting the asynchronous nature of MQSeries. The connection needs to be reliable since (in most cases) the program will not be able to perform any work unless messages are flowing properly.

It's quite likely that a system built with MQSeries and intermediate Windows NT class servers will not achieve the reliability of the back-end CICS system (which may typically deliver 99.75% or better availability). This is because such systems use new, sometimes unproven technology, introduce new points of failure, involve more physical servers, and require a long code-path. Therefore using MQSeries instead of direct APPC or TCP/IP connections could impact the reliability of a critical client/server system.

It is also possible that, unless the right options are selected, the performance in terms of end-to-end response time could be worse using MQSeries than is acceptable in an on-line client/server application. Hints for avoiding this are provided later in this article.

Finally, the plethora of MQI options means that the application programmers should follow strict guidelines, or use a standard wrapper to ensure that appropriate MQI options are chosen.

THE PROBLEM OF INTERMEDIATE SERVER UNRELIABILITY

Despite the marketing power of Microsoft, the fact remains that Windows NT servers do not approach the reliability of OS/390 systems. While an extremely well-tuned Unix system may come close to mainframe reliability, many mainframe-based large enterprises don't have experience of making Unix highly resilient. In addition, the up-front cost of such a Unix system is often prohibitive at the start of a new project.

There are many MQSeries system management packages on the market, an example of an excellent one being MQControl from Nastel Technologies. However, these products are really meant for addressing the symptoms of problems rather than their cause, despite the fact that they're very useful for administration. My advice is to avoid distributed

MQSeries servers except where they are essential to the application and client/server processing is rarely carried out on any of them.

MQSeries version 1.2 on OS/390 allows the direct attachment of MQSeries clients via TCP/IP (which is my recommendation) or APPC. By using direct attachment one can achieve much greater levels of reliability. IBM is addressing the reliability issue in other ways, and MQSeries 5.1 will permit the clustering of distributed servers, thus providing an alternative (albeit complex) solution.

While some administrators may be concerned about the effect of directly attached clients on OS/390 CPU consumption, our tests have shown that the overhead is actually similar to that of using server channels. Note that the client attach feature is a chargeable option on the OS/390 version of MQSeries.

Some other benefits of direct attachment to OS/390 are listed below.

- Savings in the purchase of Distributed Server, including licence costs and costs associated with deployment, OR and DR, etc.
- Savings in the cost of supporting Distributed Server (skills, time required, etc).
- Savings in the cost of Distributed Server system management software (which may no longer be required).
- Easier Distributed Server performance management and tuning (which may not be required).

ISSUES CONCERNING DIRECT ATTACHMENT TO OS/390

Below is a list of some of the issues that you may like to consider before deciding to implement direct attachment to OS/390.

- The number of reply queues must be considered.

  It is necessary to decide whether to use one reply queue on the MVS queue manager for all clients or a number of reply queues (or, alternatively, to use dynamic reply queues, one per PC).
- The number of concurrent client connections is limited.

There are limits in MVS to how many direct connections can be accommodated per queue manager. These limits are related to the storage available in the channel initiator address space. This restriction may be alleviated by applying the PTF for APAR PQ06157, though you should still expect a limit of around 10,000 connections.

In addition, the SYSTEM.CHANNEL.SYNC queue needs the INDEX(MSGID) option to avoid an exponential increase in the costs in terms of CPU usage for a large number of client channels. Paging space should be checked as 150 KB of EPVT region is used by each channel.

It is important to consolidate connections by ensuring that all MQSeries applications on a given PC share the same MQSeries connection handle, if at all possible.

- Ensuring that client applications support direct attachment.

  1. Use 'get message' and 'put message' options to specify whether syncpointing is required, as the default syncpointing option varies from one platform to another.

  2. All programs that issue MQGETs should specify the CONVERT option to ensure messages are presented in the appropriate character set (ASCII or EBCDIC).

  3. The maximum message size in MVS is currently 4 MB, while that on NT is 100 MB.

HOW TO AVOID END-TO-END PERFORMANCE PROBLEMS

MQSeries is surprisingly fast for both direct and indirect attachment, and end-to-end message performance should not be a problem (even if sub-second response is required) provided that some key recommendations are followed:

- Use non-persistent, 'fast' messages for everything other than database updates.

  To avoid the severe performance overhead associated with persistent messages, it is recommended that such messages are not used, with the exception of updating host information or where the application has no error recovery. This avoids most disk logging associated with message traffic, which is often the worst performance bottleneck.

- Use ‘fast’ mode channels where both client and server ends support them (MQSeries 5).

  This is a new performance feature in MQSeries Version 5 that applies to non-persistent messages. Fast mode channels result in further performance improvements by eliminating all disk logging.

- Reduce the number of messages passed to and from MQSeries for a given transaction.

  The processing of each message by MQSeries code includes a fixed overhead. Hence, consolidating traffic into fewer, bigger messages generally improves performance. However, it’s important not to overdo this.

- Time-out values should be stored in parameters so they can be changed easily by applications.

  It is necessary for the application to be able to decide when to ‘give up’ waiting for a response from MQSeries. For this reason, I recommend that time-out intervals are not hard-coded.

- Message expiry intervals should be used.

  Messages can be left on queues and not retrieved for a number of reasons, one being that the client ‘times out’ before the host is able to respond. To avoid the growth of ‘orphaned’ messages, specify a message expiry interval so that these messages are deleted.

- When reading messages from the queue, the message back-out count should be inspected.

  This prevents ‘abend loops’ where an errant message causes a program error again and again. Messages with a back-out count of more than one should be sent to a ‘poison letter’ queue.

- Use TCP/IP as the network protocol if the underlying network is IP router-based.

This will vary by installation, but in most cases TCP/IP is the most efficient protocol to use.

WHAT TO DO AT THE CICS ‘BACK-END’

The subject of processing MQSeries messages in CICS is a large one, which I intend to explore in future articles. However, one misconception I commonly encounter concerns the need for triggering.

Triggering is mentioned so often in relation to MQSeries that many people believe it is the normal way to deal with MQSeries messages. This may be true in a ‘classic’ asynchronous design, where a message may turn up at odd times and require an application to be started to process it. However, in a normal high-volume client/server environment, the CICS application is always running and should *not* be triggered by the arrival of the message for reasons of performance.

Triggering is an inefficient process that results in additional internal MQPUTs and MQGETs to and from the initiation queue. This means that the CICS/MQSeries overhead for each client/server request is more than doubled for no good reason.

The best set-up is to keep your CICS application running all the time and to keep an outstanding MQGET on the request queue. This means that, when a client/server request arrives, the outstanding MQGET is satisfied and the CICS program receives the data in a single MQI call. You may want the initial CICS program to perform a ‘request broker’ type function and route the data to another transaction – you can use the CICS comms area for this purpose. The original transaction can then immediately proceed to receive the next message, and so on. You may want more than one of these outstanding MQGETs, or even multiple CICS regions, for dealing with really high volumes of traffic.

When you examine your CICS MQSeries logic, it’s important to bear in mind that you may need to handle a large number of messages per second, so it’s important to ensure that you are not building in a bottleneck and that you provide proper user-oriented security by running the end-application transaction using the end-user’s RACF authority, where appropriate. I intend to explain this in more detail in a later article.

SOME RECOMMENDED MQI OPTIONS

In many instances, the decision may be taken to make all application programmers use a simple API ‘wrapper’ to shield them from the large array of MQI options and prevent them from choosing wrong ones. This is another subject that I intend to explore in a later article, but for now here are some essential ones to include in your standards.

MQOPEN

```
MQOO_INPUT_SHARED          for input queues
MQOO_OUTPUT                for output queues
MQOO_FAIL_IF_QUIESCING
```

MQGET

```
MQGMO_WAIT
MQGMO_NO_SYNCPOINT         or          MQGMO_SYNCPOINT
MQGMO_CONVERT
MQGMO_FAIL_IF_QUIESCING
```

MQPUT

```
MQPMO_NO_SYNCPOINT         or          MQPMO_SYNCPOINT
MQPMO_FAIL_IF_QUIESCING
```

DESCRIPTOR – MQMD

Set up message type fields, for example ‘MQMT_REQUEST’ and ‘MQMT_REPLY’. The reply-to-queue should be set on requests, while on replies this field specifies the appropriate queue to open. The format should be set to ‘MQFMT_STRING’ to enable ASCII to EBCDIC conversion.

MESSAGE HEADERS

I recommend that a standard installation header is placed at the front of all messages. This should include such fields as the application name, origin user-id, the function of the message, and so on. This aids debugging and message routing in the event of errors. You should define the installation header at the outset and, as with MQI options, the best way to enforce this standard is by using an API wrapper layer.

CONCLUSION

This article provides critical information for making successful use of MQSeries in a high-volume, client/server environment. My experience suggests that many new MQSeries applications are not truly asynchronous but have a synchronous element, and that client/server applications are the main reason for this trend.

The suggestion that directly attached MQSeries clients should be used for critical, real-time client/server applications is, perhaps, radical, but it does avoid most of the potential and inevitable problems inherent with using MQSeries in this way. It also reduces support effort and costs.

Adding System Management adds further complexity to your set-up, so this step should be undertaken only to add value, and not to overcome an unsuitable middleware architecture. An alternative to this approach may be to implement MQSeries 5.1 once it becomes available on all required platforms (assuming that it lives up to its promise of better distributed MQSeries server administration and operational reliability).

However, those sites that currently use a two-tier (in a physical sense) client/server architecture and wish to standardize on MQSeries APIs may still prefer to use direct attachment.

Peter A Toogood
MQSeries Solutions Expert and Developer (UK)

Closing the holes in MQ security

In choosing the default settings for MQSeries, IBM has had to strike a balance between making the product easy to use as quickly as possible and making it secure straight out of the box. In more recent releases, it has put more emphasis on ease of use and so relaxed the default security settings. This is one of the reasons why administrators must now reconfigure their systems if they require them to be secure. This article examines some of the potential security holes of which administrators should be aware, and also describes ways in which administrators can close these holes.

DEFAULT CHANNEL DEFINITIONS

There are a number of objects, such as SYSTEM.DEF.SVRCONN and SYSTEM.DEFAULT.LOCAL.QUEUE, that are created by default when you install and configure a queue manager. These are really intended only as definitions to be cloned for their default attributes in the creation of new objects. However, a potential infiltrator can exploit the fact that they are also well-defined objects that probably exist on your system.

Originally, on distributed platforms, the definition of channel SYSTEM.DEF.SVRCONN had its MCAUSER parameter set to 'nobody'. IBM had so many complaints from users who couldn't get clients connected that it has now changed this parameter to blank (' ').

The MCAUSER parameter specifies the userid that is checked when an inbound message is put on a queue. Setting this field to blank means that the authority of the userid running the channel (usually 'mqm') is checked. In other words, messages are always authorized to be put on all queues.

The thinking behind putting 'nobody' in this field is that no one should be allowed to put messages on queues unless the administrator actually changes settings to allow them to do so. Unfortunately this default setting was not documented and so users could not work out how they were required to change things.

There are many users who don't need client channels and so haven't even read this section of the manual. They're unaware that nowadays, with default settings in place, anyone who can connect to their machine (for instance, someone on the same LAN) can start a client channel to them called SYSTEM.DEF.SVRCONN and have access to put messages on any of their queues and – often more importantly – to get messages from any of their queues.

This is not an entirely new problem – even the original systems suffered from it, as there are other channels, such as SYSTEM.DEF.RECEIVER and SYSTEM.DEF.REQUESTER, that have always had a blank MCAUSER. With a little effort, users have always been able to connect to these and put messages on queues using full authority. If the queue manager is the default one, the infiltrator needs no prior knowledge of the system.

As previously mentioned, these definitions are used to provide defaults for the creation of new channels. This means that, in many systems, newly created channels also have MCAUSER set to blank.

It is recommended that the following commands be executed using RUNMQSC to close this loophole (note the use of the continuation character, '➤', in the code below to show that one line of code maps to more than one line of print):

```
alter chl(SYSTEM.DEF.SVRCONN) chltype(SVRCONN) trptype(LU62) +
➤  Mcauser(NOBODY)
alter chl(SYSTEM.DEF.RECEIVER) chltype(RCVR) trptype(LU62) +
➤  Mcauser(NOBODY)
alter chl(SYSTEM.DEF.REQUESTER) chltype(RQSTR) trptype(LU62) +
➤  Mcauser(NOBODY)
...
```

DO NOT START MQ USING ROOT

It's worth noting that much of this section is described in Unix terms, though it's applicable to most platforms, once Unix terms are replaced by their equivalents.

All MQSeries components should be started using the MQSeries administration userid (*mqm*). Many system administrators like to make the system administration userid (*root*) a member of the *mqm* group. This is understandable, as all administrative commands, not all of which are for MQ, can be run as *root*. However, this is a very dangerous thing for them to do as they are effectively giving *root* authority to all of the members of the *mqm* group.

For example, if the trigger monitor of the default queue manager is started by *root* using default parameters, a member of the *mqm* group whose workstation has IP address ‘myhost’ can enter the following commands using RUNMQSC:

```
DEFINE QL(MYQUEUE) TRIGGER PROCESS(MYPROCESS) +
INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE)
DEFINE PROCESS(MYPROCESS) APPLICID('xterm -display myhost:0 &')
```

and then enter the command:

```
echo hello | amqsput MYQUEUE
```

This causes a terminal to appear on their screen giving them a command line with *root* authority from which they have full control of the system.

Similarly, if a channel is started by *root*, or the channel initiator starts a channel and the channel initiator is started by *root*, then any exits called by the channel will run as *root*. So the *mqm* member could write and install an exit that again spawns a *root*-authorized xterm.

The receiver channel could have the same problems, for example, if started as *root* by the listener, inetd, or Communications Manager. A good start to overcoming this problem is to remove *root* from the *mqm* group. However, on some systems *root* will still have access to the **strmqm** command and, while it may look as though it has started the queue manager, there may be unexpected errors later when it performs commands for which the OAM checks authority.

The system administrator may find it useful to create commands that only *root* is authorized to run, which switch to the *mqm* userid before performing the instruction. For example the following shell script could be called **strmqm** and put higher in *root*’s path than the real **strmqm**.

```
#!/bin/ksh
su - mqm -c /usr/lpp/mqm/bin/strmqm $1
```

ONLY USE GROUPS ON UNIX OAM

The **setmqaut** command is used to set access to MQSeries objects. Among its parameters you may specify ‘-p PrincipalName’ or ‘-g GroupName’ to indicate to which users you intend this command to apply.

For example, the following command specifies that all members of the group *tango* are to be allowed to put messages on queue *orange.queue* on queue manager *saturn.queue.manager*

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
➤  -g tango +put
```

Similarly, the command:

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
➤  -p theuser +put
```

specifies that the userid *theuser* should be allowed to put messages on queue *orange.queue* on queue manager *saturn.queue.manager*. On most platforms this works fine. However, the implementation on Unix systems:

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
➤  -p theuser +put
```

specifies that all of the members of *theuser*'s primary group are allowed to put messages on queue *orange.queue* on queue manager *saturn.queue.manager.*

This can be very dangerous, as a system administrator can give access to a particular user unaware that in doing so he has accidentally also given access to many other users. User *theuser* may also be unhappy to be blamed by administrators for actions that they believe only he is authorized to have carried out.

The way around this problem is never to use the '-p' parameter on Unix. The same effect can be obtained by specifying '-g PrimaryGroup', which is a lot clearer.

ONLY CREATE OBJECTS AS MQM ON UNIX

As described above, MQSeries on Unix does all of its security using the primary group of a userid rather than the userid itself, as you would expect. This has other knock-on effects.

When a queue is created, access to it is automatically granted to the *mqm* group and to the primary group of the userid that created it. It's quite reasonable for someone designing the security of an MQSeries infrastructure to assume that access to all queues has been forbidden to all users except members of the *mqm* group. From here, the administrator specifies additional security settings that are needed.

This works fine when queues are created either by the *mqm* user or by someone whose primary group is *mqm*. The problem arises when another user whose primary group is, for instance, *staff*, but who is also a member of *mqm*, defines the queue. In this case authority is also granted automatically and unintentionally to all members of the *staff* group.

This also applies to the creation of queue managers. If a queue manager is created by a userid whose primary group is *staff*, then all members of *staff* by default have access to the queue manager.

The simplest solution to this problem is to enforce a policy whereby no userid other than *mqm* may create MQSeries objects or queue managers. An alternative policy is never to make a userid a member of the *mqm* group unless this is its primary group.

OAM USES UNION

The Object Authority Manager uses the union of the authority settings that it finds. So, to take the example above a step further, suppose a queue, *orange.queue*, is created by a userid whose primary group is *staff*. At some point later it is found that another userid, *worker*, who shouldn't have access to the queue, is nevertheless able to access it. *worker* is a member of *staff* but has *team* as his primary group. To resolve this problem an administrator might try running:

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
➤  -p worker -all
```

However, this will not solve the problem. While it will remove *team* from the authorization list, members of *staff*, including *worker*, still have access to the queue.

This also applies to other platforms, such as NT, that implement the '-p' parameter. Although the problem of primary groups is not present, it should be realized that, while:

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
➤  -p worker +all
```

gives full access to *worker*,

```
    setmqaut -m saturn.queue.manager -n orange.queue -t queue
    ➤  -p worker -all
```

only forbids all access if *worker* is not a member of any authorized groups.

CACHING

On some platforms, such as Unix, group membership is cached by MQSeries. This means that, if a new user joins a group and needs access to MQSeries objects, the queue manager needs to be restarted. Similarly (and probably more importantly), if a user leaves the team or company, it is not sufficient just to remove them from the group. The user retains access to objects until such time as the queue manager is restarted.

ONLY ENABLE THINGS IF YOU NEED THEM

This is no more than common sense, and the defaults are such that this won't cause problems, but for the sake of completeness the following points are worth mentioning:

- *Automatic channel definition*
  Enabling the automatic definition of channels increases the ability of machines to connect to your queue manager with little prior knowledge of your system, so this should be enabled only if definitely required.
- *Command server*
  The command server is very powerful and can render weak security even weaker. For instance, on a system running MQSeries version 2 in which users do not have the authority to use the client channel, they could still connect using a sender channel called SYSTEM.DEF.RECEIVER. This could put messages on the command server's input queue requesting it to create a channel and transmission queue back out. This could then be used for further breaches of security. If you're not confident of your system's security, it's advisable to start the command server only

when it is needed and to grant users only the minimum required levels of authority to it.

Sam Garforth
SJG Consulting Ltd (UK) © S Garforth 1999

MQSeries for MVS and TCP/IP

MQSeries for MVS is supplied with built-in LU 6.2 support to enable it to communicate with such remote platforms as MQSeries on OS/2. Further customization is necessary, however, if the same functionality is required using TCP/IP. All relevant information about this is available in the program directory – please refer to dataset MQM.SCSQINST, member CSQ8DQM3, and make the necessary TCP/IP-related changes.

NOT DOCUMENTED

TCP/IP DDNAMEs and dataset references need to be added to the MQxxCHIN procedure (STC). On SYSD, the following statements are added:

```
//PROFILE   DD DSN=.PROFILE(PROFILE),DISP=SHR and
//SYSTCPD   DD DSN=TCPIP.TCPPARMS(TCPDATA),DISP=SHR
```

Note that, once TCP/IP is defined as the communication protocol on MQSeries for MVS, all relevant CHANNELS, etc, will become inactive in the event of TCP/IP being de-activated. These then have to be RESET or, alternatively, task MQxxCHIN needs to be stopped and then re-started.

In the event that maintenance is carried out to MQSeries and to products with which it has SMP/E cross dependencies (and which it cross references, such as LE/370 and TCP/IP), then CALLIB job CSQ8CLIB in MQM.SCSQINST has to be executed.

Saida Davies (UK) © Xephon 1999

Recovery procedures

This month’s instalment concludes this article on MQSeries recovery procedures (the first part appeared in last month’s *MQ Update)*.

RECARCH

This job depends on the successful completion of job PRTBSDS.

This utility records information about the archive log in the BSDSs. If the active log files were added or deleted since the last operation to recover BSDSs from the archive log, then the changes have to be reflected in the replaced BSDS.

Use the DELETE function to remove additional log file information and the NEWLOG function if a new log file needs to be added.

RECARCH

```
// JOBID JOB CLASS=S,NOTIFY=&SYSUID
//*
//*******************************************************************
//* From job PRTBSDS
//*******************************************************************
//* Record information about the archive log in the BSDSs.
//*
//* If active log files were added or deleted since BSDS recovery
//* from the ARCHIVE LOG, then changes have to be reflected in the
//* replaced BSDS.
//*
//* Use:
//*   DELETE to remove additional LOG FILE information
//*   NEWLOG to add a new LOG FILE.
//*******************************************************************
//*
//JU003    EXEC PGM=CSQJU003
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1   DD DISP=OLD,DSN=SYS1.MQS.BSDS01
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSIN    DD *
 NEWLOG DSNAME=MQARCH.LOG1.xxxxxx.xxxxxxxx.Bxxxxxxx
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS01.COPY1
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS02.COPY1
/*
//*****************************************************************
//* Go to CSQ1LOGP
//*****************************************************************
```

RECARCH1

This job depends on the successful completion of job CSQ1LOGP.

Compare the RBA detail listed in job CSQ1LOGP with the RBA detail listed in job PRTBSDS.

If there is a mismatch, then use the DELETE function to remove additional log file information from each active log dataset in the inventory of the replacement BSDS, and use the NEWLOG function to add new log files.

RECARCH1

```
//JOBID JOB CLASS=S,NOTIFY=&SYSUID
//*
//*****************************************************************
//* From job CSQ1LOGP
//*****************************************************************
//* Compare the RBA detail listed in job PRTBSDS with RBA detail in
//* CSQ1LOGP.
//*
//* If there is a mismatch, then use the DELETE function on each
//* active log to remove additional information or the NEWLOG
//* function to add the inventory in the replacement BSDS.
//*
//* Edit the JCL wherever necessary
//*****************************************************************
//*
//JU003    EXEC PGM=CSQJU003
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1   DD DISP=OLD,DSN=SYS1.MQS.BSDS01
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSIN    DD *
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS01,COPY1
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS02,COPY1
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS01,COPY2
 DELETE DSNAME=SYS1.MQS.LOGCOPY3.DS02,COPY2
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY4.DS01,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY4.DS02,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY4.DS01,COPY2
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY4.DS02,COPY2
/*
//*****************************************************************
//* Go to RECLOG3
//*****************************************************************
```

RECLOG1

In case one active log is corrupt or lost, recover from another log file file. Delete, redefine, and copy the affected file.

RECLOG1

```
//JOBID JOB  CLASS=S,NOTIFY=&SYSUID
//*
//*****************************************************************
//* Recover one lost active log from another log file
//*****************************************************************
//*****************************************************************
//* Delete corrupt log file, redefine and copy
//*
//* Edit the JCL wherever necessary
//*****************************************************************
//*
//RECLOG1   EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//DD1       DD DISP=SHR,DSN=SYS1.MQS.LOGCOPY2.DS02
//DD2       DD DISP=SHR,DSN=SYS1.MQS.LOGCOPY1.DS02
//SYSIN     DD *
    DELETE (SYS1.MQS.LOGCOPY1.DS02) ERASE CLUSTER
    SET MAXCC = 0

    DEFINE CLUSTER                                            -
        (NAME (SYS1.MQS.LOGCOPY1.DS02)                        -
        LINEAR                                                -
        VOLUMES(SYS000)                                       -
        RECORDS(36750))                                       -
    DATA                                                      -
        (NAME(SYS1.MQS.LOGCOPY1.DS02.DATA))

    REPRO                                                     -
        INFILE(DD1)                                           -
        OUTFILE(DD2)
/*
//*
```

RECLOG1N

In case one active log is corrupt or lost, recover from another log file. DEFINE NEW and COPY.

RECLOG1N

```
//JOBID JOB  CLASS=S,NOTIFY=&SYSUID
//*
//******************************************************************
//* Recover an active log that's lost from another log file.
//******************************************************************
//******************************************************************
//* Define new log file and copy it.
//*
//* Edit the JCL wherever necessary.
//******************************************************************
//*
//RECLOG1   EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//DD1       DD DISP=SHR,DSN=SYS1.MQS.LOGCOPY2.DS02
//DD2       DD DISP=SHR,DSN=SYS1.MQS.LOGCOPY3.DS02
//SYSIN     DD *
    DEFINE CLUSTER                                           -
        (NAME (SYS1.MQS.LOGCOPY3.DS02)                       -
        LINEAR                                               -
        VOLUMES(SYS000)                                      -
        RECORDS(36750))                                      -
    DATA                                                     -
        (NAME(SYS1.MQS.LOGCOPY3.DS02.DATA))

    REPRO                                                    -
        INFILE(DD1)                                          -
        OUTFILE(DD2)
/*
//*
//******************************************************************
//* Delete old log file and record information on new log file in the
//* BSDS.
//*
//* Amend LOG FILE reference in the MQ start-up procedure
//******************************************************************
//*
//JU003    EXEC PGM=CSQJU003
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1   DD DISP=OLD,DSN=SYS1.MQS.BSDS01
```

```
//SYSUT2   DD DISP=OLD,DSN=SYS1.MQS.BSDS02
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSIN    DD *
 DELETE DSNAME=SYS1.MQS.LOGCOPY1.DS02,COPY2
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY3.DS02,COPY2
/*
//
```

RECLOG2

If both active logs are corrupt or lost, this job recovers them using the page datasets. It is necessary to ensure that the QMGR task is not executing at the time. This job performs the following:

- Verify old datasets to ensure they were properly closed when MQ terminated. The datasets will close properly, despite the error messages.
- Define new page datasets, one for each existing one, and ensure they are larger than the old ones.

RECLOG2

```
//JOBID JOB  CLASS=S,NOTIFY=&SYSUID
//*
//*******************************************************************
//* Loss of both active logs - recover from Page datasets.
//*******************************************************************
//* Verify old datasets to ensure they closed properly when MQ
//* terminated.
//*
//* This will close datasets properly even in the event of errors.
//*
//* Edit the JCL wherever necessary.
//*******************************************************************
//*
//VERIFY   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   VERIFY DATASET(SYS1.MQS.PSID00)
   VERIFY DATASET(SYS1.MQS.PSID01)
   VERIFY DATASET(SYS1.MQS.PSID02)
   VERIFY DATASET(SYS1.MQS.PSID03)
/*
//*
//*******************************************************************
//* Ensure QMGR task is not executing.
```

```
//*
//* Ensure new datasets are larger than the old.
//*
//* Define a new page dataset to correspond to each existing one.
//*******************************************************************
//*
//DEFINE    EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
    DEFINE CLUSTER                                               -
        (NAME(SYS1.MQS.NEW.PSID00)                               -
        RECORDS(50000 2500)                                      -
        LINEAR                                                   -
        VOLUMES(SYS000)                                          -
        SHAREOPTIONS(2 3))                                       -
    DATA                                                         -
        (NAME(SYS1.MQS.NEW.PSID00.DATA))

    DEFINE CLUSTER                                               -
        (NAME(SYS1.MQS.NEW.PSID01)                               -
        RECORDS(50000 2500)                                      -
        LINEAR                                                   -
        VOLUMES(SYS000)                                          -
        SHAREOPTIONS(2 3))                                       -
    DATA                                                         -
        (NAME(SYS1.MQS.NEW.PSID01.DATA))

    DEFINE CLUSTER                                               -
        (NAME(SYS1.MQS.NEW.PSID02)                               -
        RECORDS(50000 2500)                                      -
        LINEAR                                                   -
        VOLUMES(SYS000)                                          -
        SHAREOPTIONS(2 3))                                       -
    DATA                                                         -
        (NAME(SYS1.MQS.NEW.PSID02.DATA))

    DEFINE CLUSTER                                               -
        (NAME(SYS1.MQS.NEW.PSID03)                               -
        RECORDS(50000 2500)                                      -
        LINEAR                                                   -
        VOLUMES(SYS000)                                          -
        SHAREOPTIONS(2 3))                                       -
    DATA                                                         -
(NAME(SYS1.MQS.NEW.PSID03.DATA))
/*
//*
//*******************************************************************
//* FORMAT and RESETPAGE new datasets.
//*
//* Use RESET function to generate consistent page sets to be used
//* with clean BSDS and log datasets.
//*******************************************************************
//* Note that MQ cannot restart if page set zero is not available.
//*
//* Amend page set reference in MQ start-up procedure.
//*******************************************************************
//*
//RESET    EXEC PGM=CSQUTIL
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//CSQP0000 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID00
//CSQP0001 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID01
//CSQP0002 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID02
//CSQP0003 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID03
//CSQS0000 DD DISP=OLD,DSN=SYS1.MQS.PSID00
//CSQS0001 DD DISP=OLD,DSN=SYS1.MQS.PSID01
//CSQS0002 DD DISP=OLD,DSN=SYS1.MQS.PSID02
//CSQS0003 DD DISP=OLD,DSN=SYS1.MQS.PSID03
//CSQT0000 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID00
//CSQT0001 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID01
//CSQT0002 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID02
//CSQT0003 DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID03
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
FORMAT
RESETPAGE
/*
//*******************************************************************
//* Delete and redefine log files and BSDS
//*******************************************************************
//*
//RECLOG2   EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
   DELETE (SYS1.MQS.BSDS01)        ERASE CLUSTER
   DELETE (SYS1.MQS.BSDS02)        ERASE CLUSTER
   DELETE (SYS1.MQS.LOGCOPY1.DS01) ERASE CLUSTER
   DELETE (SYS1.MQS.LOGCOPY1.DS02) ERASE CLUSTER
   DELETE (SYS1.MQS.LOGCOPY2.DS01) ERASE CLUSTER
   DELETE (SYS1.MQS.LOGCOPY2.DS02) ERASE CLUSTER
    SET MAXCC = 0
    DEFINE CLUSTER                                         -
        (NAME(SYS1.MQS.BSDS01)                             -
        VOLUMES(SYS000)                                    -
        SHAREOPTIONS(2 3))                                 -
    DATA                                                   -
        (NAME(SYS1.MQS.BSDS01.DATA)                        -
        RECORDS(3000 3000)                                 -
        RECORDSIZE(4089 4089)                              -
        CONTROLINTERVALSIZE(4096)                              -
        FREESPACE(0 20)                                        -
        KEYS(4 0))                                             -
    INDEX                                                      -
        (NAME(SYS1.MQS.BSDS01.INDEX)                           -
        RECORDS(5 5)                                           -
    CONTROLINTERVALSIZE(1024))

    DEFINE CLUSTER                                             -
        (NAME(SYS1.MQS.BSDS02)                                 -
        VOLUMES(SYS000)                                        -
        SHAREOPTIONS(2 3))                                     -
    DATA                                                       -
        (NAME(SYS1.MQS.BSDS02.DATA)                            -
        RECORDS(3000 3000)                                     -
        RECORDSIZE(4089 4089)                                  -
        CONTROLINTERVALSIZE(4096)                              -
        FREESPACE(0 20)                                        -
        KEYS(4 0))                                             -
    INDEX                                                      -
        (NAME(SYS1.MQS.BSDS02.INDEX)                           -
        RECORDS(5 5)                                           -
        CONTROLINTERVALSIZE(1024))

    DEFINE CLUSTER                                             -
        (NAME (SYS1.MQS.LOGCOPY1.DS01)                         -
        LINEAR                                                 -
        VOLUMES(SYS000)                                        -
        RECORDS(36750))                                        -
    DATA                                                       -
     (NAME(SYS1.MQS.LOGCOPY1.DS01.DATA))

    DEFINE CLUSTER                                             -
        (NAME (SYS1.MQS.LOGCOPY1.DS02)                         -
        LINEAR                                                 -
        VOLUMES(SYS000)                                        -
        RECORDS(36750))                                        -
    DATA                                                       -
        (NAME(SYS1.MQS.LOGCOPY1.DS02.DATA))

    DEFINE CLUSTER                                             -
        (NAME (SYS1.MQS.LOGCOPY2.DS01)                         -
        LINEAR                                                 -
        VOLUMES(SYS000)                                        -
        RECORDS(36750))                                        -
    DATA                                                       -
        (NAME(SYS1.MQS.LOGCOPY2.DS01.DATA))

DEFINE CLUSTER                                                 -
        (NAME (SYS1.MQS.LOGCOPY2.DS02)                         -
         LINEAR                                                -
         VOLUMES(SYS000)                                       -
         RECORDS(36750))                                       -
    DATA                                                       -
    (NAME(SYS1.MQS.LOGCOPY2.DS02.DATA))
/*
//*
//*****************************************************************
//* Record information of the redefined LOG file in the BSDS
//*****************************************************************
//*
//JU003    EXEC PGM=CSQJU003
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1   DD DISP=OLD,DSN=SYS1.MQS.BSDS01
//SYSUT2   DD DISP=OLD,DSN=SYS1.MQS.BSDS02
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSIN    DD *
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY1.DS01,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY1.DS02,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY2.DS01,COPY2
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY2.DS02,COPY2
/*
//
```

RECLOG3

This job depends on the successful completion of RECARCH1.

If only two active log datasets are listed for each copy of the active log, MQ may still not restart. In this case add a new active log file for each copy of the active log and define the replacement in the BSDS.

RECLOG3

```
//JOBID JOB CLASS=S,NOTIFY=&SYSUID
//*
//*****************************************************************
//* From job RECARCH1
//*****************************************************************
//* If only two active log datasets are listed for each copy of
//* the active log, MQ may still not restart.
//*
//* In this case, add a new active log file for each copy of the
//* active log and define the replacement in the BSDS.
//*
```

```
//* Edit the JCL wherever necessary.
//*******************************************************************
//*
//RECLOG2  EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
    DEFINE CLUSTER                                                -
        (NAME (SYS1.MQS.LOGCOPY1.DS01)                            -
        LINEAR                                                    -
        VOLUMES(SYS000)                                           -
        RECORDS(36750))                                           -
    DATA                                                          -
        (NAME(SYS1.MQS.LOGCOPY1.DS01.DATA))

    DEFINE CLUSTER                                                -
        (NAME (SYS1.MQS.LOGCOPY1.DS02)                            -
        LINEAR                                                    -
        VOLUMES(SYS000)                                           -
        RECORDS(36750))                                           -
    DATA                                                          -
        (NAME(SYS1.MQS.LOGCOPY1.DS02.DATA))

    DEFINE CLUSTER                                                -
        (NAME (SYS1.MQS.LOGCOPY2.DS01)                            -
        LINEAR                                                    -
        VOLUMES(SYS000)                                           -
        RECORDS(36750))                                           -
    DATA                                                          -
        (NAME(SYS1.MQS.LOGCOPY2.DS01.DATA))

    DEFINE CLUSTER                                                -
        (NAME (SYS1.MQS.LOGCOPY2.DS02)                            -
        LINEAR                                                    -
        VOLUMES(SYS000)                                           -
        RECORDS(36750))                                           -
    DATA                                                          -
        (NAME(SYS1.MQS.LOGCOPY2.DS02.DATA))

/*
//*******************************************************************
//* Record information on the newly defined log file in the BSDS.
//*******************************************************************
//*
//JU003    EXEC PGM=CSQJU003
//STEPLIB  DD DSN=SYS1.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.SCSQSNLE,DISP=SHR
//         DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1   DD DISP=OLD,DSN=SYS1.MQS.BSDS01
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
```

```
//SYSIN    DD *
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY1.DS01,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY1.DS02,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY2.DS01,COPY2
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY2.DS02,COPY2
/*
//******************************************************************
//* Go to RCBSDS3
//******************************************************************
```

RECNLOG

Define additional log datasets and then record information about them in the BSDS.

RECNLOG

```
//JOBID JOB CLASS=S,NOTIFY=&SYSUID
//*
//******************************************************************
//* Define new log datasets
//******************************************************************
//*
    DEFINE CLUSTER                                              -
        (NAME (SYS1.MQS.LOGCOPY3.DS01)                          -
        LINEAR                                                  -
        VOLUMES(SYS000)                                         -
        RECORDS(36750))                                         -
    DATA                                                        -
        (NAME(SYS1.MQS.LOGCOPY3.DS01.DATA))
/*
//*
//******************************************************************
//* Record information about new log datasets in the BSDSs
//******************************************************************
//*
//JU003     EXEC PGM=CSQJU003
//STEPLIB   DD DSN=SYS1.SCSQANLE,DISP=SHR
//          DD DSN=SYS1.SCSQSNLE,DISP=SHR
//          DD DSN=SYS1.SCSQAUTH,DISP=SHR
//SYSUT1    DD DISP=OLD,DSN=SYS1.MQS.BSDS01
//SYSUT2    DD DISP=OLD,DSN=SYS1.MQS.BSDS02
//SYSPRINT  DD SYSOUT=*,DCB=BLKSIZE=629
//SYSIN     DD *
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY3.DS01,COPY1
 NEWLOG DSNAME=SYS1.MQS.LOGCOPY3.DS02,COPY1
/*
```

RECPAG1

Define new page datasets to replace any existing ones that may be corrupted.

RECPAG1

```
//JOBID JOB CLASS=S,NOTIFY=&SYSUID
//*
//*******************************************************************
//* Define new page sets if any existing ones are corrupted.
//*
//* Edit the JCL accordingly wherever necessary.
//*******************************************************************
//*
    DEFINE CLUSTER                                              -
        (NAME(SYS1.MQS.NEW.PSID00)                              -
        RECORDS(50000 2500)                                     -
        LINEAR                                                  -
        VOLUMES(SYS000)                                         -
        SHAREOPTIONS(2 3))                                      -
    DATA                                                        -
        (NAME(SYS1.MQS.NEW.PSID00.DATA))

    DEFINE CLUSTER                                              -
        (NAME(SYS1.MQS.NEW.PSID01)                              -
        RECORDS(50000 2500)                                     -
        LINEAR                                                  -
        VOLUMES(SYS000)                                         -
        SHAREOPTIONS(2 3))                                      -
    DATA                                                        -
        (NAME(SYS1.MQS.NEW.PSID01.DATA))

    DEFINE CLUSTER                                              -
        (NAME(SYS1.MQS.NEW.PSID02)                              -
        RECORDS(50000 2500)                                     -
        LINEAR                                                  -
        VOLUMES(SYS000)                                         -
        SHAREOPTIONS(2 3))                                      -
    DATA                                                        -
        (NAME(SYS1.MQS.NEW.PSID02.DATA))

    DEFINE CLUSTER                                              -
        (NAME(SYS1.MQS.NEW.PSID03)                              -
        RECORDS(50000 2500)                                     -
        LINEAR                                                  -
        VOLUMES(SYS000)                                         -
        SHAREOPTIONS(2 3))                                      -
    DATA                                                        -
```

```
        (NAME(SYS1.MQS.NEW.PSID03.DATA))
/*
//*
//*******************************************************************
//* FORMAT new page sets
//*******************************************************************
//*
//CSQUTIL EXEC PGM=CSQUTIL
//STEPLIB   DD DSN=SYS1.SCSQANLE,DISP=SHR
//CSQP0000  DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID00
//CSQP0001  DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID01
//CSQP0002  DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID02
//CSQP0003  DD DISP=OLD,DSN=SYS1.MQS.NEW.PSID03
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
FORMAT
/*
//*
//*******************************************************************
//* Recover from another page set.
//*
//* Amend page set reference in MQ start-up procedure.
//*******************************************************************
//*
//RECPAG1   EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//DD1       DD DISP=SHR,DSN=SYS1.MQS.PSID00
//DD2       DD DISP=SHR,DSN=SYS1.MQS.NEW.PSID01
//SYSIN     DD *

    REPRO                                                    -
        INFILE(DD1)                                          -
        OUTFILE(DD2)
/*
//
```

ORDER OF JOBS

The order in which jobs should be run to recover if both BSDSs are corrupted is listed below:

1 RCBSDS2

2 PRTBSDS

3 RECARCH

4 CSQ1LOGP

5 RECARCH1

6 RECLOG3

7 RCBSDS3.

Saida Davies (UK) © Xephon 1999

Stress-testing

INTRODUCTION

MQSeries has become an industry standard for robust messaging. Once an installation has got to grips with MQSeries' architectural implications, it is then necessary to understand how its use affects operations and, in particular, how it affects performance in a production environment. When planning a release of MQSeries-dependent software, it is necessary to gain an understanding of its requirement for server capacity and its impact on throughput. As the release of software under development approaches, stress-testing and performance tuning become important.

The two phases – planning a roll-out and stress-testing in the lead-up to implementation – can be tackled separately. MQSeries performance figures are freely available from IBM's Web site and from White Papers, while it's possible to stress test an application simply by running it repeatedly.

Alternatively, the two issues can be tackled together, and it is this approach that I intend to discuss in this article. While adopting this approach does not preclude the use of the above techniques, it does help to provide a fuller picture of performance and scalability. Firstly, while IBM's figures are undoubtedly accurate, it is difficult to extrapolate from them to your own environment. The performance constraints of MQSeries (for instance, those arising from disk IO performance or queue manager configuration) can be extremely difficult to predict. In my opinion, it is better to try out your own infrastructure as soon as possible. Secondly, when dealing with application performance, it is impractical to ask hundreds of users to test your system. It is even harder to do so under controlled conditions from which accurate conclusions may be drawn. Simulations of your system are usually a more useful tool.

What I present here is a working example of techniques similar to those I have found to be useful in my day-to-day work. It is only an example (for contractual reasons, I can't provide actual data) but, if extended, it should be useful to others.

ARCHITECTURE AND TESTING STRATEGIES

The strategy that I employ for testing is to mimic proposed systems. To do this, I need to understand the following factors:

1 What messages are to be sent (size and data)

2 The frequency of messaging

3 What response times are deemed to be acceptable

4 How many concurrent users are to be supported.

Once the above is known, it is possible to model the system on the proposed server/network infrastructure.

What I have produced is a simple module that does this. The system requires an array of messages to be specified (the messages represent the typical 'message life-cycle' of your proposed system). The system also requires pacing information to allow it to space the messages accurately. Also needed are acceptable response times (ie the delay in response from your server, via MQSeries, that has been deemed reasonable). Finally, the application requires the number of instances of the test model that are to run concurrently. Using this strategy, it is relatively straightforward to establish how response times degrade as a system approaches capacity.

With the test model defined, it is then possible to run tests under traced conditions. The application in this article measures response times, which is clearly useful to know. A more complete picture can be obtained using this in conjunction with system resource monitors, such as NT's Performance Monitor, Omegamon, or even network 'sniffers', depending on where your constraints may be.

From an MQSeries point of view, there are three basic queue configurations that are testable. It is possible to study these using the software in the code listings. My application simply puts messages on one queue and gets a 'reply' message from another queue (using the MsgId). Depending on the queues used, three scenarios are possible:

1 If your system is primarily composed of MQClients writing to and reading from queues, it is possible to put and get messages using a single local queue. This gives you an indication of the performance of your network and the number of clients that your queue manager can support.

2 If your system uses local queue managers to communicate with remote queue managers, or if it uses a 'hub and spoke' topology, channel throughput can be studied by writing to a remote queue on 'ServerA', which points to a remote queue on 'ServerB', which in turn points back to a local queue on 'ServerA'. The final local queue is the queue to be read by the stress tester.

3 Finally, if a server system is available (in whole or in part), it is possible to write messages to a queue that is serviced by your server software. Replies are then read in a way that closely approximates normal client/server applications.

IMPLEMENTING THE TESTING TOOLS

For reasons of object orientation, portability, and my own personal fondness for the language, I have coded my testing tools in Java. This has been achieved using four modest classes and one very simple interface. I have used the *com.ibm.mq* MQSeries client classes in this example as I have assumed that most people use MQSeries via clients. If direct connection to a queue manager is required, it is possible to substitute the *com.ibm.mq* package with the *com.ibm.mqbind* package, which maps all calls to the standard C API.

I should note that the performance and memory requirements of Java may be a concern in some installations. I am generally able to run between 50 to 100 instances of my test on a single Pentium II under NT without problems. If you run the queue manager on the same machine, it obviously reduces capacity. All in all, with just a handful of borrowed PCs (possibly running more than one operating system), it should be possible to test systems that are intended for thousands of users. Alternatively, a larger midrange system could be used to drive the tests. As it's usually easier to carry out performance tests out of peak user hours, when network traffic is less variable, unused machines are often available.

The code I have written is not full-featured (various MQSeries facilities have been omitted), and it also carries out very little data validation (I guessed that no-one would be interested in reading pages of validation rules). It should, therefore, either be used with care or enhanced according to your own requirements.

MQCONSTANTS

This class prompts the user for the runtime data constants to be used during the test. It uses a GUI built using Java Swing components to gather data, but makes no effort to validate the data. All fields are obligatory. The following data is gathered:

- Queue manager name.
- Queue manager address (either its TCP/IP address or name, as the Java client supports only IP).
- Client channel name (that is, the SVRCONN channel that's to be used).
- Request queue name (local or remote).
- Reply queue name (local only).
- Maximum message length (this is used only for client buffers, and shouldn't affect queue manager performance).
- Number of instances to run (from one upwards, depending on your machine's capacity).
- Number of repetitions (the number of times to loop through the message set).

- Number of messages.

Once this data is gathered, a second window is displayed prompting the user to enter the following array of message profile data, which makes up a message set:

- Message source, which is either a file name (that is, the message is generated from the contents of the file) or a message length. If you specify a message length, then a message of that length is generated that contains only spaces. This will lead to inaccurate results if compression exits, or SNA channels using SNA compression, are used.
- Acceptable time for message pair to complete, specified in milliseconds.
- Failure time (after which message is said to have failed, perhaps as a result of an over-stressed system) in milliseconds.
- Pacing time (again in milliseconds), the interval (excluding the time taken to generate and transfer messages) during which the test pauses before resuming messaging.

As a fairly sizeable amount of data is required to run the program, basic serialization is used to save users from repeatedly having to key in parameters. The serialized file (*MQConstants.ser*) can be copied to other systems only if identical Java versions are used. Once all data is captured, the interface *ConstantsListener* is used to notify other classes that they may continue (in this case, only *MQStressTester* is interested).

MQSTRESSTESTER

This class is the suite's control object. It instantiates other objects and controls the starting and stopping of tests. It uses a window with a basic response-time feedback area. Each instance of the test (each *MQStressThread*) has a visual component in this area, reporting on how it is progressing. To clarify the mass of numbers reported, colour coding is used as follows:

- *Green*
  The response time is less than the acceptable time.
- *Yellow*
  The response time is less than the failure time.
- *Red*
  The response time is greater than the failure time (or messaging has totally failed).

It's not my intention to produce a system for generating figures for statistical analysis (though the figures may be logged to disk by extending this system). The way I envisage the system being used is that the user should increase the number of simulated users until the systems starts showing 'yellow' warning responses, then tune from there. When the system cannot be tuned further, it is then in its optimal state and the maximum load is also established.

MQSERVICE

This class interfaces to MQSeries. It handles connections and queue opening. It also sets up message parameters (the subset that are used) and handles the putting and getting of messages to and from the queues. It assumes all messages exist in request-reply pairs. The time taken to handle messages is recorded here and returned to the calling class. Several extensions could be made to this class, for example, by introducing persistent messaging and error recovery.

MQSTRESSTHREAD

An object of this class is instantiated for each simulated user. The class calls the *MQService* class directly and renders the response time data accordingly. Rendering is achieved by use of a GUI label containing the response time in milliseconds. The label is colour-coded to improve clarity. Each instance of this class runs in its own separate thread.

CONCLUSION

Using the tools and the methods that I have described, it should be possible to stress test both queue managers and server applications. It may be necessary to extend the classes provided, or to re-write them totally. I am sure, however, that the general principle of testing a system by matching the intended messaging structure as closely as possible is valid. I also think that, by using a multi-threaded environment, a close approximation of user input can be achieved. It is important, however, to be aware of the resource limitations on the test machine. If a test machine (client not server) is over-utilized it will slip below the prescribed messaging rate and under-stress the server being tested.

MQCONSTANTS.JAVA

```
package com.dmitri.mqstress;
import com.sun.java.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
/**
*   @author Dmitri
*
*   This class prompts for various parameters that are thereafter
*   used to control the test. To save users the trouble of repeatedly
*   typing in data, the class persists itself to disk using standard
*   serialization.
*/
public class MQConstants extends JFrame implements Serializable {

//  The constants
  private JTextField iQueueManagerName = new JTextField();
  private JTextField iHostName = new JTextField();
  private JTextField iChannelName = new JTextField();
  private JTextField iRequestQueueName = new JTextField();
  private JTextField iReplyQueueName = new JTextField();
  private JTextField iMaxMessageLength = new JTextField();
  private JTextField iNumberOfInstances = new JTextField();
  private JTextField iNumberOfRepetitions = new JTextField();
  private JTextField iNumberOfMessages = new JTextField();

//  Arrays of 'textfields' for message details
  private JTextField [] iMessageLenthOrFile;  // The length of a blank
                                              // message or a file
  private JTextField [] iAcceptableTime;      // Acceptable time
  private JTextField [] iFailureTime;         // Time after which
                                              // message is deemed to
```

```
                                           // have failed
  private JTextField [] iPaceTime;         // Time used to delay
                                           // between messages

//  The text that is to be used for messaging.
  private String [] iMessages;

//  Buttons
  private JButton iBtnMessages = new JButton("Messages");
  private JButton iBtnReset = new JButton("Reset");
  private  JButton iBtnOk = new JButton("Ok");

//  Messsage capture dialogue box
  private JDialog iMessagedlg;

  protected static MQConstants cInstance;
  private transient Vector iListeners = new Vector();
  private static final String PERSISTENCY_NAME = "MQConstants.ser";

/**
*   Return the name of the queue manager
*/
  public String getQueueManagerName() {
    return iQueueManagerName.getText();
  }
/**
*   Return the host name
*/
  public String getHostName() {
    return iHostName.getText();
  }
/**
*   Return the channel name
*/
  public String getChannelName() {
    return iChannelName.getText();
  }
/**
*   Return the name of the request queue
*/
  public String getRequestQueueName() {
    return iRequestQueueName.getText();
  }
/**
*   Return the name of the reply queue
*/
  public String getReplyQueueName() {
    return iReplyQueueName.getText();
  }
/**
```

```
*   Return the maximum message length
*   Note : no effort is made to validate numeric values entered
*/
  public int getMaximumMessageLength() {
    return Integer.parseInt(iMaxMessageLength.getText());
  }
/**
*   Return the number of instances of the stress tester to run
*   Note : no effort is made to validate numeric values entered
*/
  public int getNumberOfInstances() {
    return Integer.parseInt(iNumberOfInstances.getText());
  }
/**
*   Return the number of repetitions to make
*   Note : no effort is made to validate numeric values entered
*/
  public int getNumberOfRepetitions() {
    return Integer.parseInt(iNumberOfRepetitions.getText());
  }
/**
*   Return the number of messages requested
*/
  public int getNumberOfMesages() {
    try {
      return Integer.parseInt(iNumberOfMessages.getText());
    } catch (NumberFormatException ex) {}
    return 0;
  }
/**
*   Return the time (in milliseconds) that is specified as acceptable
*   Note : no effort is made to validate numeric values entered
*/
  public int getAcceptableTime(int index) {
    return Integer.parseInt(iAcceptableTime[index].getText());
  }
/**
*   Return the time (in milliseconds) that represents a failure
*   Note : no effort is made to validate numeric values entered
*/
  public int getFailureTime(int index) {
    return Integer.parseInt(iFailureTime[index].getText());
  }
/**
*   Return the time (in milliseconds) that to be used to pace
*   MQ polling
*   Note : no effort is made to validate numeric values entered
*/
  public int getPaceTime(int index) {
    return Integer.parseInt(iFailureTime[index].getText());
  }
/**
*   Return the message specified by the index supplied
*/
  public String getMessage(int index) {
    return iMessages[index];
  }

/**
*   Standard singleton pattern constructor that takes care of the
*   capture of runtime parameters
*/
  protected MQConstants() {

    super("MQPing Constants");
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

    JPanel contents = new JPanel();
    getContentPane().add(contents);
    contents.setBorder(BorderFactory.createEmptyBorder(15,15,15,15));
    contents.setLayout(new BorderLayout());

    JPanel middle = new JPanel();
    middle.setLayout(new GridLayout(10, 2, 10, 10));  // Panel to
                                                      // capture constants
    contents.add(middle);

    middle.add(new JLabel("Queue Manager Name:"));
    middle.add(iQueueManagerName);
    middle.add(new JLabel("Host Name:"));
    middle.add(iHostName);
    middle.add(new JLabel("Client Channel Name:"));
    middle.add(iChannelName);
    middle.add(new JLabel("Request Queue Name:"));
    middle.add(iRequestQueueName);
    middle.add(new JLabel("Reply Queue Name:"));
    middle.add(iReplyQueueName);
    middle.add(new JLabel("Max message Length:"));
    middle.add(iMaxMessageLength);
    middle.add(new JLabel("Number of Instances:"));
    middle.add(iNumberOfInstances);
    middle.add(new JLabel("Number of Repetitions:"));
    middle.add(iNumberOfRepetitions);
    middle.add(new JLabel("Number of Messages:"));
    middle.add(iNumberOfMessages);

    JPanel buttons = new JPanel();
    contents.add(buttons, BorderLayout.SOUTH);
    iBtnOk.setEnabled(false);
    buttons.setLayout(new FlowLayout());
    buttons.add(iBtnMessages);
    buttons.add(iBtnReset);
    buttons.add(iBtnOk);

    addListeners();
    pack();

  }

/**
*   Handle hiding of the prompter frame and persisting data, which
*   is also a convenient point at which to calculate messages
*/
  private void closeEvent() {
    try {
      ObjectOutputStream oos = new ObjectOutputStream(new
                                 FileOutputStream(PERSISTENCY_NAME));
      oos.writeObject(this);
      oos.flush();
      oos.close();
    } catch (IOException ex) {
      ex.printStackTrace();
    }

    iMessages = new String [iMessageLenthOrFile.length];

    for (int i = 0; i < iMessageLenthOrFile.length; i++) {

      try {
        int buffLen =
                 Integer.parseInt(iMessageLenthOrFile[i].getText());
        char [] buffer = new char [buffLen];
        for (int j = 0; j < buffLen; j++) buffer[j] = ' ';
        iMessages[i] = new String(buffer);

      } catch (NumberFormatException ex) { // The user has
                                           // specified a filename

        try {
          BufferedReader reader = new BufferedReader(
            new FileReader( iMessageLenthOrFile[i].getText() ) );
          iMessages[i] = reader.readLine();
        } catch(IOException ioex) {
          System.out.println(ioex);
          System.out.println("Message length set to 1");
          iMessages[i] = " ";
        }
      }
    }
    setVisible(false);
```

```
        notifyConstantsListeners();
  }

/**
*   Adds appropriate inner classes for listening to buttons
*/
  private void addListeners() {

    iListeners = new Vector();
    iBtnMessages.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        setUpMessages();
        iBtnOk.setEnabled(true);
      }
    });

    iBtnReset.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        setVisible(false);
        cInstance = new MQConstants();
        cInstance.setVisible(true);
      }
    });

    iBtnOk.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        closeEvent();
      }
    });
  }
/**
*   Sets up the visual fields to capture message data
*/
  private void setUpMessages() {

    iMessagedlg = new JDialog(this, "Messages", true); // modal popup.
    iMessagedlg.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

    JPanel dlgContents = new JPanel();
    iMessagedlg.getContentPane().add(dlgContents);
    dlgContents.setBorder(
                    BorderFactory.createEmptyBorder(10,10,10,10));
    dlgContents.setLayout(new BorderLayout());

    JPanel key = new JPanel();
    key.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    key.setLayout(new GridLayout(1, 5, 5, 5));
    dlgContents.add(key, BorderLayout.NORTH);

    key.add(new JLabel("Message Number"));
    key.add(new JLabel("Message length/src"));
    key.add(new JLabel("Acceptable Time"));
    key.add(new JLabel("Failure Time"));
    key.add(new JLabel("Pace Time"));

    int msgNumbers = getNumberOfMesages();
    JPanel messageParms = new JPanel();
    JScrollPane scrPane = new JScrollPane();
    scrPane.getViewport().add(messageParms);
    dlgContents.add(scrPane);
    messageParms.setLayout(new GridLayout(msgNumbers, 5, 5, 5));

    // Instantiate the array of parameters only if the number
    // requested has changed
    if (iMessageLenthOrFile == null || iMessageLenthOrFile.length
                                               != msgNumbers) {
      iMessageLenthOrFile = new JTextField [msgNumbers];
      iAcceptableTime = new JTextField [msgNumbers];
      iFailureTime = new JTextField [msgNumbers];
      iPaceTime = new JTextField [msgNumbers];
    }

    for (int i = 0; i < msgNumbers; i++) {

      messageParms.add(new JLabel(Integer.toString(i + 1) + "."));
      if (iMessageLenthOrFile[i] == null) iMessageLenthOrFile[i]
                                               = new JTextField();
      messageParms.add(iMessageLenthOrFile[i]);
      if (iAcceptableTime[i] == null) iAcceptableTime[i]
                                               = new JTextField();
      messageParms.add(iAcceptableTime[i]);
      if (iFailureTime[i] == null) iFailureTime[i] = new JTextField();
      messageParms.add(iFailureTime[i]);
      if (iPaceTime[i] == null) iPaceTime[i] = new JTextField();
      messageParms.add(iPaceTime[i]);
    }
```

The remainder of the code for this class and the code for other classes used for MQSeries stress-testing appear in next month's issue of *MQ Update*.

MQSeries Specialist (UK) © Xephon 1999

MQ news

BMC Software has announced Patrol for MQ, the latest member of the company's family of system and network management products and the first to benefit from the company's recent (November 1998) acquisition of Boole & Babbage. Patrol for MQ manages the MQSeries layer, managing both MQSeries objects and hardware and software components that affect MQSeries availability.

The product provides managers with facilities for monitoring, automating, and managing the MQSeries layer, also providing numerous facilities for managing messages by content, including the ability to search and browse messages. Also provided is end-to-end message compression for improved performance and throughput. Another benefit of Patrol for MQ (one that the company seems to think hardly worth mentioning) is its ability to manage both Microsoft's MSMQ and IBM's MQSeries.

It's out now, and standard packages start at US$25,000.

For further information contact:
BMC Software, 2101 CityWest Blvd, Houston, TX 77042, USA
Tel: +1 713 918 8800
Fax: +1 713 918 8000
Web: http://www.bmc.com

BMC Software Limited, Compass House, 207-215 London Road, Camberley, Surrey, GU15 3EY, UK
Tel: +44 1276 24622
Fax: +44 1276 61201

* * *

IBM has announced MQSeries Integrator V2.0, the latest version of the company's message broker, which now supports dynamic publish/subscribe based on message content (bringing the product in line with the same facilities in the MQSeries V5.0 base product), integration with databases, allowing MQSeries Integrator to query databases such as DB2 and SQL Server for content for message transformation and creation, support for XML, and new APIs, including the Application Messaging Interface (AMI) V1.0 API that makes message transport functionality transparent to applications. MQSeries Integrator V2 is to be available in December this year for AIX and NT, with versions for other platforms to follow.

For further information contact your local IBM representative.

* * *

Vision Software Tools is to bundle JADE 4.1 with IBM's MQSeries (the product also ships with DB2 Connect and WebSphere Enterprise Edition). JADE includes the Vision Business Logic Server, Vision Developer Studio, and Vision Business Server Manager, which handles Web application development and deployment.

For further details contact:
Vision Software Tools Inc, 2101 Webster Street, 8th Floor, Oakland, CA 94612, USA
Tel: +1 510 238 4100
Fax: +1 510 238 4118
Web: http://www.vision-soft.com