April 2011

High Availability Solution for WebSphere Message Queue in a Single Instance Queue Manager Configuration using HAE and a Shared DASD Device

Table of Contents

Abstract

This paper describes the design and implementation of a high availability solution for a WebSphere® Message Queue Single Instance Queue Manager configuration on a pair of virtual Linux® servers running on an IBM System z10®. The solution tested during the creation of this paper used SUSE Linux Enterprise Server 11 SP1, *SUSE Linux Enterprise High Availability Extensions* (HAE), and WebSphere MQ version 7.0.1. Although the solution described in this paper has been tested by the authors only with the software versions described, the same techniques are applicable to other Linux distributions and other versions of WebSphere MQ with some modifications.

This document assumes a basic working understanding of Linux on IBM System z® including IBM networking technologies, and WebSphere MQ operation and installation.

This document builds upon an earlier document titled: **High Availability Solution for WebSphere Message Queue in a single instance queue manager configuration using Linux HA and a shared DASD device**
http://www.**ibm.com**/systems/resources/sysserv_platformtest_mq_singleinstance.pdf

Architecture at a glance

The WebSphere MQ high availability configuration consists of three resources, which will be managed by HAE:

1) WebSphere MQ Queue Manager - a WebSphere MQ system service that provides a logical container for the message queue, and is responsible for transferring data to other queue managers using message channels.
2) A DASD device that contains the WMQ queue and log files, which is used as the persistent state shared between the two cluster nodes.
3) A virtual IP address, which serves as the public interface to the WMQ QM.

In this solution two instances (or perhaps “two identically configured instances” if they really are copies) of WebSphere MQ are installed with one residing on each host system. The Queue and Log files used by the Queue Manager reside on a shared disk device that is made available to only one system at a time, which is referred to as the *active* system. The system that has control of the shared disk simultaneously has control of the virtual IP address used by the Queue Manager. In addition, this active system has an active copy of the Queue Manager running.

If the active system fails or is in the process of failing, the high availability manager stops the virtual IP address, stops the running copy of the Queue Manager, and unmounts the shared disk device if the system is intact enough for those actions to be taken. Control of all three resources, virtual IP address, shared DASD device, and running Queue Manager instance, is transferred to the *backup* system, at which time the shared disk is mounted, the IP address is made active, and the Queue Manager is started, thus enabling a new active system on the former backup system.

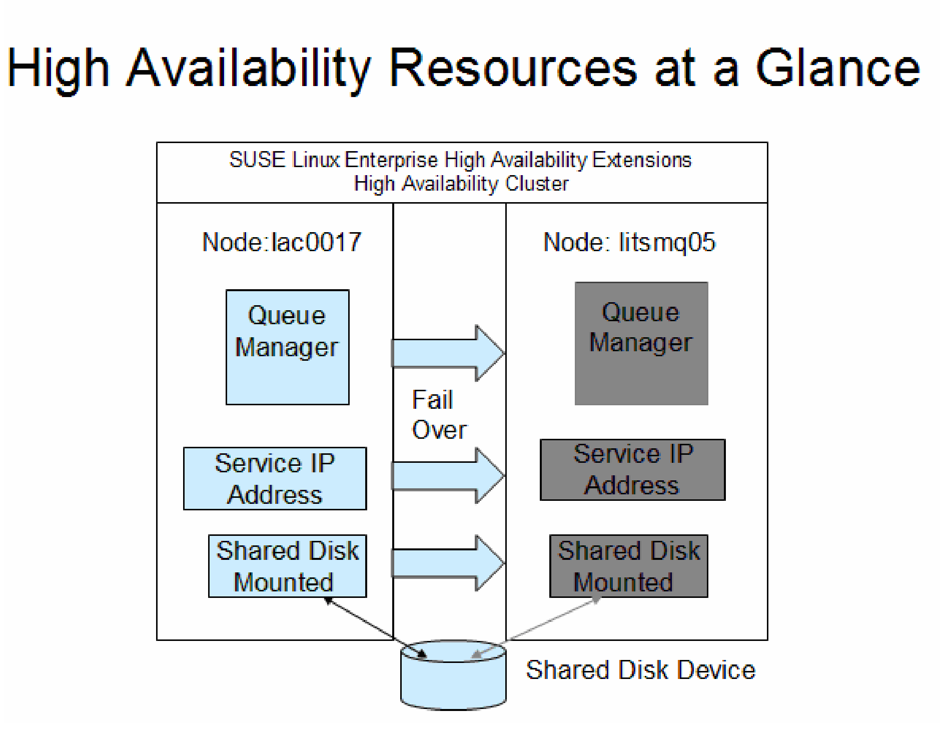


Figure1: This diagram illustrates an example of our two node HA architecture. For this example, we have selected hostnames that match the documentation in the remainder of this whitepaper. In this diagram, node lac0017 is in control of the High Availability Resources and is considered “active” while node litsmq05 is in standby mode and is the backup node. The direction of failover is shown to illustrate a single HA failover scenario. In reality, the failover is bi-directional.

SUSE Linux Enterprise High Availability Extension is an integrated suite of open source clustering technologies that provides the ability to implement and manage highly available Linux clusters.

SUSE Linux Enterprise High Availability Extension will be referred to as *HAE* in this document. HAE is an integrated suite of open source clustering technologies that provides the ability to implement highly available physical and virtual Linux clusters.

In SLES10, SUSE provided the Heartbeat version of Linux HA. Linux HA is an open source product developed by the Linux High Availability Project. In SLES11 SP0, SUSE provided a version of HAE which was based upon the Pacemaker version of Linux HA with OpenAIS, providing cluster level communication support. HAE also includes support for the *Oracle Cluster File System2* (OCFS2), a clustered version of the *Logical Volume Manager* (cLVM), and other high availability features. In SLES11 SP1, HAE is based on Pacemaker with the addition of the Corosync Cluster Engine, which enhances the capabilities of OpenAIS.

HAE provides the mechanisms for monitoring and providing fail-over for the resources required for this configuration.

Prior to WebSphere MQ V7, IBM provided the scripts need for a high availability solution in packages called WebSphere MQ SupportPacs™. Those SupportPacs are now officially withdrawn. IBM now provides sample scripts for managing WebSphere MQ in a high availability cluster on the WebSphere MQ Info Center web site:
http://publib.boulder.**ibm.com**/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.amqzag.doc/fa70230_.htm

The Info Center provides useful information on how WebSphere MQ interacts with a high availability cluster.

The sample scripts provided at the Info Center are used in this solution to stop, start, and monitor the WebSphere MQ Queue Manager. In addition to the scripts provided at the Info Center, one additional script is needed. The additional script is used by HAE to drive the other three scripts. All of the scripts required for this solution are provided in this document.

Implementing the solution

Build two Linux systems

This solution is based on SLES11 SP1.

Networking considerations

In this solution the high availability cluster contains two nodes, named lac0017 and litsmq05.

In addition to the default system network device, each system will require internal network connectivity for HAE communication. This network connection is referred to as the *bind network address.* We recommend using HiperSockets™ if deploying the HA solution within a physical host, or ideally using a VSWITCH or real OSA devices for the heartbeat network when spanning hosts. Refer to the HAE documentation link provided earlier for more information.

The systems will also each need a public IP address used to make WebSphere MQ available to the consumers and applications requiring access to this service. This IP address will be implemented as a virtual IP address, and is active on only one of the two nodes at any given time.

Storage considerations

In addition to the storage required to contain the Linux system and WebSphere MQ, a single ECKD™ DASD device is required. If it is determined that more than one DASD device is required to contain your WebSphere MQ log data and log files, it would be easy to modify the configuration presented in this document to use an LVM or RAID configuration as the shared disk device. A z/FCP SCSI could also be used in place of the DASD device. Only the single DASD device configuration will be discussed here.

Configure a shared disk device

In this solution a shared disk device is used by the WebSphere MQ Queue Manager for log and queue data. This device will also contain the scripts that Pacemaker will use to start, stop, and monitor the resources. This solution uses an ECKD DASD device.

The shared DASD device must be configured to allow multiple systems to perform both read and write operations to the device. The integrity of the device will be maintained by Pacemaker. Pacemaker will insure that the device is mounted on only one system at a time.

Shared disk considerations for z/VM

If both systems are on the same LPAR, that is they are guests of the same z/VM® system, then one system will require a MDISK (mini disk) directory control statement in it's z/VM system directory entry, while the other system will require a LINK directory statement in it's z/VM system directory entry. In this configuration a full-pack minidisk was used.

The system directory entry for lac0017 would contain this MDISK directory control statement:

MDISK 8092 3390 DEVNO 8092 MWV

Parameter	Meaning
8092	The virtual address of the device
3390	The device type
DEVNO	Specifies a full-pack minidisk.
8092	The volume serial number of the real DASD volume on which the minidisk resides.
MW	The mode of the minidisk. MW allows Multiple-write access.
V	Tells CP to use its virtual reserve/release support in the I/O operations for the minidisk. MWV means the minidisk functions with write linkage using CP's virtual reserve/release.

The system directory entry for litsmq05 would contain this LINK directory control statement:

LINK LAC0017 8092 8092 MW

Parameter	Meaning
LAC0017	The user ID in the system directory, whose entry is to be searched for the device.
8092	The virtual address that was assigned to the device on lac0017.
8092	The virtual device number that is to be assigned to the device on litsmq05.
MW	The mode of the device. MW allows Multiple-write access.

In order to update the z/VM system directory, a user ID with at least Class B privileges is needed.

It is best to dedicate a real DASD device(s) that will be shared by the individual Linux

Guests, when the guests reside on multiple LPARs. The shared disk device must be configured in the system configuration file for each z/VM system using an RDEVICE statement.

Rdevice xxxx Type DASD Shared Yes MDC OFF

Parameter	Meaning
xxxx	This is real device address of the shared disk device.
Type DASD	The device type.
Shared Yes	Indicates that the device is to be shared between multiple real systems.
MDC OFF	Tells CP to not cache data on the real device in the minidisk cache.

With the disk online, it is then necessary to format the disk, partition it, and then make a file system before it can be used by Linux. For our testing we used the ext3 file system.

Install the required software

Install HAE on your SLES11 SP1 system per the documentation provided with the installation media. Perform the basic configuration steps required to have a working two node HAE cluster.

HAE documentation can be found at this address:
http://www.novell.com/documentation/sle_ha/book_sleha/?page=/documentation/sle_ha/book_sleha/data/book_sleha.html

Install WebSphere MQ on both systems. This solution is based on WebSphere MQ 7.0.1.0. Information on installing WebSphere MQ can be found at this address:
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/topic/com.ibm.mq.amq1ac.doc/lq10220.htm

Configure the shared disk for use by Linux

The steps described in this section can be performed on either of the nodes. They need to be performed only once, on a single node.

Determine the kernel device name for the shared DASD device. The Linux lsdasd command will provide this information.

```
# lsdasd
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.0201   active      dasda     94:0     ECKD  4096   7042MB    1802880
0.0.0202   active      dasdb     94:4     ECKD  4096   7042MB    1802880
0.0.8092   active      dasdc     94:8     ECKD  4096   7043MB    1803060
0.0.0200   active      dasdd     94:12   FBA   512    1000MB    2048000
```

In the display shown above the kernel device name for device 8092, which is our shared disk device, is dasdc.

Note: The kernel device name for the shared disk cannot be the same on both nodes. This is not a concern. When this device is defined to HAE the by-path ID will be used. For DASD device 8092 the by-path ID is /dev/disk/by-path/ ccw-0.0.8092. After the 8092 device has been formatted and partitioned, the by-path ID will be /dev/disk/by-path/ ccw-0.0.8092-part1.

```
# ls /dev/disk/by-path
ccw-0.0.0200       ccw-0.0.0201-part1  ccw-0.0.0202-part1 ccw-0.0.0200-part1  ccw-0.0.0201-
part2  ccw-0.0.8092  ccw-0.0.0201       ccw-0.0.0202       ccw-0.0.8092-part1
```

Format the DASD device

The Linux dasdfmt command will format the shared disk device. In the example command the –b option is set to 4096. This is the blocking factor that will be used when the device is formatted. This is the IBM recommendation. Consult the storage administrator at your location to be sure that this meets location recommendations.

```
# dasdfmt -b 4096 -p -f /dev/dasdc

Drive Geometry: 10017 Cylinders * 15 Heads = 150255 Tracks

I am going to format the device /dev/dasdc in the following way:
   Device number of device: 0x8092
   Labelling device        : yes
   Disk label              : VOL1
   Disk identifier         : 0X8092
   Extent start (trk no)   : 0
   Extent end (trk no)     : 150254
   Compatible Disk Layout  : yes
   Blocksize               : 4096
```

```
--->> ATTENTION! <<---
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device.  This may take a while (get yourself a coffee).

cyl 10017 of 10017
|##################################################| 100%

Finished formatting the device.
Rereading the partition table... ok
```

Partition the DASD device

Now that the shared disk device has been formatted, it needs to be partitioned. The Linux fdasd command is used to partition the device. The –a option tells the fdasd command to automatically create a single partition using all the space available on the device.

```
# fdasd /dev/dasdc -a
reading volume label ..: VOL1
reading vtoc ..........: ok

auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
```

Create the ext3 filesystem

The shared disk device has now been formatted and partitioned. The last step in preparing the device for use is to create a filesystem. In the testing of this solution, an ext3 filesystem was used. Because this is the only filesystem type that was tested, it the one recommended for implementing this solution.

```
# mkfs.ext3 -b 4096 /dev/dasdc1
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
451584 inodes, 1803036 blocks
90151 blocks (5.00%) reserved for the super user
First data block=0
```

```
Maximum filesystem blocks=1849688064
56 block groups
32768 blocks per group, 32768 fragments per group
8064 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

The shared disk device is now ready to be used. The next steps describe the creation of a mount point for the device to be mounted on, mounting the device, and creating sub-directories on the device.

Setup the Shared Disks Directory Structure

1. Create a mount point for the shared disk. The /MQHA mount point needs to be created on both systems.

   ```
   # mkdir -m 755 /MQHA
   ```

   While the mount point needs to be created on both systems, the shared disk device should only be mounted on one system at a time.

2. Mount the shared disk device on one of systems:

   ```
   # mount /dev/dasdc1 /MQHA/
   ```

   The scripts used by HAE to control the Queue Manager resource will be placed on the shared disk device in the bin sub-directory.

3. Create the bin directory on the shared disk device:

   ```
   # mkdir -m 755 /MQHA/bin
   ```

   The Queue Manager requires a sub-directory for data (/MQHA/<queue_manager_name>/data. It also requires a directory for logs (/MQHA/<queue_manager_name>/log).

4. Create the sub-directories needed by the Queue Manager:

In this example the Queue Manager name is “QM1”.

```
# mkdir -m 755 /MQHA/QM1
# mkdir -m 755 /MQHA/QM1/data
# mkdir -m 755 /MQHA/QM1/log
```

Create a Queue Manager for use in the HA cluster

Create a WebSphere Queue Manager instance on one of the nodes using the MQ crtmqm command.

1. WebSphere MQ commands are usually required to be run under the MQ user ID. This is usually “mqm”. The Linux “su” command enables commands to be run with a substitute user ID. As used in the example below, the –c option on the su command indicates that a command string is to be executed. The command string is encapsulated in a pair of double quotes.

```
#su mqm -c "crtmqm -md /MQHA/QM1/data -ld /MQHA/QM1/log QM1"

WebSphere MQ queue manager created.
Directory '/MQHA/QM1/data/QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 65 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

2. The MQ command dspmqinf used with the “command” option creates a command that can be used to populate the settings needed for the Queue Manager on the backup node.

Run the dspmqinf on the same node that the first instance of the Queue Manager was created on.

```
# su mqm -c "dspmqinf -o command QM1"
addmqinf -s QueueManager -v Name=QM1 -v Directory=QM1 -v Prefix=/var/mqm -v
DataPath=/MQHA/QM1/data/QM1
```

The response from dspmqinf command is comprised of the MQ addmqinf along with all of the information required to create a second copy of the original Queue Manager instance one the second node.

3. In preparation for applying the Queue Manager instance settings on the second node copy the output of the dspmqinf command.
4. The shared disk device is no longer needed and should be unmounted using the Linux umount command:

```
# umount /MQHA
```

5. Mount the shared disk device at /MQHA on the backup node using the by-path ID.

```
# mount /dev/disk/by-path/ccw-0.0.8092-part1 /MQHA/
```

Add Queue Manager configuration to the second node

Run the “addmqinf” command string created by the “dspmqinf” command on the active node. This command must be executed using the MQ user ID.

```
l# su mqm -c "addmqinf -s QueueManager -v Name=QM1 -v Directory=QM1 -v
Prefix=/var/mqm -v DataPath=/MQHA/QM1/data/QM1"

WebSphere MQ configuration information added.
```

Configure the Scripts Used to Support the HA Configuration

HAE has the ability to execute scripts to perform actions that control the resources it manages.

The scripts needed to manage the shared disk and IP Address are provided by HAE and will be discussed later. The scripts needed to manage the WebSphere Queue Manager must be provided. The full set of scripts required to manage the Queue Manager are provided in this section.

This table shows the scripts needed to implement the solution and their function.

Script Name	Function
mqm	A script in the LSB * format used to control the WebSphere MQ Queue Manager. This script is used by HAE to start, stop, and monitor the Queue Manager.
hamqm_start_su	Used to start the WebSphere MQ Queue Manager. Called by mqm.
hamqm_stop_su	Used to stop the WebSphere MQ Queue Manager. Called by mqm.
hamqm_running_su	Used to monitor the WebSphere MQ Queue Manager. Called by mqm.

The mqm script is based on the script by the same name originally provided in the WebSphere MQ SupportPac IC91. All other scripts were taken from the WebShere MQ Info Center web site.

* LSB stands for the *Linux Standards Base which implies that the script follows a set of standards developed by the Linux Foundation (www.linuxfoundation.org).*

This table shows the scripts need to implement the configuration:

Script Name	**Script Content**

mqm	#!/bin/bash
	# %Z% %W% %I% %E% %U%
	#
	# Licensed Materials - Property of IBM
	#
	# (C) Copyright IBM Corp. 2006 All Rights Reserved.
	#
	# IC91: High Availability for WebSphere Message Broker on Distributed Platforms
	#
	# /etc/ha.d/resource.d/mqm
	#
	### BEGIN INIT INFO
	# Provides: WMQ Control
	# Required-Start: $network $remote_fs
	# Required-Stop: $network $remote_fs
	# Default-Start: 3 5
	# Default-Stop: 0 1 2 6
	# Description: Start a WMQ Queue Manager
	### END INIT INFO
	
	STRMQM=/usr/bin/strmqm
	test -x $STRMQM || exit 5
	
	ENDMQM=/usr/bin/endmqm
	test -x $ENDMQM || exit 5
	
	# Copy this script to /etc/rc.d/mqm_<your_Queue_Manager_name>
	# Example: mqm_QM1
	QM="<place_Queue_Manager_name_here>
	# Example: QM="QM1"
	
	. /etc/rc.status
	
	# Shell functions sourced from /etc/rc.status:
	# rc_check check and set local and overall rc status
	# rc_status check and set local and overall rc status
	# rc_status -v ditto but be verbose in local rc status
	# rc_status -v -r ditto and clear the local rc status
	# rc_failed set local and overall rc status to failed
	# rc_reset clear local rc status (overall remains)
	# rc_exit exit appropriate to overall rc status
	
	# First reset status of this service
	rc_reset
	
	#case "$2" in
	case "$1" in
	    start)
	        su mqm -c "/MQHA/bin/hamqm_start_su $QM"
	
	        # Remember status and be verbose
	        rc_status -v
	        ;;
	    stop)

```
    su mqm -c "/MQHA/bin/hamqm_stop_su $QM 30"

    # Remember status and be verbose
    rc_status -v
    ;;
  status)
    su mqm -c "/MQHA/bin/hamqm_running_su $QM"
    if [ $? -ne 0 ]

    then
      :echo "Queue Manager $QM is not operational"
      exit 1
    fi
    echo "Queue Manager $QM is running"
    exit 0
    ;;
  monitor)
    su mqm -c "/MQHA/bin/hamqm_running_su $QM"
    if [ $? -ne 0 ]

    then
      echo "Queue Manager $QM is not operational"
      exit 1
    fi
    echo "Queue Manager $QM is running"
    exit 0
    ;;

esac

rc_exit
```

hamqm_start_su

```
#!/bin/bash
#
# This script robustly starts the queue manager.
#
# The script must be run by the mqm user.

# The only argument is the queue manager name. Save it as QM variable
QM=$1

if [ -z "$QM" ]
then
 echo "ERROR! No queue manager name supplied"
 exit 1
fi

# End any queue manager processes which might be running.

srchstr="( |-m)$QM *.*$"
for process in amqzmuc0 amqzxma0 amqfcxba amqfqpub amqpcsea amqzlaa0 \
        amqzlsa0 runmqchi runmqlsr amqcrsta amqrrmfa amqrmppa \
        amqzfuma amqzdmaa amqzmuf0 amqzmur0 amqzmgr0
 do
 ps -ef | tr "\t" " " | grep $process | grep -v grep | \
  egrep "$srchstr" | awk '{print $2}'| \
    xargs kill -9 > /dev/null 2>&1
done
```

```
# It is now safe to start the queue manager.
strmqm ${QM}
```

hamqm_stop_su

```
#!/bin/bash
#
# The script ends the QM by using two phases, initially trying an immediate
# end with a time-out and escalating to a forced stop of remaining
# processes.
#
# The script must be run by the mqm user.
#
# There are two arguments: the queue manager name and a timeout value.
QM=$1
TIMEOUT=$2

if [ -z "$QM" ]
then
  echo "ERROR! No queue manager name supplied"
  exit 1
fi

if [ -z "$TIMEOUT" ]
then
  echo "ERROR! No timeout specified"
  exit 1
fi

for severity in immediate brutal
do
  # End the queue manager in the background to avoid
  # it blocking indefinitely. Run the TIMEOUT timer
  # at the same time to interrupt the attempt, and try a
  # more forceful version. If the brutal version fails,
  # nothing more can be done here.

  echo "Attempting ${severity} end of queue manager '${QM}'"
  case $severity in

  immediate)
    # Minimum severity of endmqm is immediate which severs connections.
    # HA cluster should not be delayed by clients
    endmqm -i ${QM} &
    ;;

  brutal)
    # This is a forced means of stopping queue manager processes.

    srchstr="( |-m)$QM *.*$"
    for process in amqzmuc0 amqzxma0 amqfcxba amqfqpub amqpcsea amqzlaa0 \
               amqzlsa0 runmqchi runmqlsr amqcrsta amqrrmfa amqrmppa \
               amqzfuma amqzdmaa amqzmuf0 amqzmur0 amqzmgr0
    do
      ps -ef | tr "\t" " " | grep $process | grep -v grep | \
         egrep "$srchstr" | awk '{print $2}'| \
            xargs kill -9 > /dev/null 2>&1
    done
  esac

  TIMED_OUT=yes
  SECONDS=0
  while (( $SECONDS < ${TIMEOUT} ))
  do
   TIMED_OUT=yes
   i=0
   while [ $i -lt 5 ]
   do
     # Check for execution controller termination
     srchstr="( |-m)$QM *.*$"
     cnt=`ps -ef | tr "\t" " " | grep amqzxma0 | grep -v grep | \
       egrep "$srchstr" | awk '{print $2}' | wc -l `
     i=`expr $i + 1`
     sleep 1
     if [ $cnt -eq 0 ]
     then
       TIMED_OUT=no
       break
     fi
   done

   if [ ${TIMED_OUT} = "no" ]
   then
     break
   fi

   echo "Waiting for ${severity} end of queue manager '${QM}'"
   sleep 1
  done # timeout loop

  if [ ${TIMED_OUT} = "yes" ]
  then
    continue        # to next level of urgency
  else
    break           # queue manager is ended, job is done
  fi

done # next phase
```

hamqm_running_su

```
#
# This script tests the operation of the queue manager.
#
# An exit code is generated by the runmqsc command:
# 0  => Either the queue manager is starting or the queue manager is running and
responds.
#       Either is OK.
# >0 => The queue manager is not responding and not starting.
#
# This script must be run by the mqm user.
QM=$1

if [ -z "$QM" ]

then
  echo "ERROR! No queue manager name supplied"
  exit 1
fi

# Test the operation of the queue manager. Result is 0 on success, non-zero on error.
echo "ping qmgr" | runmqsc $QM > /dev/null  > /dev/null 2>&1
pingresult=$?

if [ $pingresult -eq 0 ]
then # ping succeeded

  echo "Queue manager '$QM' is responsive"
  result=0

else # ping failed
  # Don't condemn the queue manager immediately, it might be starting.
  srchstr="( |-m)$QM *.*$"
  cnt=`ps -ef | tr "\t" " " | grep strmqm | grep "$srchstr" | grep -v grep \
              | awk '{print $2}' | wc -l`
  if [ $cnt -gt 0 ]
  then
    # It appears that the queue manager is still starting up, tolerate
    echo "Queue manager '$QM' is starting"
    result=0
  else
    # There is no sign of the queue manager starting
    echo "Queue manager '$QM' is not responsive"
    result=$pingresult
  fi

fi
exit $result
```

All of the scripts in the preceding table can be placed on your Linux system using copy and paste.

The file names must be kept the same as the names provided in the table. If you desire to change the file names you will have to modify the mqm script to reflect the new names.

The mqm script must be placed in /etc/rc.d. This is an HAE requirement. All other scripts must be placed in /MQHA/bin, which is located on the shared disk.

Create a unique copy of the mqm script

In order to support one or more unique Queue Managers, it is necessary to create a copy of the mqm script in /etc/rc.d with a unique name and unique setting for the Queue Manager name.

The Queue Manager name in this solution is QM1. The copy command would look like this:
cp /etc/rc.d/mqm /etc/rc.d/mqm_QM1
Edit /etc/rc.d/mqm_QM1 and find this section of code within the script:

```
# Copy this script to /etc/rc.d/mqm_<your_Queue_Manager_name>
# Example: mqm_QM1
QM="<place_Queue_Manager_name_here>
# Example: QM=”QM1”
```

Edit the QM setting to read: QM=”QM1”

If your configurations requires more than one instance of the Queue Manager you can set up mqm_<Queue_Manager_name> scripts for each instance.

Make the scripts executable

Once all of the scripts have been created in the /MQHA/bin directory you can make them executable. Run these Linux commands:

```
chmod +x /MQHA/bin/*
chmod +x /etc/rc.d/mqm*
```

At this point it is a good idea to umount the shared disk so that it will not be mounted when you implement the HAE configuration.

```
# umount /MQHA
```

Quorum state considerations in a two-node cluster

Split Brain is a condition that occurs when a high availability cluster contains less than three members. It describes a stalemate situation where no quorum exists when the high availability manager needs to take a vote to determine which node is the survivor after some type of event that disrupts communications within the cluster. Pacemaker provides a no-quorum option to allow resources to be started when a quorum is not possible.

In some high availability configurations a STONITH mechanism is used to “shoot the other node in the head” when it is believed to be in a state where resource integrity is threatened. This mechanism is also called a fencing mechanism. The purpose of fencing is to halt or isolate a distressed node so that the integrity of cluster resources is maintained. In this solution, no STONITH mechanism is in place. Pacemaker will not start resources if no fencing mechanism is in place unless the stonith_enabled option is set to “false”.

Set the no-quorum option

Since a two node cluster is being deployed in this solution, the Global Cluster Option no-quorum-policy must be set to “ignore” in order to insure that HAE is able to function regardless of the quorum state. In a two-node cluster any single node failure would result in a loss of majority. It is desirable for the cluster to carry on when one node experiences a failure.

The current no-quorum-policy setting can be checked using this command:

```
crm_attribute -n no-quorum-policy -G
```

For this solution, the no-quorum-policy option must be set to “ignore”. Use this command to change the setting:

```
crm_attribute -n no-quorum-policy -v ignore
```

This setting tells HAE to ignore the loss of quorum and permit the control of resources.

When changing CM settings it is a good idea to verify that no errors were introduced into the CRM. This command will verify the live copy of the CRM: crm_verify -L

Disable fencing

In this solution a STONITH mechanism will not be used. The Global Cluster Option stonith-enabled must be disabled. This is done by setting the stonith-enabled option to “false”. This setting effectively disables fencing.

This command sets the stonith-enabled option to “false”:

```
crm configure property stonith-enabled=false
```

Configure cluster resources

In our cluster, HAE will manage three resources: the shared DASD device, the virtual IP address, and the Queue Manager. The Pacemaker component of HAE is also known as the Cluster Resource Manager or CRM. The CRM implements the configuration specified in the Cluster Information Base or CIB. The CIB contains XML code that describes the configuration.

The CIB can be updated in three ways. The most popular option is the GUI (crm_gui) tool. It is also possible to use the cibadmin tool or various line commands. When implementing the configuration required for this solution, the CRM Command Line Interface tool (CLI) will be used. In previous releases of Pacemaker, the configuration would have to be provided in XML and would be loaded with the cibadmin tool. The CRM CLI tool supports a command language that is much easier to read and edit than XML.

The configuration that supports this solution follows in CRM CLI format:

```
primitive MQHA_Filesystem ocf:heartbeat:Filesystem \
        operations $id="MQHA_Filesystem-operations" \
        op monitor interval="120" timeout="60" \
        params device="/dev/disk/by-path/ccw-0.0.8092-part1" directory="/MQHA" fstype="ext3"
primitive MQHA_IP_addr ocf:pacemaker:IPaddr \
        params ip="192.168.70.16"
primitive MQHA_Ping ocf:pacemaker:ping \
        params host_list="192.168.70.1" multiplier="10" dampen="5s"
primitive MQHA_mqm lsb:mqm_QM1 \
        operations $id="MQHA_mqm-operations" \
        op monitor interval="60" timeout="15" \
        meta is-managed="true"
group MQHA MQHA_Filesystem MQHA_mqm MQHA_IP_addr \
        meta target-role="started"
clone MQHA_Ping_clone MQHA_Ping \
        meta clone-max="2" target-role="Started"
location MQHA_Filesystem_loc1 MQHA_Filesystem 100: lac0017
location MQHA_Filesystem_loc2 MQHA_Filesystem 100: litsmg05
location MQHA_IPaddr_loc1 MQHA_IP_addr 100: lac0017
location MQHA_IPaddr_loc2 MQHA_IP_addr 100: litsmg05
location MQHA_loc1 MQHA 100: lac0017
location MQHA_loc2 MQHA 100: litsmg05
location MQHA_mqm_loc1 MQHA_mqm 100: lac0017
location MQHA_mqm_loc2 MQHA_mqm 100: litsmg05
order MQHA_mqm_order : MQHA_Filesystem MQHA_mqm
```

This configuration describes the three resources that HAE will manage when this solution is implemented. The three resources are members of a resource group.

Configure a resource group

Cluster resources can be configured as a resource group for easier management. In this solution, all three resources are configured as a resource group. This allows the user to execute actions on the entire group as a convenience. Actions can be executed against individual resources within the group as well.

Here is the group statement for this solution:

```
group MQHA MQHA_Filesystem MQHA_mqm MQHA_IP_addr \
        meta target-role="started"
```

A look at one of the group statement:

Keyword	Group	The type of resource.
ID	MQHA	The name of this group. User selectable.
Resources	MQHA_Filesystem MQHA_mqm MQHA_IP_addr	The list of resources that are members of this group.
Meta attributes	target-role="started"	The state the group should be in.

Configuring the Filesystem resource

Here are the statements that relate the Filesystem resource:

```
primitive MQHA_Filesystem ocf:heartbeat:Filesystem \
        operations $id="MQHA_Filesystem-operations" \
        op monitor interval="120" timeout="60" \
        params device="/dev/disk/by-path/ccw-0.0.8092-part1" directory="/MQHA" fstype="ext3"
```

Here is the primitive statement for the Filesystem resource:

```
primitive MQHA_Filesystem ocf:heartbeat:Filesystem
```

Value	**Description**
ocf	Script style used for resource agent
heartbeat	The combination of class and provider tells Heartbeat where the resource agent script is located. In this case it is located in /usr/lib/ocf/resource.d/heartbeat.
Filesystem	Name of script used as resource agent

The Filesystem script requires three pieces of information; device, directory, and fstype. They are supplied on primitive statement:

Name	**Value**	**Description**
device	/dev/disk/by-path/ccw-0.0.8092-part1	This parameter provides the by-path ID of the shared disk
directory	/MQHA	The mount point of the shared disk device
fstype	ext3	The filesystem type used on the shared disk

Here is the operations statement:

```
operations $id="MQHA_Filesystem-operations" \
        op monitor interval="120" timeout="60" \
```

The information provided on the op statement consists of:

Name	Value	Description
$id	MQHA_Filesystem-operations	The ID associated with the operation. This value can be changed by the user.
op	monitor	The name of the operation
interval	120	Length of time in seconds between monitor operations
timeout	60	How many seconds to wait for the monitor operation to complete

Configuring the Queue Manager resource

Here are the statements that relate to the Queue Manager resource:

```
primitive MQHA_mqm lsb:mqm_QM1 \
        operations $id="MQHA_mqm-operations" \
        op monitor interval="60" timeout="15" \
        meta is-managed="true"
```

Here is the primitive statement for the Queue Manager resource:

```
primitive MQHA_Filesystem ocf:heartbeat:Filesystem
```

Value	Description
MQHA_mqm	The name of the primitive. This is user selectable.
lsb	The class of the resource agent. LSB class resource agents are located in /etc/rc.d.
mqm_QM1	Name of script used as resource agent. This script is created by the user. See section: **Creating a Unique Copy of the mqm script**

The mqm_QM1 script requires one piece of information, the action to be performed. This is supplied by Pacemaker depending on the action being taken. The possible actions are start, stop, monitor, and status.

Here is the operations statement:

```
        operations $id="MQHA_mqm-operations" \
        op monitor interval="60" timeout="15" \
```

The information provided on the op statement consists of:

Name	Value	Description
$id	MQHA_mqm-operations	The ID associated with the operation. This value can be changed by the user.
op	monitor	The name of the operation
interval	60	Length of time in seconds between monitor operations
timeout	15	How many seconds to wait for the monitor operation to complete
Meta attributes	target-role=”started”	The state that the resource should be in.

Configuring the IP Address resource

Here is the primitive statement for the IP Address resource:

```
primitive MQHA_IP_addr ocf:heartbeat:IPaddr \
        params ip="192.168.70.16"
```

Value	Description
MQHA_IP_addr	The name of this primitive. This is user selectable.
ocf	The class of the resource agent.
heartbeat	The combination of class and provider tells Heartbeat where the resource agent script is located. In this case it is located in /usr/lib/ocf/resource.d/heartbeat.
IPaddr	The name of the resource agent script.

The IPaddr script requires one piece of information, the IP address to be used by the Queue Manager. This is supplied with the params statement: params ip="192.168.70.16".

Configuring the Ping resource

The ping resource agent monitors connectivity to specific hosts or IP addresses. In this solution, the ping resource agent is used to monitor the IP address of the gateway. This provides a way to determine the health of the network. Ping uses a clone resource to set up an instance of the ping resource on each node in the cluster.

A look at the Ping Primitive statement:

```
primitive MQHA_Ping ocf:pacemaker:ping \
        params host_list="192.168.70.1" multiplier="10" dampen="5s"
```

Value	Description
MQHA_Ping	The name of this primitive. This is user selectable.
ocf	The class of the resource agent.
pacemaker	The provider of the resource agent is pacemaker. The combination of class and provider tells Pacemaker where the resource agent script is located. In this case it is located in /usr/lib/ocf/resource.d/pacemaker.
ping	The name of the resource agent script.

Here is an explanation of the params statement:

Name	Value	Description
host_list	192.168.70.1	This is the IP address of the network host or IP address you wish to monitor. In this solution the network gateway IP address is being monitored.
multiplier	10	A number used to help set a score that will trigger a failover when a problem is found.
dampen	5s	The number of seconds to wait before updating the CIB after a ping.

Here is a look at the Ping Clone resource:

clone MQHA_Ping_clone MQHA_Ping \

meta clone-max="2" target-role="Started

Value	Description
MQHA_Ping_clone	The name of this resource. This is user selectable.
MQHA_Ping	The resource being cloned.
meta attributes: Clone-max=2 target-role-“started”	 The number of instances of ping in the cluster. The state the resource should be in.

Constraints

Constraints tell Pacemaker where resources can run, the order resources must be started, etc.

In this solution only two types of constraints are used. They are: location and order constraints.

Location constraints tell Pacemaker where a resource can run. They also supply a score that is used by Pacemaker to help determine where resources should be located in failover situation.

```
location MQHA_Filesystem_loc1 MQHA_Filesystem 100: lac0017
location MQHA_Filesystem_loc2 MQHA_Filesystem 100: litsmg05
location MQHA_IPaddr_loc1 MQHA_IP_addr 100: lac0017
location MQHA_IPaddr_loc2 MQHA_IP_addr 100: litsmg05
location MQHA_loc1 MQHA 100: lac0017
location MQHA_loc2 MQHA 100: litsmg05
location MQHA_mqm_loc1 MQHA_mqm 100: lac0017
location MQHA_mqm_loc2 MQHA_mqm 100: litsmg05
```

In this solution the MQHA_Filesystem resource can be run on either node lac0017 or node litsmq05. A score of 100 is used. This value is somewhat arbitrary. In testing it would found that using a score of 100 produced the desired actions. The score helps determine if there is a preference for the resource to be started on one node rather than another node. In this solution all location constraints have identical scores.

A look at one of the location statements:

Keyword	location	The type of constraint.
ID	MQHA_Filesystem_loc1	The name of this location contraint. User selectable.
Resource	MQHA_Filesystem	The name of the resource this location constraint applies to.
Score	100	Arbitrary number that helps Pacemaker to determine where to place resources.
Node	lac0017	Node where the resource is allowed to run.

The other location constraints follow the same pattern.

Resource affected	**Applicable Location constraints**
The MQHA Resource Group	location MQHA_loc1 MQHA 100: lac0017 location MQHA_loc2 MQHA 100: litsmg05
The Filesystem resource	location MQHA_Filesystem_loc1 MQHA_Filesystem 100: lac0017 location MQHA_Filesystem_loc2 MQHA_Filesystem 100: litsmg05
The Queue Manager Resource	location MQHA_mqm_loc1 MQHA_mqm 100: lac0017 location MQHA_mqm_loc2 MQHA_mqm 100: litsmg05
The IP Address Resource	location MQHA_IPaddr_loc1 MQHA_IP_addr 100: lac0017 location MQHA_IPaddr_loc2 MQHA_IP_addr 100: litsmg05

Order constraints tell Pacemaker the order that resources must be started in. In this solution, the Filesystem resource must be started before staring the Queue Manager resource. Note that the IP Address resource can be started at any time in the migration or start up sequence and therefore does not need to be included on the order constraint. The Queue Manager resource will fail if the Filesystem resource is not started before it starts.

Here is the order constraint:

```
order MQHA_mqm_order : MQHA_Filesystem MQHA_mqm
```

Keyword	**Order**	**The type of statement.**
ID	MQHA_mqm_order	The name of this order constraint. User selectable.
Resources in startup order	MQHA_Filesystem MQHA_mqm	The resources this order constraint applies to, in the order that they are to be started.

Load the configuration into the CIB

To make the process of loading the configuration into the CIB as easy as possible, the configuration can be placed in a file in CRM CLI format along with the additional CRM commands required to implement the configuration. The file can be made executable and can be run as a script file.

Create the configuration file

The CRM commands are integrated with the CRM CLI configuration statements. The Linux echo command is used to feed the CRM CLI commands and the configuration to the crm command using the Linux pipe facility.

Here the layout for creating a file that will use the CRM CLI tool to load the configuration into the CIB:

```
echo ‘configure
<primitive_statements>
<group_statement>
<clone_statements>
<location_statments>
commit
quit’ | crm
```

Here is an analysis of the commands used:

```
echo ‘configure
```

Command or delimiter	Meaning
echo	Invoke the echo command
‘	The single quote delimits the text to be input to the echo command.
configure	The name of the configuration. In the example the name MQHA_config is used.

This represents the actual configuration.

```
<primitive_statements>
<group_statement>
<clone_statements>
<location_statments>
```

```
commit
quit’ | crm
```

Command or delimiter	Meaning
commit	
quit	The quit command indicated the end of the crm command string.
‘	The single quote delimits the text string being processed by the echo command.
|	The pipe character. This feeds the output from the echo command as input to the command specified.
crm	Invoke the crm command. The crm command receives the output of the echo command as input.

The first step to creating the configuration file is to copy and paste the configuration contained in the frame below to a file on your Linux system. After you have created the file you, will need to edit it. The footnotes indicate the settings that must be changed in order to tailor the configuration to match your environment. The footnote numbers will be copied when you do the copy and paste, so they will need to be removed. The single quotes wrapping the echo command’s text string may be changed to “.”s if you are using a Windows workstation and will need to be set back to single quotes. The configuration file has to be run on only one system in the cluster.

```
echo ‘configure
primitive MQHA_Filesystem ocf:heartbeat:Filesystem \
       operations $id="MQHA_Filesystem-operations" \
       op monitor interval="120" timeout="60" \
params device="/dev/disk/by-path/ccw-0.0.8092-part1" directory="/MQHA" fstype="ext3" [1]
primitive MQHA_IP_addr ocf:heartbeat:IPaddr \
       params ip="192.168.70.16" [2]
primitive MQHA_Ping ocf:pacemaker:ping \
       params host_list="192.168.70.1" multiplier="10" dampen="5s" [3]
primitive MQHA_mqm lsb:mqm \
       operations $id="MQHA_mqm-operations" \
       op monitor interval="60" timeout="15" \
       meta is-managed="true"
group MQHA MQHA_Filesystem MQHA_mqm MQHA_IP_addr \
       meta target-role="started"
clone MQHA_Ping_clone MQHA_Ping \
       meta clone-max="2" target-role="Started"
location MQHA_Filesystem_loc1 MQHA_Filesystem 100: lac0017 [4]
location MQHA_Filesystem_loc2 MQHA_Filesystem 100: litsmg05 [5]
location MQHA_IPaddr_loc1 MQHA_IP_addr 100: lac0017 4
location MQHA_IPaddr_loc2 MQHA_IP_addr 100: litsmg05 5
location MQHA_loc1 MQHA 100: lac0017 4
location MQHA_loc2 MQHA 100: litsmg05 5
location MQHA_mqm_loc1 MQHA_mqm 100: lac0017 4
location MQHA_mqm_loc2 MQHA_mqm 100: litsmg05 5
order MQHA_mqm_order : MQHA_Filesystem MQHA_mqm
commit
quit’ | crm
```

Make the configuration file executable

Run the Linux command:

chmod +x <your_config_file>

[1] In this line you will need to change the device settings and possibly the directory setting if you chose a mountpoint that differs from the one used in the sample solution.
[2] Specify the IP address used by the Queue Manager.
[3] Specify the IP address you monitor to determine the health of your network. The gateway address is recommended.
[4] Specify the node name of the first node in your cluster.
[5] Specify the node name of the second node in your cluster.

Execute the configuration file:

Execute the config file:

```
./<your_config_file>
```

Verify the configuration

The crm_verify command can be used to verify the integrity of the newly loaded configuration.

Run the command as follows:

```
crm_verify –L
```

The “-L” option specifies that the live configuration is to be examined. When there are no errors, no response will be given to the command.

Summary

If the configuration was entered with no errors, it should become active as soon as it is entered. Assuming that WebSphere MQ was installed correctly, the Queue Manager should be started as well as the related resources.

This configuration is easily expanded. It would be fairly easy to add a resource for the MQ Listener.

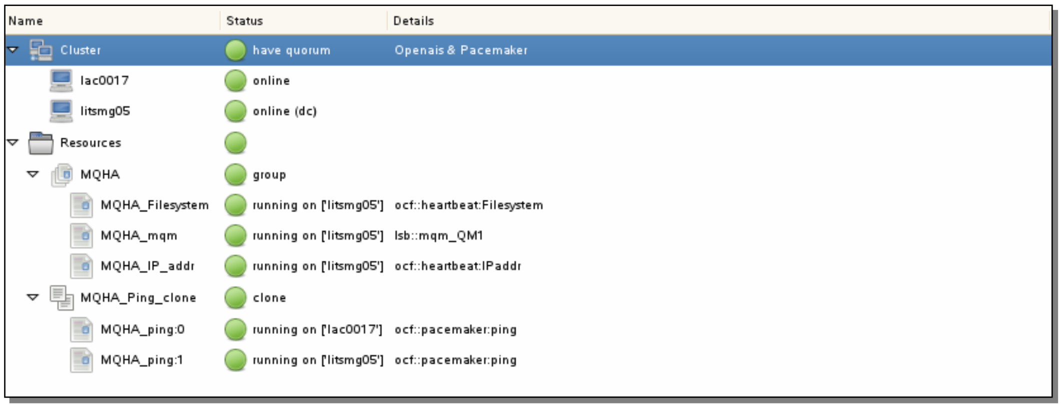
Since the mqm_<queue_manager_name> is specific to a single Queue Manager instance, it can easily be modified to supply optional parameters such as port number, and so forth.

The testing done with this solution included a simulated network failure, a total node failure (reboot), and a Queue Manager failure. In all cases HAE took between 1 and 2 minutes to detect the problem and execute the failover procedure.

In testing the network was stopped using the "rcnetwork stop" command on the z/VM console to simulate a network failure on that system. A total system failure was produced by doing a reboot of the Linux system from the z/VM console. A Queue Manager failure was generated by simply forcing the Queue Manager to end prematurely using the kill command. In all cases HAE took between 1 and 2 minutes to detect the problem and execute the failover procedure.

Management view of active configuration

This is Management view in the crm_gui tool:



Name	Status	Details
Cluster	have quorum	Openais & Pacemaker
lac0017	online	
litsmg05	online (dc)	
Resources		
MQHA	group	
MQHA_Filesystem	running on ['litsmg05']	ocf::heartbeat:Filesystem
MQHA_mqm	running on ['litsmg05']	lsb::mqm_QM1
MQHA_IP_addr	running on ['litsmg05']	ocf::heartbeat:IPaddr
MQHA_Ping_clone	clone	
MQHA_ping:0	running on ['lac0017']	ocf::pacemaker:ping
MQHA_ping:1	running on ['litsmg05']	ocf::pacemaker:ping

ZSW03187-USEN-00