WebSphere MQ for HP-UX - Implementing with Multi-Computer/ServiceGuard

Version 2.0

30th December 2002

Andrew Ford
MQSeries Development
IBM UK

Take Note!

Before using this report be sure to read the general information under "Notices".

Second Edition, December 2002

This edition applies to Version 2.0 of *MQSeries for HP-UX - Implementing with MCSG*, and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

This SupportPac is intended to help the customer or IBM systems engineer configure MQSeries for HP-UX V5.2 or V5.3 in a highly available manner using HP-UX Multi-Computer/ServiceGuard (MC/ServiceGuard) product. The information in this report is not intended as the specification of any programming interfaces that are provided by MQSeries or MC/ServiceGuard.

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

MC/ServiceGuard certification for this product was carried out by Hewlett Packard and any questions about MC/ServiceGuard should be directed to your Hewlett Packard representative.

While the information may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.

The data contained in this report was determined in a controlled environment, and therefore results obtained in other operating environments may vary significantly.

The following paragraph does not apply to any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact Laboratory Counsel, Mail Point 151, IBM United Kingdom Laboratories, Hursley Park, Winchester, Hampshire SO21 2JN, England. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Trademarks

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

IBM
MQ
MQSeries

The following are trademarks of Hewlett Packard in the United States, or other countries, or both:

HP-UX
Multi-Computer Service Guard (MC/ServiceGuard)

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Acknowledgments

This SupportPac is based upon high availability SupportPacs written by Mark Taylor and Graham Wallis for AIX and Solaris.

Summary of Changes

Date	Changes
October 2001	Initial release
December 2002	Updated to support Websphere MQ V5.3

Bibliography

The expected audience for this document is MC/ServiceGuard and MQSeries trained professionals. It is expected that the reader is familiar with the concepts and activities involved in setting up and running MC/ServiceGuard clusters and MQSeries queue managers. The following sources of information may be useful in this regard:

- MC/ServiceGuard product manuals
- MQSeries product manuals, especially
  - MQSeries System Administration, SC33-1873-01
  - MQSeries Command Reference, SC33-1369-11

The MQSeries book is available online under

http://www.ibm.com/software/ts/mqseries/library/manualsa/index.htm#Latest family books

Chapter 1. Introduction

Concepts

HP-UX is Hewlett Packard's UNIX operating system which runs on its 9000 series systems.

Multi-Computer/ServiceGuard (MC/ServiceGuard) is a control application that can link HP-UX servers into highly available clusters. Clustering servers enables parallel access to data, which can help provide the redundancy and fault resilience required for business-critical applications.

MQSeries for HP-UX provides asynchronous message and queuing capabilities with assured, once-only delivery of messages.

By using MQSeries for HP-UX and MC/ServiceGuard for HP-UX together, it possible to further enhance the availability of MQSeries queue managers. With a suitably configured MC/ServiceGuard cluster, it is possible for failures of power supplies, nodes, disks, disk controllers, networks, network adapters or queue manager processes to be detected and automatically trigger recovery procedures to bring an affected Queue Manager back online as quickly as possible.

This SupportPac provides notes and sample scripts to assist with the installation and configuration of MQSeries for HP-UX in an MC/ServiceGuard environment. Used in conjunction with the documentation for HP-UX, MC/ServiceGuard and MQSeries, it shows how to create and configure MQSeries queue managers such that they are amenable to operation within an MC/ServiceGuard cluster. It also shows how to configure MC/ServiceGuard to take control of such queue managers.

Certification

The certification process for MQSeries with MC/ServiceGuard was carried out by the Hewlett Packard Partner Technology Access Center. Included with this service pack is a copy of the test reports they generated, it contains a number of MC/ServiceGuard set up scripts that could be used as templates for creating your own implementation. The document also expands on Hewlett Packard specific areas that are not covered in this document. The document for the testing of MQSeries V5.2 is called *mc6bcertHP52.doc*, the document for Websphere MQ V5.3 is called *mc6bcert53.doc.*

Definition of the word *"cluster"*

The word "cluster" has a number of different meanings within the computing industry. Throughout this document, unless explicitly noted otherwise, the word "cluster" is used to describe an "MC/ServiceGuard cluster", which is a collection of nodes and resources (such as disks and networks) which cooperate to provide high availability of services running within the cluster. It is worth making a clear distinction between such an "MC/ServiceGuard cluster" and the use of the phrase "MQSeries Cluster", which refers to a collection of queue managers which can allow access to their queues by other queue managers in the cluster. The relationship between these two types of cluster is described in "Relationship to MQSeries Clusters" later in this chapter.

Functional Capabilities

Cluster Configurations

This SupportPac can be used to help set up either **standby** or **takeover** configurations, including mutual takeover where all cluster nodes are running MQSeries workload.

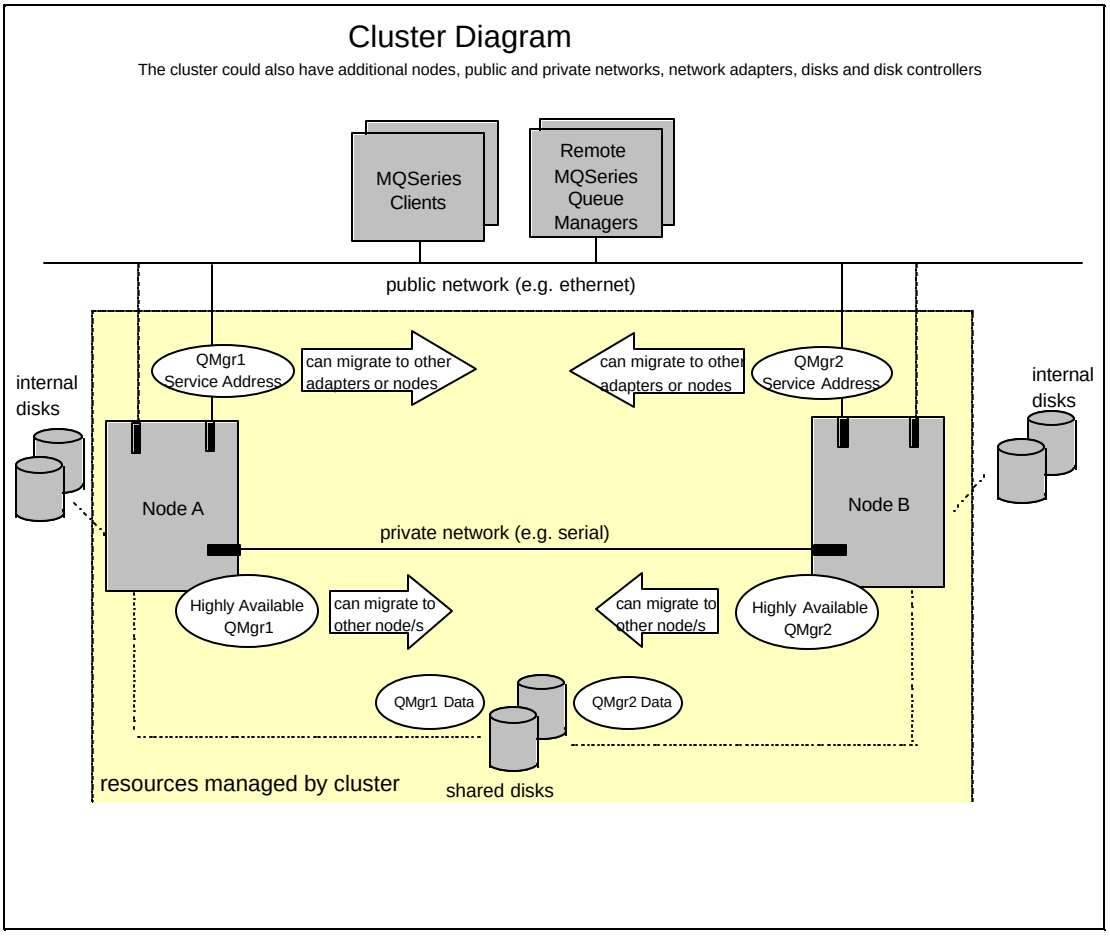
A **standby** configuration is the most basic cluster configuration in which one node performs work whilst the other node acts only as standby. The standby node is referred to as the "adoptive node" and does not perform work; this configuration is sometimes called "cold standby". Such a configuration requires a high degree of hardware redundancy. To economise on hardware, it is

possible to extend this configuration to have multiple worker nodes with a single standby node, the idea being that the standby node can take over the work of any worker node. This is still referred to as a standby configuration and sometimes as an "N+1" configuration.

A **takeover** configuration is a more advanced configuration in which all nodes perform some kind of work and critical work can be taken over in the event of a node failure. This type of cluster configuration is sometimes referred to as "Active/Active" to indicate that all nodes are actively processing critical workload.

In an “Active/Active” configuration all nodes perform highly available (movable) work, and consideration must be given to the peak load which may be placed on any node which can take over the work of other nodes. Such a node must possess sufficient capacity to maintain an acceptable level of performance.

Cluster Diagram



Relationship to MQSeries Clusters

MQSeries Clusters reduce administration and provide load balancing of messages across instances of cluster queues. They also offer higher availability than a single queue manager, because following a failure of a queue manager, messaging applications can still access surviving instances of a cluster queue. However, MQSeries Clusters (alone) will not provide automatic detection of queue manager failure and automatic triggering of queue manager restart or failover. MC/ServiceGuard clusters provide these features. The two types of cluster can used together to good effect.

MQSeries Clients

MQSeries Clients which are communicating with a queue manager that may be subject to a restart or failover should be written to tolerate a broken connection and should repeatedly attempt to reconnect.

User Exits

It is also possible to define optional user exits which are invoked just after a queue manager is brought online (started) and just before a queue manager is taken offline (ended). These allow you to start and stop additional processes, or provide notification of these cluster actions to other systems. This is described in detail in "Chapter 3. Configuration - Step 5. Configure the Application Server".

Specified Operating Environment

Hardware Requirements

The minimum hardware requirements for a cluster are at least two HP-UX servers, a shared disk enclosure, a private (non-IP) network (e.g. serial) and a public (IP) network. Refer to the MC/ServiceGuard Planning Guide for details of how to select appropriate disk technologies and networks. As described in the Planning Guide, it is likely that you will want to include additional networks and network adapters to eliminate these single points of failure. The hardware requirements for a cluster which will support MQSeries are the same as the above, but you should plan carefully the layout of filesystems and disks. Refer to "Chapter 3. Configuration" of this document for assistance in planning the filesystem layout.

The example scripts included in this SupportPac were tested by Hewlett Packard on a pair of HP9000 `A' class servers connected to a shared disk subsystem. Communication between the servers was via Ethernet and serial networks.

Please note that the certification carried out by Hewlett Packard is ONLY for the software listed in the requirements below, this does not mean tht other combinations will work only that they have not been certified to work.

Software Requirements

Versions of software tested are:

- HP-UX Version 11.0 withMC/ServiceGuard Version 11.12 and MQSeries for HP-UX Version 5.2
- HP-UX Version 11i with MC/ServiceGuard Version 11.14 and Websphere MQ for HP-UX Version 5.3

Networks

The SupportPac has been tested with TCP/IP public networks, and serial private networks. TCP/IP networks could be used for both public and private networks.

Chapter 2. Installation

HP-UX, MQSeries and MC/ServiceGuard should already be installed, using the normal procedures. It is recommended that you install MQSeries and MC/ServiceGuard onto internal disks on each of the nodes. It is important that under normal operating conditions you are running identical versions of HP-UX, MC/ServiceGuard and MQSeries software on all cluster nodes.

Ignore the advice in the MQSeries documentation about creating separate /var/mqm and /var/mqm/log filesystems. This is not the preferred configuration in an MC/ServiceGuard environment. See under "Chapter 3. Configuration" for more details.

When installing MQSeries in a cluster, it is essential that the "mqm" username and "mqm" groupname have been created and each have the same numeric value on all of the cluster nodes.

Installing the SupportPac

Create the /MQHA/bin directory on each of the cluster nodes and ensure it is writable by the mqm userid. Download the SupportPac onto each of the cluster nodes and untar it into /MQHA/bin. This is the working directory assumed by example scripts. You could use a different location if you wanted to, but you'd need to change the example scripts.

The SupportPac contains the following files.

Scripts

- hacrtmqm - creates a queue manager which is amenable to MC/ServiceGuard operation.
- halinkmqm - modifies the directory structure of a queue manager for MC/ServiceGuard operation.
- hadltmqm - deletes a queue manager created and modified by the above scripts.
- hamqm_start - robust start method for a queue manager, called by MC/ServiceGuard for start and restart.
- hamqm_stop - robust stop method for a queue manager, called by MC/ServiceGuard for stop and cleanup.

Internal utilities:-

- hamqm_start_su - used internally by hamqm_start, provides start sequence logic.
- hamqm_stop_su - used internally by hamqm_stop, provides stop sequence logic.
- hamqm_applmon_su - used to monitor the health of the queue manager under MC/ServiceGuard.
- hamqm_running - used by other scripts for a quick check of queue manager status.
- hamqm_status - used by other scripts for a quick check of queue manager status.
- hamqm_qm_directory - used by other scripts to locate the queue manager data directory.

Chapter 3. Configuration

The unit of failover in MC/ServiceGuard is a package. A package has to contain all the processes and resources needed to deliver a highly available service and ideally should contain only those processes and resources. This approach maximises the independence of each package, providing flexibility and minimising disruption during a failure or planned maintenance. The smallest unit of failover of MQSeries is a queue manager, since you cannot move part of a queue manager without moving the whole thing. It follows that the optimal configuration is to place each queue manager in a separate package, with the resources upon which it depends. The package should therefore contain the shared disks used by a queue manager, which should be in a volume group reserved exclusively for the package, the IP address used to connect to the queue manager (the package IP address) and an application server which represents the queue manager.

You *could* put multiple queue managers into a package, but if you did so they would all have to failover to another node together, even if the problem causing the failover were confined to one queue manager. This would cause unnecessary disruption to the other queue managers.

It is assumed that if mirroring or RAID are used to provide protection from disk failures then references in the following text to physical disks should be taken to mean the disk or *group of disks* that are being used to store the data being described.

A queue manager that is to be used in an MC/ServiceGuard cluster needs to have its logs and data on shared disks, so that they can be accessed by a surviving node in the event of a node failure. A node running a queue manager must also maintain a number of files on internal disks. These files include files that relate to all queue managers on the node, such as /var/mqm/mqs.ini, and queue manager specific files that are used to generate internal control information. Files related to a queue manager are therefore divided between internal and shared disks.

Regarding the queue manager files that are stored on shared disk it would, in principle, be possible to use a single shared disk for all the recovery data (logs and data) related to a queue manager. However, for optimal performance, it is recommended practice to place logs and data in separate filesystems such that they can be separately tuned for disk i/o. The example scripts included in this SupportPac use separate filesystems. The layout is described in "Step 2. Configure the Shared Disks".

If the MC/ServiceGuard cluster will contain multiple queue managers then, depending on your chosen cluster configuration, two or more queue managers may need to run on the same node, due perhaps to a takeover. To provide correct routing of MQ channel traffic to the queue managers, you should use a different TCP/IP port number for each queue manager. The standard MQSeries port is 1414. It is common practice to use a range of port numbers immediately above 1414 for additional queue managers. Note that whatever port number you assign to a queue manager, that port needs to be consistently defined on all cluster nodes that may host the queue manager, and all channels to that queue manager need to refer to the port.

When configuring a listener for incoming MQ connections you can choose between inetd and runmqlsr. If you use inetd then you do not need to perform any start or stop of the listener from within the cluster scripts. If you use runmqlsr then you should refer configure a user exit as described in "Chapter 3. Configuration - Step 5. Configure the Application Server".

The example scripts and utilities provided in the SupportPac and the descriptions of the configuration steps deal with one queue manager at a time. For additional queue managers, repeat Steps 2 through 8.

Step 1. Configure the Cluster

The MC/ServiceGuard configuration is straightforward and should present no difficulties for a trained implementer.

Actions:

1. Create and configure the template ASCII package configuration file:
   - Set the PACKAGE_NAME,
   - Set the NODE_NAME,
   - Set the RUN_SCRIPT,
   - Set the HALT_SCRIPT,
   - Set the SERVICE_NAME,
   - Set the SUBNET.
2. Create and configure the template package control script:
   - Set VG, LV and IP,
   - Set the SUBNET,
   - Set the SERVICE_NAME,
   - Set up the customer_defined_run_cmds function to use the supplied hamqm_start script,
   - Set up the customer_defined_stop_cmds function to use the supplied hamqm_stop script.
3. Disable Node Fail Fast. This causes the machine to panic and is only necessary if MQSeries services are so tightly integrated with other services on the node that it is impractical to failover MQSeries independently.
4. Enable Package Switching so that packages can be moved between nodes

Step 2. Configure the shared disks

This step creates the volume group and filesystems needed for the queue manager. The suggested layout is based on the advice earlier that each queue manager should be put into a separate package. You should perform this step and the subsequent steps for each queue manager that you wish to make highly available.

So that this queue manager can be moved from one node to another without disrupting any other queue managers, you should designate a volume group containing shared disks which is used exclusively by this queue manager and no others.

For performance, it is recommended[1] that a queue manager uses separate filesystems for logs and data. The suggested layout therefore creates two filesystems within the volume group.

You can optionally protect each of the filesystems from disk failures by using mirroring or RAID, this is not shown in the suggested layout.

Mount points must all be owned by the mqm user.

[1] MQSeries SupportPac MP67 may be of interest.

You will need the following filesystems:

Per node:

/var on internal disks - this is a standard HP-UX filesystem which will already exist. You only need one of these per node regardless of the number of queue managers that the node may host. It is important that all queue managers that may run on this node use one filesystem for some of their internal control information and the example scripts designate /var for this purpose. With the suggested configuration, not much MQSeries data is stored in /var, so it should not need to be extended greatly.

Per queue manager:

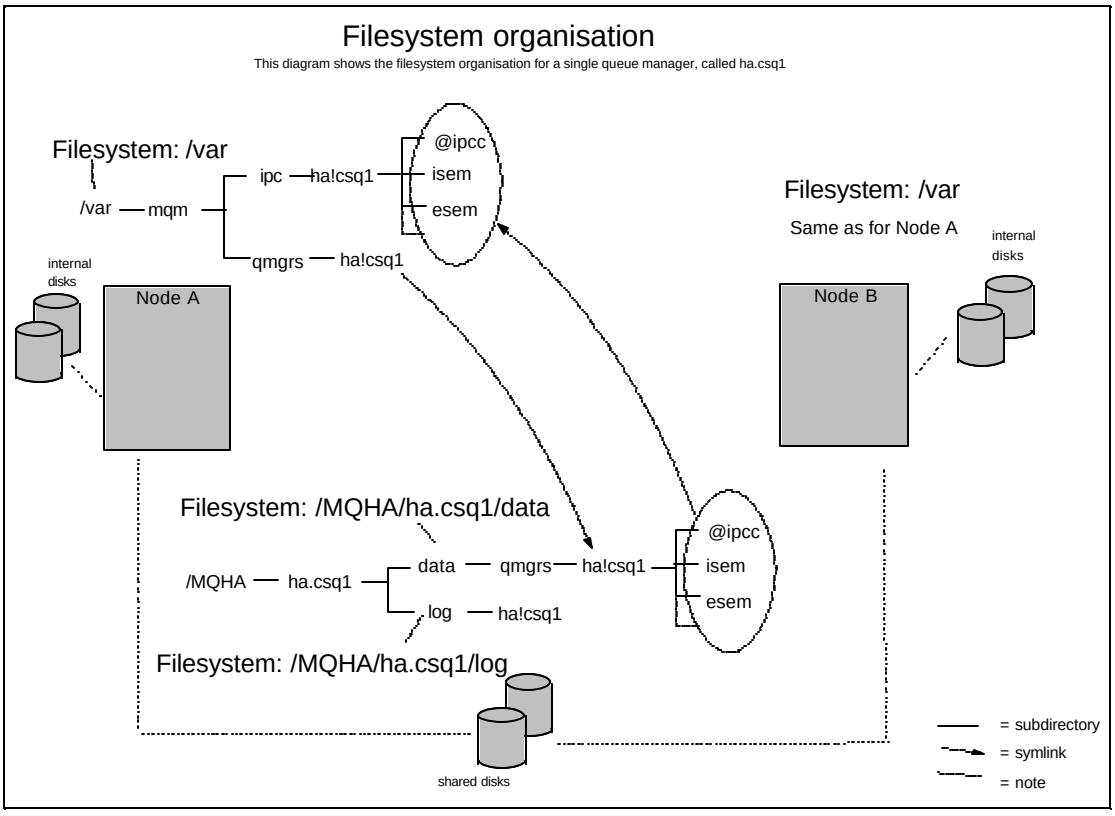
/MQHA/$qmgr/data on shared disks - this is where the queue manager data directory will reside. /MQHA is the top level directory used in the example scripts.

/MQHA/$qmgr/log on shared disks - this is where the queue manager log directory will reside.

The filesystems are shown in the following diagram. The subdirectories and symlinks are all created automatically in the next step - you only need to create the filesystems that are on shared disk (e.g /MQHA/ha.csq1/data and /MQHA/ha.csq1/log), then proceed to Step 3.

Actions:

1. Create the volume group that will be used for this queue manager's data and log files (e.g. /dev/vg01).
2. Create a logical volume in this volume group (e.g. /dev/vg01/$qmgr).
3. Mount the logical volume to be shared at /MQHA/$qmgr.
4. Create the /MQHA/$qmgr/data and /MQHA/$qmgr/log filesystems using the volume group created above.
5. Unmount /MQHA/$qmgr.
6. Issue a *vgchange -a n /dev/vg01/$qmgr.*
7. Issue a *vgchange -c y /dev/vg01/$qmgr.*
8. Issue a *vgexport -m /tmp/mq.map -s -p -v /dev/vg01.*
9. Copy the mq.map file created in 8 above to /tmp the adoptive node.
10. On the adoptive node create a the same logical volume and volume group as in steps 1 and 2 above.
11. Issue a *vgimport -m /tmp/mq.map -s -v /dev/vg01*
12. Mount the volume group on the adoptive to check the configuration is correct.



Step 3. ServiceGuard Configuration

The following steps show how to configure a cluster under MC/ServiceGuard and how to configure nodes into the cluster. This information was supplied by Hewlett Packard and you should consult your MC/ServiceGuard documentation for a full explanation of the commands. The example commands are to set up a cluster of 2 machines called ptaca2 and ptaca3. Examples of the files mentioned below are contained in the appendices of the Hewlett Packard documentation supplied with this SupportPac

Actions to configure the cluster:

1. Create the ascii template file:

   cmquerycl -v -C /etc/cmcluster/cluster.ascii -n ptaca2 -n ptaca3

2. Modify this template to reflect the environment and then verify the cluster configuration:

   cmcheckconf -v -C /etc/cmcluster/cluster.ascii

3. Apply the configuration file, this creates the cluster and automatically distributes the "cmclconfig" file throughout the cluster:

   cmapplyconf -v -C /etc/cmcluster/cluster.ascii

4. Start and stop the cluster to check that the above has worked.

   cmruncl -v -n ptaca2 -n ptaca3

   cmviewcl -v

cmhalt -f -v

cmruncl -n ptaca2 -n ptaca3

The next steps show how to create a package (called mq1) on the first node.

Actions to configure the ServiceGuard package on the first node:

1. Create and tailor the mq1 package for you environment:

   cd /etc/cmcluster

   mkdir mq1

   cmmakepkg -s mq1.conf

2. Edit the mq1.conf file to reflect your environment.
3. Change into the mq1 directory created above.
4. Issue the following command:

   cmmakepkg -s mq1.cntl

5. Shutdown the cluster:

   cmhaltcl -f -v

6. Distribute the configuration files:

   cmapplyconf -v -C /etc/cmcluster/cluster.ascii -P /etc/cmcluster/mq1/mq1.conf

Actions to test the cluster and package startup:

1. Shutdown MQSeries (if it is running).
2. Unmount all logical volumes in the volume group you created earlier (e.g. /dev/vg01)
3. Deactivate the volume group
4. Start the cluster:

   cmruncl

5. Check that the package has started:

   cmviewcl -v

Actions the assign the dynamic IP address of the package:

1. Halt the MQSeries package:

   cmhaltpkg mq1

2. Edit the mq1.cntl script to add the package's IP address.
3. Restart the MQSeries package:

   cmrunpkg -v mq1

4. Check the package has started and has clients:

   cmviewcl -v

Actions to add the second node to the cluster:

1. Edit the mq1.conf file and add the following line:

   NODE_NAME ptaca2

2. Apply the new configuration:

   cmapplyconf -v -C /etc/cmcluster/cluster.ascii -P mq1.ascii

3. Halt the cluster:

   cmhaltcl -f -v

4. Restart the cluster:

   cmruncl -v

Actions to test package switching:

1. Halt the mq1 package:

   cmhaltpkg mq1

2. Start the mq1 package on the machine ptaca3:

   cmrunpkg -n ptaca3 mq1

3. Enable package switching for the mq1 package on ptaca3:

   cmmodpkg -e mq1

4. Halt the mq1 package:

   cmhaltpkg mq1

5. Start the mq1 package on the machine ptaca2:

   cmrunpkg -n ptaca2 mq1

6. Enable package switching for the mq1 package on ptaca2:

   cmmodpkg -e mq1

Step 4. Create the Queue Manager

Select a node on which to create the queue manager. The choice is fairly arbitrary, but you should use one of the nodes which may host the queue manager.

When you create the queue manager, it is strongly advised that you should use the hacrtmqm script included in the SupportPac. It is possible to create the queue manager manually, but using hacrtmqm will save a lot of effort. For example, hacrtmqm moves and relinks some subdirectories. The move and relink of these subdirectories is to ensure smooth coexistence of queue managers which may run on the same node.
hacrtmqm has a companion script, called hadltmqm, which can be used to delete a queue manager which has been relinked in this way, leaving the system tidy. Refer to Step 8 for more information.

Actions:

1. Select a node on which to perform the following actions
2. If not already mounted mount the filesystems on the selected node.
3. Create the queue manager on this node, using the hacrtmqm script described in the first of the following frames.
4. Start the queue manager manually, using the strmqm command.
5. Create any queues and channels.
6. Test the queue manager.
7. End the queue manager manually, using endmqm.

8. On the other nodes, which may takeover the queue manager, run the halinkmqm script described in the second following frames. Unmount the shared disk and remount it in turn on the machine(s) that will need to run halinkmqm. When complete remount the shared disk onto the cluster node that will initially run the queue manager.

hacrtmqm command

The hacrtmqm command will create the queue manager and will ensure that its directories are arranged to allow for High Availability (HA) operation.

hacrtmqm makes use of two environment variables to determine where the data and log directories should be created:

```
export MQHAFSDATA="/MQHA/$qmgr/data"
export MQHAFSLOG="/MQHA/$qmgr/log"
```

The invocation of the hacrtmqm command uses exactly the same parameters that you would normally use for crtmqm. You do not need to set MQSPREFIX or specify the -ld parameter for the log directory - these will be both be handled automatically by hacrtmqm. You must be root to run the hacrtmqm command.

Syntax

hacrtmqm <crtmqm parameters>

Parameters

- crtmqm parameters - are exactly the same as for the regular MQSeries crtmqm command

Example:

```
MQHAFSDATA="/MQHA/ha.csq1/data"
MQHAFSLOG="/MQHA/ha.csq1/log"

hacrtmqm -c "Highly available queue manager" ha.csq1
```

halinkmqm command

Internally, hacrtmqm uses a script called halinkmqm to relink the subdirectories used for IPC keys and create a symlink from /var/mqm/qmgrs/$qmgr to the /MQHA/$qmgr/data/qmgrs/$qmgr directory. As shown at the end of hacrtmqm, you need to run halinkmqm on the cluster nodes which will act as standby nodes for this queue manager. Do not run halinkmqm on the node on which you created the queue manager with hacrtmqm - it has already been run there.

The halinkmqm command will create the necessary symlinks and will insert a stanza for the queue manager into the mqs.ini file.

Syntax

halinkmqm <qmgr name> <mangled qmgr directory name> <qmgr data directory name>

Parameters

- qmgr name - The name of the queue manager as you specified it to hacrtmqm (e.g. ha.csq1)
- mangled qmgr directory name - The name of the directory under /var/mqm/qmgrs/ which closely resembles the qmgr name.
- qmgr data directory name - The directory you selected for the queue manager data, and to which you set MQHAFSDATA before issuing hacrtmqm. (e.g. /MQHA/ha.csq1/data).

Example:

```
halinkmqm ha.csq1 ha!csq1 /MQHA/ha.csq1/data
```

Step 5. Configure the Application Server

The queue manager is represented within the package by an application server. The SupportPac includes example application server start and stop methods which allow MC/ServiceGuard to start and end the queue manager, in response to cluster commands or cluster events. These start and stop methods are described below.

Both the example start and stop scripts allow you to specify a user exit to be invoked just after the queue manager is brought online or just before it is taken offline. The provision of the user exit is optional. The purpose of the user exit is to allow you to start or stop additional processes following the start of a queue manager or just before ending it. For example, you may wish to start a listener, a trigger monitor or a command server. You may also want to start some other application that uses the queue manager. You may wish to send a notification to an application, a monitoring system, or a human administrator. The creation of the user exit is described in Step 6 listed below.

To allow MC/ServiceGuard to check that the MQSeries queue manager is still responding the hamqm_applmon_script is provided in the SupportPac. This script uses the MQSeries ping command to check on the queue manager. If there is no response it performs a quick check to ensure that the queue manager is not being started up before returning an error code. This queue manager checking script is placed under the control of MC/ServiceGuard and MC/ServiceGuard will run the script to check on he queue manager.

Actions:

1. The start and stop scripts contained in the SupportPac may be used unmodified, or may be used as a basis from which you can develop customized scripts. The examples are called **hamqm_start** and **hamqm_stop** and are described in the following frames.

hamqm_start

The example start script is called hamqm_start. This script is robust in that it does not assume anything about the state of the QM on entry. It accepts a single command line parameter which is the queue manager name, so when you define the start command in MC/ServiceGuard, include the parameter.

Example

```
"/MQHA/bin/hamqm_start  ha.csq1"
```

To define this command so as it can be run as the user mqm under ServiceGuard control create a wrapper function in the package control script (/etc/cmcluster/mq1/mq1.cntl) that contains the following line:

su mqm -c "/MQHA/bin/hamqm_start_su $qmgr"
Where $qmgr is the name of the queue manager you wish MC/ServiceGuard to start.

For an example function see Appendix 4 of the Hewlett Packard documentation supplied with this SupportPac.

hamqm_stop

The example stop script is called hamqm_stop. The stop script accepts two command line parameters, the first is the queue manager name, the second parameter is the timeout (in seconds) to use on each of the levels of severity of stop. The stop script attempts to shutdown the queue manager gracefully. If this takes longer than the specified time, it attempts progressively more severe stop methods. When you define the stop command in MC/ServiceGuard you should include the parameters.

Example

```
"/MQHA/bin/hamqm_stop  ha.csq1 30"
```

If the stop command is defined in this way, then when MC/ServiceGuard has reason to invoke the stop command, it will issue an immediate stop of the queue manager (endmqm -i) and will allow up to 30 seconds for it to complete. If it does not complete within that time, a preemptive stop (endmqm -p) is issued, and up to a further 30 seconds is allowed. If the queue manager still hasn't stopped within that time, then the hamqm_stop command terminates the queue manager forcefully. It is clearly better if the queue manager can be shutdown gracefully, because a clean shutdown will lead to a faster restart and is less disruptive to clients and applications.

To define this command so that can be run as the user mqm under Service Guard control create a wrapper function in the package control script (/etc/cmcluster/mq1/mq1.cntl) that contains the following line:

```
su mqm -c "/MQHA/bin/hamqm_stop_su $qmgr 30"
```
Where $qmgr is the name of the queue manager you wish MC/ServiceGuard to stop.

For an example function see Appendix 4 of the Hewlett Packard documentation supplied with this supportpac.

2. Create a monitoring script file for use by MC/ServiceGuard this script can initially be a renamed copy of the /MQHA/bin/hamqm_applmon_su script supplied with this supportpac, which checks the health of a named queue manager using the MQSeries ping command. You may wish to add extra features to your copy of this script to check that other processes essential to your MQSeries environment are functioning correctly. If you wish to have one common script for all queue managers under MC/ServiceGuard control then they can all use the same copy of the script, however if you wish the monitoring of different queue managers to check different things then it is suggested that you call your script $qmgr.mon after the queue manager it is used for. The monitoring script should ideally be run regularly but should be a short lived process doing a quick check or checks and then disappearing to avoid slowing down MQSeries and the machine it runs on

   The monitoring script is then called from the package control script for the queue manager *(/etc/cmcluster/mq1/mq1.cntl)* . This is achieved by adding the following lines to mq1.cntl file created during Service Guard configuration:

```
SERVICE_NAME[n]=mq1
SERVICE_COMMAND[n]="su mqm -c \"/etc/cmcluster/mq1/mq1.mon QMA \""
```

   Where mq1.mon is a renamed version of hamqm_applmon_script,
   mq1 is the name of the MQSeries package being monitored,
   QMA is the name of the queue manager to monitor,
   n is the number of the service being monitored

.

Step 6. Create a user exit (optional).

Optionally, create a user exit in */MQHA/bin/rc.local* as described in the following frame

/MQHA/bin/rc.local

If you decide to make use of the user exit capability, create a script called /MQHA/bin/rc.local. The script can contain anything you like. It is passed two parameters, the first is the queue manager name and the second is a string describing the phase of processing, which will be set to either "pre_offline" or "post_online". Ensure that the rc.local script is executable. The start and stop scripts will invoke the rc.local script as user "mqm". It is recommended that you test the exit by invoking it manually, before putting it under cluster control.

Syntax

```
/MQHA/bin/rc.local <qmgr  name> <phase>
```

Parameters

- qmgr name - the name of the queue manager which this invocation relates to
- phase - either "pre_offline" or "post_online", to indicate what is about to happen or what has just happened.

The examples are written such that the script is invoked asynchronously (in the background). This is a conservative policy that aims to reduce the likelihood that an errant script could delay, possibly indefinitely, other cluster operations which need to be performed. The asynchronous invocation policy does have the disadvantage that the exit script cannot make any assumptions about the state of the queue manager, since it may change immediately after the script is invoked. The script should therefore be written in a robust style.

Step 7. Removal of Queue Manager from Cluster

Should you decide to remove the queue manager from the cluster, it is sufficient to remove the application server and monitoring script from the MC/ServiceGuard configuration. You may also decide to delete the package. **This does not destroy the queue manager**, which should continue to function normally, but under manual control.

Once the queue manager has been removed from the MC/ServiceGuard configuration, it will not be highly available, and will remain on one node. Other nodes will still remember the queue manager and you may wish to tidy up the other nodes. Refer to Step 8 for details.

Actions:

1. Delete the application server.
2. Delete the monitoring script.

3. Remove the filesystem, service label and volume group resources from the package.

Step 8. Deletion of Queue Manager

If you decide to delete the queue manager, then you should first remove it from the cluster configuration, as described in Step 7. Then, to delete the QM, perform the following actions.

Actions:

1. Make sure the queue manager is stopped, by issuing the endmqm command.
2. On the node which currently has the queue manager's shared disks and has the queue manager's filesystems mounted, run the hadltmqm script provided in the SupportPac.

hadltmqm command

The hadltmqm command will delete a queue manager. This will destroy its log files, and control files and *on the owning node* (only) will remove the definition of the queue manager from the /var/mqm/mqs.ini file. This is similar to the behaviour of the dltmqm command, which the hadltmqm command uses internally. The hadltmqm command deletes the symlinks used to locate the IPC subdirectories and deletes the subdirectories themselves.

Syntax

```
hadltmqm <qmgr name>
```

Parameters

- qmgr name - the name of the queue manager to be deleted

3. You can now destroy the filesystems /MQHA/$qmgr/data and /MQHA/$qmgr/log**.**
4. You can also destroy the volume group.
5. On each of the other nodes in the cluster,
   a. run the hadltmqm command as above, which will clean up the subdirectories related to the queue manager.
   b. Manually remove the queue manager stanza from the /var/mqm/mqs.ini file.

The queue manager has now been completely removed from the cluster and the nodes.

Chapter 4. Suggested Test

You may find that the following tests are helpful when determining whether your configuration is working as intended.

Create a queue manager, e.g. QM1, and put it under MC/ServiceGuard control, as defined in the preceding chapters.

Start the QM1queue manager and use runmqsc to define the following objects:

```
************************************************************
*                                                          *
* Create the queues for QMgr (QM1) clustered QMgr          *
*                                                          *
************************************************************

************************************************************
* Define the outbound xmit queue                           *
************************************************************
DEFINE QLOCAL(XMITQ1) +
        USAGE(XMITQ) +
        DEFPSIST(YES) +
        TRIGGER +
        INITQ(SYSTEM.CHANNEL.INITQ) +
        REPLACE

************************************************************
* Define the inbound/outbound message queue                *
************************************************************
DEFINE QREMOTE(QM1_INBOUND_Q) +
        RNAME(QM2_INBOUND_Q) +
        RQMNAME(QM2) +
        XMITQ(XMITQ1) +
        DEFPSIST(YES) +
        REPLACE

************************************************************
* Define the channels between QM1 <-> QM2                  *
************************************************************

* Channel 1
DEFINE CHANNEL(QM2_SDR.TO.QM1_RCV) +
        CHLTYPE(RCVR) +
        TRPTYPE(TCP) +
        HBINT(30) +
        REPLACE

* Channel 2
DEFINE CHANNEL(QM1_SDR.TO.QM2_RCV) +
        CHLTYPE(SDR) +
        TRPTYPE(TCP) +
        CONNAME(QM2_ip_address) +
        XMITQ(XMITQ1) +
        HBINT(30) +
        REPLACE
```

Create another "out-of-cluster" queue manager, e.g. QM2, start it and create the following objects:

```
*************************************************************
*                                                           *
* Create the queues for "out-of-cluster" QMgr QM2           *
*                                                           *
*************************************************************

*************************************************************
* Define the inbound message queue                          *
*************************************************************
DEFINE QLOCAL(QM2_INBOUND_Q) +
        DEFPSIST(YES) +
        REPLACE

*************************************************************
* Define the outbound xmit queue                            *
*************************************************************
DEFINE QLOCAL(XMITQ1) +
        USAGE(XMITQ) +
        DEFPSIST(YES) +
        INITQ(SYSTEM.CHANNEL.INITQ) +
        REPLACE

*************************************************************
* Define the outbound message queue                         *
*************************************************************
DEFINE QREMOTE(QM2_OUTBOUND_Q) +
        RNAME(QM1_INBOUND_Q) +
        RQMNAME(QM1) +
        XMITQ(XMITQ1) +
        DEFPSIST(YES) +
        REPLACE


*************************************************************
* Define the channels between QM2 <-> QM1                   *
*************************************************************

* Channel 1
DEFINE CHANNEL(QM2_SDR.TO.QM1_RCV) +
        CHLTYPE(SDR) +
        TRPTYPE(TCP) +
        CONNAME(QM1_ip_address) +
        XMITQ(XMITQ1) +
        HBINT(30) +
        REPLACE

* Channel 2
DEFINE CHANNEL(QM1_SDR.TO.QM2_RCV) +
        CHLTYPE(RCVR) +
        TRPTYPE(TCP) +
        HBINT(30) +
        REPLACE
```

On the machine running QM2, run a script similar to the following, which will prime the transmit queue with a number of messages.

```
#!/bin/sh

# $1 controls size of message buffer
# Actual amount sent is between $1 and 2*$1 KBytes
#
# e.g. QM2_put 10
#

rm -f message_buffer

# Construct the set of messages to send
SIZE=`expr $1 \* 1000`
MSG=0
date >> message_buffer
while [ $MSG -le $SIZE ]
do
    # double the previous buffer
    cp message_buffer .previous
    cat .previous .previous > message_buffer
    MSG=`ls -l message_buffer | awk '{ print $5 }'`
done
echo "Putting $MSG Bytes onto outbound queue"
cat message_buffer | /usr/lpp/mqm/samp/bin/amqsput QM2_OUTBOUND_Q QM2
```

Each line of text in the message_buffer file will be sent as an MQ message. Run initially with a small number of messages. The script will prime the transmission queue. When the transmission queue is primed, use runmqsc to start the QM2_SDR.TO.QM1_RCVR channel, and the messages will be transmitted to QM1and then routed via QM1's transmission queue back to QM2's inbound queue. When you are happy that your definitions are correct and you get end to end transmission of messages, run the script again with a much large number of messages to be sent. When the transmission queue is primed, note how many messages it contains. Once again, start the sender channel - but after only a few seconds of transmission of messages, reset the node on which QM1 is running so that it fails over to another cluster node. The QM1 queue manager should restart on the takeover node and the channels should be restarted. All the messages should be received at QM2's inbound queue. You could use runmqsc to inspect the queue, or run the amqsget sample to retrieve the messages into a file.

Chapter 5. Comments

If you have any comments on this SupportPac, please send them to:

E-mail:

aford@uk.ibm.com

Post:

Andrew Ford
MailPoint 211
IBM UK Laboratories Ltd.
Hursley Park
Winchester
Great Britain
SO21 2JN