Implementing MC/LockManager and Oracle Parallel Server

Bob Trojak
btrojak@telespectrum.com
TeleSpectrum Worldwide, Inc.
Phone: 610-878-7959

Introduction

This paper will discuss the MC/LockManager implementation performed at TeleSpectrum Worldwide, Inc. The paper will be divided into four sections:

Section One:	The Specifications.
Section Two	The Design.
Section Three:	Configuration and Implementation.
Section Four:	Lessons Learned

1. The Specifications

The initial specifications called for a High Volume Reporting Database and a Highly Available OLTP Database and an environment with the ability to expand. With these specifications in mind, a two-node ServiceGuard implementation was chosen. Initially two K-Class servers were thought to be sufficient to house these applications, but after all single points of failures were eliminated and an afterthought (backups), were considered, it was determined that two K-Class servers would not be sufficient and two V2250's were the hardware of choice. The K-Class has an I/O limitation due to available slots, but the V2250 has 24 pci slots available for expansion. After the initial ServiceGuard design was agreed upon, there were questions as to whether one or two databases would be a better fit. It was still necessary to provide a Highly Available environment, but the specifications had changed and it was not feasible to have two databases, since both database contained the same information. In steps Oracle Parallel Server and MC/LockManager. With OPS and MC/LockManager, we are able to take advantage of the processing power of two system running separate instances of oracle accessing the same data and provide a High Available Environment.

2. The Design

I believe that initial ***DESIGN and PLANNING*** is the key to a successful ServiceGuard/LockManager installation. With a successful design and planning stage, a successful implementation is well at hand. What information is necessary for you to begin planning your implementation? Below are some questions that need to be answered.

1. What are the system requirements?
2. What are the disk requirements?
3. What are the network requirements?

4. What are the applications?

We will take a look at each of these items on an individual basis, but there is one important item to keep in mind when reviewing these requirements, locating and eliminating all single points of failure. In the lessons learned section, I will show some instances where we did not exactly achieve this in the planning stages. Below I will use both text and diagrams to show how the final hardware configuration was achieved.

System Requirements

As described above, two V2250's were the systems of choice in December 1998. With up to 16 processors, 16 GB of RAM and 24 PCI slots available, the systems provided the expandability that TeleSpectrum required. Initially, two V2250's with 3 processors and 4GB of RAM each were sufficient to support the application. To eliminate a power single point of failure, external mirrored root disks in separate cabinets were implemented. Figure 1 below shows the hardware design at this point:

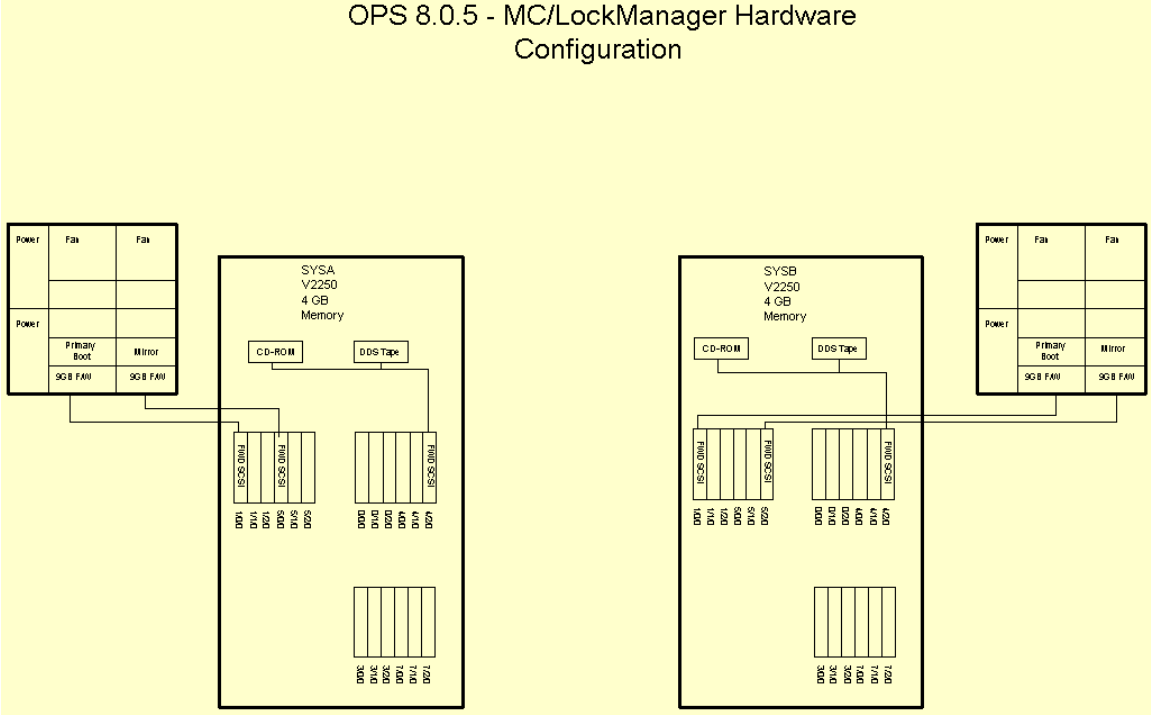

Figure 1.

As you can see, we don't have any single points of failure above. The root disks are mirrored across two channels eliminating the possibility of an interface card failure, cable failure or controller failure. Each processor and each system disk are attached to separate power sources.

Disk Requirements

Next, lets look at the disk requirements. Initially 600 GB of Highly Available disk was required. An EMC 3700 with 47GB useable (94GB raw) disks was chosen. The system was populated with 26 drives that is equivalent to 1.2 TB raw or 600 GB mirrored. Each system will connect into the EMC with 4 fibre channel interfaces. Below in Figure 2 is our hardware configuration including the EMC and fibre channel connections:

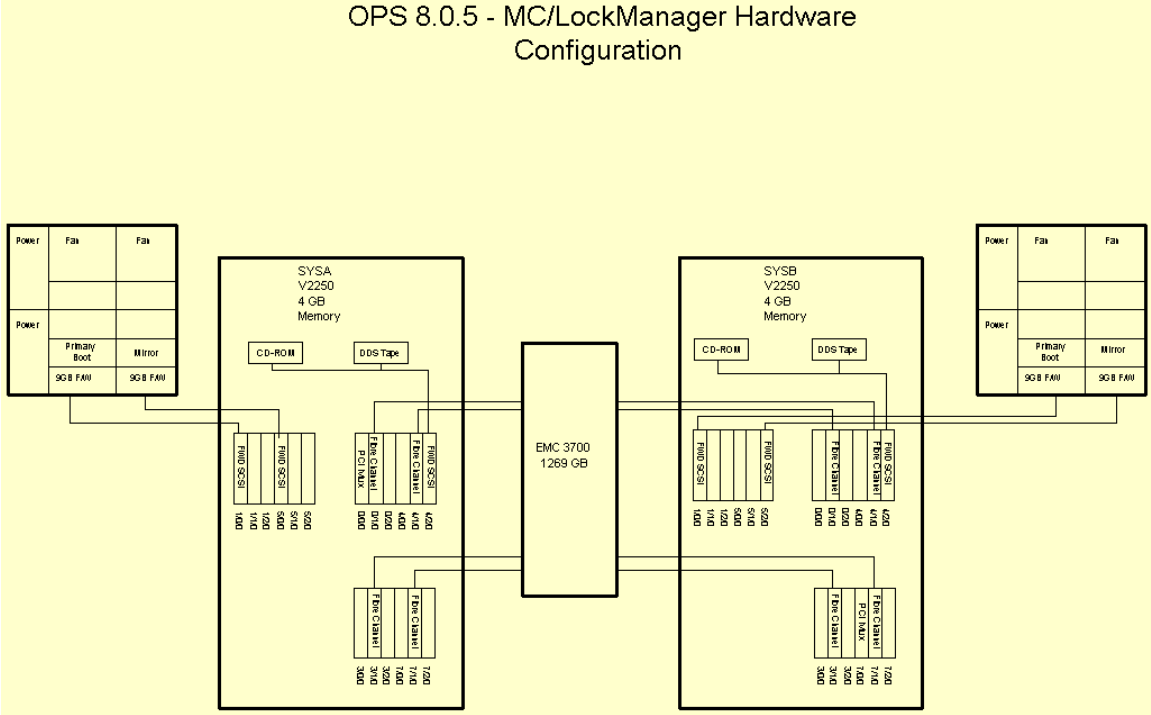

Figure 2.

Again to avoid single points of failure, 4 fibre channel connects via PV Links from each system are used. PV links allow data to flow across an alternate fibre channel connection in the case of a fibre channel interface card or cable failure. Also, Shared Logical Volume Manager that is available with MC/LockManager is used to allow both systems to access the same physical devices simultaneously. We will discuss Shared Logical Volume Manager later in the configuration section.

Network Requirements

The network requirements are three fold. First the applications will require access to the system. Second, an isolated network for backups was required and since the load from GMS (Group Membership Services, this will be discussed later in the configuration section) was unknown, an isolated network for this service was also required. Figure 3 shows the final hardware configuration including the network requirements:

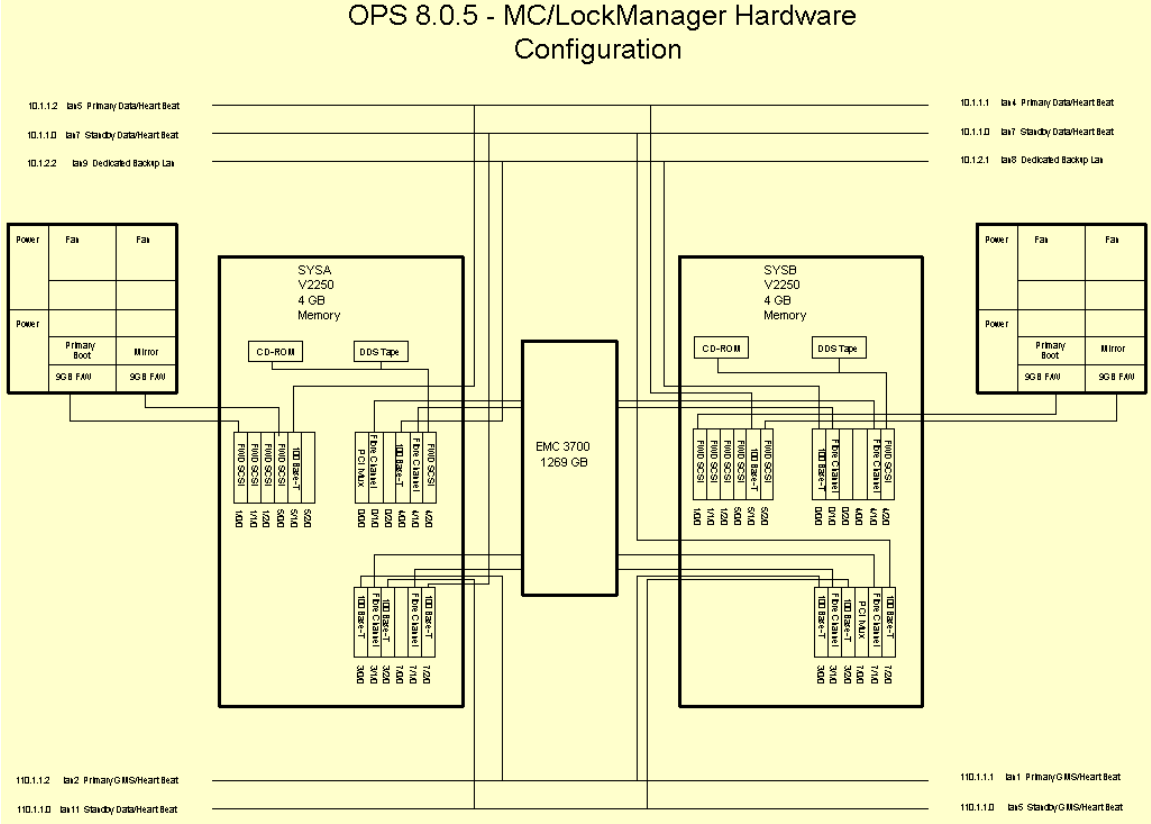

Figure 3.

In the above network configuration, we have eliminated single points of failure in the primary application connection and the GMS connection by providing a hot stand by connection. We have also provided two heartbeat paths to insure that if a heartbeat packet is missed, it is due to a system failure and not a network error. With the configuration above, we could have monitored the heartbeat via a serial connection, but that would limit us to a two-node configuration.

Application Requirements

Finally, any applications that depend on the database must be fault tolerant. An HA package must be configured for each application that will run on the same server/servers that the database resides. If one of the systems fails, the application will be required to fail over to another server so that the database and application will continue to be available to the end users.

In Review

Hopefully, you can tell that the exercise above was intended to show how to prevent single points of failure. The one resource that I did not illustrate is power circuits and supplies can also be single points of failure. In the diagrams above we do have some power single points of failures, can you point them out.

3. Configuration and Implementation

Now that we have completed the design phase, the configuration and implementation phase should be simple and straightforward. The first step is to mirror the root volume group and create our OPS volumes.

Logical Volume Manager Configuration

Root Volume Group

First we will look at the steps necessary to configure the disk space required by the database, applications and operating system. The operating system and application may be configured using mirrored standalone disks, but this type of configuration is not supported for OPS datafiles. Also, OPS datafiles must reside on raw logical volumes. In our configuration, we are using an EMC mass storage unit that mirrors data at the hardware level. On each system in the cluster, we must mirror the root disk to eliminate this single point of failure. Below are the steps necessary to mirror the root volume group:

1. Create physical volume for use with LVM

    pvcreate –B /dev/rdsk/c5t6d0

2. Extend physical volume into root volume group

    vgextend vg00 /dev/dsk/c5t6d0

3. Make disk bootable

    mkboot /dev/rdsk/c5t6d0

4. Copy AUTO file

    mkboot –a “hpux (;0)/stand/vmunix” /dev/rdsk/c5t6d0

5. Create mirrors of logical volumes

    lvextend –m 1 /dev/vg00/lvol1 /dev/dsk/c5t6d0
    lvextend –m 1 /dev/vg00/lvol2 /dev/dsk/c5t6d0

lvextend –m 1 /dev/vg00/lvol3 /dev/dsk/c5t6d0
lvextend –m 1 /dev/vg00/lvol4 /dev/dsk/c5t6d0
lvextend –m 1 /dev/vg00/lvol5 /dev/dsk/c5t6d0
lvextend –m 1 /dev/vg00/lvol6 /dev/dsk/c5t6d0
lvextend –m 1 /dev/vg00/lvol7 /dev/dsk/c5t6d0
lvextend –m 1 /dev/vg00/lvol8 /dev/dsk/c5t6d0

6. Update boot information in BDRA

lvlnboot –b /dev/vg00/lvol1
lvlnboot –r /dev/vg00/lvol3
lvlnboot –s /dev/vg00/lvol2

7. Insure that the BDRA is correct

lvlnboot –v

OPS Volume Groups

Configuring the volume group for the OPS datafiles is a two step process. First you must create the volume group and logical volume and then export the volume group on the first system and import it on the second. I will illustrate creating one OPS volume group. Our volume group naming convention was created to allow the DBA to identify which logical unit a volume group resides on. Part of the EMC volume id is used in the volume group name. Below are the step used to create an OPS volume group:

1. Create physical volume for use with LVM

pvcreate /dev/rdsk/c6t0d5

2. Create volume group directory and create group file

mkdir /dev/vg01aD4
mknod /dev/vg01aD4/group c 64 0x010000

3. Extend physical volume into root volume group

vgcreate vg01aD4 /dev/dsk/c6t0d5 /dev/dsk/c8t8d5

4. Create logical volumes

lvcreate –L 4100 /dev/vg01aD4/re001

5. Mark the volume as part of a cluster and make it sharable

vgchange –c y –S y vg01aD4

6. Export the volume group on SYSA

vgexport –p –s –m /tmp/vg01aD4.map vg01aD4

7. Copy the map file to SYSB and create the volume group directory and group file

mkdir /dev/vg01aD4
mknod /dev/vg01aD4/group c 64 0x010000

8. Import the volume group

vgimport –s –m /tmp/vg01aD4.map vg01aD4

Finally, let's take a minute and discuss Shared Logical Volume Manager. When MC/LockManager is installed, Shared Logical Volume Manager is installed. Shared LVM allows two systems to access a disk at the same time. At the time the volume group is activated, the first system to activate the volume group is the server and the other is a client. Below is a partial vgdisplay listing showing the client/server relationship:

vgdisplay -v vg01aD4

```
--- Volume groups ---
VG Name                     /dev/vg01aD4
VG Write Access             read/write
VG Status                   available, shared, client
Max LV                      255
Cur LV                      30
Open LV                     30
Max PV                      16
Cur PV                      12
Act PV                      12
Max PE per PV               2232
VGDA                        24
PE Size (Mbytes)            4
Total PE                    26772
Alloc PE                    24687
Free PE                     2085
Total PVG                   1
Total Spare PVs             0
Total Spare PVs in use      0

  ---  ---

  SYSA                      Server
  SYSB                      Client
```

Cluster Configuration Files

The next step is to create the cluster configuration file. The acronym for our application is TESS, therefore we will call the MC/LockManager configuration file tess.conf. To create the cluster template file issue the following command:

cmquerycl –v –C /etc/cmcluster/tess.conf –n SYSA –n SYSB

The file shown below in Figure 4. is a modified file that resulted from the command above. The comments in bold below are fields which were required to produce the final cluster configuration file.

```
# **********************************************************************
# ********* HIGH AVAILABILITY CLUSTER CONFIGURATION FILE ***************
# ***** For complete details about cluster parameters and how to     ****
# ***** set them, consult the cmquerycl(1m) manpage or your manual. ****
# **********************************************************************

# Enter a name for this cluster.  This name will be used to identify the
# cluster when viewing or manipulating it.

CLUSTER_NAME          tess

# Cluster Lock Device Parameters.  This is the volume group that
# holds the cluster lock which is used to break a cluster formation
# tie.  This volume group should not be used by any other cluster
# as cluster lock device.

FIRST_CLUSTER_LOCK_VG          /dev/vg01aD4

Modify above line with Cluster Lock Volume Group

# Definition of nodes in the cluster.
# Repeat node definitions as necessary for additional nodes.

NODE_NAME             SYSB
  NETWORK_INTERFACE     lan7
  NETWORK_INTERFACE     lan4
    HEARTBEAT_IP        10.1.1.1
  NETWORK_INTERFACE     lan1
    HEARTBEAT_IP        110.1.1.1
  NETWORK_INTERFACE     lan5
  FIRST_CLUSTER_LOCK_PV /dev/dsk/c6t0d5
# List of serial device file names
# For example:
# SERIAL_DEVICE_FILE    /dev/tty0p0

#   Possible standby Network Interfaces for lan4: lan7.
#   Possible standby Network Interfaces for lan1: lan5.

NODE_NAME             SYSA
  NETWORK_INTERFACE     lan5
```

```
  HEARTBEAT_IP       10.1.1.2
 NETWORK_INTERFACE    lan7
 NETWORK_INTERFACE    lan2
  HEARTBEAT_IP       110.1.1.2
 NETWORK_INTERFACE    lan11
 FIRST_CLUSTER_LOCK_PV /dev/dsk/c5t0d5
# List of serial device file names
# For example:
# SERIAL_DEVICE_FILE    /dev/tty0p0

#   Possible standby Network Interfaces for lan2: lan11.
#   Possible standby Network Interfaces for lan5: lan7.

# Cluster Timing Parmeters (microseconds).

HEARTBEAT_INTERVAL            2000000
NODE_TIMEOUT          6000000

# Configuration/Reconfiguration Timing Parameters (microseconds).

AUTO_START_TIMEOUT     600000000
NETWORK_POLLING_INTERVAL       2000000

# Package Configuration Parameters.
# Enter the maximum number of packages which will be configured in the cluster.
# You can not add packages beyond this limit.
# This parameter is required.
MAX_CONFIGURED_PACKAGES        10
```

Modify above line with maximum number of packages required.

```
# List of cluster aware Volume Groups. These volume groups will
# be used by package applications via the vgchange -a e command.
# For example:
# VOLUME_GROUP         /dev/vgdatabase.
# VOLUME_GROUP         /dev/vg02.
VOLUME_GROUP          /dev/vg1GB
VOLUME_GROUP          /dev/vg8gb2
VOLUME_GROUP          /dev/vg8gb3
VOLUME_GROUP          /dev/vg8gb4
VOLUME_GROUP          /dev/vg8gb5
VOLUME_GROUP          /dev/vggenesys
```

Above list all cluster aware volume groups, these are the volume groups for application packages.

```
# List of OPS Volume Groups.
# Formerly known as DLM Volume Groups, these volume groups
# will be used by OPS cluster applications via
# the vgchange -a s command. (Note: the name DLM_VOLUME_GROUP
# is also still supported for compatibility with earlier versions.)
# For example:
# OPS_VOLUME_GROUP            /dev/vgdatabase.
# OPS_VOLUME_GROUP            /dev/vg02.

OPS_VOLUME_GROUP             /dev/vg01aD4
OPS_VOLUME_GROUP             /dev/vg01aC4
```

```
OPS_VOLUME_GROUP            /dev/vg16aC1
OPS_VOLUME_GROUP            /dev/vg16aD2
OPS_VOLUME_GROUP            /dev/vg08aD2
OPS_VOLUME_GROUP            /dev/vg07bC4
OPS_VOLUME_GROUP            /dev/vg02bC2
OPS_VOLUME_GROUP            /dev/vg16bC2
OPS_VOLUME_GROUP            /dev/vg10bC2
OPS_VOLUME_GROUP            /dev/vg15bC2
OPS_VOLUME_GROUP            /dev/vg10aD2
OPS_VOLUME_GROUP            /dev/vg16bD4
OPS_VOLUME_GROUP            /dev/vg08aC2
```

List all Oracle Parallel Server volume groups above. These are the volume groups that were created above.

```
# DLM parameters.
# When using Oracle Parallel Server versions prior to 8.0, set
# DLM_ENABLED to YES, enter values for the other parameters as
# specified in your Oracle documentation, and set GMS_ENABLED to NO below.

DLM_ENABLED                 NO
DLM_CONNECT_TIMEOUT         30000000
DLM_PING_INTERVAL           20000000
DLM_PING_TIMEOUT            60000000
DLM_RECONFIG_TIMEOUT        300000000
DLM_COMMFAIL_TIMEOUT        270000000
DLM_HALT_TIMEOUT            240000000
```

Since we are using OPS 8.0.5, ignore the above section.

```
# GMS parameters.
# When using Oracle Parallel Server version 8.0x or later, set
# DLM_ENABLED to NO above, set GMS_ENABLED to YES below, and
# enter values for the other parameters as specified in your
# Oracle documentation.

GMS_ENABLED                 YES
GMS_CONNECT_TIMEOUT         30000000
GMS_LOCATION                /u01/app/oracle/product/8.0.5.32/bin/ogms
```

Change GMS_ENABLED to YES and enter the GMS location.

Figure 4.

What is GMS?

GMS (Group Management Services) is the way OPS communicates with MC/LockManager. In previous versions of OPS and MC/LockManager, DLM (Distributed Lock Manager) was used to insure consistency between OPS instances. When an OPS instance starts, it registers itself with the GMS. If an OPS instance fails, GMS notifies all other OPS instances to perform recovery. With DLM, we could

configure which network interface the DLM would communicate over, but with GMS this configuration option is not available. By default GMS will communicate over the primary network interface. How do we change this? First we must set our /etc/resolve.conf file to look at the local host file first. Then we must trick OPS with a custom hostfile shown below:

```
10.1.1.34       SYSBb
10.1.2.33       SYSAa
10.1.2.34       SYSBa
110.1.1.34      SYSB
110.1.1.33      SYSA
10.1.1.33       SYSAb
```

GMS will choose the ip that is associated with the configured host. In our case the configured hosts are SYSA and SYSB. The actual ip address used via dns for SYSA and SYSB are 10.1.1.33 and 10.1.1.34 respectively. Since we want the GMS traffic to go across and isolated lan, we change the local host table to and the ips for SYSA and SYSB.

Package Configuration Files

Next we must create a package configuration file to start each OPS instance. In figure 5 below is the package configuration file used to control the OPS instance on SYSB. The sample configuration file was created with the following command:

mkdir /etc/cmcluster/ops
cmmakepkg –p /etc/cmcluster/ops/ops-sysb.conf

Below is the resulting configuration file. The modification will be shown in bold.

```
# ******************************************************************************
# ****** HIGH AVAILABILITY PACKAGE CONFIGURATION FILE (template) *******
# ******************************************************************************
# ******* Note: This file MUST be edited before it can be used. ********
# * For complete details about package parameters and how to set them, *
# * consult the MC/ServiceGuard or MC/LockManager manpages or manuals. *
# ******************************************************************************

# Enter a name for this package.  This name will be used to identify the
# package when viewing or manipulating it.  It must be different from
# the other configured package names.

PACKAGE_NAME                ops-sysb

# Enter the failover policy for this package. This policy will be used
# to select an adoptive node whenever the package needs to be started.
# The default policy unless otherwise specified is CONFIGURED_NODE.
# This policy will select nodes in priority order from the list of
# NODE_NAME entries specified below.
#
```

The alternative policy is MIN_PACKAGE_NODE. This policy will select
the node, from the list of NODE_NAME entries below, which is
running the least number of packages at the time this package needs
to start.

FAILOVER_POLICY CONFIGURED_NODE

Enter the failback policy for this package. This policy will be used
to determine what action to take when a package is not running on
its primary node and its primary node is capable of running the
package. The default policy unless otherwise specified is MANUAL.
The MANUAL policy means no attempt will be made to move the package
back to its primary node when it is running on an adoptive node.
#
The alternative policy is AUTOMATIC. This policy will attempt to
move the package back to its primary node whenever the primary node
is capable of running the package.

FAILBACK_POLICY MANUAL

Enter the names of the nodes configured for this package. Repeat
this line as necessary for additional adoptive nodes.
Order IS relevant. Put the second Adoptive Node AFTER the first
one.
Example : NODE_NAME original_node
NODE_NAME adoptive_node

NODE_NAME SYSB

Enter the complete path for the run and halt scripts. In most cases
the run script and halt script specified here will be the same script,
the package control script generated by the cmmakepkg command. This
control script handles the run(ning) and halt(ing) of the package.
If the script has not completed by the specified timeout value,
it will be terminated. The default for each script timeout is
NO_TIMEOUT. Adjust the timeouts as necessary to permit full
execution of each script.
Note: The HALT_SCRIPT_TIMEOUT should be greater than the sum of
all SERVICE_HALT_TIMEOUT specified for all services.

RUN_SCRIPT /etc/cmcluster/ops/ops-sysb.ctl
RUN_SCRIPT_TIMEOUT NO_TIMEOUT
HALT_SCRIPT /etc/cmcluster/ops/ops-sysb.ctl
HALT_SCRIPT_TIMEOUT NO_TIMEOUT
Change the run and halt scripts to the name of the scripts that you will create shortly

Enter the SERVICE_NAME, the SERVICE_FAIL_FAST_ENABLED and the
SERVICE_HALT_TIMEOUT values for this package. Repeat these
three lines as necessary for additional service names. All
service names MUST correspond to the service names used by
cmrunserv and cmhaltserv commands in the run and halt scripts.
#
The value for SERVICE_FAIL_FAST_ENABLED can be either YES or
NO. If set to YES, in the event of a service failure, the
cluster software will halt the node on which the service is
running. If SERVICE_FAIL_FAST_ENABLED is not specified, the

```
# default will be NO.
#
# SERVICE_HALT_TIMEOUT is represented in the number of seconds.
# This timeout is used to determine the length of time (in
# seconds) the cluster software will wait for the service to
# halt before a SIGKILL signal is sent to force the termination
# of the service.  In the event of a service halt, the cluster
# software will first send a SIGTERM signal to terminate the
# service.  If the service does not halt, after waiting for the
# specified SERVICE_HALT_TIMEOUT, the cluster software will send
# out the SIGKILL signal to the service to force its termination.
# This timeout value should be large enough to allow all cleanup
# processes associated with the service to complete.  If the
# SERVICE_HALT_TIMEOUT is not specified, a zero timeout will be
# assumed, meaning the cluster software will not wait at all
# before sending the SIGKILL signal to halt the service.
#
# Example: SERVICE_NAME                DB_SERVICE
#          SERVICE_FAIL_FAST_ENABLED     NO
#          SERVICE_HALT_TIMEOUT          300
#
# To configure a service, uncomment the following lines and
# fill in the values for all of the keywords.
#
#SERVICE_NAME                 <service name>
#SERVICE_FAIL_FAST_ENABLED    <YES/NO>
#SERVICE_HALT_TIMEOUT         <number of seconds>

# Enter the network subnet name that is to be monitored for this package.
# Repeat this line as necessary for additional subnet names.  If any of
# the subnets defined goes down, the package will be switched to another
# node that is configured for this package and has all the defined subnets
# available.

#SUBNET

# The following keywords (RESOURCE_NAME, RESOURCE_POLLING_INTERVAL, and
# RESOURCE_UP_VALUE) are used to specify Package Resource Dependencies.  To
# define a Package Resource Dependency, a RESOURCE_NAME line with a fully
# qualified resource path name, and one or more RESOURCE_UP_VALUE lines are
# required.  A RESOURCE_POLLING_INTERVAL line (how often in seconds the resource

# is to be monitored) is optional and defaults to 60 seconds.  An operator and
# a value are used with RESOURCE_UP_VALUE to define when the resource is to be
# considered up.  The operators are =, !=, >, <, >=, and <=, depending on the
# type of value.  Values can be string or numeric.  If the type is string, then
# only = and != are valid operators.  If the string contains whitespace, it
# must be enclosed in quotes.  String values are case sensitive.  For example,
#
#                              Resource is up when its value is
#                              --------------------------------
#      RESOURCE_UP_VALUE       = UP                "UP"
#      RESOURCE_UP_VALUE       != DOWN                 Any value except "DOWN"
#      RESOURCE_UP_VALUE       = "On Course"           "On Course"
#
# If the type is numeric, then it can specify a threshold, or a range to
```

```
# define a resource up condition.  If it is a threshold, then any operator
# may be used.  If a range is to be specified, then only > or >= may be used
# for the first operator, and only < or <= may be used for the second operator.
# For example,
#                               Resource is up when its value is
#                               --------------------------------
#       RESOURCE_UP_VALUE     = 5             5                 (threshold)
#       RESOURCE_UP_VALUE     > 5.1           greater than 5.1    (threshold)
#       RESOURCE_UP_VALUE     > -5 and < 10     between -5 and 10   (range)
#
# Note that "and" is required between the lower limit and upper limit
# when specifying a range.  The upper limit must be greater than the lower
# limit.  If RESOURCE_UP_VALUE is repeated within a RESOURCE_NAME block, then
# they are inclusively OR'd together.  Package Resource Dependencies may be
# defined by repeating the entire RESOURCE_NAME block.
#
# Example : RESOURCE_NAME             /net/interfaces/lan/status/lan0
#           RESOURCE_POLLING_INTERVAL   120
#           RESOURCE_UP_VALUE         = RUNNING
#           RESOURCE_UP_VALUE         = ONLINE
#
#           Means that the value of resource /net/interfaces/lan/status/lan0
#           will be checked every 120 seconds, and is considered to
#           be 'up' when its value is "RUNNING" or "ONLINE".
#
# Uncomment the following lines to specify Package Resource Dependencies.
#
#RESOURCE_NAME            <Full_path_name>
#RESOURCE_POLLING_INTERVAL  <numeric_seconds>
#RESOURCE_UP_VALUE         <op> <string_or_numeric> [and <op> <numeric>]

# The default for PKG_SWITCHING_ENABLED is YES. In the event of a
# failure, this permits the cluster software to transfer the package
# to an adoptive node.  Adjust as necessary.

PKG_SWITCHING_ENABLED           NO
```

Insure that package switching is set to NO. With a ops package, the package does not switch because the same package is running on the other systems.

```
# The default for NET_SWITCHING_ENABLED is YES.  In the event of a
# failure, this permits the cluster software to switch LANs locally
# (transfer to a standby LAN card).  Adjust as necessary.

NET_SWITCHING_ENABLED           YES
```

Set network switching to YES in case a network card or connection fails.

```
# The default for NODE_FAIL_FAST_ENABLED is NO.  If set to YES,
# in the event of a failure, the cluster software will halt the node
# on which the package is running.  Adjust as necessary.
NODE_FAIL_FAST_ENABLED  NO
```

Figure 5.

Package Control Script

In the package configuration file above, we modified the lines for the RUN and HALT scripts. Below is the file that controls the starting and stopping of the OPS instance. This script performs volume group (de)activation, ip address configuration and running the customer defined run and halt scripts. Using the following command creates the control script:

cmmakepkg –s /etc/cmcluster/ops/ops-sysb.ctl

The resulting template is shown below in Figure 6. Again the modifications to the file are in bold.

```
#"(#) A.11.04         $Revision: 81.15 $ $Date: 98/08/18 11:36:00 $"
# ******************************************************************************
# *                                                                            *
# *       HIGH AVAILABILITY PACKAGE CONTROL SCRIPT (template)         *
# *                                                                            *
# *       Note: This file MUST be edited before it can be used.        *
# *                                                                            *
# ******************************************************************************

# UNCOMMENT the variables as you set them.

# Set PATH to reference the appropriate directories.
PATH=/sbin:/usr/bin:/usr/sbin:/etc:/bin

# VOLUME GROUP ACTIVATION:
# Specify the method of activation for volume groups.
# Leave the default ("VGCHANGE="vgchange -a e") if you want volume
# groups activated in exclusive mode. This assumes the volume groups have
# been initialized with 'vgchange -c y' at the time of creation.
#
# Uncomment the first line (VGCHANGE="vgchange -a e -q n"), and comment
# out the default, if your disks are mirrored on separate physical paths,
#
# Uncomment the second line (VGCHANGE="vgchange -a e -q n -s"), and comment
# out the default, if your disks are mirrored on separate physical paths,
# and you want the mirror resynchronization to ocurr in parallel with
# the package startup.
#
# Uncomment the third line (VGCHANGE="vgchange -a y") if you wish to
# use non-exclusive activation mode. Single node cluster configurations
# must use non-exclusive activation.
#
# VGCHANGE="vgchange -a e -q n"
# VGCHANGE="vgchange -a e -q n -s"
# VGCHANGE="vgchange -a y"
VGCHANGE="vgchange -a s"
Uncomment the line above to activate the volume group in shared mode

# VOLUME GROUPS
# Specify which volume groups are used by this package. Uncomment VG[0]=""
```

```
# and fill in the name of your first volume group. You must begin with
# VG[0], and increment the list in sequence.
#
# For example, if this package uses your volume groups vg01 and vg02, enter:
#       VG[0]=vg01
#       VG[1]=vg02
#
# The volume group activation method is defined above. The filesystems
# associated with these volume groups are specified below.
#
VG[0]=vg01aD4
VG[1]=vg01aC4
VG[2]=vg16aC1
VG[3]=vg16aD2
VG[4]=vg08aD2
VG[5]=vg07bC4
VG[6]=vg02bC2
VG[7]=vg16bC2
VG[8]=vg10bC2
VG[9]=vg15bC2
VG[10]=vg10aD2
VG[11]=vg16bD4
VG[12]=vg08aC2
```

List the volume groups to activate

```
# FILESYSTEMS
# Specify the filesystems which are used by this package. Uncomment
# LV[0]=""; FS[0]=""; FS_MOUNT_OPT[0]="" and fill in the name of your first
# logical volume, filesystem and mount option for the file system. You must
# begin with LV[0], FS[0] and FS_MOUNT_OPT[0] and increment the list in
# sequence.
#
# For example, if this package uses the file systems pkg1a and pkg1b,
#:wq!
# write options enter:
#       LV[0]=/dev/vg01/lvol1; FS[0]=/pkg1a; FS_MOUNT_OPT[0]="-o rw"
#       LV[1]=/dev/vg01/lvol2; FS[1]=/pkg1b; FS_MOUNT_OPT[1]="-o rw"
#
# The filesystems are defined as triplets of entries specifying the logical
# volume, the mount point and the mount options for the file system. Each
# filesystem will be fsck'd prior to being mounted. The filesystems will be
# mounted in the order specified during package startup and will be unmounted
# in reverse order during package shutdown. Ensure that volume groups
# referenced by the logical volume definitions below are included in
# volume group definitions above.
#

# FILESYSTEM UNMOUNT COUNT
# Specify the number of unmount attempts for each filesystem during package
# shutdown.  The default is set to 1.
#LV_UMOUNT_COUNT=1

# IP ADDRESSES
# Specify the IP and Subnet address pairs which are used by this package.
# Uncomment IP[0]="" and SUBNET[0]="" and fill in the name of your first
# IP and subnet address. You must begin with IP[0] and SUBNET[0] and
```

```
# increment the list in sequence.
#
# For example, if this package uses an IP of 192.10.25.12 and a subnet of
# 192.10.25.0 enter:
#         IP[0]=192.10.25.12
#         SUBNET[0]=192.10.25.0 # (netmask=255.255.255.0)
#
# Hint: Run "netstat -i" to see the available subnets in the Network field.
#
# IP/Subnet address pairs for each IP address you want to add to a subnet
# interface card.  Must be set in pairs, even for IP addresses on the same
# subnet.
#
#IP[0]=""
#SUBNET[0]=""
IP[0]=10.1.1.38
SUBNET[0]=10.1.1.0
```

Specify the ip address and subnet for the package

```
# SERVICE NAMES AND COMMANDS.
# Specify the service name, command, and restart parameters that are
# used by this package. Uncomment SERVICE_NAME[0]="", SERVICE_CMD[0]="",
# SERVICE_RESTART[0]="" and fill in the name of the first service, command,
# and restart parameters. You must begin with SERVICE_NAME[0], SERVICE_CMD[0],
# and SERVICE_RESTART[0] and increment the list in sequence.
#
# For example:
#         SERVICE_NAME[0]=pkg1a
#         SERVICE_CMD[0]="/usr/bin/X11/xclock -display 192.10.25.54:0"
#         SERVICE_RESTART[0]=""  # Will not restart the service.
#
#         SERVICE_NAME[1]=pkg1b
#         SERVICE_CMD[1]="/usr/bin/X11/xload -display 192.10.25.54:0"
#         SERVICE_RESTART[1]="-r 2"   # Will restart the service twice.
#
#         SERVICE_NAME[2]=pkg1c
#         SERVICE_CMD[2]="/usr/sbin/ping"
#         SERVICE_RESTART[2]="-R" # Will restart the service an infinite
#                                 number of times.
#
# Note: No environmental variables will be passed to the command, this
# includes the PATH variable. Absolute path names are required for the
# service command definition.  Default shell is /usr/bin/sh.
#
#SERVICE_NAME[0]=""
#SERVICE_CMD[0]=""
#SERVICE_RESTART[0]=""

# DTC manager information for each DTC.
# Example: DTC[0]=dtc_20
#DTC_NAME[0]=

# START OF CUSTOMER DEFINED FUNCTIONS

# This function is a place holder for customer define functions.
# You should define all actions you want to happen here, before the service is
```

```
# started.  You can create as many functions as you need.

function customer_defined_run_cmds
{
# ADD customer defined run commands.
: # do nothing instruction, because a function must contain some command.

        /etc/cmcluster/ops/prda.sh start
        /etc/cmcluster/ops/prod.sh start
        test_return 51
Add your start scripts here
}

# This function is a place holder for customer define functions.
# You should define all actions you want to happen here, before the service is
# halted.

function customer_defined_halt_cmds
{
# ADD customer defined halt commands.
: # do nothing instruction, because a function must contain some command.
        /etc/cmcluster/ops/prda.sh shutdown
        /etc/cmcluster/ops/prod.sh shutdown
        test_return 52
Add your stop scripts here
}

# END OF CUSTOMER DEFINED FUNCTIONS
```

Figure 6.

Note that this is not the entire file, but no further modification occurred beyond this point.

OPS Startup and Shutdown scripts

Figure 7 below show the script that actually starts and stops the database. The script is provided as part of the Enterprise Cluster Master Toolkit. This toolkit contains startup and shutdown script templates for all major databases. The script uses svrmgrl to start and stop the database.

```
#!/usr/bin/sh
#VERSION="@(#) A.10.10   $Revision: 1.13 $ $Date: 98/07/23 10:18:02 $"
# ***************************************************************************
# ****** Startup, Shutdown and Monitor Script for ORACLE (Template) ******
# ***************************************************************************
# ******** Note: This file MUST be edited before it can be used. *********
# ***************************************************************************
#
# NEW FEATURES IN CURRENT RELEASE:
#
# Improved performance in the monitor loop by checking the PID only
# for each process instead of looping with "ps -ef"
#
# Added support for 7.3.3 Oracle.
#
# This shell script supports Oracle Version 7 only. This script executes one
# Oracle instance only. If you are running more than one instance on one node,
# this script has to be duplicated for each instance. This template should be
# edited and moved to
# '/etc/cmcluster/${SID_NAME}/${SID_NAME}.sh,
# where ${SID_NAME} is the name of the Oracle instance. Consult the
# README file for more information.
#
# Set the environment variables:
#
# ORA_7_3_3:          Set to "yes" if you are running Oracle version
#                     7.3.3 or greater, or set to "no" if you are
#                     using and earlier version.
#
# SID_NAME:           The session name is often called the session ID (SID)
#                     in Oracle text.
#
# ORACLE_HOME:        The home directory of Oracle.
#
# MONITOR_INTERVAL:   The interval of time in seconds, this script should
#                     wait between checks of the Oracle database running.
#
# MONITOR_PROCESSES:  Set the array MONITOR_PROCESSES to contain the names
#                     of all processes that must be executing to assume this
#                     instance is still up and running.
#
# PACKAGE_NAME:       The name of the package as defined in the
#                     MC/ServiceGuard package configuration file.
#
# TIME_OUT:           The amount of time in seconds you want to wait for the
#                     Oracle abort to complete before resorting to killing
```

```
#                   the oracle processes.  The TIME_OUT variable is to
#                   protect against the worst case of a hung database
#                   that would prevent the halt script from completing
#                   and the standby node from starting up the data base.
#                   The value of TIME_OUT must be less than the time out
#                   variable set in the package configuration file.
#                   The time out variable does not impact any
#                   failover times, it is only a fail safe.
# Examples:
#
# ORA_7_3_3=yes
# SID_NAME=owas
# ORACLE_HOME=/mnt1/base/app/oracle/product/7.3.3
# SQLNET=yes
# SQLNET_NAME=LSNR_${SID_NAME}
# SQLNET_PASS=1DF5C2FD0FE9CFA2
# MONITOR_INTERVAL=30
# set -A MONITOR_PROCESSES ora_smon_${SID_NAME} ora_pmon_${SID_NAME} ora_lgwr_${
SID_NAME} ora_dbwr_${SID_NAME} ${SQLNET_NAME}
# PACKAGE_NAME=dbs
# TIME_OUT=10

ORA_7_3_3=yes
SID_NAME=PRDATS01
ORACLE_HOME=/u01/app/oracle/product/8.0.5.32
SQLNET=
SQLNET_NAME=
SQLNET_PASS=
MONITOR_INTERVAL=
set -A MONITOR_PROCESSES
PACKAGE_NAME=
TIME_OUT=

HOST=`hostname`
DATE=`date`
PATH=${ORACLE_HOME}/bin:/sbin:/usr/bin:/usr/sbin:/etc:/bin
export ORACLE_SID=${SID_NAME}
export ORACLE_HOME

##############################################################################
# Function: oracle_run_cmds
#
# This function is used to start the ORACLE instance defined in the environment
# variables ORACLE_HOME and SID_NAME.
##############################################################################

function oracle_run_cmds
{
#   The Oracle initialization file (init${SID_NAME}.ora) can contain node
#   specific information. A large memory configuration might not run on a
#   smaller backup system so you might need to create node specific
#   configuration files as shown below:
#
#         ${ORACLE_HOME}/dbs/init${SID_NAME}.ora.${HOST}
#
#   Note: You must have node specific files for ALL NODES in the cluster.
```

```
   PFILE=${ORACLE_HOME}/dbs/init${SID_NAME}.ora

   if [[ -f ${PFILE}.${HOST} ]]
   then
      PFILE=${PFILE}.${HOST}
   fi

#   Make sure that the configuration file exists, if not then the
#   database cannot be started on this node.

   if [[ ! -f ${PFILE} ]]
   then
      print "ORACLE: The file ${PFILE} does not exist."
      print "\tERROR:  Failed to start Oracle database."
      exit 1
   fi

#   If using MC/ServiceGuard A.10.10 without any patches, you must uncomment
#   the following line; otherwise, this script will be killed before the startup

#   can be completed.

#   trap "" SIGTERM

#   Startup Oracle using the standard Oracle 'dbstart' command.
#   This command does not return any error codes, so anything that
#   goes wrong after this point will cause a fail over to another node.
#
   if [[ ${ORA_7_3_3} = yes ]]
   then
su oracle -c ${ORACLE_HOME}/bin/svrmgrl <<EOF
connect internal
startup pfile=${PFILE}
exit
EOF
   else
su oracle -c ${ORACLE_HOME}/bin/sqldba lmode=y <<EOF
connect internal
startup pfile=${PFILE}
exit
EOF
   fi

   if [[ $? != 0 ]]
   then
      print "Oracle startup failed."
   else
      print "Oracle startup done."
   fi

#   Set the SQLNET variable to start up a listener process for Oracle
#   SQL*Net V2. See the README file for more suggestions on how to setup SQL*Net
.

   if [[ ${ORA_7_3_3} = yes ]]
```

```
   then
      if [[ ${SQLNET} = yes ]]
      then
         su oracle -c "${ORACLE_HOME}/bin/lsnrctl start ${SQLNET_NAME}"
         if [[ $? != 0 ]]
         then
            print "Oracle lsnrctl start failed."
         else
            print "Oracle lsnrctl start done."
         fi
      fi
   fi
}

################################################################################
# Function: oracle_shutdown_cmds
#
# Use "oracle_<SID_NAME>_cntl shutdown" to do a clean shutdown of the
# database, do not use "cmhaltpkg" to shutdown the database, as
# cmhaltpkg does a database abort and all transactions in memory will be lost.
################################################################################

function oracle_shutdown_cmds
{
   if [[ ${ORA_7_3_3} = yes ]]
   then
      su oracle -c ${ORACLE_HOME}/bin/svrmgrl <<EOF
connect internal
shutdown immediate
EOF
   else
     su oracle -c ${ORACLE_HOME}/bin/sqldba lmode=y <<EOF
connect internal
shutdown immediate
EOF
   fi

   if [[ $? != 0 ]]
   then
      print "Oracle shutdown failed."
   else
      print "Oracle shutdown done."
   fi
}

################################################################################
# Function: oracle_abort_cmds
#
# The command below is this scripts calling itself with the kill option to
# kill any process that will not die normally after waiting for the TIME_OUT
# period.
################################################################################

function oracle_abort_cmds
{
#   In case something is wrong we abort.  If you wish to shutdown the
```

```
#   database without an abort, you must manually execute "${SID_NAME}.sh
#   shutdown" from the command line instead of using cmhaltpkg.  Abort is
#   used, because immediate and normal will wait for all users to log off
#   and they are not deterministic.
#
#   The command below is this scripts calling itself with the kill option to
#   kill any process that will not die normally after waiting for the TIME_OUT
#   period.

   set -m
   ${0} kill &
   KILL_PID=$!

#   Set the SQLNET variable to stop a listener process for Oracle SQL*net V2.

   if [[ ${ORA_7_3_3} = yes ]]
   then
      su oracle -c ${ORACLE_HOME}/bin/svrmgrl <<EOF
connect internal
shutdown abort
exit
EOF
      if [[ ${SQLNET} = yes ]]
      then
         su oracle -c ${ORACLE_HOME}/bin/lsnrctl <<EOF
set password ${SQLNET_PASS}
stop ${SQLNET_NAME}
exit
EOF
      fi
   else
      su oracle -c ${ORACLE_HOME}/bin/sqldba lmode=y <<EOF
connect internal
shutdown abort
EOF
      if [[ ${SQLNET} = yes ]]
      then
         su oracle -c ${ORACLE_HOME}/bin/lsnrctl <<EOF
set password ${SQLNET_PASS}
stop ${SQLNET_NAME}
exit
EOF
      fi
   fi

   if [[ $? != 0 ]]
   then
      print "Oracle abort failed."
   else
      print "Oracle abort done."
   fi

#   Make sure all processes have gone away before saying shutdown is complete.
#   This stops the other node from starting up the package before it has been
#   stopped and the file system has been unmounted.
```

```
    typeset -i c
    typeset -i num_procs=${#MONITOR_PROCESSES[@]}

    while true
    do
        for i in ${MONITOR_PROCESSES[@]}
        do
            ps -ef | grep ${i} | grep -v grep > /dev/null
            if [[ $? != 0 ]]
            then
                print "\n *** ${i} process has stopped. ***\n"
                c=0
                while (( c < $num_procs ))
                do
                    if [[ ${MONITOR_PROCESSES[$c]} = $i ]]
                    then
                        unset MONITOR_PROCESSES[$c]
                        c=$num_procs
                    fi
                    (( c = c + 1 ))
                done
            fi
        done

        if [[ ${MONITOR_PROCESSES[@]} = "" ]]
        then
#           Looks like shutdown was successful, so get rid of the script to
#           kill any hung processes, which we started earlier.  Check to see if
#           the script is still running.  If jobs returns that the script is
#           done, then we don't need to kill it.

            job=$(jobs | grep -v Done)
            if [[ ${job} != "" ]]
            then
                print "Killing the kill_hung_processes script\n"
                kill %1
            fi
            exit
        fi

        sleep 5
    done
}

###############################################################################
# Function: terminate
#
# Called when a SIGTERM is trapped.
###############################################################################

function terminate
{
    print "This monitor script has been signaled to terminate\n"
    exit
}
```

```
##############################################################################
# Function: kill_hung_processes
#
# In case of a database hang we spawn a background process that will issue
# a SIGKILL to each of the monitored processes, this allows us to guarantee
# a time limit for clean halt attempts.
##############################################################################

function kill_hung_processes
{
#   Wait for the TIME_OUT period before sending a kill signal.  The TIME_OUT
#   value should be less than the MC/Serviceguard package time out.

   sleep ${TIME_OUT}

   for i in ${MONITOR_PROCESSES[@]}
   do
      id=`ps -fu oracle | awk '/'${i}'/ { print $2 }'`
      if [[ ${id} != "" ]]
      then
         kill -9 ${id}
         if [[ $? != 0 ]]
         then
            print "\n *** ${0} kill_hung_processes function did NOT find pro
cess ***\n *** ${i} running. ***\n"
         else
            print "\n *** ${0} kill_hung_processes function did find process
 ***\n *** ${i} running.  Sent SIGKILL. ***\n"
         fi
      else
         print " *** kill_hung_processes function did NOT find ANY process ru
nning. ***\n"
      fi
   done
}

##############################################################################
# Function: monitor_processes
#
# Monitor the Oracle processes by making sure that all required processes
# are running.  First get the process ID of each the processes being
# monitored, so the less cpu intensive "kill -s" can be used instead of
# "ps -ef".
##############################################################################

function monitor_processes
{
   typeset -i n=0

   for i in ${MONITOR_PROCESSES[@]}
   do
      MONITOR_PROCESSES_PID[$n]=`ps -fu oracle | awk '/'${i}'/ { print $2 }'`
      print "Monitored process = ${i}, pid = ${MONITOR_PROCESSES_PID[$n]}"
      if [[ ${MONITOR_PROCESSES_PID[$n]} = "" ]]
      then
         print "\n\n"
```

```
        ps -ef
        print "\n *** ${i} has failed at startup time.  Aborting Oracle. ***
"
        set -m
        nohup ${0} fault &    # The script calls itself with the fault optio
n.
        set +m
        sleep 999999
      fi
      (( n = n + 1 ))
   done

   sleep ${MONITOR_INTERVAL}

   while true
   do
      for i in ${MONITOR_PROCESSES_PID[@]}
      do
         kill -s 0 ${i} > /dev/null
         if [[ $? != 0 ]]
         then
            print "\n\n"
            ps -ef
            print "\n *** ${i} has failed.  Aborting Oracle. ***"
            set -m
            nohup ${0} fault &    # The script calls itself with the fault o
ption.
            set +m
            sleep 999999
         fi
      done
      sleep ${MONITOR_INTERVAL}
   done
}

################################################################################
# Function: halt_package
#
# Because there is no move command in MC/Serviceguard so the package has to be
# halted first, then disabled from running on the host, then enabled
# to run in the cluster.
################################################################################

function halt_package
{
   cmhaltpkg ${PACKAGE_NAME}
   cmmodpkg -d -n ${HOST} ${PACKAGE_NAME}
   sleep 1
   cmmodpkg -e ${PACKAGE_NAME}
}

################################################################################
# FUNCTION STARTUP SECTION.
#
# Test to see if we are being called to run the application, or halt the
# application.
```

```
##############################################################################

print "\n *** $0 called with $1 argument. ***\n"
case $1 in

   fault)
      halt_package
   ;;

   kill)
      kill_hung_processes
   ;;

   monitor)
      monitor_processes
   ;;

   start)
      print "\n \"${HOST}\": Starting Oracle SESSION $SID_NAME at ${DATE} "
      oracle_run_cmds
   ;;

   halt)
      print "\n \"${HOST}\": Aborting Oracle SESSION $SID_NAME at ${DATE} "
      oracle_abort_cmds
   ;;

   shutdown)
      print "\n \"${HOST}\": Shutting down Oracle SESSION $SID_NAME at ${DATE}
 "
      oracle_shutdown_cmds
   ;;

   *)
      print "Usage: ${0} [ shutdown | halt | start | monitor ]"
   ;;
esac
```

Figure 7.

Creating the cluster

Now that we have created all of the configuration files necessary to create the cluster, we must check the configuration and then apply it. The command to perform these tasks are listed below:

cmcheckconf –v –C /etc/cmcluster/tess.conf –p /etc/cmcluster/ops/ops-sysa –p /etc/cmcluster/ops/ops-sysb

When the configuration check completes without errors, apply the configuration with the following command:

cmcheckapply –v –C /etc/cmcluster/tess.conf –p /etc/cmcluster/ops/ops-sysa –p /etc/cmcluster/ops/ops-sysb

Now that the cluster and package configurations have been applied, it is now time to start the cluster:

cmruncl –v

Next start the ops packages on each node:

cmrunpkg –n sysa –v ops-sysa
cmrunpkg –n sysb –v ops-sysb

You have now successfully implemented a MC/LockManager configuration.

Lessons Learned

Now the section that we all have been waiting for, what lessons did I learn.

1. Remember to get all applications that are running on the systems that should be fault tolerant. Even thought the databases are fault tolerant, the developers can introduce single points by allowing certain process to occur on one system. An example is if file are ftp'ed and process on only one system. If that system were to fail, the ftp processing would stop. If this process it critical to the operation of the application, the application will stop. **REMEMBER** to insure that all critical processing is fault tolerant.

2. We are currently experiencing an issue in which an application (NT Server) is using on of the databases. If the database is stopped, the application will not fail to the other instance because the oracle listener is still running. The reason the listener is still running is because we have more that just OPS oracle instances running on this system. From the beginning I informed the developers and DBAs to use the ip addresses associated with the packages. Each ops package has its own ip address. To resolve this issue, we are going to redo the startup and shutdown of the listener process. We will assign an ip address to the various listener process for each database. Again **REMEMBER** to insure that all critical applications will fail over to an alternate node if this is the desired actions.

3. **TEST TEST TEST.** Remember to test these configurations and procedures to insure that the provide the desired results.