White Paper : Network Node Manager 7.x and MC/ServiceGuard

The purpose of this document is to present the issues and methodology for improving the availability of network management using Hewlett-Packard's OpenView Network Node Manager (NNM) together with MC/ServiceGuard. Two approaches are provided for implementation: a MxN setup and conventional setup. Concepts of MxN are provided later in the document.

The primary goal from an HP perspective is to increase the availability of mission critical applications when compared to standard availability, but to do so at a reasonable cost. It is important to first understand that the fundamental architecture and databases of NNM have not changed. The conditions under which it was possible to have a corrupted NNM database will exist in the new configuration with MC/ServiceGuard. While a system crash will force the migration of the NNM processes to the surviving system, the crash can still potentially corrupt an NNM database. User precautions and backup techniques previously employed to recover from a corrupt database are still applicable in an NNM configuration with MC/ServiceGuard..

MC/ServiceGuard is only available on HP9000 Series 800 Servers. The configuration information is only applicable to those systems. MC/ServiceGuard information is not applicable to NNM on Series 700 Workstations, SUN Solaris and NT.

In addition, these notes do not address the issue of combining an MC/ServiceGuard configuration of NNM with an MC/ServiceGuard configuration of IT/Operations.

NNM distributed consoles are supported as part of the NNM MC/ServiceGuard enhancements. They are implemented using a combination of MC/ServiceGuard NFS package configuration and a new version of ovw, run on the client, called ovwrs. The details of implementing management consoles are discussed later in the 'Implementation Notes for Network Node Manager and MC/ServiceGuard'. Users must work with their OpenView Solution Partners to determine whether their NNM applications will function with NNM in an MC/ServiceGuard configuration and whether their NNM applications can also be configured with MC/ServiceGuard. To ensure a common understanding a brief discussion of the concepts of High Availability and the features and functions of MC/ServiceGuard is provided below. The paper continues with a description of implementing Network Node Manager with MC/ServiceGuard. The description is only provided as a guideline. Special customization is left to the user. The methodology can be applied to either a new installation or an existing configuration. A current implementation of Network Node Manager with all its user and map customizations can be migrated to an MC/ServiceGuard configuration. As always, the user is advised to backup their system and NNM configuration before proceeding with any upgrade.

The terms package and instance mean the same in this document and could be interchanged as applicable.

It should be noted that NetComplete 4.0, which NNM 7.5 is a component, is now HA-compliant. Support has been given to run NetComplete 4.0 (Extended Topology 4.0) on HA systems.

An Overview of High Availability

Hewlett-Packard's High Availability (HA) solutions seek to reduce the number and length of business application downtime by providing redundant hardware and rapid failover capabilities. HP's HA solutions address the issue of single points of failure (SPOFs) of the system and environment where the failure of any item may cause the entire system to become unavailable. Availability typically does not take into account planned downtime.

Examples of SPOFs include SPU (system processing unit), disks and disk controllers, LAN interface cards and cables, and power connection. These potential SPOFs are removed by clustering SPUs, mirroring and/or using RAID technology, providing redundant LAN interface cards, and attaching UPSs to the system. Clustering also facilitates rolling OS and application upgrades. HA solutions cannot protect against some failures such as bugs in applications and OS panics.

It should be noted that HP's HA solution MC/ServiceGuard does not provide customers with a continuously available system. For futher discussion on levels of availability please see Clusters for High Availability: a Primer of HP-UX Solutions by Peter Weygant, Prentice-Hall, 1996.

An Overview of Network Node Manager and MC/ServiceGuard

The following description is only provided as a brief outline of the features, functions and capabilities of MC/ServiceGuard. MC/ServiceGuard should only be implemented after thorough training. MC/ServiceGuard is a high availability solution that monitors system, process and LAN failures. MC/ServiceGuard supports a configuration of up to 8 systems. However, for the purposes of this paper only a configuration of two systems is considered. The solution is designed around the concept of moving the service point from one system to another. In the event of a failure on one system, the designated processes and LAN connection are moved to a standby (failover) system. To provide access to the applications/processes regardless of the system on which they are running a relocatable IP address is assigned to each set of application resources grouped into an MC/ServiceGuard package. In this case, Network Node Manager processes and volatile data are the package. NNM will use the relocatable IP address to monitor the network and interact with NNM Collection Stations. If the primary system fails, the backup system acquires the relocatable IP address of NNM, activates the shared disks and starts the NNM processes. The shared disks are only accessed by one system at a time (primary or standby system), even though the disks are connected to both systems.

MC/ServiceGuard ensures that the NNM package will run on only one system at a time. The cluster will automatically reconfigure itself when it detects that a system has gone down. Heartbeat messages are exchanged between the primary and failover systems to monitor each other's health. If the two systems cannot communicate with each other via heartbeat messages, the cluster will reform automatically. Each system will attempt to obtain the lock on the designated cluster lock disk (see the MC/ServiceGuard manual for more information on the definition and use of a cluster lock disk). Whichever system gains control of the cluster lock disk will reform itself as a one-system cluster. The other system will crash immediately to prevent two servers from running NNM concurrently.

Package switching occurs when a failure (an interruption of the execution of the NNM package and services, regardless of whether the system continues running) is detected. It is a feature of MC/ServiceGuard that no more than one minute will pass between the detection of a failure and the start of the NNM package's startup scripts on another available system. The time for all the NNM services to be running is dependent on the condition of the NNM database and the normal startup associated with NNM. The NNM package can fail over from either the designated primary system to the designated standby system or vice versa. Refer to the MC/ServiceGuard manual for the details and conditions for automated failover. As stated earlier, if the primary system fails NNM will restart on the standby system. The NNM package will not automatically fail back to the primary system when that system is repaired. Fail back is the responsibility of the administrator of the cluster. Fail back is usually performed during off-hours to minimize interruption of NNM monitoring. Alternatively, NNM can be left running on the standby system. This will effectively reverse the roles of the systems.

When the NNM package is started on either system or is restarted on either system after a failure, NNM daemons are started as they would at boot time on servers not running NNM under MC/ServiceGuard.

Neither MC/ServiceGuard nor NNM require that the two systems have similar values for date and time. However, different time settings may increase the difficulty of debugging the MC/ServiceGuard log files. In addition, NNM puts time stamps into its database files. Different time settings may negatively impact NNM synchronization. For these reasons the use of Network Time Protocol is highly recommended.

MC/ServiceGuard provides a Cluster Manager SNMP subagent. Whenever a cluster is monitored by NNM (whether or not NNM is running on that cluster), this subagent should be enabled on each node in the cluster. This subagent provides NNM with additional information to handle the floating IP address. The subagent is enabled by editing the file */etc/rc.config.d/cmsnmpagt* and started by */sbin/init.d/cmsnmpagt*. Start the SNMP master agent *(/usr/sbin/ snmpdm*) before the subagent.

Implementation notes for standard two node configuration:

These implementation notes assume the reader is familiar with HP-UX system administration, Logical Volume Manager, MC/ServiceGuard and Network Node Manager. Sample NNM MC/ServiceGuard configuration files are provided in the Appendix of this paper.

Install NNM on both systems in the cluster. Alternatively, it is possible to maintain a single set of NNM executables in */opt/OV* that is shared between the two systems. With NNM version 7.x, a single license is available which is shared by all instances of NNM. This license is bound to the floating IP address, specified in the NNM_INTERFACE field of the ov.conf file.

NNM should be started and tested on both of the systems at this time. This will ensure that NNM is properly installed on each system. Any NNM discovery or map customization can be done on one system at this time without compromising moving the configuration under MC/ServiceGuard. This also implies that a current implementation of Network Node Manager can be upgraded to an MC/ServiceGuard configuration. Alternatively, the customization can be done after moving to an MC/ServicGuard configuration. Before proceeding stop the NNM daemons on both systems. NNM operators defined or planned for on one system must be defined on both systems with the same user and group IDs.

The data in */etc/opt/OV/share* and */var/opt/OV/share* must be available on all the systems in the cluster depending on the type of implementation (described later in the document) of the NNM package. A volume group and two logical volumes must be created on a shared disk. This disk can also serve as the cluster lock disk. The sizes of these two file systems depend on the configuration. Refer to the “NNM Performance and Configuration Guide” for assistance in this area. These new file systems can be HFS or JFS. JFS file systems are recommended for High Availability configurations, but are not required. It is beyond the scope of this paper to describe the performance differences between these two file system types when used for the NNM data files.

The following procedure is done on the primary system unless otherwise noted. After creating the volume group and file systems and stopping the NNM daemons copy the contents of the two file systems noted earlier to the shared file systems. This copy is done on only one system. Retain file ownership and permissions when doing the copy. The original contents of */etc/opt/OV/share* and */var/opt/OV/share* can now be deleted. (Note: Had these shared file systems been mounted, NNM could have been installed directly into these shared file systems.) Unmount the shared file systems and remount them on */etc/opt/OV/share* and */var/opt/OV/share*. Restart the NNM daemons to ensure that NNM is still properly functioning. Unmount the shared file systems and deactivate the shared volume group. The original contents of */etc/opt/OV/share* and /*var/opt/OV/share* on the second system can now be deleted. Import the volume group information onto the second system in the cluster. Do NOT attempt to manually activate the shared volume group, mount the shared file systems and try to run NNM on the second system at this time.

A new configuration file (*ov.conf*) has been added to NNM. This file is applicable to MC/SG and non-MC/SG configurations. This paper only addresses the use of this file in an MC/SG environment. Copy the sample file provided in the patch from */opt/OV/newconfig/OVNNM-RUN/conf/ov.conf* to */etc/opt/OV/share/conf/ov.conf*. Sample configurations of this file is provided in the Appendix E.

Note: If the ov.conf file does not exist, then NNM reverts to its default behavior.

Edit the authorization files */etc/opt/OV/share/conf/ovw.auth, /etc/opt/OV/share/conf/ovwdb.auth* and */etc/opt/OV/share/conf/ovspmd.auth* to contain entries for both systems in the cluster and the relocatable IP address name. If LOOPBACK is enabled in *ov.conf*, then entries for localhost, loopback and loghost must also appear in the authorization files.

Create the MC/ServiceGuard cluster configuration file. A sample cluster configuration file is provided in the Appendix. The cluster configuration file must contain a minimum of two (three are recommended) LAN definitions. Edit the other parameters in the file as normally done for a MC/ServiceGuard configuration.

Create the NNM package configuration file. At this time NNM does not require any special naming convention for the package name or package services. A sample package configuration file is provided in the Appendix B. The sample file contains a single service definition. This service is used to monitor the NNM daemons. While a service is recommended it is not a requirement for a successful implementation. The sample file also defines a network for MC/ServiceGuard to monitor. While monitoring the health of the network is recommended it is not a requirement. If this network is unavailable to the system running NNM, the NNM package will gracefully fail over (an *ovstop* is executed to halt the NNM processes) to the other system (provided it has current access to that network). If neither system has access to the network, the NNM package will gracefully shut down and would restart on gaining access to the network.

Note: if NNM management consoles are used, an NFS service definition should be added to the package configuration file as well and the suggested modifications to be made are defined later in this whitepaper.

Create the NNM run/halt script. A sample script is provided in the Appendix C. Use the volume group and file systems described earlier in this paper. Define the relocatable IP address. This must be on the same network defined in the NNM package definition file and must resolve to the same DNS name used in the */etc/opt/OV/share/ conf/ov.conf* file described earlier. Add the commands:

cp /etc/opt/OV/share/conf/ov.conf.`hostname` /etc/opt/OV/share/conf/ov.conf
/opt/OV/bin/ovstart –v

to the *customer_defined_run_cmds* function. The verbose option of ovstart is recommended for debug and troubleshooting but is not required. Add the commands

/opt/OV/bin/ovstop –v
rm /etc/opt/OV/share/conf/ov.conf
rm /var/opt/OV/share/databases/openview/ovwdb/ovserver

to the *customer_defined_halt_cmds* function. The verbose option of *ovstop* is recommended for debug and troubleshooting but is not required. The removal of the two files is not required, but may facilitate running NNM outside of MC/ServiceGuard. Complete the service definition if a service was defined in the package configuration file.

A sample service monitor script is provided in the Appendix D. Defining one restart of the service is recommended but is not required. The service monitor script checks for the presence of a maintenance file. This is helpful for troubleshooting or making modifications to files in the shared file system. When /tmp/maint_NNM exists, NNM can be safely stopped and restarted without MC/ServiceGuard attempting

to restart stopped processes. If a maintenance file is created be sure to delete the file after the modifications are made.

The sample service monitor script checks all the major NNM background daemons. If a daemon is not running the script will attempt to restart it. The ovstart command in this case will only start daemons that are not running; this will not affect running daemons. After the restart, if there are NNM daemons that still are not running, the script will exit. MC/ServiceGuard will either restart the service monitor or gracefully fail over the NNM package to the other system depending on the service definition. Remember to copy this file to both systems in the cluster. Also ensure that the executable properties of the file are retained.

Important note: If management consoles are to be used, then the Package script should contain the appropriate NFS directives for exporting the server-shared file systems to the appropriate client machines. and the purchase of the MC/ServiceGuard NFS Toolkit is highly recommended. This toolkit contains all the necessary scripts for implementing HA NFS within the cluster. In addition, only a single package in a cluster may be configured with NFS. The processes rpc.statd and rpc.lockd are killed and restarted when the package configured with NFS halts. This will impact any other applications or packages using NFS processes. See the MC/ServiceGuard manual for more details. Refer to the section later in the document indicating the suggested changes to be made in the package configuration and control scripts . When management consoles are used, *ovwrs* instead of *ovw* should be run on the client. *ovwrs* will detect a loss of connection to the server, as would be the case if the package failed between cluster nodes. When the connection to the server is restored *ovwrs* will restart the *ovw* session, including the original map if a map is specified in the parameter list when *ovwrs* is invoked. The *ovw* session will restart with the configured home submap and not the open submap at the time of the failure. For more information see the *ovwrs(1)* man page.

The run/halt script should also contain scripting for the start and stop of the NFS processes.

When configuring the NNM management consoles, outside of the special considerations for MC/ServiceGuard, all other management console configuration procedures should be followed. Lastly, when mounting the NFS exported directories, use the floating IP address and/or hostname associated with this address, rather than the hostname associated with the static IP address. When ovw -server is run on the client, it should return the name of the floating IP address.Use *cmcheckconf* and *cmapplyconf* to check and create the cluster configuration binary file and distribute it to the systems in the cluster. If there are no errors, execute *cmruncl* to start the cluster.

Maintenance Notes for Network Node Manager and MC/ServiceGuard:

Rolling upgrades of the Operating System, Network Node Manager and MC/ServiceGuard are supported. As always be sure a backup is done before the upgrade. Most NNM patches do not directly affect the files in the shared file systems. However, when applying NNM patches check the log files for errors and warnings. If possible, each system should be patched when the shared file systems are mounted and the NNM daemons are not running (i.e., the NNM package is running in maintenance mode, but the NNM processes are stopped). The following sequence may be used when patching NNM:

Touch /tmp/maint_NNM
Stop all ovw sessions
Stop NNM daemon processes
Remove the ov.conf file
Remove the ovserver file
Install the patch
Restore the applicable ov.conf file for that server
Restart the NNM daemon proceses

Remove /tmp/maint_NNM

Known Issues for Implementing a Single Set of Binaries

The nettl process, which is started at system boot time, has a dependency on /opt/OV/lib/libovextfmt.sl. If the NNM binaries are relocated to the shared disk, then this library will not be available at system boot time. As a result, nettl fails to start.

Of the possible options for working around this problem, Hewlett-Packard recommends moving this library to /usr/lib on the system's local disk. After moving the library to /usr/lib update the NNM entries in /etc/nettlgen.conf to indicate the new path to this library:

SS:82:OVS:13:u:/usr/lib/libovextfmt.sl:NULL:ss84fmt::OpenView
SS:84:OVEXTERNAL:12:u:/usr/lib/libovextfmt.sl:NULL:ss84fmt::OpenView
SS:80:OVW:12:u:/usr/lib/libovextfmt.sl:NULL:ss84fmt::OpenView
SS:85:OVWAPI:12:u:/usr/lib/libovextfmt.sl:NULL:ss84fmt::OpenView

NNM 7.x considerations

The modifications that affect MC/ServiceGuard in migrating to 7.5 are:

Daemons: ovuispmd, ovalarmsrv, httpd
The file ovpause.lock
Elimination of ovwsessions and ovwlistsessions
Web access
Licensing
DynamicView user credential informations

If using the monitor script, the new daemons should be added. See the MUSTRUN variable in the monitor example below.

With the advent of ovpause and ovresume, a file, */var/opt/OV/tmp/ovpause.lock*, will be created when a pause (ovpause command) is done. It is possible that in the event of a system failure, the file might still exist, which would bring up the NNM daemons in a paused state the next time that they are started on that system. To eliminate this possibility, there should be a check for this file when the package is run and the halting of the package should remove the file. See the example template for the halt and run commands below.

In the past, the mechanism to stop ovw sessions was to use ovwlistsessions and ovwsessions. This was recommended as part of the halt script for the package. With NNM 7.5, these are not needed as the shutting down of the ovw sessions is controlled by ovuispmd.

For web access, there is a configuration file, */opt/OV/httpd/conf/httpd.conf*, which will specify the server name as the physical nodename when the installation is done. For MC/ServiceGuard, the physical nodename should be changed to the fully-qualified logical nodename for the following lines:

ServerAdmin root@spike.hp.com
ServerName spike.hp.com

(spike.hp.com is the fully-qualified logical nodename). For client applications, web access should be done using the logical nodename for all operations.

A single license can now be shared by all instances of NNM installed on the cluster nodes. This license is bound to the floating IP address, specified in the NNM_INTERFACE field of the ov.conf file. Prior to NNM version 6.x, a nodelock license was required for each of the cluster nodes on which NNM was installed.

DynamicView user credentail informations are stored under $OV_AS/webapps/topology/WEB-INF/dynamicViewsUsers.xml file. As this resides under /opt/OV, it is not transferable between the clusters. It is always recommended to manually maintain such files.

NNM 7.5 considerations

The NNM 7.5 reporting component, which requires additional configuration steps to run in a ServiceGuard environment. This section assumes that the previous steps to configure NNM in the ServiceGuard cluster have already been performed.

For the report scheduler daemon (ovrequestd), data in the */var/opt/OV/analysis/ovrequestd* directory must be available to both systems in the cluster. Be sure to do an "ovstop" before performing this step. The easiest way to do this is to move this directory from the primary system to */var/opt/OV/share/ovrequestd* on the shared file system, delete */var/opt/OV/analysis/ovrequestd* on the secondary system, and then create a symbolic link from the new directory /var/ opt/OV/share/ovrequestd back to */var/opt/OV/analysis/ovrequestd* on each system. Reports can be configured on the primary system either before or after the move, since all report configurations will be stored under */var/opt/OV/analysis/ovrequestd*. Logging by ovrequestd will be performed locally to */var/opt/OV/log/ovrequestd.log* for debugging purposes. Reports are stored in */var/opt/OV/www/htdocs*. If there is a desire to see past reports from both systems after a failover, or if the systems are to be kept consistent, this directory must be shared by both systems and therefore mounted by the package, along the same lines as */var/opt/OV/share* and /*etc/opt/OV/share*. The following procedure is done on the primary system unless otherwise noted. After creating the volume group and file system and stopping the NNM daemons, copy the contents of /*var/opt/OV/www/htdocs* to the shared file system. This copy is done on only the primary system. Retain file ownership and permissions when doing the copy. The original contents of */var/ opt/OV/www/htdocs* can now be deleted. Unmount the shared file system and remount it on */var/opt/OV/www/htdocs*. Restart the NNM daemons to ensure that NNM is still properly functioning. Unmount the shared file systems and deactivate the shared volume group. The original contents of */var/opt/OV/www/htdocs* on the second system can now be deleted. Import the volume group information onto the second system in the cluster. Do NOT attempt to manually activate the shared volume group; mount the shared file systems and try to run NNM on the second system at this time.

NNM 7.5 / NC 4.0 considerations

Some of the older versions of NetComplete did not support HA environments. While the NNM component of this did, the Extended Topology component was not qualified. With NetComplete 4.0, the Extended Topology component was qualified, so it is now supported.

To operate the Extended Topology component in the HA environment, there is only one additional step to the configuration for NNM. The file, /etc/opt/OV/share/conf/remoteConfAllow.conf, needs entries for the IP addresses for cluster nodes and external web browsers that will access the views.

For NNM 7.5, there was the support of AutoPass, the latest licensing technology for NNM. Of key interest for the HA environment is that this technology is not HA-compliant in two modes. First, the licenses reside in a file /var/opt/OV/HPOvLIC/LicFile.txt. Some licenses are the newer OvKey4 license format with NNM using OvKey3, so these licenses have to be converted back to OvKey3 for NNM's usage. For NNM and older applications, these licenses are converted or copied back to the $OV_CONF/.license file for usage.

Overall, this does not appear to be a problem for NNM and most applications, but some applications expect that the LicFile.txt file and the .license file contain the same licenses. As newer products roll out, more will have the issue that the LicFile.txt file between cluster nodes might not be in sync. Given this, it

is recommended that all licenses tied to package IP addresses be stored in a single file, and this be used to populate the LicFile.txt files on each system. If new licenses are being added, then the added licenses should be distributed to all systems by merging the existing licenses and the license file, removing duplicates, and then copying this information into the LicFile.txt if they apply to a HA package. A simple model of this is:

cat /var/opt/OV/HPOvLIC/LicFile.txt <common license file> | sort –u > /tmp/licenses.HA

cp /tmp/licenses.HA /var/opt/OV/HPOvLIC/LicFile.txt

If such licenses do not show up in the $OV_CONF/.license file, then running ovnnmInstallLic –migrate will correct this.

The second problem with AutoPass is that it is now the model for requesting licenses. While ovnnmPassword still exists, it now uses the AutoPass GUI interface for requesting licenses. The request interface of AutoPass, however, does not acknowledge HA systems. If licenses are being requested for floating IP addresses, the user must click on the "No Internet Connection" radio button when requesting the license, accept the physical IP address for the license, and save the license information to a file. After saving the license information to a file, the file can be edited to change the IP address to the floating IP address as needed. This can then be faxed or e-mailed to HP for processing.

Upon receiving the licenses, it is recommended that these be installed using ovautoLic –import. It is possible that some licenses might be the new OvKey4 format, and only AutoPass understands these. As part of the import process, these licenses will be copied or converted back into the $OV_CONF/.license file.

Running Multiple Instances of NNM in the cluster environment : Note: The essential part here is to understand that there are other ways of implenting the **MxN** setup and it is upto the user to decide on the configuration that is most suitable for the intended environment. The setup of MxN means M+N number of systems are participating in the cluster, where M instance of NNM are running in M systems, and N number of adaptive systems are designated. In this setup, typically N is less than or equal to M and thus N out of M NNM instance can failover to N adaptive systems in the cluster. During the failover scenario, any instance of NNM can failover to any adaptive system and this is purely incidental and non deterministic. So, From the standpoint of the adaptive systems, this leads to a situation of more than one instance of NNM trying to failover to one adaptive system. This is described as n:1 resource sharing/contention problem, where 'n' number of instances that are trying to occupy one adaptive system. So, if this n:1 resource contention problem is avoided, the configuration of MxN NNM in an MC/SG cluster becomes practical.

The MxN and n:1 systems could be well understood if we consider the following example of 3x2 set up. Here, 5 systems will participate in the cluster. In which 3 instance of NNMs run in three systems and 2 adaptive systems are configured. So, the possible scerios could be

i. One out of three NNM instances is trying to failover to any one of the adaptive system.

   From the standpoint of the adaptive system, this is same as that of the usual scenario where one application instance is failing over to one of the adaptive system. The choice of which adaptive system is selected is left to the basic MC/SG system. And so there is no special case associated with this it is same as that of the classical NNM MG/SG behaviour.

ii. Two out of three NNM instances are trying to failover to the available two adaptive systems almost at the same time.

In this case the possibilities are, the first NNM instance can failover to any one of the adapative system successfully before the second NNM instance starts its failover in reality. This case will behave like the classical NNM MC/SG scenaio and when the second NNM instance failover there is only one adpative system available and it would also follow the classical NNM MG/SG set up behaviour and there wont be any special behaviour observed.

If both the NNM instances try to failover at the same time, it is possible that both might choose the same adaptive system to failver. In this case there is a 2:1 resouce contention problem, meaning to say that two instances of NNM are trying to failover to one adaptive system. The adaptive system should only permit one NNM instance to failover successfully and reject the other NNM instance. The solution for the 2:1 failover is given below and described as n:1 solution model. The rejected NNM instance should failver to the other available adaptive system without any problem.

iii. Three out of three NNM instances are trying to failover to the available two adaptive systems almost at the same time.

Here again, if all the three NNM instances try to failover to one of the two adaptive systems, it leads to 3:1 resourse contention problem and explained below as the n:1 solution mode. In this case since there are only two adaptive systems in the cluster, only two out of the three NNM instances will finally succeed in failing over.

Thus MxN configuration of NNM could be achieved if n:1 set could be addressed with the below mentioned setup. Other than this there is no other difference in the behaviour of NNM in an the classical NNM in MC/SG setup (1:1) to the MxN setup. Though the example configuration scripts provided in the annexures are for implementing n:1 model, the same could be extropolated to accommodate MxN setup also.

n:1 solution model for NNM under MG/SG setup

The concept of **n:1** is the existence of **n** systems cluster, with each system executing an instance of NNM and having **1** system designated as the adoptive system. The execution of NNM instance on a failover to the backup system is dependent on a first come first served basis, potentially leading to a resource contention like mounting of shared disk containing the data in */var/opt/OV/share* and */etc/opt/OV/share* directories.. The first instance that establishes the file lock process gets to execute on the adoptive node. NNM instances that tries to failover to its adoptive node exits on detecting the file lock process. It is mandatory to execute the lockfile process to avoid the resouce contention.

For understanding purposes, let **n = 3** (3:1 setup), implying that there are three systems (nnmHAs1, nnmHAs2 and nnmHAs3) and three instances of NNM (nnmHAp1, nnmHAp2 and nnmHAp3) with another additional system (nnmHAs4) as an adoptive node for the three instances of NNM. This implies that there would be a package control, package config and OpenView configuration file per instance of NNM, amounting to three sets of scripts in the */etc/cmcluster/nnm* directory via: *nnm1.config, nnm2.config, nnm3.config, nnm1.control, nnm2.control, nnm3.control, ov.conf_nnmHAs1, ov.conf_nnmHAs2* and *ov.conf_nnmHAs3* placed in the adoptive node *nnmHAs4*. The files *nnm1.config, nnm1.control* and *ov.conf_nnmHAs1* would be placed in the system *nnmHAs1*, while *nnm2.config, nnm2.control* and *ov.conf_nnmHAs2* would be placed in the system *nnmHAs2* and also *nnm3.config, nnm3.control* and *ov.conf_nnmHAs3* would be placed in the system *nnmHAs3* A copy of

nnm_file_lock.ovpl, Monitor is to be distributed on all the four systems in the setup. All these files are to be placed in the directory */etc/cmcluster/nnm*.

On a failover of the package *nnmHA1* to the adoptive node *nnmHAs4*, it would check if the */etc/cmcluster/nnm/ Nnm_lockNnm_lock* file could be created. On an error, the startup of the package *nnmHA1*on the adoptive node *nnmHAs4* would be terminated, implying that there is already a package that is currently executing there. A success in creating the lock file is an indication that the adoptive node *nnmHAs4* is free to execute the package *nnmHA1* that has failed over.

The package *nnmHA1* on the adoptive node *nnmHAs4* would then bind the package i/p address to *nnmhas4* and continue to execute as a normal NNM instance. Messages generated by the execution of NNM package would be available in the log file **/etc/cmcluster/nnm/nnm.control.log** and **/var/adm/syslog/syslog.log** for the message generated by the cluster node participants.

Disclaimer: Behavior of the package on the node cannot be assured if the files *nnm_lock* in /etc/cmcluster/nnm directory are deleted manually. There could be problems while cut and pasting these templates and using them as scripts, please ensure removal of incorrect insertions of newlines. The suggested templates in this document are per- instance of NNM and suitable number of templates are to be prepared and distributed to all the system in the setup appropriately to ensure the predicted behavior of NNM instances in the setup.

In the Appendices A, B, C, D, E and F are the templates suggested for the implementation of the **standard two node** or **n:1** configuration setup of MC/ServiceGuard for NNM and need appropriate modifications in tune with the environment that would have the setup. Described further are the guidelines that indicate the modifications to be made for the templates.

Suggested modification for Appendix A:

The template is used for cluster configuration and could be named **cluster.config** to be stored in the directory **/etc/cmcluster/nnm**. Please consult your systems administrator while modifying or applying the cluster configuration and is dependent on the type of MC/SerivceGuard setup.

Suggested modification for Appendix B:

The template is used for package configuration and could be named **nnm.config** to be stored in the directory **/etc/cmcluster/nnm**. The changes that are to be made are:

PACKAGE_NAME: Modify the value according to value of **n** in the **n:1** setup.

NODE_NAME: Specify the primary node name and then repeat the variable with the node name of the adoptive node.

SUBNET: This should be the subnet id that the cluster would be existing in and is generally identical for the cluster participants.

If NFS is used add the following service definition:

SERVICE_NAME	NFS
SERVICE_FAIL_FAST_ENABLED	NO
SERVICE_HALT_TIMEOUT	300

Suggested modification for Appendix C:

The template is used for package control and could be named **nnm.control** to be stored in the directory **/etc/cmcluster/nnm**. Mounting of the shared data could be either through NFS or shared disks connected to the cluster participant.

The suggested changes to be made if shared disk is used for shared data:

VG[0]: Add the volume group that have been made available for the shared data of NNM, to be used by the primary node and later by the adoptive node on a failover. This configuration change has to be made for each volume groups that would be mounted for the package. Indicated in the template is the definition of a single volume group. Consult your sysadmin for the details on the number of volume groups configured for the cluster.

LV[0], FS[0] and FS_MOUNT_OPT[0]: Modify these variables according the logical volumes partitioned in the volume group mentioned above. The logical volume partitions index depends on the configuration.

IP[0]: Virtual i/p address assigned to the package is to be provided. This virtual i/p address should not be assigned to any physical device, but should be resolvable by the domain name server.

SUBNET[0]: The subnet id that IP[0] belongs to and could be identical to the SUBNET defined in **nnm.config** file.

SERVICE_RESTART[0]="-r 2": The numeric **2** is the number of times a restart of NNM is attempted before a failover would be initiated. This value is to be modified depending on the requirement.

The suggested changes to be made if NFS is used for shared data:

XFS[0]="-o root=robot /etc/opt/OV/share"
XFS[1]="-o root=robot /var/opt/OV/share"

The service must also be defined in the Package script.
SERVICE_NAME[1]="NFS"
SERVICE_CMD[1]="/etc/cmcluster/nnm/nfs.mon"
SERVICE_RESTART[1]=

Suggested modification for Appendix D:

This template could be named Monitor to be placed in the directory **/etc/cmcluster/nnm**. There are **no** changes required for this template.

Suggested modification for Appendix E:

There are three fields to be configured in the *$OV_CONF/ov.conf* file: HOSTNAME, NNM_INTERFACE, USE_LOOPBACK.

HOSTNAME=<>: The value for this field is the actual hostname of the machine currently running NNM. This entry must reflect the hostname which resolves to the static IP Address of the system on which NNM is currently running.

NNM_INTERFACE=<>: The value for this field is the network IP name or IP address through which NNM discovers and manages the network. In a MC/ServiceGuard configuration, the value for this field is the relocatable IP name or address associated with the NNM package.

USE_LOOPBACK=<>: The value for this field determines whether loopback is enabled (ON or OFF). To maintain backward compatibility the default value for this field is OFF). The recommended value for this field is ON. When set to ON, NNM processes will continue even if all LAN interfaces are down. The entries are position/line independent.

Copy this file to */etc/opt/OV/share/conf/ov.conf.<hostname1>* and */etc/opt/OV/share/conf/ov.conf.<hostname2>*, where <*hostname1*> and <*hostname2*> are the hostnames of the two systems in the cluster. Edit the HOSTNAME field in each of these files to reflect the respective hostname. Two files are necessary since each system has a different hostname. Two files are used to simplify NNM package switching between the systems in the cluster, but automating an edit of a single ov.conf before NNM starts on each system is also acceptable. See the ov.conf(4) man page for a more detailed description of these fields.

Suggested modification for Appendix F:

This template could be named **nnm_file_lock.ovpl** and is to be placed in the directory **/etc/cmcluster/nnm**. The changes required for this template are:

Path to the perl location is to be modified as per the installation of NNM. and the version of Perl has to be at least that of the perl provided along with NNM installation.

Do not introduce any type of quote in the above mentioned files.

Appendix A : Sample cluster configuration file

Suggested directory to store: **/etc/cmcluster/nnm**

File permissions: **0755**

Ownership: **root:sys**

Note: please refer to the suggestion made earlier in the document for the template modification guideline:

```
# **********************************************************************
# ********* 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                    NNM

# 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/vgfmpsvr

# Definition of nodes in the cluster.
# Repeat node definitions as necessary for additional nodes.
NODE_NAME               nnmha1
NETWORK_INTERFACE           lan0
HEARTBEAT_IP                    15.70.182.29
FIRST_CLUSTER_LOCK_PV       /dev/dsk/c0t15d0

# List of serial device file names
# For example:
#SERIAL_DEVICE_FILE          /dev/tty1p0

# Primary Network Interfaces on Bridged Net 1: lan0.
#   Warning: There are no standby network interfaces on bridged net 1.
NODE_NAME               nnmha2
NETWORK_INTERFACE           lan0
HEARTBEAT_IP                    15.70.182.30
FIRST_CLUSTER_LOCK_PV       /dev/dsk/c0t15d0
# List of serial device file names
# For example:
#SERIAL_DEVICE_FILE          /dev/tty1p0

# Primary Network Interfaces on Bridged Net 1: lan0.
#   Warning: There are no standby network interfaces on bridged net 1.

# Cluster Timing Parmeters (microseconds).
HEARTBEAT_INTERVAL                  1000000
NODE_TIMEOUT                    2000000

# 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

# List of cluster aware Volume Groups. These volume groups will
# be used by clustered applications via the vgchange -a e command.
# For example:
# VOLUME_GROUP              /dev/vgdatabase.
# VOLUME_GROUP              /dev/vg02.

# List of cluster aware Volume Groups. These volume groups will
# be used by DLM applications via the vgchange -a s command.
# For example:
# DLM_VOLUME_GROUP                  /dev/vgdatabase.
# DLM_VOLUME_GROUP                  /dev/vg02.

DLM_VOLUME_GROUP                    /dev/vgfmpsvr
DLM_VOLUME_GROUP                    /dev/vgfmpora1
DLM_VOLUME_GROUP                    /dev/vgfmpora2
DLM_VOLUME_GROUP                    /dev/vgfmpmd

# DLM parameters.
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
```

Appendix B: **Sample NNM package configuration file**

Suggested directory to store: **/etc/cmcluster/nnm**

File permissions: **0755**

Ownership: **root:sys**

Note: Please refer to the suggestion made earlier in the document for the template modification guideline**:**

```
# ********************************************************************
# ****** 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                nnm

# 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           nnmha1
NODE_NAME           nnmha2

# 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/nnm/nnm.control
RUN_SCRIPT_TIMEOUT      NO_TIMEOUT
HALT_SCRIP          /etc/cmcluster/nnm/nnm.control
HALT_SCRIPT_TIMEOUT     NO_TIMEOUT

# 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>

SERVICE_NAME                 nnm
SERVICE_FAIL_FAST_ENABLED      NO
SERVICE_HALT_TIMEOUT           300

# 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   15.70.182.0

# 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/lan/lan0/res1
#           RESOURCE_POLLING_INTERVAL     120
#           RESOURCE_UP_VALUE             = RUNNING
#           RESOURCE_UP_VALUE             = ONLINE
#
#           Means that the value of resource /net/lan/lan0/res1 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           YES

# 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

# 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
```

Appendix C: Sample NNM package control file

Suggessted directory to store: **/etc/cmcluster/nnm**
File permissions: 0755
Ownership: root:sys

Note: Please refer to the suggestion made earlier in the document for the template modification guideline. This sample file does not include the entries for NFS that would be required to support distributed/management consoles:

```
#"(#) A.10.12                          $Date: 09/08/1999 $"
# ************************************************************************
# *                                                      *
# *       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=/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 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 y"
# VGCHANGE="vgchange -a e"          # Default

VGCHANGE="vgchange -a e"            # Default


# 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]=""

VG[0]=vgfmpsvr

# 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,
# which are mounted on the logical volumes lvol1 and lvol2 with read and
# 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.
#
#LV[0]=""; FS[0]=""; FS_MOUNT_OPT[0]=""

LV[0]=/dev/${VG[0]}/lvfmpetc
FS[0]=/etc/opt/OV/share
FS_MOUNT_OPT[0]="-o rw"

LV[1]=/dev/${VG[0]}/lvfmpvar
FS[1]=/var/opt/OV/share
FS_MOUNT_OPT[1]="-o rw"


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

FS_UMOUNT_COUNT=1


# FILESYSTEM MOUNT RETRY COUNT.
# Specify the number of mount retrys for each filesystem.
# The default is 0. During startup, if a mount point is busy
# and FS_MOUNT_RETRY_COUNT is 0, package startup will fail and
# the script will exit with 1.  If a mount point is busy and
# FS_MOUNT_RETRY_COUNT is greater than 0, the script will attempt
# to kill the user responsible for the busy mount point
```

```
# and then mount the file system.  It will attempt to kill user and
# retry mount, for the number of times specified in FS_MOUNT_RETRY_COUNT.
# If the mount still fails after this number of attempts, the script
# will exit with 1.
# NOTE: If the FS_MOUNT_RETRY_COUNT > 0, the script will execute
# "fuser -ku" to freeup busy mount point.


FS_MOUNT_RETRY_COUNT=0


# 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]="15.70.182.114"
SUBNET[0]="15.70.182.0"


# SERVICE NAMES AND COMMANDS.
# Specify the service name, command, and restart parameters which 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]=""

SERVICE_NAME[0]="nnm"
SERVICE_CMD[0]="/etc/cmcluster/nnm/Monitor"
SERVICE_RESTART[0]="-r 2"

# 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.


# this function has been added for nnm to ensure that the process locking
# the /etc/cmcluster/nnm/nnm_lock is safely removed using SIGINT.  the lock
# file is neither deleted nor truncated by the process locking it.
function nnm_exit
{
# store the input value passed to this function
    typeset received_value
    receved_value=$1

# access the pid from nnm_lock file, now kill the lock process
     kill -2 `cat /etc/cmcluster/nnm/nnm_lock` 2>/dev/null
     echo "------------------------terminated the lock file process having PID = ${killpid} at `date`"

# the file is removed just in case it is still dangling
     if [ -e "/etc/cmcluster/nnm/nnm_lock" ]
     then
        rm /etc/cmcluster/nnm/nnm_lock 2>/dev/null
     fi

# exit from the script with the value received as a parameter to this function
     exit ${received_value}
}
# end of function nnm_exit


function customer_defined_run_cmds
{
 # ensure the specified ov.conf file is avialable for NNM to startup
    cp /etc/opt/OV/share/conf/ov.conf.`hostname`  /etc/opt/OV/share/conf/ov.conf
    echo "copied the ov.conf file with system name"
```

```
 # found a previously left over pause lock file ?
   if [ -f /var/opt/OV/tmp/ovpause.lock ]
   then
      rm /var/opt/OV/tmp/ovpause.lock
      echo "removed the ovpause lock file"
   fi

 # startup NNM as per the configuration
  /opt/OV/bin/ovstart -v

   test_return 51
}

# 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
{
 # stop all the configured process of NNM
      /opt/OV/bin/ovstop -v
      echo "stopped all NNM processes"

 # remove the ovserver file name
      rm /var/opt/OV/share/databases/openview/ovwdb/ovserver 2>/dev/null
      echo "removed the ovserver file"

 # found a previously left over pause lock file ?
      if [ -f /var/opt/OV/tmp/ovpause.lock ]
      then
         rm /var/opt/OV/tmp/ovpause.lock 2>/dev/null
         echo "removed the ovpause lock file"
      fi

      test_return 52
}

# END OF CUSTOMER DEFINED FUNCTIONS


# START OF RUN FUNCTIONS


function activate_volume_group
{
for I in ${VG[@]}
do
          if [[ "${VGCHANGE}" = "vgchange -a y" ]]
          then
          print "$(date '+%b %e %X') - Node \"$(hostname)\": Activating volume group $I with non-
exclusive option."
          else
          print "$(date '+%b %e %X') - \"$(hostname)\": Activating volume group $I with exclusive option."
          fi

          $VGCHANGE $I
```

```
        test_return 1
done
}

#This function is used to kill the user to freeup a mountpoint
#that could be busy and then do the mount operation.
#freeup_busy_mountpoint_and_mount_fs(x, y, z)
#       x  =  Logical volume group to be mounted.
#       y  =  File System where the logical volume is to be mounted.
#     z  =  Mount Options to be used for mount operation
#
function freeup_busy_mountpoint_and_mount_fs
{
typeset vol_to_mount
typeset mount_pt
typeset fs_mount_opt

vol_to_mount=$1
mount_pt=$2
shift 2
fs_mount_opt=$*

print "\tWARNING:   Running fuser on ${mount_pt} to remove anyone using the busy mount point
directly."
UM_COUNT=0
RET=1

# The control script exits, if the mount failed after
# retrying FS_MOUNT_RETRY_COUNT times.

while (( $UM_COUNT < $FS_MOUNT_RETRY_COUNT && $RET != 0 ))
do
      (( UM_COUNT = $UM_COUNT + 1 ))
      fuser -ku ${mount_pt}
      if (($UM_COUNT == $FS_MOUNT_RETRY_COUNT))
      then
            mount ${fs_mount_opt} ${vol_to_mount} ${mount_pt}
            test_return 17
      else
            mount ${fs_mount_opt} ${vol_to_mount} ${mount_pt}
            (( RET = $? ))
            sleep 1
      fi
done
}

# For each {file system/logical volume} pair, fsck the file system
# and mount it.

function check_and_mount
{
integer R=0

for I in ${LV[@]}
do
        if [[ $(mount -p | awk '$1 == "'$I'"') = "" ]]
```

```
        then
                RLV[$R]="${I%/*}/r${I##*/}"

        if [ -x /usr/sbin/fstyp ]
        then
                        fstype[$R]=$(fstyp $I)
                fi
                (( R = $R + 1 ))
        fi
done

# Verify that there is at least one file system to check and what type.
if [[ ${RLV[@]} != "" ]]
then
        print -n "$(date '+%b %e %X') - Node \"$(hostname)\": "
        print "Checking filesystems:"
        print ${LV[@]} | tr ' ' '\012' | sed -e 's/^/   /'

        # If there is more than one filesystem type being checked
        # then each filesystem is check individually.
        #
        R=$(print ${fstype[*]} | tr ' ' '\012' | sort -u | wc -l)
        if (( R > 1 ))
        then
           R=0
           while (( R < ${#RLV[*]} ))
           do
                case ${fstype[$R]} in

                   hfs)   fsck -F hfs -P ${RLV[$R]}
                                test_return 2
                                ;;

                   vxfs)  fsck -F vxfs -y ${RLV[$R]}
                                test_return 2
                                ;;

                   unk*) fsck ${RLV[$R]}
                                test_return 2
                                ;;

                   *)           if [[ ${fstype[$R]} = "" ]]
                                then
                                   fsck ${RLV[$R]}
                                else
                                   fsck -F ${fstype[$R]} ${RLV[$R]}
                                fi
                                test_return 2
                                ;;
                esac
                (( R = R + 1 ))
           done

        # If there is only one filesystem type being checked, then
        # multiple invocations of fsck can be avoided. All filesystems
        # are specified on the command line to one fsck invocation.
```

```
        #
        else
           case ${fstype} in

                hfs)    fsck -F hfs -P ${RLV[@]}
                        test_return 2
                        ;;

                vxfs)   fsck -F vxfs -y ${RLV[@]}
                        test_return 2
                        ;;

                unk*)   fsck ${RLV[@]}
                        test_return 2
                        ;;

                *)      if [[ ${fstype} = "" ]]
                        then
                           fsck ${RLV[@]}
                        else
                           fsck -F ${fstype} ${RLV[@]}
                        fi
                        test_return 2
                        ;;
           esac
        fi
fi

# Check exit value (set if any proceeding fsck calls failed)

if (( $exit_value == 1 ))
then
      deactivate_volume_group
      print "\n\t############ Node \"$(hostname)\": Package start failed  at $(date) ############"
      nnm_exit 1
fi

integer F=0
for I in ${LV[@]}
do
        if [[ $(mount | grep -E $I" ") = "" ]]
        then
                print "$(date '+%b %e %X') - Node \"$(hostname)\": Mounting $I at ${FS[$F]}"

                #if there is permission to kill the user, we can
                #run fuser to kill the user, on the mount point.
             #This would freeup the mount point, if it is busy

                if (( $FS_MOUNT_RETRY_COUNT > 0 ))
                then
                        mount ${FS_MOUNT_OPT[$F]} $I ${FS[$F]}
                        if (( $? != 0 ))
                        then
                                freeup_busy_mountpoint_and_mount_fs $I ${FS[$F]}
${FS_MOUNT_OPT[$F]}
                        fi
```

```
                else
                        mount ${FS_MOUNT_OPT[$F]} $I ${FS[$F]}
                        test_return 3
                fi

        else
                print "$(date '+%b %e %X') - Node \"$(hostname)\": WARNING:  File system \"${FS[$F]}\"
was already mounted."
        fi
        (( F = $F + 1 ))
done
}

# For each {IP address/subnet} pair, add the IP address to the subnet
# using cmmodnet(1m).

function add_ip_address
{
integer S=0
integer error=0

for I in ${IP[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Adding IP address $I to subnet
${SUBNET[$S]}"
        XX=$( cmmodnet -a -i $I ${SUBNET[$S]} 2>&1 )
        if (( $? != 0 ))
        then
                YY=$( netstat -in | awk '$4 == "${I}"')
                if [[ -z $YY ]]
                then
                        print "$XX" >> $0.log
                        print "\tERROR:  Failed to add IP $I to subnet ${SUBNET[$S]}"
                        (( error = 1 ))
                else
                        print "\tWARNING:  IP $I is already configured on the subnet ${SUBNET[$S]}"
                fi
        fi
        (( S = $S + 1 ))
done

if (( error != 0 ))
then

# `let 0` is used to set the value of $? to 1. The function test_return
# requires $? to be set to 1 if it has to print error message.

        let 0
        test_return 4
fi

}

# Own and reset the DTC connections

function get_ownership_dtc
```

```
{
for I in ${DTC_NAME[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Assigning Ownership of the DTC $I"
        dtcmodifyconfs -o $I
        test_return 5

        for J in ${IP[@]}
        do
                print "$(date '+%b %e %X') - Node \"$(hostname)\": Resetting the DTC connections to IP
address $J"
                dtcdiag -Q $J -q -f $I
                test_return 6
        done
done
}


# For each {service name/service command string} pair, start the
# service command string at the service name using cmrunserv(1m).

function start_services
{
integer C=0
for I in ${SERVICE_NAME[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Starting service $I using"
        print "   \"${SERVICE_CMD[$C]}\""
        #
        # Check if cmrunserv should be called the old
        # way without a restart count.
        #
        if [[ "${SERVICE_RESTART[$C]}" = "" ]]
        then
           cmrunserv $I ">> $0.log 2>&1 ${SERVICE_CMD[$C]}"
        else
           cmrunserv ${SERVICE_RESTART[$C]} $I ">> $0.log 2>&1 ${SERVICE_CMD[$C]}"
        fi
        test_return 8
        (( C = $C + 1 ))
done
}

# END OF RUN FUNCTIONS.



# START OF HALT FUNCTIONS

# Halt each service using cmhaltserv(1m).

function halt_services
{
for I in ${SERVICE_NAME[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Halting service $I"
```

```
        cmhaltserv $I
        test_return 9
done
}

# Disown the DTC.

function disown_dtc
{
for I in ${DTC_NAME[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Disowning the DTC $I"
        dtcmodifyconfs -d $I
        test_return 11
done
}

# For each IP address/subnet pair, remove the IP address from the subnet
# using cmmodnet(1m).

function remove_ip_address
{
integer S=0
integer error=0

for I in ${IP[@]}
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Remove IP address $I from subnet
${SUBNET[$S]}"
        XX=$( cmmodnet -r -i $I ${SUBNET[$S]} 2>&1 )
        if (( $? != 0 ))
        then
                echo $XX | grep "is not configured on the subnet"
                if (( $? != 0 ))
                then
                        print "$XX" >> $0.log
                        (( error = 1 ))
                fi
        fi
        (( S = $S + 1 ))
done
if (( $error != 0 ))
then

# `let 0` is used to set the value of $? to 1. The function test_return
# requires $? to be set to 1 if it has to print error message.

        let 0
        test_return 12
fi
}

# Unmount each logical volume.

function umount_fs
{
```

```
integer UM_CNT=${FS_UMOUNT_COUNT:-1}
integer ret

set -A LogicalVolumes ${LV[@]} ${CVM_LV[@]}

if [[ $UM_CNT < 1 ]]
then
   UM_CNT=1
fi

integer L=${#LogicalVolumes[*]}
while (( L > 0 ))
do
        (( L = L - 1 ))
        I=${LogicalVolumes[$L]}
        mount | grep -e $I" " > /dev/null 2>&1
        if (( $? == 0 ))
        then
                print "$(date '+%b %e %X') - Node \"$(hostname)\": Unmounting fil
esystem on $I"
                umount $I; ret=$?
                if (( ret != 0 ))
                then
                   print "\tWARNING:   Running fuser to remove anyone using the
file system directly."
                fi

 UM_COUNT=$UM_CNT
                while (( ret != 0 && UM_COUNT > 0 ))
                do
                  fuser -ku $I
                  umount $I; ret=$?
                  if (( ret != 0 ))
                  then
                       if (( $UM_COUNT == 1 ))
                       then
                            let 0
                            test_return 13
                       fi
                       (( UM_COUNT = $UM_COUNT - 1 ))
                       sleep 1
                       if (( $UM_COUNT > 0 ))
                       then
                            print "\t$(date '+%b %e %X') - Unmount failed, tr
ying again."
                       fi
                fi
                done
        fi
done
}


function deactivate_volume_group
{
for I in ${VG[@]}
```

```
do
        print "$(date '+%b %e %X') - Node \"$(hostname)\": Deactivating volume group $I"
        vgchange -a n $I
        test_return 14
done
}

# END OF HALT FUNCTIONS.

# FUNCTIONS COMMON TO BOTH RUN AND HALT.

# Test return value of functions and exit with NO RESTART if bad.
# Return value of 0 - 50 are reserved for use by Hewlett-Packard.
# System administrators can use numbers above 50 for return values.
function test_return
{
if (( $? != 0 ))
then
        case $1 in
                1)
                print "\tERROR:  Function activate_volume_group"
                print "\tERROR:  Failed to activate $I"
                deactivate_volume_group
                nnm_exit 1
                ;;

                2)
                print "\tERROR:  Function check_and_mount"
                print "\tERROR:  Failed to fsck one of the logical volumes."
                exit_value=1
                ;;

                3)
                print "\tERROR:  Function check_and_mount"
                print "\tERROR:  Failed to mount $I to ${FS[$F]}"
                umount_fs
                deactivate_volume_group
                nnm_exit 1
                ;;

                4)
                print "\tERROR:  Function add_ip_address"
                print "\tERROR:  Failed to add IP address to subnet"
                remove_ip_address
                umount_fs
                deactivate_volume_group
                nnm_exit 1
                ;;

                5)
                print "\tERROR:  Function get_ownership_dtc"
                print "\tERROR:  Failed to own $I"
                disown_dtc
                remove_ip_address
                umount_fs
                deactivate_volume_group
```

```
nnm_exit 1
;;

6)
print "\tERROR:  Function get_ownership_dtc"
print "\tERROR:  Failed to switch $I"
disown_dtc
remove_ip_address
umount_fs
deactivate_volume_group
nnm_exit 1
;;

8)
print "\tERROR:  Function start_services"
print "\tERROR:  Failed to start service ${SERVICE_NAME[$C]}"
halt_services
customer_defined_halt_cmds
disown_dtc
remove_ip_address
umount_fs
deactivate_volume_group
nnm_exit 1
;;

9)
print "\tFunction halt_services"
print "\tWARNING: Failed to halt service $I"
;;

11)
print "\tERROR:  Function disown_dtc"
print "\tERROR:  Failed to disown $I from ${SUBNET[$S]}"
exit_value=1
;;

12)
print "\tERROR:  Function remove_ip_address"
print "\tERROR:  Failed to remove $I"
exit_value=1
;;

13)
print "\tERROR:  Function umount_fs"
print "\tERROR:  Failed to unmount $I"
exit_value=1
;;

14)
print "\tERROR:  Function deactivate_volume_group"
print "\tERROR:  Failed to deactivate $I"
exit_value=1
;;

17)
print "\tERROR:  Function freeup_busy_mountpoint_and_mount_fs"
```

```
                print "\tERROR:  Failed to mount $I to ${FS[$F]}"
                umount_fs
                deactivate_volume_group
                nnm_exit 1
                ;;

                51)
                print "\tERROR:  Function customer_defined_run_cmds"
                print "\tERROR:  Failed to RUN customer commands"
                halt_services
                customer_defined_halt_cmds
                disown_dtc
                remove_ip_address
                umount_fs
                deactivate_volume_group
                nnm_exit 1
                ;;

                52)
                print "\tERROR:  Function customer_defined_halt_cmds"
                print "\tERROR:  Failed to HALT customer commands"
                exit_value=1
                ;;

                *)
                print "\tERROR:  Failed, unknown error."
                ;;
        esac
fi
}

# END OF FUNCTIONS COMMON TO BOTH RUN AND HALT

#-------------------MAINLINE Control Script Code Starts Here-----------------
#
# FUNCTION STARTUP SECTION.

typeset MIN_VERSION="A.10.03"  # Minimum version this control script works on

integer exit_value=0
typeset CUR_VERSION

#
# Check that this control script is being run on a A.10.03 or later release
# of MC/ServiceGuard or MC/LockManager.  The control scripts are forward
# compatible but are not backward compatible because newer control
# scripts use commands and option not available on older releases.

CUR_VERSION="$(/usr/bin/what /usr/lbin/cmcld | /usr/bin/grep "Date" | \
                        /usr/bin/egrep '[AB]\...\...|NTT\...\...' | \
                                cut -f2 -d" ")"

if [[ "${CUR_VERSION}" = ""  ]] || \
                        [[ "${CUR_VERSION#*.}" < "${MIN_VERSION#*.}" ]]
then
   print "ERROR:  Mismatched control script version ($MIN_VERSION).  You cannot run"
```

```
    print "\ta version ${MIN_VERSION} control_script on a node running pre"
    print "\t${MIN_VERSION} MC/ServiceGuard or MC/LockManager software"
    exit 1
fi

# Test to see if we are being called to run the package, or halt the package.


if [[ $1 = "start" ]]
then
          print "\n\t############ Node \"$(hostname)\": Starting package at $(date) ############"

# added for NNM High Availability
# to ensure that there are no contender for the shared resource for executing
# nnm pakcages on the adoptive server.  a lock file would by created by the
# package at start on a first come first server basis.  this lock file exists
# until a package exits server.  the lock file would be removed by a SIGINT
# given to the the lock file process at the termination of the package
# execution on the server.
# logic for startup of process is that if the lock process exist then exit,
# elseif the lock file exists, remove it and startup the lock file process.

    typeset adoptnode
    adoptnode=`cmviewcl -l package -v | grep -v grep | grep "(current)" | awk '{print $1}'`
    if [ "${adoptnode}" = "Alternate" ]
    then
       typeset lockfile="/etc/cmcluster/nnm/nnm_lock"

# startup the lock file process, parent process returns while child continues
# to hold the lock file
       /etc/cmcluster/nnm/nnm_file_lock.ovpl

# check for the return status from the parent process of lockfile process.
# if exit value = 0, then this script has the right to execute package on this
# node
# if the exit value = 1, then there is already an existence of another lockfile
# process, hence exit from further execution
# if the exit value = 2, then there is a problem with the node

       case $? in
         0)   echo "Secured the lock file, continuing with the package startup\n"
              break;;

         1)   echo "The parent lockfile process could not write into ${lockfile} with the child lockfile process,
hence exiting from package startup"
              exit 1 ;;

         2)   echo "The parent lockfile process could not either close or unlink ${lockfile} after terminating
the child lockfile process due to a failure in writing the child process pid into ${lockfile}"
              exit 2;;

         3)   echo "the lockfile parent process has encountered a fork failure.  please remove the
/etc/cmcluster/nnm/nnm_lock file after analysis of the reason for exit"
              exit 3;;

         *)   echo "unknown exit received, hence exiting from further execution"
```

```
            exit 255;;
        esac

# display the status of the the execution
        echo "started the lock file process at `date` with lock file $lockfile with a PID = `cat $lockfile` "

    fi
## end of addition for NNM High Availability

            activate_volume_group

            check_and_mount

            add_ip_address

            get_ownership_dtc

            customer_defined_run_cmds

            start_services

# Check exit value

            if (( $exit_value == 1 ))
            then
                        print "\n\t############ Node \"$(hostname)\": Package start failed  at $(date)
############"
                        nnm_exit 1
            else
                        print "\n\t############ Node \"$(hostname)\": Package start completed  at $(date)
############"
                        exit 0
            fi

elif [[ $1 = "stop" ]]
then
            print "\n\t############ Node \"$(hostname)\": Halting package at $(date) ############"


            halt_services

            customer_defined_halt_cmds

            disown_dtc

            remove_ip_address

            umount_fs

            deactivate_volume_group

# Check exit value
            if (( $exit_value == 1 ))
            then
                        print "\n\t############ Node \"$(hostname)\": Package halt failed at $(date)
############"
```

```
                nnm_exit 1
        else
                print "\n\t############ Node \"$(hostname)\": Package halt completed at $(date)
############"
                nnm_exit 0
        fi

fi
```

Appendix D: Sample NNM process monitor script

Suggested directory to store: **/etc/cmcluster/nnm**

File permissions: **0755**

Ownership: **root:sys**

Note: Please refer to the suggestion made earlier in the document for the template modification guideline.

```
#!/usr/bin/sh
# ************************************************************************
# ************ NNM SERVICE GUARD Monitor Daemon Script *****************
# ************************************************************************
#
#              MONITOR Shell Script for NNM &
#                   MC/ServiceGuard
#
# This shell script monitors NNM by making sure that the necessary
# NNM background processes are up and running.
#

PATH=/opt/OV/bin:/bin:/usr/bin:$PATH

# Set MUSTRUN to contain names of NNM processes that should be monitored.
MUSTRUN="netmon ovspmd ovtrapd ovwdb ovtopmd ovuispmd ovalarmsrv httpd"

trap "exit " 15

##############################################################################
# Monitor the NNM processes by making sure that all required processes are running.

while true
do
   for i in $MUSTRUN
   do
      # Check that MUSTRUN processes are running
      ps -ef | grep $i | grep -v grep > /dev/null 2>&1
      if [ $? -eq 1 ]
      then
            # If a process is not running, then rerun ovstart to restart
            # the failed processes.  ovstart will only restart failed
            # processes.
            /opt/OV/bin/ovstart
            sleep 5   # wait ovspmd execution of daemon to fail
            for j in $MUSTRUN
            do
                  ps -ef | grep $j | grep -v grep > /dev/null 2>&1
                  if [ $? -eq 1 ]
                  # If a process won't run, then exit
                  then
                     echo "Process $j won't run exiting service"
                     exit 1
```

```
                fi
            done
        fi
    done
    sleep 30
    echo "Found all processes `date`, message from /etc/cmcluster/nnm/Monitor script"
done

exit 0
```

Appendix E: **Sample NNM ov.conf file**

Suggested directory to store: **/etc/cmcluster/nnm**

File permissions: **0755**

Ownership: **root:sys**

Note: Please refer to the suggestion made earlier in the document for the template modification guideline:

```
# Sample file for /etc/opt/OV/share/conf/ov.conf.sgtest1
HOSTNAME=sgtest1.hp.com
NNM_INTERFACE=spike.hp.com
USE_LOOPBACK=OFF

#Sample file for /etc/opt/OV/share/conf/ov.conf.sgtest2
HOSTNAME=sgtest2.hp.com
NNM_INTERFACE=spike..hp.com
USE_LOOPBACK=OFF
```

AppendixF : Sample NNM lockfile process script

Suggested directory to store: **/etc/cmcluster/nnm**

File permissions: **0755**

Ownership: **root:sys**

Note: Please refer to the suggestion made earlier in the document for the template modification guideline. This code is to be used for the n:1 setup:

```
#!/opt/OV/bin/Perl/bin/perl
# @(#) revision S.01.00 (01/02/2002 ) Copyright : Hewlett-Packard Company Ltd.

##############################################################################
#   file name           :  nnm_file_lock.ovpl
#   version             :
#   description         :  nnm_file_lock.ovpl is for avoiding resource
#       contention on a failover to the backup server.  the logic behind is
#       check for creating the lock file in the /etc/cmcluster/nnm/.
#       directory with a file name nnm_lock containing the pid of the process
#       that is assigned at execution of this perl script.  the signal handler
#       is set for SIGINT which is used to terminate this process.  untill the
#       SIGINT is received, the process would be in sleep loop, on termination
#       the file handle is close & the lock file removed.  it is now time for a
#       clean exit.  the SIGINT would be delivered at the time of the package
#       shutdown by the /etc/cmcluster/nnm/nnm.control file.
#   usage               :  nnm_file_lock.ovpl
#   note                :  at any cost do not remove the lock file created by
#       this process without killing the process indicated in the lock file.
#       if in case this process does not exist, the lock file could be removed.
#   creation date       :  1st Feb 2002
#   modification details  :
#   copyright           :  Hewlett-Packard
##############################################################################

# modules used are
use Fcntl;

# setup the local variables used in this script
local $lockfile="/etc/cmcluster/nnm/nnm_lock";    # lock file location
local $childpid;


##############################################################################
#   function name       :  received_signal
#   description         :  on receiving signal from the o/s, close the file
#       descriptor that was used to lock the /etc/cmcluster/nnm/nnm_lock
#       file & exit after cleanup.  the input parameter is the signal name
#       placed in the variable $received.  also record the signal received in
#       the log file /etc/cmcluster/nnm/nnm.control.log which is done
#       automatically of executed from the mc/sg environment.  the functionality
```

```
#       in this function would be to close the file descriptor used to write
#       into the nnm_lock, unlink the file & then exit with value 2 so that it
#       would be received by nnm.control script for further processing.  this
#       function is the exit point for the script.  if there is an error while
#       closing or unlinking the file then the exit value would be set to 2 &
#       acted upon by nnm.control script for error handling.
#   called from          :  main
#   calling function     :  nil
#   input                :  signal type through positional parameter
#   output               :  exit value of either 0 or 1 or 2.
#   note                 :  certain code is repeated for logic traceablity.
#   modification details :
################################################################################
sub received_signal
{
    local $exitvalue=2;  # this value => nnm_lock to be removed manually
    local ($received) = @_;         # get the signal received

    print "\nReceived the signal $received for the process "
                . "nnm_file_lock.ovpl with process id $$ \n\n";  # for ref.

# the parent failed to write the child pid into the lock file after the fork,
# remove the child process before exiting
# this is an easy way of implementing the case structure

# this is reached if the parent could not get the lock on the lockfile
    if ( $received eq "OpenFail" )
    {
        $exitvalue = 1;
        print "Could not get the lockfile, $lockfile file open failed : $!\n";
    };

# this is reached if the parent process fails on a writing the pid of the
# child process into nnm_lock, so kill the child with SIGINT
    if ( $received eq "WriteFail" )
    {
        $exitvalue = 2;
        kill 2, $childpid;
        print "Killed child lockfile pid $childpid by the parent pid $$\n";
        close( file_hndl ) or $exitvalue = 2;
        unlink $lockfile or $exitvalue = 2;
    };

# this is reached if the parent process failed to fork the child lockfile
# process
    if ( $received eq "ForkFail" )
    {
        $exitvalue = 3;
        print "Fork failed by parent lockfile process with pid $$ : $!\n";
        close(file_hndl) or $exitvalue = 3;
        unlink $lockfile or $exitvalue = 3;
    };

# received a SIGINT for the child process to perform a normal exit, this value
# would not be caught by the nnm.control script, but the message would be
# logged in /etc/cmcluster/nnm/nnm.control.log file
```

```
    if ( $received eq "INT" )
    {
        $exitvalue = 0;
        print "SIGINT received for $$ \n";
        close( file_hndl )or $exitvalue = 4;
        unlink $lockfile or $exitvalue = 4;
        if ( $exitvalue == 4 )
        {
            print "\nChild lockfile process could not either close "
                  . "or unlink the $lockfile, please remove the $lockfile "
                  . "after analysis or the reason for failure";
        }
        else
        {
            print "\nNormal exit of child lockfile process with $$ pid, "
                  . "unlinked $lockfile";
        }
    };

# normal exit from the script & would be used by the parent process
    if ( $received eq "NormalExit" )
    {
       $exitvalue = 0;
       print "Normal exit from nnm_file_lock.ovpl\n";
    };

# do an exit as per the setting
    print "doing an exit with value $exitvalue\n";
    exit($exitvalue);
}
# end of function received_signal


##############################################
#          * main logic *                    #
##############################################

# create the lock file in the /etc/cmcluster/nnm/. directory
# open the file that contains the child pid, on failure of sysopening would be
# handled by the function received_signal
sysopen( file_hndl, $lockfile, O_CREAT|O_TRUNC|O_WRONLY|O_EXCL )
              or received_signal(OpenFail);
print "nnm_lock file created by parent process $$ \n";

# ignore the interrupt
$SIG{INT} = 'IGNORE';

# fork a child
$pid = fork;

if ( $pid == 0 )
{
    # child process - reset the signal handler function to received_signal
    $SIG{INT} = 'received_signal';

    # get the childpid
```

```
    $childpid = $$;

    # sleep & then wakeup on a signal to close the file
    print "\nChild going to sleep with pid $$\n";
    sleep ;  # infinite sleep

    # close the file handler that was opened for the file directory
    # /etc/cmcluster/nnm/nnm_lock if it gets past the sleep loop just in case.
    # THE CONTROL SHOULD NOT REACH THIS CODE
    received_signal(INT);
}
else
{
    # parent process - check for fork error, error handled by the function
    # received_signal
    if ( $pid == -1 )
    {
        # detected a fork failure
        print "detected a fork failure in the parent process $$: $!\n";
        received_signal(ForkFail);
    }

    # store the pid of the child process in the lock file so that it could be
    # picked up for terminating this scirpt with SIGINT.  also put the process
    # id in the /etc/cmcluster/nnm/nnm.control.log file for ref.  failure of
    # write would be handled by the function received_signal
    syswrite file_hndl, "$pid \n", 512, 0 or received_signal(WriteFail);
    print "\nProcess id = $pid for nnm_file_lock.ovpl ";   # for ref

    # exit the parent process
    received_signal(NormalExit);          # normal exit
}

# THE CONTROL SHOULD NOT REACH THIS CODE
received_signal(INT);          # normal exit
```