EMC²
where information lives®

Integrating Linux and Linux based Storage Management Software with RAID System-based Replication

EMC Proven Professional Knowledge Sharing 2009

Diedrich Ehlerding,
Fujitsu Technology Solutions
diedrich.ehlerding@ts.fujitsu.com

EMC Proven Professional
EMCSA Symmetrix Business Continuity Specialist
EMCIE CLARiiON Solution Specialist

EMC² PROVEN PROFESSIONAL

Contents

Abstract

Linux is more frequently being used as an operating system for enterprise class applications with large amounts of data and high availability requirements. All major database and ERP software vendors release their products on Linux. As with other operating systems, the demands for shorter backup windows, faster restore processes, and faster system copy processes makes RAID array functionality necessary.

The legacy device names traditionally used in Linux without any storage management software are inappropriate for enterprise class configurations. The problem is that these name spaces are not persistent over server reboots. They cannot guarantee that the system will find its data at the same device node where it found it before the reboot.

This article discusses various naming spaces within Linux, IO multipathing software, volume management layers, and file system layers. All these layers create their own naming spaces. We must look at these names to select the proper naming scheme in a configuration with RAID system based replication. Some naming schemes depend on hardware properties of the storage environment, implying that an array-based replica will have a different name; other naming spaces reside on the disk and are replicated with the data. Some naming spaces are automatically unique and are created by hardware or software, others may be influenced by the use. Some naming spaces are unique and persistent within a server, but may be totally different on a second server that can see the same data.

This article reviews naming schemes with respect to persistence and replication issues. In detail, it reviews the software layers:

- Linux native layers (legacy sd names, device mapper names)
- multipathing drivers: EMC PowerPath® names and Linux native multipath names
- LVM as an example of a volume manager
- file system issues (labelled file systems, file system uuids)

and replication and shared storage usage scenarios for:

- local cluster
- stretched cluster / disaster recovery
- off-host backup
- system copy

This article explains why traditional device node names are inappropriate in standard configurations for enterprise class applications due to non-persistence. It will present some typical usage scenarios for RIAD system-based or SAN-based functionality, namely clustering, disaster recovery, off-host backup, and application system copies. I will explain which objectives should be met in these environments with respect to device names. The third part of the article describes several methods to obtain persistent names, and it discusses which methods are appropriate for which usage scenario. The final section offers an example of how these layers interact.

Linux' Legacy Device Node Names are not Persistent

Usually, we mount file systems on Linux servers via fstab entries. An fstab entry links a device name to a directory name where the file system on the device name will be mounted. Traditionally, the device name is the name of a SCSI disk or a partition on a SCSI disk - something like /dev/sda or /dev/sdac1, i.e. sd driver instances. Although newer distributions (e. g. Novell's SLES 10 SP1 ff.) propose using other device names (so-called udev names), many Linux administrators still prefer these /dev/sdX device nodes.

However, these legacy names can change. There is no guarantee that a disk which your server recognized as /dev/sdc during one session of Linux will have the same device name in the next session. The first SCSI disk detected during the boot process or afterwards will be /dev/sda, the second will be /dev/sdb etc. When the system is rebooted after the device configuration has changed, it scans the attached disks and may find more or fewer disks than it had before, and it may detect them in a different sequence. There is no persistent memory to keep the relation of disk device and device name over a reboot. Moreover, if several servers of a cluster can access the same disks, there is no guarantee that all cluster nodes will refer to the same disk with the same device node name.

Here are some scenarios that will cause device node confusion, even in standard configurations without any RAID system functionality:

- If you have more than one RAID array connected to your system, and you have one logical disk in array one and another in RAID array two, you will see these disk as (e.g.) /dev/sdb (the first array) and /dev/sdc (the second array). You add a second logical disk in array one to increase capacity. Depending on your kernel version, you may make this disk visible without reboot, and your system will call it /dev/sdd. You may put a partition onto it, you may create a filesystem on this partition (/dev/sdd1), and you can mount that filesystem. But after the next reboot, your server will name the LUN in your first array /dev/sdb and /dev/sdc, and will call the LUN in the second

array /dev/sdd! Not only your new fstab entry for the new file system on (formerly) /dev/sdd1 is incorrect, but also the fstab entry for the file system that formerly resided on /dev/sdc.

- Or consider several RAID arrays in a large SAN. Linux will scan the devices in the order that the fibre channel switch presents them to the server. Brocade switches present local devices first and then devices at remote switches in the fabric. The local devices are presented in the order of port numbers, remote devices in the order of domain IDs and remote port IDs. Now imagine that you encounter an SFP failure and plug on the second RAID system from port 15 to port 0 - it will be scanned as the first system after a reboot.

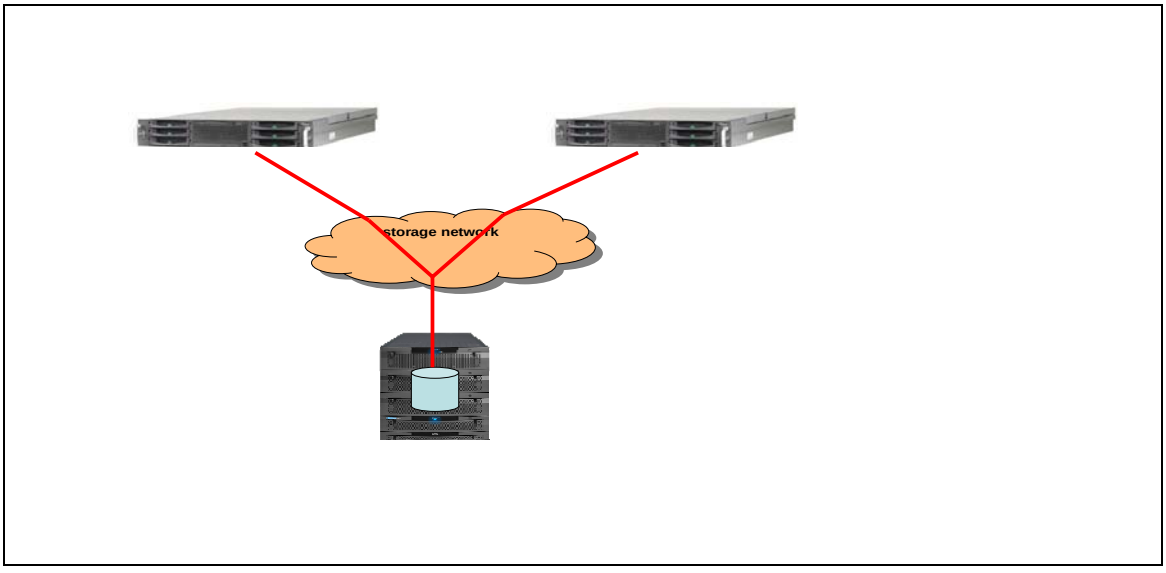
The problems with legacy device names are even more complicated when RAID system functionality is used (simple disk sharing as well as local or remote replication). We must find methods to properly identify the replica.

Sample Usage Scenarios for Shared Storage and RAID System-based Replication

In this section, I will describe sample usage scenarios and sample configurations, and will discuss the demands for device identification associated with each.

Local Cluster

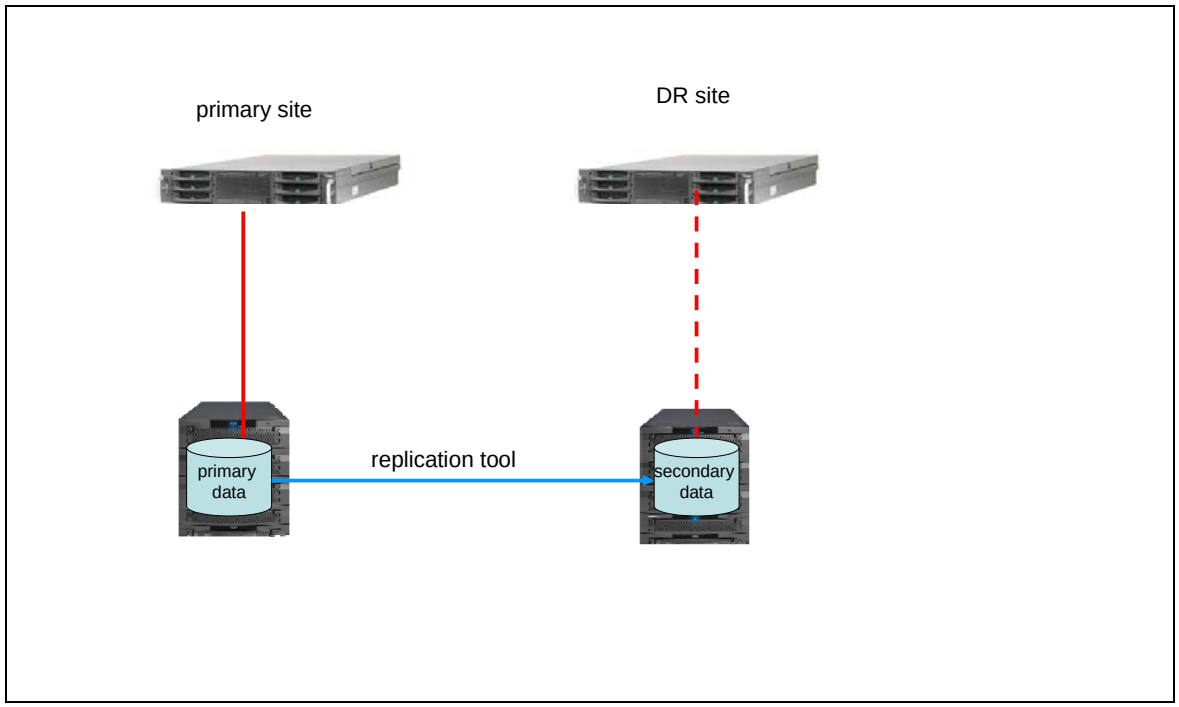
A local cluster is a set of servers sharing the same disks. The servers are connected to an external storage array over some kind of storage network, usually fibre channel or iSCSI networks. If one server fails, a surviving server will (automatically or manually) take over the failing server's applications.



The main objective of a local cluster is that all servers carrying a certain application will give the *same* name to the *same* device - otherwise they will not be able to take over. We need a method to ensure that all servers can mount the same device on the same mountpoint.

Stretched Cluster / Disaster Recovery

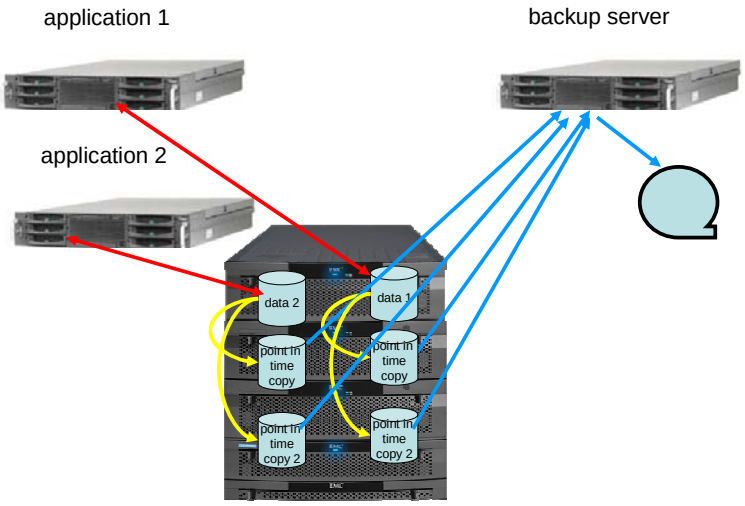
A stretched cluster is a configuration that involves a RAID system or SAN-based mirroring, usually for disaster recovery purposes, e.g. by synchronous or asynchronous SRDF® or MirrorView®, or RecoverPoint/CRR. One cluster node runs the application on the primary storage array. The RAID array or SAN functionality copies the data into a second array, the secondary mirror. The secondary mirror is usually a few kilometers away (for synchronous copies), or very far away for asynchronous replication. The secondary array may be a completely different type depending on the replication tool (e.g. with EMC RecoverPoint as remote replication tool). The secondary mirror is activated if a disaster occurs, made visible to another host, and the application will run on the DR site.



As opposed to the local cluster scenario, the objective is to give the failover cluster node access to *different* disks, not to the same disks that the primary cluster node is attached to. The standby server must use the *same* name for a *different* disk. The disks that the failover node can see will have a different identity, they may even be connected over a different type of network. A second objective is to reconfigure the standby server before application takeover since it remembers certain properties of the disks, such as read only state.

Off-host Backup

Off-host backup uses RAID-system based point in time copies (e.g. BCVs, snapshots, or clones in EMC array terminology). An application host runs an application. The RAID system creates a point in time copy at a convenient point in time (when the application is in a consistent state, e.g. a database is put into backup mode). Another server, the backup server, can save the point in time copy to tape while the server continues the application. This minimizes backup windows. This backup server also needs access to the original disks for restore purposes.



A backup server may serve several applications. The objective, with respect to device names, is to properly identify the replica. It may happen that the backup server can see replicas of several application systems, it must know which application resides on which disk. Depending on the customer solution, there may be different point in time copies of the same application data. The backup server will restore the data to its original place if a restore is necessary.

This scenario must solve the problems of a local cluster (identify the application's original devices on a second server) as well as the problems of a stretched cluster (properly identify a replica of the original devices). Moreover, the backup server will have to distinguish the point in time copy from the original data in order to restore to the desired location. It will be subject to some kind of device reconfiguration whenever a replica is presented to this host or removed from this host.

System Copy

ERP applications such as SAP frequently create a complete copy of the application for evaluation and quality assurance. While traditional approaches restore a backup, RAID system-based replicas are also frequently used for this purpose. Some customers desire to have this application system copy running within the same cluster - i.e. they use one cluster node for the production application, the standby cluster nodes runs the quality assurance system as long as the production system is alive, and only in the case of a production system outage, the standby cluster nodes will discontinue the quality assurance system and take over the production system.

This scenario, creating the quality assurance system by RAID-system functionality and mounting the copy within the same server or cluster, requires additional effort of comparison to cluster configurations that do not simultaneously access several (point in time) copies of the same data.

IO Reconfiguration

IO reconfiguration dynamically adds and removes devices on a running system. It is necessary for all usage scenarios that dynamically create and remove replicas - i.e for the scenarios “off-host backup” and “system copy” described above, and also for the stretched cluster scenario.

The reasons to reconfigure a Linux system, and the methods described to reconfigure and deconfigure devices into and from a Linux system without reboot, are closely related and very similar to the methods described in [Ehl2008] for the same task in a Solaris environment. Reconfiguration is necessary whenever a replica will be presented to a host for the first time, or whenever it is to be removed from that host. Knowing how to reconfigure a Linux system allows us to make use of RAID-system functionality for higher level processes.

IO reconfiguration is also necessary in a stretched cluster configuration (e.g. SRDF), since Linux (or at least some distribution kernels, my tests were performed with SLES10 SP2) remembers certain properties of disks detected at boot time. For example, it remembers that a disk was read-only when it was detected (e.g. an SRDF R2 disk). Even if the RAID array swaps SRDF personality, Linux still believes that this device is read-only, and refuses to mount it read-write. Reconfiguration is necessary for Linux to rediscover these properties.

All major distributions today are based on kernel 2.6 and have sysfs and devfs support. In this environment, it is fairly easy to cause a SCSI adapter to rescan its environment (iSCSI and fibre channel controllers are SCSI controllers, too, from Linux' point of view). The rescan procedure is fairly simple; every host bus adapter is listed in /sys/class/SCSI_host. Although every driver has its own type of entries in this sysfs subtree, the drivers for most widely used fibre channel HBAs (Emulex of QLogic controllers) are sysfs aware, as well as Linux' iSCSI driver. Every sysfs aware driver provides a pseudo file "scan" in its sysfs subtree. Writing into this pseudo file causes the controller to rescan its environment.

Here is a simple procedure to rescan all fibre channel HBAs:

```
cd /sys/class/fc_host
HBAS=`ls`
cd /sys/class/SCSI_host
for i in $HBAS/scan
do
       echo "- - -" > $i
done
```

where "- - -" means "rescan all channels, targets, and LUNs."(Precautions may be necessary with certain SCSI controller drivers and/or early 2.6 kernels, e.g. QLogic HBAs before kernel 2.6.12. I skip these precautions here; if you have such older kernels, refer e.g. to [QLA2007])

Having rescanned the disk layer, rescan multipathing layers if you are using a multipathing driver, and rescan volume manager layers if necessary.

Conversely, removing a device node - say: /dev/sdd - from Linux is also achieved by writing into sysfs.

For example:

```
echo 1 >/sys/block/sdd/device/delete.
```

Perform IO Reconfiguration and deconfiguration properly. Shut down your application working on the affected devices and unmount all file systems that you are going to deconfigure. Deactivate all affected volume groups if you are using a volume manager, then deconfigure any multipathing layer before you remove a disk from Linux' kernel. Be aware that Linux will immediately remove your device node; it does not check if this device is still open. Removing the wrong device may lead to problems in higher layers.

Persistent Device Names

In this section, I will discuss methods to obtain a persistent name space for you data to replace legacy device names. Several methods are available. Consider whether the disk is identified by some unique *hardware ID*, or if the disk is identified on the basis of its *content*, i.e. on a unique ID or magic number, or a user defined name created by software that is written to a disk label.

All these hardware or software based IDs are frequently called “uid”, “uuid” or “wwid”, i.e. the same notions are in use for hardware and software generated magic numbers. I will use the term “uuid” for all these ids.

Hardware-based Names and uuids

This section describes the hardware based IDs of EMC arrays (Symmetrix/DMX® and CLARiiON®). Every LUN of an array has its own unique ID to identify the disk. The device uuid can be read by the host with SCSI inquiry commands, usually on code page ox83, or by the administrator with Linux’ utility “SCSI_id”

Example: get uuid of /dev/sde

```
db82:~ # /sbin/SCSI_id -g -u -ppre-spc3-83 -s /block/sde
360060480000290104212533030303233
db82:~ #
```

(Linux prepends a “3” to the Symmetrix’ or CLARiiON’s device uuids)

As described below, these uuids are used by certain drivers, either directly as a device name or as an index in some kind of memory (a configuration file) to correlate device and device node in a persistent manner.

Symmetrix generated unique names

The device uuid of a Symmetrix device can be obtained from Solutions Enabler. Example:

```
nsr1:~ # symdev -sid 08 show 02F| grep WWN
    Device WWN               : 60060480002901041085330303032 46
nsr1:~ #
```

Looking at this uuid, it contains the Symmetrix’ serial number and the symdev (or symlabel in older Enginuity versions, e.g. on DMX1000 arrays). We are considering device 02F; this device has a symlabel “S0002F”; and 533030303246 is “S0002F” in ascii encoding. It is therefore easy to correlate these device uuids with your Symmetrix’ layout. We can be sure that an SRDF R2 device has another uuid than its R1 partner device - it resides in another Symmetrix.

As for point in time copies, keep in mind that a snapshot device (VDEV) is predefined within the Symmetrix' configuration and has its own predictable uuid, as well as clones or BCVs. If you have several point in time copies of one original Symmetrix device, you will see two different uuids. If you use the same VDEV for one application today and for another application tomorrow, a host will detect the same uuid in a SCSI inquiry.

If you reconfigure your Symmetrix by symconfigure, EMC ControlCenter®, or Symmetrix management console, and thus create new devices within the Symmetrix, you may delete one symdev and create another with the same symdev ID. This device will get the same device uuid that previously identified the deleted device.

CLARiiON Generated Unique Names

A CLARiiON LUN has a unique id that is visible in the Navisphere® GUI via GUI or CLI. Example:

```
hvr0007a:~ # navicli -h 172.30.4.29 getLUN 0 | grep UID
UID:                         60:06:01:60:C6:45:14:00:14:10:21:36:9D:49:DD:11
hvr0007a:~ #
```

This id is created when the LUN is bound. Alas, there is no easy way to decode this uuid, either with respect to the identification of the array or with respect to identifying the LUN within the array. This uuid will change If the LUN is unbound and recreated; it will change even if the LUN is unbound and recreated with the same properties (same RAID group, same size, same array LUN). Unlike the Symmetrix, there is no easy way to predict the device uuid of a CLARiiON LUN; a CLARiiON device uuid will never be reused.

As for point in time copies, a clone device in a CLARiiON is a device with its own uuid. Unlike the Symmetrix, there are no predefined snapshot devices. A snapshot's uuid is a property of the *snapshot* - not a property of the snapshot *session*. It can be obtained by navicli; example:

```
hvr0007a:~ # naviseccli -h 172.30.4.29 snapview -listsnapshots

SnapView logical unit name:  Snap_LUN101
SnapView logical unit ID:  60:06:01:60:C6:45:14:00:34:84:54:3A:14:DE:DD:11
Target Logical Unit:  101
State:  Active
Session Name:  Snap1_LUN101
```

This uuid remains as long as the snapshot is not deleted. You may start different sessions within this same snapshot and present them to a host; it will always detect the same uuid for these different sessions. As soon as the snapshot is deleted, the uuid is gone forever. If the snapshot is recreated, it may have the same name and the same properties as before but it will have a new, different uuid. If you need persistent hardware ids, you should not delete snapshots and recreate them; instead, start and stop snapshot sessions within the snapshot.

(However, be aware that some CLARiiON firmware update (NDU) procedures demand to terminate all layered applications. In this case, the hardware generated uuids for all snapshots will change!)

Hardware-based udev Name Spaces

This is a very short overview. A more detailed description of device mapper names may be found in [But2007] .

The udev subsystem introduced with kernel 2.6. udev can create device names depending on rule sets that are evaluated during device discovery. These rule sets may differ between distributions, but the major distributions (RedHat, SuSE) create rather similar device names.

You will find interesting directories within /dev/disk for disk devices. The sub-directories usually contain symlinks to the legacy device name that this device obtained in the Linux session, thus correlating a persistent name to an sd instance.

/dev/disk/by-path contains a symlink for every disk path based on PCI slot, HBA, and (for fibre channel disks) the WWN and LUN of the device. Example (two HBAs, one device is visible over two FA ports; some lines concerning internal disks are omitted). The PCI slot of the HBA is highlighted in red, the Symmetrix' WWN is green, and the LUN address is blue (i.e. this host can see a LUN 0 (the VCM device), and LUNs 0x10 and 0x12).

```
db82:~ # ls -l /dev/disk/by-path
total 0
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:03:00.0-fc-
0x5006048c52a8e507:0x0000000000000000 -> ../../sda
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:03:00.0-fc-
0x5006048c52a8e507:0x0010000000000000 -> ../../sdb
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:03:00.0-fc-
0x5006048c52a8e507:0x0012000000000000 -> ../../sdc
[…]
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:0a:00.0-fc-
0x5006048c52a8e508:0x0000000000000000 -> ../../sdd
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:0a:00.0-fc-
0x5006048c52a8e508:0x0010000000000000 -> ../../sde
lrwxrwxrwx 1 root root  9 Jan 16 13:11 pci-0000:0a:00.0-fc-
0x5006048c52a8e508:0x0012000000000000 -> ../../sdf
db82:~ #
```

Although this name space is persistent within one server (as long as the hardware configuration remains the same, HBA slots, front-end mapping within the Symmetrix, or ALU/HLU relation in a CLARiiON), it is fairly complicated to read. It can be useful to detect the hardware configuration and to check SAN zoning and mapping.

The **/dev/disk/by-id** directory contains the names based on hardware uuids mentioned above. This is a good method to create a persistent name space for single pathed servers, e.g. local clusters accessing the same disks on a Symmetrix**.** Be aware that a disk reports the same uuid over all channels where it is visible. Everything is fine if your server is attached over one single path to your array. If you have several paths to your array, /dev/disk/by-id will contain only one symlink pointing to one of these paths. Same example as above: sdc and sdf refer to the same device, but only one of them is mentioned in /dev/disk/by-id:

```
db82:~ # ls-l /dev/disk/by-id| grep sdc
lrwxrwxrwx 1 root root  9 Jan 16 13:11 SCSI-360060480000290104212533030303242 ->
../../sdc
db82:~ # ls-l /dev/disk/by-id| grep sdf
db82:~ #
```

Since you will attach a single HBA host to a CLARiiON via at least two paths (one to SPA and another one to SPB), you cannot predict if /dev/disk/by_id will refer to the current owner SP of your CLARiiON or not. You will have some kind of multipathing software with a CLARiiON.

/dev/disk may contain more subdirectories that do not refer to hardware properties but to disk contents; see below for file system labels and uuids.

Multipathing Software

Multipathing drivers also create their own name spaces, which provide persistent names. PowerPath and Linux native multipath are the most widely used multipathing drivers in Linux servers attached to EMC arrays.

EMC PowerPath

PowerPath is the standard IO multipathing driver for EMC arrays and can be used for other arrays. PowerPath is a filtering driver, i.e. it intercepts IOs to the legacy devices which represent the same LUN and redirects IOs if necessary. Therefore, in the CLARiiON example above, a symlink in /dev/disk/by-id can be used as a persistent device name in a PowerPath environment. PowerPath will redirect this IO to the proper path if it refers to the inactive SP.

PowerPath creates its own name space in addition to filtering IOs to the legacy device nodes. The first disk detected by PowerPath is /dev/emcpowera, the second one /dev/emcpowerb etc. These names are persistent over reboots, and they are also persistent if you add or remove paths to your arrays. PowerPath inquires the device uuid and maintains a configuration file on the server's root file system to properly correlate device uuid and

emcpower name. So, emcpower names are a good choice to obtain a stable and persistent name space on one server.

However, the relation of PowerPath device names to devices is not necessarily identical on several cluster nodes accessing the same device. A Powerpath utility (emcpadm) allows maintenance of this mapping. It is possible to change the device names within one server; e.g. to establish some kind of naming scheme - call LUNs in your first array emcpoweraa, emcpowerab, etc, and call LUNs in another array emcpowerba, emcpowerbb, etc, to easily identify your arrays. emcpadm also allows you to export the mapping created on one host and import it on another host. These options may be useful for local and stretched clusters. In a local cluster environment, select one of the nodes, export its configuration, and import it on the second cluster node providing the same emcpower device names on both nodes. In a stretched cluster environment, identify the remote mirror relations and rename the standby server's devices in a way that the standby server will call the remote mirror of emcpowera with the same name.

As for reconfiguration, you may instruct PowerPath to rescan its configuration by a utility (powermt config). It may also be instructed to forget a uuid-to-emcpower relation by powermt remove, but then it will be unnecessary to get the same name if it becomes visible again.

For details, see cf. [EMC2008a] and [EMC2008b]

Linux Native Multipath

Linux native multipath, or "device mapper multipath" or "dm_multipath", appeared some time after the release of kernel 2.6 for many arrays, including EMC arrays. For details, especially for installation and configuration, cf. [EMC2006]

The standard configuration of Linux native multipath uses the same device uuids as described for /dev/disk/by-id, but replaces the symlinks with symlinks to /dev/dm-X nodes. In a native multipath configuration without special configuration, /dev/disk/by-id/SCSI-3<uuid> is a name of the multipath device. These devices also appear in /dev/mapper, but there are no symlinks in /dev/mapper; instead, it contains device node for major number 253 (the device mapper multipath driver). SuSE SLES9 formerly put symlinks into /dev/disk/by-name too, and SLES10 still does that for backward compatibility, RedHat creates /dev/mpath as a container for multipath devices. I recommend using /dev/mapper nodes as they are available on both distributions and on other distributions as well.

As for reconfiguration, a multipath device is removed by multipath -f <device>, or all multipath devices which are currently not open are removed by multipath -F. Just running multipath causes the driver to rescan, usually after the HBAs are rescanned.

Device names that directly contain a device uuid are automatically unique and persistent. Even if a device node is removed for deconfiguration, it will reappear with the same name as soon as the device is visible again. The device names are identical on all nodes of a local cluster sharing the same disk configuration. If you decide to use this name space, there is no need to deploy any mapping to all cluster nodes; the device names are identical.

There is an option to create a mapping file if you find these names unreadable. Activate “user friendly names” in /etc/multipath.conf; then the devices will be called /dev/mapper/mpatha, /dev/mapper/mpathb etc. - very similar to emcpower names. As with emcpower names, these names are persistent within one server, but the mapping will be different on another server. Instead of exporting and importing the mapping with a utility, simply deploy the configuration file /var/lib/multipath/bindings. This is a simple text file, you can create your own name space by manually editing this file. Here is an example:

Activation of user friendly names and then running multipath -F && multipath yields:

```
db82:/var/lib/multipath # cat bindings
# Multipath bindings, Version : 1.0
# NOTE: this file is automatically maintained by the multipath program.
# You should not need to edit this file in normal circumstances.
#
# Format:
# alias wwid
#
mpatha 360060480000290104212533030303242
mpathb 360060480000290104212533030303233
```

Edit the file to contain names of your choice:

```
db82:/var/lib/multipath # cat bindings
# Multipath bindings, Version : 1.0
# NOTE: this file is automatically maintained by the multipath program.
# You should not need to edit this file in normal circumstances.
#
# Format:
# alias wwid
#
LUN18  360060480000290104212533030303242
LUN16  360060480000290104212533030303233
```

After multipath -F && multipath; multipath -ll displays:

```
db82:/var/lib/multipath # multipath -ll
LUN18 (360060480000290104212533030303242) dm-0 EMC,SYMMETRIX
[size=61G][features=0][hwhandler=0]
\_ round-robin 0 [prio=2][active]
 \_ 4:0:0:18 sdf 8:80  [active][ready]
 \_ 2:0:0:18 sdc 8:32  [active][ready]
LUN16 (360060480000290104212533030303233) dm-1 EMC,SYMMETRIX
```

```
[size=61G][features=0][hwhandler=0]
\_ round-robin 0 [prio=2][active]
 \_ 4:0:0:16 sde 8:64  [active][ready]
 \_ 2:0:0:16 sdb 8:16  [active][ready]
db82:/var/lib/multipath #
```

and the devices are called:

```
db82:/var/lib/multipath # ls -l /dev/mapper
total 0
brw------- 1 root root 253, 1 Jan 17 11:42 LUN16
brw------- 1 root root 253, 0 Jan 17 11:42 LUN18
```

For stretched cluster configurations, maintain one mapping file on the production node and another on the DR site node to get identical device nodes on both servers.

You should no longer use /dev/disk/by-id/SCSI-3<uuid> as names for your multipath disks if "user friendly names" will be activated. In this case, Linux native multipath will only create symlinks for the user defined names, but it will not create symlinks of the device uuid to device mapper nodes; these symlinks still refer to one of the legacy sd nodes. Unlike PowerPath, native multipath is not a filtering driver; it will not intercept these IOs. Using /dev/mapper as described above, or using /dev/mpath on RedHat, avoids this pitfall.

Persistent Names Based on Disk Contents

There are other methods to create persistent device nodes that rely on the *contents* of a disk instead of *hardware IDs.* These usually depend on a "magic number" in the label of the disk or file system that identifies the disk, or on a user-defined name written on the disk. Since the disk content will be replicated with all kinds of array based replication, these labels and magic numbers are replicated too; the replica will contain the same identification. Some usage scenarios become easier with disk content based names, and others become more complicated.

Volume Manager Names (LVM)

This is a very short description of LVM; for details concerning LVM, cf. [Lew2006] and [Nov2008]. There are other volume managers in a Linux environment, e.g. evms, or commercial products, such as Veritas storage Foundation; this article only considers LVM.

LVM is a frequently used volume manager in Linux environments. Like other volume managers, it groups disks into groups ("disk groups" or "volume groups"), and creates logical volumes somewhere within this group. The volume group has a user defined name. A volume can be created within the group with a certain size and with other properties (striping

or not, etc.). A volume may span several disks, and several volumes may reside on a disk. The volume names are user-defined. LVM creates a device node for every volume that can then be used to create a file system etc. Example: a volume group “colours ” containing three volumes “red”, “blue”, and “green” will be accessible via /dev/colours/red, /dev/colours/blue, and /dev/colours/green.

A volume group can be active or inactive. Activating a volume group causes Linux to scan its disks to look for a set of disks containing this volume group. It then creates the device nodes for the volumes. Deactivating a group removes these nodes. Since the application will use the volume manager device nodes instead of disk device nodes, non-persistent disk names are no longer harmful in a volume manager configuration. In a local cluster environment, and in the simplest setup, a volume group will be active on one cluster node and inactive on the other nodes. (Hint: SuSE’s and RedHat’s LVM versions slightly differ with respect to cluster awareness, and to activation and deactivation of volume groups at boot time. Be particularly careful with SuSE SLES servers and PowerPath and LVM. Some versions of PowerPath, 3.4 through 5.1, do not properly observe SuSE’s /etc/sysconfig/lvm file; current versions may no longer have this bug).

The information concerning the structure of a volume group, including user-defined names, resides entirely on the disk. If two servers have access to the same volume group, they will create the same device nodes on activation of this group, irrespective of the hardware ID or the disks legacy device name. A server that will get access to a replica will immediately be able to activate the replica volume group, and will create exactly the same device nodes for the replica as the production server used for the original data. The “local cluster” and “stretched cluster” scenarios are extremely easy using LVM.

LVM correlates the disks belonging to one volume group via “magic numbers” (also called uuids, but do not denote hardware generated IDs here, these are generated by software upon creation of the group or the volume). Such a uuid will be created for the volume group, for every physical volume (the disk), and for every logical volume, at creation. These uuids are also stored in the LVM label on all the disks. LVM may become somewhat confused if it detects duplicate volume groups and/or duplicate physical volume IDs. The scenario “off-host backup” should therefore care for proper reconfiguration, and should not present several point in time copies of the same application to the backup host simultaneously. The scenario “system copy” will be discussed below in deep detail; see page 21 ff.

For reconfiguration, use pvscan and vgscan to detect disks that became visible.

Linux Software RAID (md) Names

The Linux software RAID driver (md; “metadisk”) has been available for a long time. It can concatenate disks or disk partitions, create software RAID, and perform software striping. Customers who don’t have RAID system based tools sometimes use Software RAID (mirroring). The device nodes are /dev/md0, /dev/md1 etc.

The disks belonging to an md array are identified via a magic number residing on the disk. The mapping of this uuid to a metadisk instance can be made persistent in /etc/mdadm.conf: After creating a metadisk array, read the uuid by

```
hvr0007a:~ # mdadm --examine --scan /dev/md0
ARRAY /dev/md0 level=raid1 num-devices=2 UUID=900175ae:b02eeb34:e21ad84e:2be65fb1
hvr0007a:~ #
```

and add the result (the red line) to /etc/mdadm.conf. It will always call this array “/dev/md0” even if this host can see several md arrays. /etc/mdadm.conf is a text file and can be deployed to all nodes of a cluster. In cluster environments, make sure that only one cluster node assembles an md array, otherwise data will be lost.

Theoretically, it is also possible to deploy mdadm.conf to an off-host backup server, and the backup server may use another md instance (i.e. you may edit the /dev/md0 entry if you want your backup server to see this disk as /dev/md1 instead of /dev/md0). But using md mirroring with array based replication is difficult. The point in time copy will be created in one array, but software mirrors usually reside in two arrays. The backup host will believe that the mirror is broken; the backup solution must be aware of this situation. As in LVM environments, make sure that a server will never see more than one point in time copy of the same md array.

File System Labels and File System uuids

A file system may be “labelled”, i.e. this file system has a user defined name in its metadata, and may be mounted without mentioning the device where the file system resides. A file system has also uuid (a magic number) that resides in the file system’s metadata on the disk. It is also possible to mount a file system using its uuid.

File system uuids are created at mkfs time. Labels can be created at the same time or may be modified afterwards; uuids may also be modified afterwards. See the man pages of mkfs, tune2fs, xfs_admin, mount etc.

Linux enumerates the file systems that it can see in /dev/disk/by-label, and also enumerates the file system uuids that it can see in /dev/disk/by-uuid. These directories contain symlinks. Caution: file system labels and/or file system uuids do not appear automatically after mkfs and/or after reconfiguration; you will have to trigger udevd (refer to “man udevtrigger”) to populate these directories after reconfiguration.

Check in advance that they properly interact with your multipathing driver if you want to use these name spaces. You will not have a problem with PowerPath because PowerPath intercepts these IOs if /dev/disk/by-label contains a symlink to a legacy device. If you use native multipath, verify that your distribution populates these directories with symlinks to /dev/dm-X nodes instead of symlinks to legacy nodes.

As with all other software generated name spaces, these labels and uuids reside on the disk and are replicated by array based mechanisms. This makes it easy to identify the disk on another host, but will cause problems with duplicate IDs, i.e. in the system copy scenario.

Selecting the Proper Naming Scheme

General Considerations

The advantage of hardware based IDs is that they are automatically unique; there is no danger of duplicates. The disadvantage of hardware based IDs is that they are unique; they are not easily usable for all kinds of replication because the replica will have a different name. In other words, hardware based IDs, especially udev names and the native dm_multipath names, are excellent for creating local clusters but are a bad choice for replication (or at least require some configuration efforts).

As soon as you must consider replication, hardware based IDs will need some kind of mapping from hardware names to proper device node names. This mapping must be properly defined and maintained on all affected servers. Methods include maintaining emcpower names or maintaining dm_multipath names. Keep in mind that the production host will need different mappings than the host that can see the replica.

A very interesting method to deal with unique hardware based IDs in a replication context is contained in a script collection “emcadm”. This is a “global tool” from EMC, developed for automation of replication processes on Symmetrix and CLARiiON arrays. The script solution contains functions to create systematically constructed dm_multipath names that make it fairly easy to correlate the replica’s device node name to the corresponding original device

node, and to deploy this name space over several hosts. For details, contact the author; see [Wie2008]

Conversely, the main advantage of software generated IDs contained on the disk is the fact that this ID will be replicated with the data. A host addressing the replica will see the same software ID as the host that processes the production data. It will therefore be able to use the same device nodes, the same fstab entries etc. Again, however, the main advantage is also the main disadvantage. If the replica is made visible to the same host, or if a host happens to see several replicas of the same original data, then this host may see the same software ID twice.

Scenario "Local Cluster"

This scenario can be set up with all the methods mentioned above. You can safely choose unique hardware ids as the base of your persistent name space, or you can rely on disk content based software IDs.

If you choose hardware based IDs, native multipath will automatically create identical (but ugly) device nodes. Edit the mapping to your desired names and don't forget to deploy the mapping file to all servers in the cluster if you want them to look less ugly. You should export the device-to-emcpower mapping on one node and import it on all other cluster nodes using emcpower names. Repeat these processes to keep the mapping in sync whenever you change your configuration, especially if you add LUNs to the configuration.

Persistent names based on software IDs are even simpler. LVM device nodes automatically have the same name on all cluster nodes. If you want to use md mirrors, maintain and deploy mdadm.conf. File system labels and file system uuids are automatically identical on all nodes of a local cluster.

Scenario "Stretched Cluster"

Using hardware based IDs requires effort since stretched cluster nodes will see different disks. It is neither possible to export emcpower mappings and import them on the DR site, nor will native multipath and/or udev create the same device names automatically, nor can the dm_multipath mapping table be deployed. In a more or less static configuration, it should be possible to create a PowerPath or multipath mapping suitable for the DR host. If you want hardware ID based names in a stretched cluster, I repeat my hint concerning emcadm.

This scenario can be managed much more easily with software based names, preferably a volume manager like LVM, but also with labelled file systems or file system uuids. Since these names reside on the disks, they will be identical on all affected servers, irrespective of the disk device node on which they reside.

Scenario "Off-host Backup"

The same considerations apply as in the stretched cluster scenario. The backup host will see other devices than the production hosts, therefore hardware based IDs are not a good choice. Maintaining PowerPath device nodes may become difficult (it is not possible to prepare a device-to-emcpower mapping before a server was attached to a snapshot or VDEV).

As in the stretched cluster scenario, a software ID based naming scheme will make the process much easier. However, all software id based processes require proper reconfiguration of the backup host to avoid trouble with duplicate ids.

Scenario "System Copy"

This is the most complicated scenario. Hardware based IDs are difficult to use for the same reasons as in the off-host backup scenario. You must consider duplicate IDs visible to the same server if software IDs or user generated names are used. Linux layers may become seriously confused if they see the same metadata on two different disks. Therefore, it is necessary to rename all these layers properly to prevent trouble (see next section).

Renaming Procedures for Replicas

I will now give a complete example of how to rename the replica of an LVM volume group so it may be activated simultaneously on the same host as the original group, and then give some hints with respect to file system labels and file system uuids.

Renaming Volume Manager Groups

Here is a complete example explaining how the device reconfiguration, the multipathing layer and its method to map device IDs to readable names, CLARiiON snapshots, and LVM can interact to mount a RAID system replica (here: a CLARiiON snapshot) of a volume group on the same host. This is an LVM configuration on top of native multipath. Create a CLARiiON snapshot of the LUNs belonging to that volume group. The snapshot will then be made visible to the same host on which the volume group is active, and then the replica volume group will be renamed.

The procedure described here to rename a volume group was inspired by a similar task in a Solaris environment with Veritas Volume manager; cf. [Ver2002]. In fact, you must perform the same steps:

- dump the original metadata into a textfile
- destroy the old metadata on the replica disks, but preserve the user data
- invent new metadata IDs for all objects (here: volume group, physical volumes, and logical volumes)
- edit the metadata dump appropriately; insert the invented metadata IDs
- re-initialize the disks with the metadata IDs you invented
- restore the modified metadata dump in order to create the volumes on the replica

The screenshots below refer to a CLARiiON CX500 array attached to a SLES10 SP2 Linux host. The host can see the CLARiiON via two fibre channel controllers; the host uses native multipath. At the beginning, the host can see two LUNs in the CX500 (array LUN 101 and 201). A snapshot is created for the LUNs, but no session is active. A volume group is defined on these two LUNs; two volumes within this group are active (one of them is striped over the two LUNs).

Here are hardware IDs of the affected LUNs and snapshots:

```
hvr0007a:~ # navicli getLUN 101|  grep UID
UID:                          60:06:01:60:C6:45:14:00:DA:BD:C7:61:9D:49:DD:11
hvr0007a:~ # navicli getLUN 201|  grep UID
UID:                          60:06:01:60:C6:45:14:00:04:16:0A:C0:9D:49:DD:11
hvr0007a:~ # navicli snapview -listsnapshots|grep unit
SnapView logical unit name:  Snap_LUN101
SnapView logical unit ID:  60:06:01:60:C6:45:14:00:34:84:54:3A:14:DE:DD:11
SnapView logical unit name:  Snap_LUN201
SnapView logical unit ID:  60:06:01:60:C6:45:14:00:35:84:54:3A:14:DE:DD:11
```

These devices are mapped to readable names:

```
hvr0007a:~ # grep data /var/lib/multipath/bindings
data1  360060160c6451400dabdc7619d49dd11
data2  360060160c645140004160ac09d49dd11
snap_data2 360060160c64514003584543a14dedd11
snap_data1 360060160c64514003484543a14dedd11
hvr0007a:~ #
```

The volume group resides on /dev/mapper/data1 and /dev/mapper/data2:

```
hvr0007a:~ # pvscan
  PV /dev/mapper/data1        VG data   lvm2 [200.00 GB / 197.00 GB free]
  PV /dev/mapper/data2        VG data   lvm2 [200.00 GB / 199.00 GB free]
  Total: 2 [399.99 GB] / in use: 2 [399.99 GB] / in no VG: 0 [0   ]
```

Now find out which device mapper instance is called /dev/data1 and /dev/data2 (here: dm-5 is the device node currently belonging to the device “data2”, and “data1” refers to dm-6):

```
hvr0007a:~ # multipath -l data1
data1 (360060160c6451400dabdc7619d49dd11) dm-6 DGC,RAID 10
 […]
hvr0007a:~ # multipath -l data2
data2 (360060160c645140004160ac09d49dd11) dm-5 DGC,VRAID
[…]
```

Two volumes are mounted:

```
hvr0007a:~ # mount | grep mapper/data
/dev/mapper/data-vol1 on /tmp/data/vol1 type ext2 (rw)
/dev/mapper/data-volstripe on /tmp/data/volstripe type ext2 (rw)
```

As a first step, save the metadata of the volume group as it contains the entire description of which logical volume is located where, and on which disk. In this case, one logical volume is striped over both physical volumes, and one volume is only on one disk.

```
hvr0007a:~ # vgcfgbackup -f vgcfg_data data
  Volume group "data" successfully backed up.
```

This metadata dump file contains the entire description of the volume group, both the physical and logical volumes. Every physical volume, every logical volume, and the volume group itself have a name and a software uuid. The user defined the names of the volume group and the volumes when he created the volume group; the physical volumes have standard names “pv0” etc created by LVM, and LVM also c created the uuids.

This is the dumpfile; the user defined names and the uuids are highlighted:

```
hvr0007a:~ # cat vgcfg_data
# Generated by LVM: Fri Jan  9 16:26:43 2009

contents = "Text Format Volume Group"
version = 1

description = "vgcfgbackup -f vgcfg_data data"

creation_host = "hvr0007a"      # Linux hvr0007a 2.6.16.60-0.29-smp #1 SMP Thu Aug
28 09:26:55 UTC 2008 i686
creation_time = 1231514803      # Fri Jan  9 16:26:43 2009

data {
        id = "ZVVxGb-MZLT-eXqa-oNay-FFkN-DS6g-mF14ty"
        seqno = 3
        status = ["RESIZEABLE", "READ", "WRITE"]
        extent_size = 8192              # 4 Megabytes
        max_lv = 0
        max_pv = 0

        physical_volumes {

                pv0 {
                        id = "3iktbX-kL63-xV2J-oUxO-PTYN-v6Fw-C9WA3j"
                        device = "/dev/dm-6"    # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 419430400    # 200 Gigabytes
                        pe_start = 384
                        pe_count = 51199        # 199.996 Gigabytes
                }

                pv1 {
                        id = "YejBAM-NvzP-qKXg-9HKr-1oo8-2BT8-1jH18y"
                        device = "/dev/dm-5"    # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 419430400    # 200 Gigabytes
                        pe_start = 384
                        pe_count = 51199        # 199.996 Gigabytes
                }
        }

        logical_volumes {

                vol1 {
                        id = "8x7tKa-bupX-p469-D7bh-RR6o-vg0Y-bDJy1M"
                        status = ["READ", "WRITE", "VISIBLE"]
                        segment_count = 1

                        segment1 {
                                start_extent = 0
                                extent_count = 512      # 2 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

                                stripes = [
                                        "pv0", 0
                                ]
                        }
                }

                volstripe {
                        id = "HZ9pH2-WLJD-w12c-gLbk-6sBE-5GKD-4F6FCx"
                        status = ["READ", "WRITE", "VISIBLE"]
                        segment_count = 1
```

```
                        segment1 {
                                start_extent = 0
                                extent_count = 512      # 2 Gigabytes

                                type = "striped"
                                stripe_count = 2
                                stripe_size = 128       # 64 Kilobytes

                                stripes = [
                                        "pv1", 0,
                                        "pv0", 512
                                ]
                        }
                }
        }
}
```

As you see, the configuration backup only contains as a hint the device mapper nodes **(/dev/dm-5** and **/dev/dm-6)**; we need this information to set up the configuration for the replica properly. At this moment, we have to determine and to keep in mind that /dev/dm-5 is /dev/mapper/data2 and is called “pv1” within this group, and /dev/dm-6 is /dev/mapper/data1 which is called pv0 within this group (see the multipath -l listing above for the dm-X instances). It is crucial to understand the relation of the LVM internal “pvX” name, the device mapper node /dev/dm-Y, and the /dev/mapper name.

Now, you must modify this config file by changing the volume group name and inventing new LVM uuids for all objects (i.e. for the volume group, the physical volumes, and the logical volumes). Modified lines are marked red above. I created a new config file from the backup by modifying all these uuids to contain the string “SNAP.”. Here are only the relevant lines; all others are identical to the file quoted above. (There may be better methods to create these uuids; cf. “man uuidgen” if you want some kind of automation).

```
hvr0007a:~ # cat vgcfg_snap
[…]
snap {
        id = "vgSNAP-MZLT-eXqa-oNay-FFkN-DS6g-mF14ty"
        […]

        physical_volumes {

                pv0 {
                        id = "pvSNAP-kL63-xV2J-oUxO-PTYN-v6Fw-C9WA3j"
                        […]
                }

                pv1 {
                        id = "pvSNAP-NvzP-qKXg-9HKr-1oo8-2BT8-1jH18y"
[…]
                }
        }
```

```
        logical_volumes {

                vol1 {
                        id = "lvSNAP-bupX-p469-D7bh-RR6o-vg0Y-bDJy1M"
[…]
                }

                volstripe {
                        id = "lvSNAP-WLJD-w12c-gLbk-6sBE-5GKD-4F6FCx"
[…]
                }
        }
}
```

All omitted lines remain unchanged.

Now, start a snap session on both LUNs and activate it (omitted). Start the session in a crash-consistent manner, i.e. use “navicli startsession -consistent”, or use Enginuity consistency assist on a Symmetrix, if you have volumes that span over more than one disk. Then, put the snapshot into the host’s storage group. The storage group in the CLARiiON now contains the two LUNs and the two snapshots.

```
hvr0007a:~ # navicli storagegroup -list
[…]
HLU/ALU Pairs:

  HLU Number     ALU Number
  ----------     ----------
    2               101
    3               201

HLU/SLU Pairs:

 HLU No.  SNAP SHOT UID                                    SNAP SHOT NAME
 ------  -------------------------------------------------  --------------
 10      60:06:01:60:C6:45:14:00:34:84:54:3A:14:DE:DD:11      Snap_LUN101
 11      60:06:01:60:C6:45:14:00:35:84:54:3A:14:DE:DD:11      Snap_LUN201
```

Rescan HBAs, rescan multipath (omitted). Now the host can see the data and the snapshot:

```
hvr0007a:~ # multipath -l | grep data
snap_data2 (360060160c64514003584543a14dedd11) dm-0 DGC,VRAID
snap_data1 (360060160c64514003484543a14dedd11) dm-1 DGC,RAID 10
data2 (360060160c645140004160ac09d49dd11) dm-5 DGC,VRAID
data1 (360060160c6451400dabdc7619d49dd11) dm-6 DGC,RAID 10
hvr0007a:~ #
```

but LVM gets confused:

```
hvr0007a:~ # pvscan
  Found duplicate PV YejBAMNvzPqKXg9HKr1oo82BT81jH18y:
    using /dev/mapper/data2 not /dev/mapper/snap_data2
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/mapper/data1 not /dev/mapper/snap_data1
  PV /dev/mapper/data1   VG data   lvm2 [200.00 GB / 197.00 GB free]
  PV /dev/mapper/data2   VG data   lvm2 [200.00 GB / 199.00 GB free]
  Total: 2 [399.99 GB] / in use: 2 [399.99 GB] / in no VG: 0 [0   ]
```

Therefore, destroy the metadata on the replica:

```
hvr0007a:~ # pvremove -ff /dev/mapper/snap_data2
  Found duplicate PV YejBAMNvzPqKXg9HKr1oo82BT81jH18y:
    using /dev/dm-5 not /dev/dm-0
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/dm-6 not /dev/dm-1
  Found duplicate PV YejBAMNvzPqKXg9HKr1oo82BT81jH18y:
    using /dev/mapper/snap_data2 not /dev/dm-5
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/mapper/snap_data1 not /dev/dm-6
  Found duplicate PV YejBAMNvzPqKXg9HKr1oo82BT81jH18y:
    using /dev/mapper/data2 not /dev/mapper/snap_data2
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/mapper/data1 not /dev/mapper/snap_data1
Really WIPE LABELS from physical volume "/dev/mapper/snap_data2" of volume group
"data" [y/n]? y
  WARNING: Wiping physical volume label from /dev/mapper/snap_data2 of volume group
"data"
  Labels on physical volume "/dev/mapper/snap_data2" successfully wiped
hvr0007a:~ # pvremove -ff /dev/mapper/snap_data1
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/dm-6 not /dev/dm-1
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/mapper/snap_data1 not /dev/dm-6
  Found duplicate PV 3iktbXkL63xV2JoUxOPTYNv6FwC9WA3j:
    using /dev/mapper/data1 not /dev/mapper/snap_data1
Really WIPE LABELS from physical volume "/dev/mapper/snap_data1" of volume group
"data" [y/n]? y
  WARNING: Wiping physical volume label from /dev/mapper/snap_data1 of volume group
"data"
  Labels on physical volume "/dev/mapper/snap_data1" successfully wiped
```

Ignore pvremove's complaints. Recreate the physical volumes using the uuids invented above. As noted above, "pv0" was on /dev/dm-6, and /dev/dm-6 is /dev/mapper/data1; therefore /dev/mapper/snap_data1 must be initialized with the newly invented UUID of pv0, and /dev/mapper/snap_data2 with the new UUID of pv1.

```
hvr0007a:~ # pvcreate -u pvSNAP-kL63-xV2J-oUxO-PTYN-v6Fw-C9WA3j \
/dev/mapper/snap_data1

  Physical volume "/dev/mapper/snap_data1" successfully created

hvr0007a:~ # pvcreate -u pvSNAP-NvzP-qKXg-9HKr-1oo8-2BT8-1jH18y \
/dev/mapper/snap_data2

  Physical volume "/dev/mapper/snap_data1" successfully created
hvr0007a:~ #
```

Recreate the volume group:

```
hvr0007a:~ # vgcreate snap /dev/mapper/snap_data*
  Volume group "snap" successfully created

hvr0007a:~ # pvscan
  PV /dev/mapper/data1        VG data   lvm2 [200.00 GB / 197.00 GB free]
  PV /dev/mapper/data2        VG data   lvm2 [200.00 GB / 199.00 GB free]
  PV /dev/mapper/snap_data1   VG snap   lvm2 [200.00 GB / 200.00 GB free]
  PV /dev/mapper/snap_data2   VG snap   lvm2 [200.00 GB / 200.00 GB free]
  Total: 4 [799.98 GB] / in use: 4 [799.98 GB] / in no VG: 0 [0  ]
```

Everything is clean now.

The volume group is still empty and does not contain any volumes. Now, restore the modified metadata onto this group, activate it, fsck the volumes (the filesystems were snapped when they were mounted, and therefore they are dirty), and mount the logical volumes:

```
hvr0007a:~ # vgcfgrestore -f vgcfg_snap snap
  Restored volume group snap

hvr0007a:~ # vgchange -ay snap
  2 logical volume(s) in volume group "snap" now active

hvr0007a:~ # fsck /dev/snap/vol1
fsck 1.38 (30-Jun-2005)
e2fsck 1.38 (30-Jun-2005)
/dev/snap/vol1 was not cleanly unmounted, check forced.
[…]
/dev/snap/vol1: 12/262144 files (8.3% non-contiguous), 9005/524288 blocks

hvr0007a:~ # fsck /dev/snap/volstripe
fsck 1.38 (30-Jun-2005)
e2fsck 1.38 (30-Jun-2005)
/dev/snap/volstripe was not cleanly unmounted, check forced.
[…]
/dev/snap/volstripe: 12/262144 files (8.3% non-contiguous), 9005/524288 blocks

hvr0007a:~ # mount /dev/snap/vol1 /tmp/snap/vol1
hvr0007a:~ # mount /dev/snap/volstripe /tmp/snap/volstripe
hvr0007a:~ # mount | grep vol
/dev/mapper/data-vol1 on /tmp/data/vol1 type ext2 (rw)
/dev/mapper/data-volstripe on /tmp/data/volstripe type ext2 (rw)
/dev/mapper/snap-vol1 on /tmp/snap/vol1 type ext2 (rw)
/dev/mapper/snap-volstripe on /tmp/snap/volstripe type ext2 (rw)
hvr0007a:~ #
```

At this moment, we have successfully renamed the volume group on the replica and mounted the volumes simultaneously on the same host.

Renaming File System Labels and File System uuids

Whether it is necessary to “rename” file systems on disk replicas, i.e. to modify their metadata before the can be mounted, depends on the file system type. It is possible to mount an ext2/ext3 file system on a replica on the same host which already mounted the source file system; at least as long as the ext2/ext3 file system is mounted via the device node and not via the file system uuid or file system label. Renaming replicas is a best practice in a production environment to prevent confusion. If a server can see two disks (or partitions, or volumes) containing the same file system uuid, /dev/disk/by-uuid may point to one of those devices but you cannot predict which one. Don’t use file system label or file system uuid based names in this scenario.

A file system can be renamed by the file system's utility. An ext2/3 file system can be modified by "tune2fs -U random" to create a new random file system uuid, or by "tune2fs -L newlabel" to write a new label. These new uuids or labels will then be visible in /dev/disk/by-label or /dev/disk/by-label (after triggering udevd).

Depending on the file system type, you may need to perform renaming operations on a dedicated host. ocfs2 is an example of such a file system type; unlike ext2/ext3, it is not possible to mount two ocfs2 file systems with the same uuid on the same host or ocfs2 cluster, nor can a replica be relabelled on the same host that mounted the original file system. The replica may be mounted on the original host after renaming it on a separate host outside the o2cb cluster.

For more details, refer to the manpages of tunefs.ocfs2, tune2fs, xfs_admin etc.

References

[But2007] Michelle Butler, SAN Persistent Binding and Multipathing in the 2.6 Kernel
http://dims.ncsa.uiuc.edu/set/san/src/linux-mpio.pdf

[Ehl2008] Diedrich Ehlerding, How to Reconfigure Fibre Channel Attached Disk Storage Devices in Solaris. EMC Proven Professional Knowledge Sharing contest 2007/2008,
https://education.emc.com/knowledgesharing/reconfigure_fibre_channel.pdf

[EMC2006] Native Multipath Failover Based on DM-MPIO for v2.6.x Linux Kernel and EMC Storage Arrays
EMC P/N 300-003-575 REV A04

[EMC2008a] EMC® PowerPath® for Linux Version 5.1 Installation Guide
EMC P/N 300-005-942 Rev A01

[EMC2008b] EMC® PowerPath® Version 5.1 Product Guide
EMC P/N 300-005-165 REV A04

[Lew2006] A. Lewis, LVM HOW TO
http://tldp.org/HOWTO/LVM-HOWTO/

[Nov2008] SUSE Linux Enterprise Server Storage Administration Guide,
http://www.novell.com/documentation/sles10/stor_evms/index.html?page=/documentation/sles10/stor_evms/data/bookinfo.html

[QLA2007]
http://solutions.qlogic.com/KanisaSupportSite/dynamickc.do?command=show&externalId=Post-278000&forward=threadedKC&kcId=Post-278000&sliceId=Post-278000

[Ver2002] How to move a disk between disk groups
http://support.veritas.com/docs/182857

[Wie2008] Uwe Wienand, personal communication from the author of emcadm, an EMC global tool. Contact wienand_uwe@emc.com , or contact lewellin_gavin@emc.com

Biography

Diedrich Ehlerding is a (title) at Fujitsu-Siemens in Germany. He has achieved the following EMC Proven Professional certifications:

- EMCSA Symmetrix Business Continuity Specialist
- EMCIE CLARiiON Solution Specialist