LVM: Enterprise computing with the Linux Logical Volume Manager

VARIABLE DIVISION

A Logical Volume Manager (LVM) makes it possible to adapt disk capacity to dynamically changing requirements while the system is still in use. Heinz Mauelshagen explains why LVM is indispensable for business-critical applications

One of the most important requirements in the field of professional IT is to be able to reconfigure computer systems online and without halting operations. In this regard, logical volume management plays a major role. The advantages are obvious: time and costs are saved, as back up and restore tasks are dispensed with, and applications don't have to be interrupted, so there are no expensive system stoppages.

This is achieved by decoupling block devices and physical disk partitions. The latter, as physical storage media (Physical Volumes, PVs for short) form the lowest level of a three-level architecture. One or more PVs are combined on the second level into virtual disks (Volume Groups, VGs). The full memory capacity (minus a small metadata portion per PV) can be assigned to virtual partitions in the third level (Logical Volumes, LVs). The LVs are addressed as regular Linux block device files, so that any filesystems can be set up on them (see Figure 1).

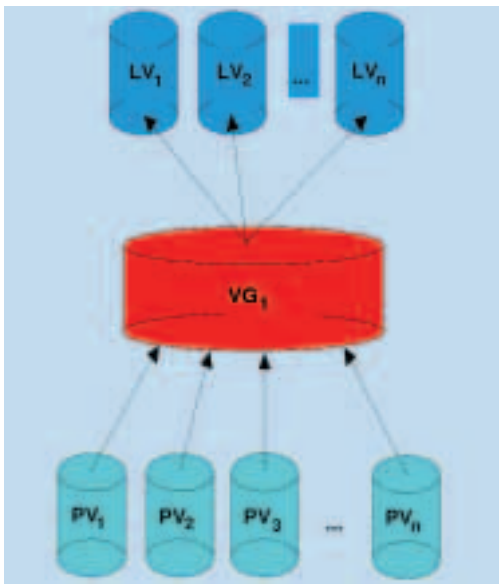

Figure 1: The three-level memory architecture of the Linux LVM

HP-UX as godfather

When LVM is used, physical disks can be added to a system and the capacity of existing volumes can be assigned to them. After many years of experience with commercial LVM variants of HP, IBM, Sun and Veritas, my fingers were itching to expand Linux by LVM functionalities. The LVM project started in February 1997, and version 0.1 was launched in July 1997. After some wide-ranging functional expansions in the past few years, version 1.0 was released in August of this year.

Since the LVM in HP-UX displays a highly intuitive command line interface and thereby considerably cuts down on the costs of learning, my own implementation is largely based on this. Incidentally, LVM was originally developed by IBM and adopted by the OSF (now the Open Group). The LVM implementation in HP-UX is based on this.

To be able to use the logic block devices provided by the LVs under Linux too, and to set up filesystems on them, an expansion of the Linux kernel is necessary. This is done by means of a device driver.

Queuing magic

In this article, I will limit myself to the principle and the application of LVM under Linux 2.4. If anyone would like to delve somewhat deeper, I would advise studying the source code of the kernel and Linux LVM – assuming you have knowledge of C. Hence this article will also make references to these sources.

Under Linux 2.4, in addition to elementary functions such as open, close, read and write, a block device driver registers a make request function, which is invoked by a central function of the Linux block device layer (see /usr/ src/ linux/ drivers/ block/ ll_rw_blk.c; functions ll_rw_block, submit_bh and generic_make_request), before an I/O request is ranked in a device-specific queue.

Queues serve the best possible processing of I/O requests, by holding these for a (very) short time in the queue (device plugging). This is in order to put them in the best possible sequence before they are passed on to the device for processing (such as an ATA-adapter) (device-unplugging).

Since LVM has to convert between logical and subordinate (physical) devices, it implements a re-mapping driver. This contains its own make request function and registers it, so that before submitting a request (call up of generic_make_request in /usr/ src/ linux/ drivers/ block/ ll_rw_blk.c), it can perform manipulations on its administration data.

The administration data relevant to us is in the buffer_ head structure, which is set up by the kernel for each buffer containing I/O data. All I/O data

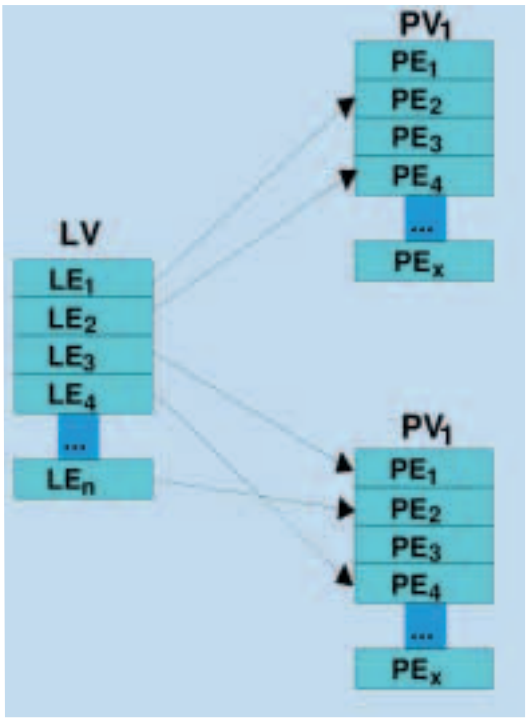

Figure 2: Example of the mapping of an LV onto two different PVs

buffers in the disk cache – which the kernel maintains and dynamically adapts in size for performance reasons – have a *buffer_head* structure.

Linux 2.4, unlike Version 2.2 and predecessors, now only performs caching in a page cache and uses buffer_ head only at the interface with the block device layer, whose central function is ll_rw_ block. buffer_head (see */usr/ src/ linux/ include/ linux/ fs.h) has, in addition to several members, a real sector address – b_rsector – and the address of a real device – b_rdev – as content.*

After opening a logical volume (LV) by mke2fs there follows some read and write accesses, so as to save the Ext2 filesystem structures on it. The ll_rw_block calls executed at this point lead directly to the invocation of the LVM driver Make-Request function, which is called the lvm_make_request_fn and is defined in /usr/ src/ linux/ drivers/ md/ lvm.c. The function invoked by lvm_make_request_fn, lvm_map requires a table in which the addresses of the devices – namely those of the physical volumes and the sector addresses thereon – are listed.

To avoid the need for a table entry for every individual sector – which would end up as a gigantic table – a number of sectors lying one behind the other are combined into physical extents (PE) and assigned one to one in the logical address space of LV, as logical extents (LE) of the same size. The mapping table thus contains an entry for each assigned PE, describing the address (b_rdev) and the real start sector (b_rsector) on the respective PV (Figure 2).

LVM application

Once all the LVM elements (PV, VG, LV, LE and PE) have been pre-set, it's the turn of the application itself. Similarly to PVs, whose names are defined by the device files issued by Linux (such as /dev/ hdb2), VGs and LVs also receive a name when they are re-made. VG names appear in the form of subdirectories in /dev/ and LV names appear as block device files in the VG subdirectories.

The user interface of the Linux LVM is implemented as a CLI (Command Line Interface) with 35 commands, which correspond to the three levels of the memory architecture. All commands for manipulation of the PVs begin with pv; all those for the VGs with vg; and those belonging to LVs with lv.

Since almost every level is involved with creating, removing, displaying, extending, reducing, renaming, scanning or changing attributes, most of the command names are produced from a combination of the prefixes pv, vg or lv with these abilities (Table 1).

In addition to these, there are commands for backing up and restoring the metadata stored on the PVs; to move VGs from one system to another; to combine two VGs into one or to split up one VG; to move LEs or LVs of assigned PEs; and to change the size of an LV including the Ext2 filesystem (Table 2).

Don't be scared off by the number of commands at this point, since only three commands are necessary to create the first LV: pvcreate, vgcreate and lvcreate. There are manuals available for all the

Table 1: Basic LVM commands

pvcreate	Create a PV
pvdisplay	Display the attributes of PVs
pvscan	Scan for existing PVs
pvchange	Alter attributes of PVs
vgcreate	Create a new VG
vgremove	Remove an empty VG without LVs
vgextend	Extend a VG by additional PVs
vgreduce	Reduce a VG by empty PVs
vgdisplay	Display the attributes of VGs
vgrename	Rename a VG
vgscan	Scan for existing VGs
vgchange	Change attributes and activate/deactivate VGs
lvcreate	Create an LV
lvremove	Remove inactive LVs
lvextend	Extend an LV
lvreduce	Reduce an LV
lvdisplay	Display the attributes of LVs
lvrename	Rename an LV
lvscan	Scan for all existing LVs
lvchange	Change attributes of an LV

Table 2: Extended LVM commands

pvdata	Debug displays of the attributes of PVs
pvmove	Move LV data online
vgcfgbackup	Perform back up of metadata of VGs
vgcfgrestore	Restore metadata on PVs of a VG
vgck	Check consistency of metadata of VGs
vgexport	Log off a VG, in order to move its PVs to another system
vgimport	Make moved VG known to the destination system
vgmerge	Combine two VGs into one
vgmknodes	Remake the device files of VGs
vgsplit	Split one VG into two
e2fsadm	Change size of LV and Ext2 file system
lvmchange	Reset LVM
lvmsadc	Collect statistical data
lvmsar	Display collected statistical data
lvmcreate_initrd	Create initial RAM disk to boot with root file system on LV
lvmdiskscan	Scan for devices supported as PV

commands. To get you started, there is a basic introduction with a list of all the commands (man lvm).

Fdisk indispensable

To avoid unintentionally overwriting a partition already in use with pvcreate, partitions must be set via fdisk to the type reserved for LVM, 0x8E; only then can pvcreate be used on them. It is in any case advisable to create at least one partition, even if the whole disk is to be used as PV under LVM. The advantage is that this then appears under /proc/partitions and is displayed in fdisk, so it simply cannot appear unused later by mistake, if one invokes fdisk -l, for example. The disadvantage – that the partitions table (one sector) for the PV gets lost – is an acceptable price to pay.

Simple practical examples

If you have created /dev/ sde1 as described and have set the type to 0x8E, you can use

```
pvcreate /dev/sde1
```

to create a first PV, then with

```
vgcreate mevg /dev/sde1
```

a first VG named mevg (my first VG). If this is successful, vgcreate automatically loads the necessary metadata in the LVM driver, so that subsequently the mapping tables of existing LVs are available or tables of newly created LVs can be loaded. Seen another way, vgcreate creates our first virtual disk, which (still) contains a physical disk partition, and activates the VG for further use.

The first LV is created with the command

```
lvcreate -n melv -L100 mevg
```

which creates an LV named melv (my first LV) with 100Mb. This LV has the device name /dev/ mevg/melv. Via

```
mke2fs /dev/mevg/melv
```

a filesystem can now be installed, which can be mounted as usual in any directory of your choice that uses a normal partition-based filesystem.

These first few steps do not yet display any particular strengths since all we have done is create a virtual partition on a physical one. If the LV becomes too small and there is still free capacity in the VG, we can expand it without re-installation or a reboot. This is done with the command

```
lvextend -L+200M /dev/mevg/melv
```

which adds a further 200Mb to the 100Mb. Since the filesystem stored in the LV is not (yet) automatically expanded at the same time, a filesystem command has to take over this task. If one uses Ted Ts'o's resize2fs, the filesystem must not be mounted; on the other hand, Andreas Dilger's ext2online is capable of expanding Ext2 filesystems in mounted condition – providing you have the necessary kernel patch for this:

```
resize2fs /dev/mevg/melv
```

Both tools are supported by the e2fsadm program, which is supplied with Linux LVM, which executes lvextend and resize2fs in Ext2 resizings.

```
e2fsadm -L+200M /dev/mevg/melv
```

Figure 3 shows the main inputs and outputs for the

Figure 3: Installing and expanding a filesystem

Figure 4: Instead of calling up lvextend and resize2fs separately, one can also use e2fsadm

little sample session. The last two commands (lvextend and resize2fs) can be replaced by calling up e2fsadm (see Figure 4).

Since expanding or newly creating LVs can easily make our VG reach the limits of its capacity, it's possible to add additional disk space after installation. New PVs are created as described above. If /dev/ sdb1 is available as an additional partition, the new PV is initialised via

```
pvcreate /dev/sdb1
```

The VG mevg is then expanded using:

```
vgextend mevg /dev/sdb1
```

Then the extra storage space is immediately available for the creation or expansion of LVs. If it is not possible to install additional hard drives in advance, all the steps for expanding the VG can be done without rebooting. Owners of Hot-Plug SCSI do not even have to reboot after installing new SCSI disks.

Removing disks

If you want to remove PVs from a VG, these must be empty. In other words, none of their PEs must be assigned to any LVs. This can be checked using the instruction:

```
pvdisplay -v  /dev/sde1
```

The mapping of a specific LV can be found with:

```
lvdisplay -v /dev/mevg/melv
```

If PEs are occupied, but sufficient capacity is free on other PVs of the same VG, the PV can be emptied with

```
pvmove /dev/sde1
```

(the option -v displays the relocation of the individual LEs). Data in assigned PEs can be relocated to PEs of other PVs without data loss, with the aid of pvmove. After that, the free PV can be removed from the VG with

```
vgreduce mevg /dev/sde1
```

to add it to another VG whose capacity has now become too small, for example.

Another instance when pvmove may be used is to replace partition A on a disk, which is too small or too slow (/dev/ sde) by a larger or faster one (B on /dev/ sdb). Provided there is a free connection for B, one would first add this to the VG, in order to then move all data with

```
pvmove /dev/sde1 /dev/sdb1
```

from A to B. If only A and B are contained in the VG, it is superfluous to specify the device file of B (/dev/ sdb1), since apart from this, no other destination PV exists. Figure 5 shows the expansion and reduction of the VG together with the relocation of data with pvmove.

You've been introduced to some of the standard applications of the Linux Logical Volume Manager. For more in-depth information, the LVM-Howtos and the general guide over at *http://www.sistina.com* are highly recommended.

Info

Alessandro Rubini & Jonathon Corbet – Linux Device Drivers (O'Reilly)

LVM HOWTOs *http://tech.sistina.com/lvm/doc/lvm_howto*

LVM homepage *http://www.sistina.com/products_lvm.htm*

Figure 5: pvmove relocates data onto other disks

The author

Heinz Mauelshagen is the author of the Logical Volume Manager for Linux. He works at Sistina Software, Inc., which specialises in file system development; in addition to LVM, they also maintain GFS.