Workshop: Setting up and Managing Software RAID

Software safety first

Security of the data on your hard disks doesn't always have to rely on hardware solutions. If your machine has a fast enough processor and you can do without a hot swap facility then Software RAID should meet your needs. The only question now is how to migrate your your current installation with a minimum of fuss?

BY CARSTEN WIESE

Experts continue to argue about the usefulness of Software Raid systems. Statements such as, "Software Raid? Couldn't you afford a hardware solution?" are typical.

Of course, Software Raid isn't exactly a high-end solution, but in the low to midrange segment it is an alternative to a hard disk controller with Raid capabilities, where the driver and thus again the main CPU does all the work. So why not opt for Software Raid?

The Multiple Device driver allows the kernel to support up to 256 multiple devices (*/dev/md*X) independently of your hardware configuration. A multiple device is a virtual block device comprising of two or more partitions that are addressed as an array. The performance of this array will depend on the clock speed and processor load as well as the transfer rate of the individual disks.

THE AUTHOR

Carsten Wiese works as a Systems Integrator for Höft and Wessel, in Hannover, Germany. Amongst other tasks, Carsten designs and implements Raid systems and high availability solutions, unfortunately, not only for Linux.

Never Mind the Quality, feel the RAID

The Raid principle was described in a paper called "A Case for Redundant Arrays of Inexpensive Disks (RAID)" [1], which was published at Berkeley in 1988 by Katz, Gibson and Patterson. Information is distributed across multiple (redundant) disks, and depending on the logical relationship between the disks, this will either improve the transfer rate or provide redundancy.

Since then about a dozen Raid levels have been specified. Version 0.90 of the Linux Multiple Device supports five distinct Raid levels (refer to the "RAID Levels" box).

Migrating On-the-Fly

The following workshop discusses how to convert an existing Linux system to a Raid 5 system with three disks on-the-fly. This approach has the advantage that the information on the existing disk do not need to be copied to a fourth, however, it is definitely a good idea to back up your data – simply mistaking one name could lead to you overwriting your system disk.

The Raid tools we will be using, and the *mdadm* program, are included with SuSE Linux 8.2 Professional and Red Hat 9.0.

The starting point for our migration task is a system with three IDE disks. The disks do not need to be identical for a Software Raid, but they should be more or less the same size and have approximately the same transfer rate.

hdc on our sample system already contains a working Linux system with the following partitions: *hdc1* (later *md0*) with 200 MBytes as */boot* set to use an ext3 filesystem, *hdc2* (later *md1*) with 1 GBytes as swap and *hdc3* (later *md3*) with 5 GBytes as the top level root partition */*, again using ext3 as the filesystem. To allow the system to boot under as many (error) conditions as possible, we decided to use Raid 1 for the */boot* partition.

Lilo instead of Grub

We opted to use lilo as our boot loader, as lilo version 22.0 or later can store emergency boot code in the master boot record on the other Raid drives. If *hda* fails, we can still boot form *hdb* or *hdc* in this case. Grub does not have this capability. We will specifying the *boot = /dev/hda* option for lilo while migrating the system.

The swap partition will be placed on the Raid 5 array. Of course, you could use three single swap partitions. If system stability in case of hard disk failure is very important to you, you should definitely put swap on a Raid 5 array. Even if a disk fails while the swap partition is being accessed, the kernel will still handle the failure gracefully.

During the migration phase, the kernel will view the current system disk *hdc* as a failed third disk in the Raid array. If you are creating a Raid 1 array, you can use this approach with two disks.

Kernel Options

You will need to enable multiple device support for the 2.4 kernel, as is shown in Figure 1. Additionally, the root filesystem should be configured in the kernel. Patches are available from [2] for the older kernels, 2.0 and 2.2.

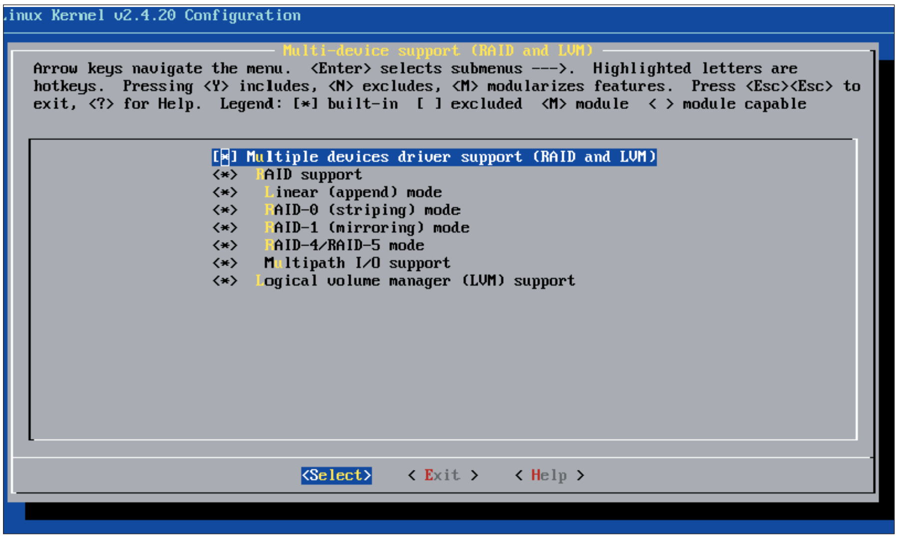

Figure 1: Multiple device support must be enabled for the kernel – default for most current distributions

The next step, after compiling and installing the kernel, is to set up and manage the Raid array. We will be using the Raid Tools package (version 1.0, [4]) and the *mdadm* [5] program. The following example uses the Raid Tools.

Step by Step to RAID 5

After completing the preparation work, type *swapoff -a* to disable the swap area, and then comment out the */etc/fstab* entry. Then use *fdisk* to create three *0xfd Linux raid auto* type partitions on disks *hda* and *hdb* (see "Partitioning the Raid Disks"). The *Linux raid auto* partition type allows the kernel to recognize the Software Raid on booting.

RAID Levels

Linear Mode

At least two disks are banded to form a virtual drive. If the capacity of the first drive is exhausted, write operations continue on the second. This mode will not provide redundancy or increase the transfer rate. The total capacity is equal to the sum of the capacities of all disks.

Raid 0 – Data Striping

Requires two or more drives. Striping will divide the data into chunks and distribute it evenly across the disks to boost performance. Again the total capacity is equal to that of all disks, however, this level does not provide redundancy.

Raid 1 – Disk Mirroring, Disk Duplexing

Raid 1 writes the same data simultaneously to two drives. As this is a redundant solution, the capacity is equal to that of the smaller drive. Performance may improve for read operations.

Raid 4 – Data Striping with Parity Drive

This mode is uncommon, as Raid 5 uses distribute parity to improve performance. At least three disks are required. One disk stores parity information while the other two are used for data striping as in Raid 0. Performance will depend on the transfer rate of the parity drive, and redundancy is provided. The total capacity is equal to the sum of all disks minus the parity disk.

Raid 5 – Data Striping with Distributed Parity

The data chunks and parity information are distributed evenly across at least three disks. If one disk fails, its content can be reconstructed from the remaining disks. The total capacity is equal to that of Raid 4 with improved performance.

Building Multiple Devices

The *mkraid /dev/md0* command creates the new Raid 1, *md0*, by reference to the first part of */etc/raidtab* without the hdc1 partition, as this partition is marked as failed in */etc/raidtab*. Amongst other information, the output shows the partition size for *hda1* and *hdb1*, and the position of the Raid super-block, 200704 KBytes. You can query the current status of a multiple device at any time using *cat /proc/mdstat*; at this stage, it should look something like Listing 2.

Most distributions create device files *md0* through *md255* by default. If these entries are missing below */dev*, you will need to use the *mknod -m 0660 /dev/mdX b 9 X* syntax to create the devices, with *root* as the owner and *disk* as the group. Now create the */etc/raidtab* file, Listing 1 shows an example.

The multiple devices *md1* and *md2* are created using the same approach. You can then go on to format the multiple devices, creating a swap partition on *md1*, and an Ext3 partition on both *md1* and *md2*.

We will be placing the root filesystem on the multiple device */dev/md2* to allow lilo to discover the new root system; (*root = /dev/md2* in */etc/lilo.conf*). The *boot = /dev/md1* entry must remain as it is for the time being.

Copying Data to the RAID

Switch your Linux system to single-user mode and mount *md2* below */mnt/system*, and *md0* below */mnt/boot*. You can now copy the data from the system disk *hdc3* to */mnt/system* – using the following syntax, for example, *find . -xdev | cpio -pm /mnt/system*.

Delete the contents of the */mnt/system/boot* directory and then copy all the entries in */boot*, including symbolic links, to */mnt/boot*. Then in the */mnt/system/etc/fstab* file swap the entries for */dev/hdc1* and */dev/md0*, */dev/hda3* and */dev/md2*, and *swap* (still commented out) */dev/hdc2* and */dev/md1*. Run *lilo* to apply the new configuration, and reboot to single-user mode. Check whether *md0* and *md2* have mounted correctly.

Rebooting the RAID

After completing these steps, you can now add *hdc* to the Raid array. To do so, partition *hdc* as described in the "Partitioning the Raid Disks" insert. Comment out the *failed disk* in */etc/raidtab* and reinstate the *raid-disk* entries. The following commands:

```
raidhotadd /dev/md0 /dev/hdc1
raidhotadd /dev/md1 /dev/hdc2
raidhotadd /dev/md2 /dev/hdc3
```

will add the three partitions to the Raid array. Wait until the rebuild process has completed before performing the following steps. *cat /proc/mdstat* shows you the current progress state.

The second to last step is to change the boot entry in the */etc/lilo.conf* file so that it knows about these new *boot* partitions. The boot option should now read:

```
boot=/dev/md0
```

and add a new line:

```
raid-boot-extra=/dev/hda,↩
/dev/hdb,/dev/hdc
```

Run *lilo* as a command so that these changes are applied to the system.≠ You can then reinstate swap in */etc/fstab*. The next reboot is the exciting bit – but you will need to wait for the Raid array to complete rebuilding.

Mdadm or RAID-Tools?

So far, we have only used the Raid Tools. Which of these tools is the better will depend to a large extent on the task in hand. In contrast to the Raid Tools package, which comprises multiple programs, the *mdadm* program provides the required functionality within a single tool, and can even create Raid arrays without a configuration file.

The tool's most interesting feature is the *--monitor* parameter, which allows you to monitor multiple devices and mail the administrator, or even launch a program, in case of a failure. Additionally, *mdadm --examine /dev/md*X can read the Raid superblock.

The RAID Superblock's mystery

When you create a multiple device, the Raid Superblock is created by parsing the information in the */etc/raidtab* file, assuming that the *persistent-superblock* option in this file is set to *1*. During the

Listing 1: The Raid Tools configuration file

```
#/etc/raidtab
# /boot md0 as RAID 5
raiddev         /dev/md0  # This
is the name used to access the
device
raid-level          1  # The RAID
Level
nr-raid-disks          3  # The
number of disks in the RAID array
chunk-size           64  #
Irrelevant for RAID1 but must be
present
persistent-superblock  1  #
Required for booting the disks
nr-spare-disks         0  #
Define a spare disk here
device          /dev/hda1
raid-disk              0
device          /dev/hdb1
raid-disk              1
device          /dev/hdc1
#raid-disk             2  #
Required after migration
failed-disk            2  # as
hdc is the current system disk
#------------------------------
---
# swap md1 as RAID 5
raiddev          /dev/md1
raid-level             5
nr-raid-disks          3
chunk-size            64
parity-algorithm left-symmetric
persistent-superblock  1
nr-spare-disks         0
device          /dev/hda2
raid-disk              0
device          /dev/hdb2
raid-disk              1
device          /dev/hdc2
#raid-disk             2
failed-disk            2
#------------------------------
---
# /root md2 as RAID 5
raiddev          /dev/md2
raid-level             5
nr-raid-disks          3
chunk-size            64
parity-algorithm left-symmetric
persistent-superblock  1
nr-spare-disks         0

device          /dev/hda3
raid-disk              0

device          /dev/hdb3
raid-disk              1

device          /dev/hdc3
#raid-disk             2
failed-disk            2
```

Listing 2: Raid Status (*/proc/mdstat*)

```
Personalities : [linear] [raid0] [raid1] [raid5] [multipath]
read_ahead 1024 sectors
md0 : active raid1 hdb1[1] hda1[0]
      200704 blocks [3/2] [UU_]
unused devices: <none>
handling MD device /dev/md1
analyzing super-block
disk 0: /dev/hda2, 987997kB, raid superblock at 987904kB
disk 1: /dev/hdb2, 987997kB, raid superblock at 987904kB
disk 2: /dev/hdc2, failed
```

Partitioning the Raid Disks

The partitions on the new Raid disks *hda* and *hdb* should be identical; the partition list for *hda* follows:

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		1	25	200781	fd	Linux raid autodetect
/dev/hda2		26	148	987997+	fd	Linux raid autodetect
/dev/hda3		149	757	4891792+	fd	Linux raid autodetect

boot process, the kernel will search for Raid superblocks on any drives attached to the system.

The system uses the information stored in the superblock and the partition type *0xfd - Linux raid autodetect* to mount multiple devices. The 4 KByte data block, is located at the beginning of the first even 64 KByte block on each multiple device partition – thus, a maximum of 128 KBytes of a partition's total space can be used for the superblock.

Listing 3 shows the contents of the superblocks as output by the *mdadm --examine /dev/hda1* command. Line 13 shows the health status of the array. *no-errors* indicates that everything is OK. No need to worry about the *dirty* entry – this simply means that some information is waiting to be written out to disk. If the system is downed gracefully, this entry will automatically toggle to *clean*.

How do I get rid of the superblock? This question crops up regularly when a disk is removed from a multiple device array for re-use. The *mdadm --zero-superblock /dev/hda1* command should get rid of the intractable beast. You can use *dd if=/dev/zero of=/dev/hda1 bs=1 seek=200704 count=4* instead. The last command uses *dd* to write four 1 KByte blocks of null characters to /dev/hda1 starting at position 200704 KBytes. The exact position, this is 200704 KBytes in our case, is output on booting or can be derived by reading the Raid superblock itself.

Monitoring a Disk

Of course, if you use a Raid system, you will want to be alerted if things go wrong. One way of doing this is to use *mdadm≠* in monitor mode, as previously described. A quick look at the */proc/mdstat* file, either using a script or manually, will also indicate the fail state of a multiple device, or a disk. If you use the BigBrother monitoring tool, you can use the *bb-mdstat.sh* script from [6] to monitor your Software Raid.

In case of disk failure, most admins will be interested in automatic recovery. Just like most Raid controllers, our Software Raid is also capable of providing a spare disk (hot spare). If a spare disk is available, the kernel can automatically fail over to the spare, if a Raid disk fails. To allow this to happen, you will need to modify the */etc/raidtab* file again using the Raid Tools.

Some manual recovery steps are unavoidable if a you do not have a spare disk. In the case of the Raid 5 array we just created, we would need to down the system and add a new disk with the same partitioning to replace the failed disk. After rebooting the system, *raidhotadd /dev/mdX /dev/hdY* will add the new disk to the Raid; the rebuild should start a few minutes later.

Finally, you might like to refer to the Software Raid Howto [7] for additional information. ■

Listing 3: The Raid Superblock

```
01 /dev/hda1:
02           Magic : a92b4efc
03         Version : 00.90.00
04            UUID : 243e03bb:e3a486c3:ebf23ec9:fc518e36
05   Creation Time : Fri May  9 17:17:28 2003
06      Raid Level : raid1
07     Device Size : 200704 (196.00 MiB 205.52 MB)
08    Raid Devices : 3
09   Total Devices : 4
10 Preferred Minor : 0
11
12     Update Time : Fri May  9 21:57:15 2003
13           State : dirty, no-errors
14  Active Devices : 3
15 Working Devices : 3
16  Failed Devices : 1
17   Spare Devices : 0
18        Checksum : 16cd8686 - correct
19          Events : 0.25
20       Number   Major   Minor   RaidDevice State
21 this     0       3       1        0      active sync   /dev/hda1
22    0     0       3       1        0      active sync   /dev/hda1
23    1     1       3      65        1      active sync   /dev/hdb1
24    2     2      22       1        2      active sync   /dev/hdc1
```

INFO

[1] David A. Patterson, Garth A. Gibson and Randy H. Katz: "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *http://sunsite.berkeley.edu/TechRepPages/CSD-87-391*

[2] Raid Kernel Patches: *http://people.redhat.com/mingo/raidpatches*

[3] Kernel Howto: *http://www.tldp.org/HOWTO/Kernel-HOWTO.html*

[4] Raid Tools: *http://people.redhat.com/mingo/raidtools*

[5] "mdadm": *http://www.cse.unsw.edu.au/~neilb/source/mdadm/*

[6] BigBrother Script: *http://www.deadcat.net/cgi-bin/download.pl?section=1&file=bb-mdstat.sh*

[7] Software Raid Howto: *http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html*