Software−RAID−HOWTO

Table of Contents

Table of Contents

The Software−RAID HOWTO

Jakob Østergaard (jakob@ostenfeld.dk)

v. 0.90.7 19th of January 2000

This HOWTO describes how to use Software RAID under Linux. It addresses a specific version of the Software RAID layer, namely the 0.90 RAID layer made by Ingo Molnar and others. This is the RAID layer that will be standard in Linux−2.4, and it is the version that is also used by Linux−2.2 kernels shipped from some vendors. The 0.90 RAID support is available as patches to Linux−2.0 and Linux−2.2, and is by many considered far more stable that the older RAID support already in those kernels.

1.Introduction

For a description of the older RAID layer, the one which is standard in 2.0 and 2.2 kernels, see the excellent HOWTO from Linas Vepstas (linas@linas.org) available from the Linux Documentation Project at linuxdoc.org.

The home site for this HOWTO is http://ostenfeld.dk/~jakob/Software−RAID.HOWTO/, where updated versions appear first. The howto is written by Jakob Østergaard based on a large number of emails between the author and Ingo Molnar (mingo@chiara.csoma.elte.hu) −− one of the RAID developers −−, the linux−raid mailing list (linux−raid@vger.rutgers.edu) and various other people.

The reason this HOWTO was written even though a Software−RAID HOWTO already exists is, that the old HOWTO describes the old−style Software RAID found in the standard 2.0 and 2.2 kernels. This HOWTO describes the use of the new−style RAID that has been developed more recently. The new−style RAID has a lot of features not present in old−style RAID.

If you want to use the new−style RAID with 2.0 or 2.2 kernels, you should get a patch for your kernel, either from ftp://ftp.[your−country−code].kernel.org/pub/linux/daemons/raid/alpha, or more recently from http://people.redhat.com/mingo/ The standard 2.2 kernels does not have direct support for the new−style RAID described in this HOWTO. Therefore these patches are needed. *The old−style RAID support in standard 2.0 and 2.2 kernels is buggy and lacks several important features present in the new−style RAID software.*

As of this writing, the new−style RAID support is being merged into the 2.3 development kernels, and will therefore (most likely) be present in the 2.4 Linux kernel when that one comes out. But until then, the stable kernels must be patched manually.

You might want to use the `−ac` kernel releases done by Alan Cox, for RAID support in 2.2. *Some* of those contain the new−style RAID, and that will save you from patching the kernel yourself.

Some of the information in this HOWTO may seem trivial, if you know RAID all ready. Just skip those parts.

1.1 Disclaimer

The mandatory disclaimer:

Although RAID seems stable for me, and stable for many other people, it may not work for you. If you lose all your data, your job, get hit by a truck, whatever, it's not my fault, nor the developers'. Be aware, that you use the RAID software and this information at your own risk! There is no guarantee whatsoever, that any of the software, or this information, is in anyway correct, nor suited for any use whatsoever. Back up all your data before experimenting with this. Better safe than sorry.

That said, I must also say that I haven't had a single stability problem with Software RAID, I use it on quite a few machines with no problems what so ever, and I haven't seen other people having problems with random crashes or instability caused by RAID.

1.2 Requirements

This HOWTO assumes you are using a late 2.2.x or 2.0.x kernel with a matching raid0145 patch and the 0.90 version of the raidtools, or that you are using a late 2.3 kernel (version > 2.3.46) or eventually 2.4. Both the patches and the tools can be found at ftp://ftp.fi.kernel.org/pub/linux/daemons/raid/alpha, and in some cases at http://people.redhat.com/mingo/. The RAID patch, the raidtools package, and the kernel should all match as close as possible. At times it can be necessary to use older kernels if raid patches are not available for the latest kernel.

2.Why RAID ?

There can be many good reasons for using RAID. A few are; the ability to combine several physical disks into one larger ``virtual'' device, performance improvements, and redundancy.

2.1 Technicalities

Linux RAID can work on most block devices. It doesn't matter whether you use IDE or SCSI devices, or a mixture. Some people have also used the Network Block Device (NBD) with more or less success.

Be sure that the bus(ses) to the drives are fast enough. You shouldn't have 14 UW−SCSI drives on one UW bus, if each drive can give 10 MB/s and the bus can only sustain 40 MB/s. Also, you should only have one device per IDE bus. Running disks as master/slave is horrible for performance. IDE is really bad at accessing more that one drive per bus. Of Course, all newer motherboards have two IDE busses, so you can set up two disks in RAID without buying more controllers.

The RAID layer has absolutely nothing to do with the filesystem layer. You can put any filesystem on a RAID device, just like any other block device.

2.2 Terms

The word ``RAID'' means ``Linux Software RAID''. This HOWTO does not treat any aspects of Hardware RAID.

When describing setups, it is useful to refer to the number of disks and their sizes. At all times the letter **N** is used to denote the number of active disks in the array (not counting spare−disks). The letter **S** is the size of the smallest drive in the array, unless otherwise mentioned. The letter **P** is used as the performance of one disk in the array, in MB/s. When used, we assume that the disks are equally fast, which may not always be true.

Note that the words ``device'' and ``disk'' are supposed to mean about the same thing. Usually the devices that are used to build a RAID device are partitions on disks, not necessarily entire disks. But combining several partitions on one disk usually does not make sense, so the words devices and disks just mean ``partitions on different disks''.

2.3 The RAID levels

Here's a short description of what is supported in the Linux RAID patches. Some of this information is absolutely basic RAID info, but I've added a few notices about what's special in the Linux implementation of the levels. Just skip this section if you know RAID. Then come back when you are having problems :)

The current RAID patches for Linux supports the following levels:

- **Linear mode**
    - Two or more disks are combined into one physical device. The disks are ``appended'' to each other, so writing to the RAID device will fill up disk 0 first, then disk 1 and so on. The disks does not have to be of the same size. In fact, size doesn't matter at all here :)
    - There is no redundancy in this level. If one disk crashes you will most probably lose all your data. You can however be lucky to recover some data, since the filesystem will just be missing one large consecutive chunk of data.
    - The read and write performance will not increase for single reads/writes. But if several users use the device, you may be lucky that one user effectively is using the first disk, and the other user is accessing files which happen to reside on the second disk. If that happens, you will see a performance gain.
- **RAID−0**
    - Also called ``stripe'' mode. Like linear mode, except that reads and writes are done in parallel to the devices. The devices should have approximately the same size. Since all access is done in parallel, the devices fill up equally. If one device is much larger than the other devices, that extra space is still utilized in the RAID device, but you will be accessing this larger disk alone, during writes in the high end of your RAID device. This of course hurts performance.
    - Like linear, there's no redundancy in this level either. Unlike linear mode, you will not be able to rescue any data if a drive fails. If you remove a drive from a RAID−0 set, the RAID device will not just miss one consecutive block of data, it will be filled with small holes all over the device. e2fsck will probably not be able to recover much from such a device.
    - The read and write performance will increase, because reads and writes are done in parallel on the devices. This is usually the main reason for running RAID−0. If the busses to the disks are fast enough, you can get very close to N*P MB/sec.
- **RAID−1**
    - This is the first mode which actually has redundancy. RAID−1 can be used on two or more disks with zero or more spare−disks. This mode maintains an exact mirror of the information on one disk on the other disk(s). Of Course, the disks must be of equal size. If one disk is larger than another, your RAID device will be the size of the smallest disk.
    - If up to N−1 disks are removed (or crashes), all data are still intact. If there are spare disks available, and if the system (eg. SCSI drivers or IDE chipset etc.) survived the crash, reconstruction of the mirror will immediately begin on one of the spare disks, after detection of the drive fault.
    - Write performance is the slightly worse than on a single device, because identical copies of the data written must be sent to every disk in the array. Read performance is *usually* pretty bad because of an oversimplified read−balancing strategy in the RAID code. However, there has been implemented a much improved read−balancing strategy, which might be available for the Linux−2.2 RAID patches (ask on the linux−kernel list), and which will most likely be in the standard 2.4 kernel RAID support.
- **RAID−4**
    - This RAID level is not used very often. It can be used on three or more disks. Instead of completely mirroring the information, it keeps parity information on one drive, and writes data to the other disks in a RAID−0 like way. Because one disks is reserved for parity information, the size of the array will be (N−1)*S, where S is the size of the smallest drive in the array. As in RAID−1, the disks should either be of equal size, or you will just have to

accept that the S in the (N−1)*S formula above will be the size of the smallest drive in the array.
  - If one drive fails, the parity information can be used to reconstruct all data. If two drives fail, all data is lost.
  - The reason this level is not more frequently used, is because the parity information is kept on one drive. This information must be updated *every* time one of the other disks are written to. Thus, the parity disk will become a bottleneck, if it is not a lot faster than the other disks. However, if you just happen to have a lot of slow disks and a very fast one, this RAID level can be very useful.
- **RAID−5**
  - This is perhaps the most useful RAID mode when one wishes to combine a larger number of physical disks, and still maintain some redundancy. RAID−5 can be used on three or more disks, with zero or more spare−disks. The resulting RAID−5 device size will be (N−1)*S, just like RAID−4. The big difference between RAID−5 and −4 is, that the parity information is distributed evenly among the participating drives, avoiding the bottleneck problem in RAID−4.
  - If one of the disks fail, all data are still intact, thanks to the parity information. If spare disks are available, reconstruction will begin immediately after the device failure. If two disks fail simultaneously, all data are lost. RAID−5 can survive one disk failure, but not two or more.
  - Both read and write performance usually increase, but it's hard to predict how much.

Spare disks

Spare disks are disks that do not take part in the RAID set until one of the active disks fail. When a device failure is detected, that device is marked as ``bad'' and reconstruction is immediately started on the first spare−disk available.

Thus, spare disks add a nice extra safety to especially RAID−5 systems that perhaps are hard to get to (physically). One can allow the system to run for some time, with a faulty device, since all redundancy is preserved by means of the spare disk.

You cannot be sure that your system will survive a disk crash. The RAID layer should handle device failures just fine, but SCSI drivers could be broken on error handling, or the IDE chipset could lock up, or a lot of other things could happen.

2.4 Swapping on RAID

There's no reason to use RAID for swap performance reasons. The kernel itself can stripe swapping on several devices, if you just give them the same priority in the fstab file.

A nice fstab looks like:

```
/dev/sda2       swap            swap    defaults,pri=1   0 0
/dev/sdb2       swap            swap    defaults,pri=1   0 0
```

```
/dev/sdc2       swap            swap    defaults,pri=1   0 0
/dev/sdd2       swap            swap    defaults,pri=1   0 0
/dev/sde2       swap            swap    defaults,pri=1   0 0
/dev/sdf2       swap            swap    defaults,pri=1   0 0
/dev/sdg2       swap            swap    defaults,pri=1   0 0
```

This setup lets the machine swap in parallel on seven SCSI devices. No need for RAID, since this has been a kernel feature for a long time.

Another reason to use RAID for swap is high availability. If you set up a system to boot on eg. a RAID−1 device, the system should be able to survive a disk crash. But if the system has been swapping on the now faulty device, you will for sure be going down. Swapping on the RAID−1 device would solve this problem.

There has been a lot of discussion about whether swap was stable on RAID devices. This is a continuing debate, because it depends highly on other aspects of the kernel as well. As of this writing, it seems that swapping on RAID should be perfectly stable, *except* for when the array is reconstructing (eg. after a new disk is inserted into a degraded array). When 2.4 comes out this is an issue that will most likely get addressed fairly quickly, but until then, you should stress−test the system yourself until you are either satisfied with the stability or conclude that you won't be swapping on RAID.

You can set up RAID in a swap file on a filesystem on your RAID device, or you can set up a RAID device as a swap partition, as you see fit. As usual, the RAID device is just a block device.

3.Hardware issues

This section will mention some of the hardware concerns involved when running software RAID.

3.1 IDE Configuration

It is indeed possible to run RAID over IDE disks. And excellent performance can be achieved too. In fact, today's price on IDE drives and controllers does make IDE something to be considered, when setting up new RAID systems.

- **Physical stability:** IDE drives has traditionally been of lower mechanical quality than SCSI drives. Even today, the warranty on IDE drives is typically one year, whereas it is often three to five years on SCSI drives. Although it is not fair to say, that IDE drives are per definition poorly made, one should be aware that IDE drives of *some* brand *may* fail more often that similar SCSI drives. However, other brands use the exact same mechanical setup for both SCSI and IDE drives. It all boils down to: All disks fail, sooner or later, and one should be prepared for that.
- **Data integrity:** Earlier, IDE had no way of assuring that the data sent onto the IDE bus would be the same as the data actually written to the disk. This was due to total lack of parity, checksums, etc. With the Ultra−DMA standard, IDE drives now do a checksum on the data they receive, and thus it becomes highly unlikely that data get corrupted.
- **Performance:** I'm not going to write thoroughly about IDE performance here. The really short story is:

    - IDE drives are fast (12 MB/s and beyond)
    - IDE has more CPU overhead than SCSI (but who cares?)
    - Only use **one** IDE drive per IDE bus, slave disks spoil performance
- **Fault survival:** The IDE driver usually survives a failing IDE device. The RAID layer will mark the disk as failed, and if you are running RAID levels 1 or above, the machine should work just fine until you can take it down for maintenance.

It is **very** important, that you only use **one** IDE disk per IDE bus. Not only would two disks ruin the performance, but the failure of a disk often guarantees the failure of the bus, and therefore the failure of all disks on that bus. In a fault−tolerant RAID setup (RAID levels 1,4,5), the failure of one disk can be handled, but the failure of two disks (the two disks on the bus that fails due to the failure of the one disk) will render the array unusable. Also, when the master drive on a bus fails, the slave or the IDE controller may get awfully confused. One bus, one drive, that's the rule.

There are cheap PCI IDE controllers out there. You often get two or four busses for around $80. Considering the much lower price of IDE disks versus SCSI disks, I'd say an IDE disk array could be a really nice solution if one can live with the relatively low (around 8 probably) disks one can attach to a typical system (unless of course, you have a lot of PCI slots for those IDE controllers).

IDE has major cabling problems though when it comes to large arrays. Even if you had enough PCI slots, it's unlikely that you could fit much more than 8 disks in a system and still get it running without data corruption (caused by too long IDE cables).

3.2 Hot Swap

This has been a hot topic on the linux−kernel list for some time. Although hot swapping of drives is supported to some extent, it is still not something one can do easily.

Hot−swapping IDE drives

Don't ! IDE doesn't handle hot swapping at all. Sure, it may work for you, if your IDE driver is compiled as a module (only possible in the 2.2 series of the kernel), and you re−load it after you've replaced the drive. But you may just as well end up with a fried IDE controller, and you'll be looking at a lot more down−time than just the time it would have taken to replace the drive on a downed system.

The main problem, except for the electrical issues that can destroy your hardware, is that the IDE bus must be re−scanned after disks are swapped. The current IDE driver can't do that. If the new disk is 100% identical to the old one (wrt. geometry etc.), it *may* work even without re−scanning the bus, but really, you're walking the bleeding edge here.

Hot−swapping SCSI drives

Normal SCSI hardware is not hot−swappable either. It **may** however work. If your SCSI driver supports re−scanning the bus, and removing and appending devices, you may be able to hot−swap devices. However, on a normal SCSI bus you probably shouldn't unplug devices while your system is still powered up. But then again, it may just work (and you may end up with fried hardware).

The SCSI layer **should** survive if a disk dies, but not all SCSI drivers handle this yet. If your SCSI driver dies when a disk goes down, your system will go with it, and hot−plug isn't really interesting then.

Hot−swapping with SCA

With SCA, it should be possible to hot−plug devices. However, I don't have the hardware to try this out, and I haven't heard from anyone who's tried, so I can't really give any recipe on how to do this.

If you want to play with this, you should know about SCSI and RAID internals anyway. So I'm not going to write something here that I can't verify works, instead I can give a few clues:

- Grep for **remove−single−device** in **linux/drivers/scsi/scsi.c**
- Take a look at **raidhotremove** and **raidhotadd**

Not all SCSI drivers support appending and removing devices. In the 2.2 series of the kernel, at least the Adaptec 2940 and Symbios NCR53c8xx drivers seem to support this, others may and may not. I'd appreciate if anyone has additional facts here...

4.RAID setup

4.1 General setup

This is what you need for any of the RAID levels:

- A kernel. Preferably a stable 2.2.X kernel, or the latest 2.0.X. (If 2.4 is out when you read this, go for that one instead)
- The RAID patches. There usually is a patch available for the recent kernels. (If you found a 2.4 kernel, the patches are already in and you can forget about them)
- The RAID tools.
- Patience, Pizza, and your favorite caffeinated beverage.

All this software can be found at `ftp://ftp.fi.kernel.org/pub/linux` The RAID tools and patches are in the `daemons/raid/alpha` subdirectory. The kernels are found in the `kernel` subdirectory.

Patch the kernel, configure it to include RAID support for the level you want to use. Compile it and install it.

Then unpack, configure, compile and install the RAID tools.

Ok, so far so good. If you reboot now, you should have a file called `/proc/mdstat`. Remember it, that file is your friend. See what it contains, by doing a `cat /proc/mdstat`. It should tell you that you have the right RAID personality (eg. RAID mode) registered, and that no RAID devices are currently active.

Create the partitions you want to include in your RAID set.

Now, let's go mode−specific.

4.2 Linear mode

Ok, so you have two or more partitions which are not necessarily the same size (but of course can be), which you want to append to each other.

Set up the `/etc/raidtab` file to describe your setup. I set up a raidtab for two disks in linear mode, and the file looked like this:

```
raiddev /dev/md0
        raid-level      linear
        nr-raid-disks   2
        chunk-size      32
        persistent-superblock 1
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

Spare−disks are not supported here. If a disk dies, the array dies with it. There's no information to put on a spare disk.

You're probably wondering why we specify a `chunk−size` here when linear mode just appends the disks into one large array with no parallelism. Well, you're completely right, it's odd. Just put in some chunk size and don't worry about this any more.

Ok, let's create the array. Run the command

```
  mkraid /dev/md0
```

This will initialize your array, write the persistent superblocks, and start the array.

Have a look in `/proc/mdstat`. You should see that the array is running.

Now, you can create a filesystem, just like you would on any other device, mount it, include it in your fstab and so on.

4.3 RAID−0

You have two or more devices, of approximately the same size, and you want to combine their storage capacity and also combine their performance by accessing them in parallel.

Set up the `/etc/raidtab` file to describe your configuration. An example raidtab looks like:

```
raiddev /dev/md0
        raid-level      0
        nr-raid-disks   2
        persistent-superblock 1
        chunk-size     4
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

Like in Linear mode, spare disks are not supported here either. RAID−0 has no redundancy, so when a disk dies, the array goes with it.

Again, you just run

```
  mkraid /dev/md0
```

to initialize the array. This should initialize the superblocks and start the raid device. Have a look in `/proc/mdstat` to see what's going on. You should see that your device is now running.

/dev/md0 is now ready to be formatted, mounted, used and abused.

4.4 RAID−1

You have two devices of approximately same size, and you want the two to be mirrors of each other. Eventually you have more devices, which you want to keep as stand−by spare−disks, that will automatically become a part of the mirror if one of the active devices break.

Set up the `/etc/raidtab` file like this:

```
raiddev /dev/md0
        raid-level      1
        nr-raid-disks   2
        nr-spare-disks  0
        chunk-size     4
        persistent-superblock 1
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

If you have spare disks, you can add them to the end of the device specification like

```
        device          /dev/sdd5
        spare-disk      0
```

Remember to set the nr−spare−disks entry correspondingly.

Ok, now we're all set to start initializing the RAID. The mirror must be constructed, eg. the contents (however unimportant now, since the device is still not formatted) of the two devices must be synchronized.

Issue the

```
  mkraid /dev/md0
```

command to begin the mirror initialization.

Check out the `/proc/mdstat` file. It should tell you that the /dev/md0 device has been started, that the mirror is being reconstructed, and an ETA of the completion of the reconstruction.

Reconstruction is done using idle I/O bandwidth. So, your system should still be fairly responsive, although your disk LEDs should be glowing nicely.

The reconstruction process is transparent, so you can actually use the device even though the mirror is currently under reconstruction.

Try formatting the device, while the reconstruction is running. It will work. Also you can mount it and use it while reconstruction is running. Of Course, if the wrong disk breaks while the reconstruction is running, you're out of luck.

4.5 RAID−4

Note! I haven't tested this setup myself. The setup below is my best guess, not something I have actually had up running.

You have three or more devices of roughly the same size, one device is significantly faster than the other devices, and you want to combine them all into one larger device, still maintaining some redundancy information. Eventually you have a number of devices you wish to use as spare−disks.

Set up the /etc/raidtab file like this:

```
raiddev /dev/md0
        raid-level      4
        nr-raid-disks   4
        nr-spare-disks  0
        persistent-superblock 1
        chunk-size      32
        device          /dev/sdb1
        raid-disk       0
        device          /dev/sdc1
        raid-disk       1
        device          /dev/sdd1
        raid-disk       2
        device          /dev/sde1
        raid-disk       3
```

If we had any spare disks, they would be inserted in a similar way, following the raid−disk specifications;

```
        device         /dev/sdf1
        spare-disk     0
```

as usual.

Your array can be initialized with the

```
   mkraid /dev/md0
```

command as usual.

You should see the section on special options for mke2fs before formatting the device.

4.6 RAID−5

You have three or more devices of roughly the same size, you want to combine them into a larger device, but still to maintain a degree of redundancy for data safety. Eventually you have a number of devices to use as spare−disks, that will not take part in the array before another device fails.

If you use N devices where the smallest has size S, the size of the entire array will be (N−1)*S. This ``missing'' space is used for parity (redundancy) information. Thus, if any disk fails, all data stay intact. But if two disks fail, all data is lost.

Set up the /etc/raidtab file like this:

```
raiddev /dev/md0
        raid-level      5
        nr-raid-disks   7
        nr-spare-disks  0
        persistent-superblock 1
        parity-algorithm        left-symmetric
        chunk-size      32
        device          /dev/sda3
        raid-disk       0
        device          /dev/sdb1
        raid-disk       1
        device          /dev/sdc1
        raid-disk       2
        device          /dev/sdd1
        raid-disk       3
        device          /dev/sde1
        raid-disk       4
        device          /dev/sdf1
        raid-disk       5
        device          /dev/sdg1
        raid-disk       6
```

If we had any spare disks, they would be inserted in a similar way, following the raid−disk specifications;

```
        device         /dev/sdh1
        spare-disk     0
```

And so on.

A chunk size of 32 KB is a good default for many general purpose filesystems of this size. The array on which the above raidtab is used, is a 7 times 6 GB = 36 GB (remember the (n−1)*s = (7−1)*6 = 36) device. It holds an ext2 filesystem with a 4 KB block size. You could go higher with both array chunk−size and filesystem block−size if your filesystem is either much larger, or just holds very large files.

Ok, enough talking. You set up the raidtab, so let's see if it works. Run the

```
mkraid /dev/md0
```

command, and see what happens. Hopefully your disks start working like mad, as they begin the reconstruction of your array. Have a look in `/proc/mdstat` to see what's going on.

If the device was successfully created, the reconstruction process has now begun. Your array is not consistent until this reconstruction phase has completed. However, the array is fully functional (except for the handling of device failures of course), and you can format it and use it even while it is reconstructing.

See the section on special options for mke2fs before formatting the array.

Ok, now when you have your RAID device running, you can always stop it or re−start it using the

```
raidstop /dev/md0
```

or

```
raidstart /dev/md0
```

commands.

Instead of putting these into init−files and rebooting a zillion times to make that work, read on, and get autodetection running.

4.7 The Persistent Superblock

Back in ``The Good Old Days'' (TM), the raidtools would read your `/etc/raidtab` file, and then initialize the array. However, this would require that the filesystem on which `/etc/raidtab` resided was mounted. This is unfortunate if you want to boot on a RAID.

Also, the old approach led to complications when mounting filesystems on RAID devices. They could not be put in the `/etc/fstab` file as usual, but would have to be mounted from the init−scripts.

The persistent superblocks solve these problems. When an array is initialized with the `persistent−superblock` option in the `/etc/raidtab` file, a special superblock is written in the beginning of all disks participating in the array. This allows the kernel to read the configuration of RAID devices directly from the disks involved, instead of reading from some configuration file that may not be available at all times.

You should however still maintain a consistent `/etc/raidtab` file, since you may need this file for later reconstruction of the array.

The persistent superblock is mandatory if you want auto−detection of your RAID devices upon system boot. This is described in the **Autodetection** section.

4.8 Chunk sizes

The chunk−size deserves an explanation. You can never write completely parallel to a set of disks. If you had two disks and wanted to write a byte, you would have to write four bits on each disk, actually, every second bit would go to disk 0 and the others to disk 1. Hardware just doesn't support that. Instead, we choose some chunk−size, which we define as the smallest ``atomic'' mass of data that can be written to the devices. A write of 16 KB with a chunk size of 4 KB, will cause the first and the third 4 KB chunks to be written to the first disk, and the second and fourth chunks to be written to the second disk, in the RAID−0 case with two disks. Thus, for large writes, you may see lower overhead by having fairly large chunks, whereas arrays that are primarily holding small files may benefit more from a smaller chunk size.

Chunk sizes must be specified for all RAID levels, including linear mode. However, the chunk−size does not make any difference for linear mode.

For optimal performance, you should experiment with the value, as well as with the block−size of the filesystem you put on the array.

The argument to the chunk−size option in `/etc/raidtab` specifies the chunk−size in kilobytes. So ``4'' means ``4 KB''.

RAID−0

Data is written ``almost'' in parallel to the disks in the array. Actually, `chunk-size` bytes are written to each disk, serially.

If you specify a 4 KB chunk size, and write 16 KB to an array of three disks, the RAID system will write 4 KB to disks 0, 1 and 2, in parallel, then the remaining 4 KB to disk 0.

A 32 KB chunk−size is a reasonable starting point for most arrays. But the optimal value depends very much on the number of drives involved, the content of the file system you put on it, and many other factors. Experiment with it, to get the best performance.

RAID−1

For writes, the chunk−size doesn't affect the array, since all data must be written to all disks no matter what. For reads however, the chunk−size specifies how much data to read serially from the participating disks. Since all active disks in the array contain the same information, reads can be done in a parallel RAID−0 like manner.

RAID−4

When a write is done on a RAID−4 array, the parity information must be updated on the parity disk as well. The chunk−size is the size of the parity blocks. If one byte is written to a RAID−4 array, then `chunk-size` bytes will be read from the N−1 disks, the parity information will be calculated, and `chunk-size` bytes written to the parity disk.

The chunk−size affects read performance in the same way as in RAID−0, since reads from RAID−4 are done in the same way.

RAID−5

On RAID−5 the chunk−size has exactly the same meaning as in RAID−4.

A reasonable chunk−size for RAID−5 is 128 KB, but as always, you may want to experiment with this.

Also see the section on special options for mke2fs. This affects RAID−5 performance.

4.9 Options for mke2fs

There is a special option available when formatting RAID−4 or −5 devices with mke2fs. The `−R stride=nn` option will allow mke2fs to better place different ext2 specific data−structures in an intelligent way on the RAID device.

If the chunk−size is 32 KB, it means, that 32 KB of consecutive data will reside on one disk. If we want to build an ext2 filesystem with 4 KB block−size, we realize that there will be eight filesystem blocks in one array chunk. We can pass this information on the mke2fs utility, when creating the filesystem:

```
mke2fs −b 4096 −R stride=8 /dev/md0
```

RAID−{4,5} performance is severely influenced by this option. I am unsure how the stride option will affect other RAID levels. If anyone has information on this, please send it in my direction.

The ext2fs blocksize *severely* influences the performance of the filesystem. You should always use 4KB block size on any filesystem larger than a few hundred megabytes, unless you store a very large number of very small files on it.

4.10 Autodetection

Autodetection allows the RAID devices to be automatically recognized by the kernel at boot−time, right after the ordinary partition detection is done.

This requires several things:

1. You need autodetection support in the kernel. Check this
2. You must have created the RAID devices using persistent−superblock
3. The partition−types of the devices used in the RAID must be set to **0xFD** (use fdisk and set the type to ``fd'')

NOTE: Be sure that your RAID is NOT RUNNING before changing the partition types. Use `raidstop`

`/dev/md0` to stop the device.

If you set up 1, 2 and 3 from above, autodetection should be set up. Try rebooting. When the system comes up, cat'ing `/proc/mdstat` should tell you that your RAID is running.

During boot, you could see messages similar to these:

```
 Oct 22 00:51:59 malthe kernel: SCSI device sdg: hdwr sector= 512
  bytes. Sectors= 12657717 [6180 MB] [6.2 GB]
 Oct 22 00:51:59 malthe kernel: Partition check:
 Oct 22 00:51:59 malthe kernel:  sda: sda1 sda2 sda3 sda4
 Oct 22 00:51:59 malthe kernel:  sdb: sdb1 sdb2
 Oct 22 00:51:59 malthe kernel:  sdc: sdc1 sdc2
 Oct 22 00:51:59 malthe kernel:  sdd: sdd1 sdd2
 Oct 22 00:51:59 malthe kernel:  sde: sde1 sde2
 Oct 22 00:51:59 malthe kernel:  sdf: sdf1 sdf2
 Oct 22 00:51:59 malthe kernel:  sdg: sdg1 sdg2
 Oct 22 00:51:59 malthe kernel: autodetecting RAID arrays
 Oct 22 00:51:59 malthe kernel: (read) sdb1's sb offset: 6199872
 Oct 22 00:51:59 malthe kernel: bind<sdb1,1>
 Oct 22 00:51:59 malthe kernel: (read) sdc1's sb offset: 6199872
 Oct 22 00:51:59 malthe kernel: bind<sdc1,2>
 Oct 22 00:51:59 malthe kernel: (read) sdd1's sb offset: 6199872
 Oct 22 00:51:59 malthe kernel: bind<sdd1,3>
 Oct 22 00:51:59 malthe kernel: (read) sde1's sb offset: 6199872
 Oct 22 00:51:59 malthe kernel: bind<sde1,4>
 Oct 22 00:51:59 malthe kernel: (read) sdf1's sb offset: 6205376
 Oct 22 00:51:59 malthe kernel: bind<sdf1,5>
 Oct 22 00:51:59 malthe kernel: (read) sdg1's sb offset: 6205376
 Oct 22 00:51:59 malthe kernel: bind<sdg1,6>
 Oct 22 00:51:59 malthe kernel: autorunning md0
 Oct 22 00:51:59 malthe kernel: running: <sdg1><sdf1><sde1><sdd1><sdc1><sdb1>
 Oct 22 00:51:59 malthe kernel: now!
 Oct 22 00:51:59 malthe kernel: md: md0: raid array is not clean --
  starting background reconstruction
```

This is output from the autodetection of a RAID−5 array that was not cleanly shut down (eg. the machine crashed). Reconstruction is automatically initiated. Mounting this device is perfectly safe, since reconstruction is transparent and all data are consistent (it's only the parity information that is inconsistent − but that isn't needed until a device fails).

Autostarted devices are also automatically stopped at shutdown. Don't worry about init scripts. Just use the /dev/md devices as any other /dev/sd or /dev/hd devices.

Yes, it really is that easy.

You may want to look in your init−scripts for any raidstart/raidstop commands. These are often found in the standard RedHat init scripts. They are used for old−style RAID, and has no use in new−style RAID with autodetection. Just remove the lines, and everything will be just fine.

4.11 Booting on RAID

There are several ways to set up a system that mounts it's root filesystem on a RAID device. At the moment, only the graphical install of RedHat Linux 6.1 allows direct installation to a RAID device. So most likely you're in for a little tweaking if you want this, but it is indeed possible.

The latest official lilo distribution (Version 21) doesn't handle RAID devices, and thus the kernel cannot be loaded at boot−time from a RAID device. If you use this version, your `/boot` filesystem will have to reside on a non−RAID device. A way to ensure that your system boots no matter what is, to create similar `/boot` partitions on all drives in your RAID, that way the BIOS can always load data from eg. the first drive available. This requires that you do not boot with a failed disk in your system.

With redhat 6.1 a patch to lilo 21 has become available that can handle `/boot` on RAID−1. Note that it doesn't work for any other level, RAID−1 (mirroring) is the only supported RAID level. This patch (`lilo.raid1`) can be found in `dist/redhat-6.1/SRPMS/SRPMS/lilo-0.21-10.src.rpm` on any redhat mirror. The patched version of LILO will accept `boot=/dev/md0` in `lilo.conf` and will make each disk in the mirror bootable.

Another way of ensuring that your system can always boot is, to create a boot floppy when all the setup is done. If the disk on which the `/boot` filesystem resides dies, you can always boot from the floppy.

4.12 Root filesystem on RAID

In order to have a system booting on RAID, the root filesystem (/) must be mounted on a RAID device. Two methods for achieving this is supplied bellow. Because none of the current distributions (that I know of at least) support installing on a RAID device, the methods assume that you install on a normal partition, and then − when the installation is complete − move the contents of your non−RAID root filesystem onto a new RAID device.

Method 1

This method assumes you have a spare disk you can install the system on, which is not part of the RAID you will be configuring.

- First, install a normal system on your extra disk.
- Get the kernel you plan on running, get the raid−patches and the tools, and make your system boot with this new RAID−aware kernel. Make sure that RAID−support is **in** the kernel, and is not loaded as modules.
- Ok, now you should configure and create the RAID you plan to use for the root filesystem. This is standard procedure, as described elsewhere in this document.
- Just to make sure everything's fine, try rebooting the system to see if the new RAID comes up on boot. It should.
- Put a filesystem on the new array (using `mke2fs`), and mount it under /mnt/newroot
- Now, copy the contents of your current root−filesystem (the spare disk) to the new root−filesystem (the array). There are lots of ways to do this, one of them is

```
cd /
find . −xdev | cpio −pm /mnt/newroot
```

- You should modify the `/mnt/newroot/etc/fstab` file to use the correct device (the `/dev/md?` root device) for the root filesystem.
- Now, unmount the current `/boot` filesystem, and mount the boot device on `/mnt/newroot/boot` instead. This is required for LILO to run successfully in the next step.
- Update `/mnt/newroot/etc/lilo.conf` to point to the right devices. The boot device must still be a regular disk (non−RAID device), but the root device should point to your new RAID. When done, run

  ```
  lilo −r /mnt/newroot
  ```

  This LILO run should complete with no errors.
- Reboot the system, and watch everything come up as expected :)

If you're doing this with IDE disks, be sure to tell your BIOS that all disks are ``auto−detect'' types, so that the BIOS will allow your machine to boot even when a disk is missing.

Method 2

This method requires that you use a raidtools/patch that includes the failed−disk directive. This will be the tools/patch for all kernels from 2.2.10 and later.

You can **only** use this method on RAID levels 1 and above. The idea is to install a system on a disk which is purposely marked as failed in the RAID, then copy the system to the RAID which will be running in degraded mode, and finally making the RAID use the no−longer needed ``install−disk'', zapping the old installation but making the RAID run in non−degraded mode.

- First, install a normal system on one disk (that will later become part of your RAID). It is important that this disk (or partition) is not the smallest one. If it is, it will not be possible to add it to the RAID later on!
- Then, get the kernel, the patches, the tools etc. etc. You know the drill. Make your system boot with a new kernel that has the RAID support you need, compiled into the kernel.
- Now, set up the RAID with your current root−device as the `failed−disk` in the `raidtab` file. Don't put the `failed−disk` as the first disk in the `raidtab`, that will give you problems with starting the RAID. Create the RAID, and put a filesystem on it.
- Try rebooting and see if the RAID comes up as it should
- Copy the system files, and reconfigure the system to use the RAID as root−device, as described in the previous section.
- When your system successfully boots from the RAID, you can modify the `raidtab` file to include the previously `failed−disk` as a normal `raid−disk`. Now, `raidhotadd` the disk to your RAID.
- You should now have a system that can boot from a non−degraded RAID.

4.13 Making the system boot on RAID

For the kernel to be able to mount the root filesystem, all support for the device on which the root filesystem resides, must be present in the kernel. Therefore, in order to mount the root filesystem on a RAID device, the kernel *must* have RAID support.

The normal way of ensuring that the kernel can see the RAID device is to simply compile a kernel with all necessary RAID support compiled in. Make sure that you compile the RAID support *into* the kernel, and *not* as loadable modules. The kernel cannot load a module (from the root filesystem) before the root filesystem is mounted.

However, since RedHat−6.0 ships with a kernel that has new−style RAID support as modules, I here describe how one can use the standard RedHat−6.0 kernel and still have the system boot on RAID.

Booting with RAID as module

You will have to instruct LILO to use a RAM−disk in order to achieve this. Use the `mkinitrd` command to create a ramdisk containing all kernel modules needed to mount the root partition. This can be done as:

```
mkinitrd --with=<module> <ramdisk name> <kernel>
```

For example:

```
mkinitrd --with=raid5 raid-ramdisk 2.2.5-22
```

This will ensure that the specified RAID module is present at boot−time, for the kernel to use when mounting the root device.

4.14 Pitfalls

Never NEVER **never** re−partition disks that are part of a running RAID. If you must alter the partition table on a disk which is a part of a RAID, stop the array first, then repartition.

It is easy to put too many disks on a bus. A normal Fast−Wide SCSI bus can sustain 10 MB/s which is less than many disks can do alone today. Putting six such disks on the bus will of course not give you the expected performance boost.

More SCSI controllers will only give you extra performance, if the SCSI busses are nearly maxed out by the disks on them. You will not see a performance improvement from using two 2940s with two old SCSI disks, instead of just running the two disks on one controller.

If you forget the persistent−superblock option, your array may not start up willingly after it has been stopped. Just re−create the array with the option set correctly in the raidtab.

If a RAID−5 fails to reconstruct after a disk was removed and re−inserted, this may be because of the ordering of the devices in the raidtab. Try moving the first ``device ..." and ``raid−disk ..." pair to the bottom of the array description in the raidtab file.

Most of the ``error reports'' we see on linux−kernel, are from people who somehow failed to use the right RAID−patch with the right version of the raidtools. Make sure that if you're running 0.90 RAID, you're using the raidtools for it

---

5.Testing

If you plan to use RAID to get fault−tolerance, you may also want to test your setup, to see if it really works. Now, how does one simulate a disk failure ?

The short story is, that you can't, except perhaps for putting a fire axe thru the drive you want to ``simulate'' the fault on. You can never know what will happen if a drive dies. It may electrically take the bus it's attached to with it, rendering all drives on that bus inaccessible. I've never heard of that happening though. The drive may also just report a read/write fault to the SCSI/IDE layer, which in turn makes the RAID layer handle this situation gracefully. This is fortunately the way things often go.

5.1 Simulating a drive failure

If you want to simulate a drive failure, then plug out the drive. You should do this with the **power off**. If you are interested in testing whether your data can survive with a disk less than the usual number, there is no point in being a hot−plug cowboy here. Take the system down, unplug the disk, and boot it up again.

Look in the syslog, and look at `/proc/mdstat` to see how the RAID is doing. Did it work ?

Remember, that you **must** be running RAID−{1,4,5} for your array to be able to survive a disk failure. Linear− or RAID−0 will fail completely when a device is missing.

When you've re−connected the disk again (with the power off, of course, remember), you can add the ``new'' device to the RAID again, with the `raidhotadd` command.

5.2 Simulating data corruption

RAID (be it hardware− or software−), assumes that if a write to a disk doesn't return an error, then the write was successful. Therefore, if your disk corrupts data without returning an error, your data *will* become corrupted. This is of course very unlikely to happen, but it is possible, and it would result in a corrupt filesystem.

RAID cannot and is not supposed to guard against data corruption on the media. Therefore, it doesn't make any sense either, to purposely corrupt data (using `dd` for example) on a disk to see how the RAID system will handle that. It is most likely (unless you corrupt the RAID superblock) that the RAID layer will never find out about the corruption, but your filesystem on the RAID device will be corrupted.

This is the way things are supposed to work. RAID is not a guarantee for data integrity, it just allows you to keep your data if a disk dies (that is, with RAID levels above or equal one, of course).

6.Reconstruction

If you've read the rest of this HOWTO, you should already have a pretty good idea about what reconstruction of a degraded RAID involves. I'll summarize:

- Power down the system
- Replace the failed disk
- Power up the system once again.
- Use `raidhotadd /dev/mdX /dev/sdX` to re−insert the disk in the array
- Have coffee while you watch the automatic reconstruction running

And that's it.

Well, it usually is, unless you're unlucky and you RAID has been rendered unusable because more disks than the ones redundant failed. This can actually happen if a number of disks reside on the same bus, and one disk takes the bus with it as it crashes. The other disks, however fine, will be unreachable to the RAID layer, because the bus is down, and they will be marked as faulty. On a RAID−5 where you can spare one disk, loosing two or more disks can be fatal.

The following section is the explanation that Martin Bene gave to me, and describes a possible recovery from the scary scenario outlined above. It involves using the `failed−disk` directive in your `/etc/raidtab`, so this will only work on kernels 2.2.10 and later.

6.1 Recovery from a multiple disk failure

The scenario is:

- A controller dies and takes two disks offline at the same time,
- All disks on one scsi bus can no longer be reached if a disk dies,
- A cable comes loose...

In short: quite often you get a *temporary* failure of several disks at once; afterwards the RAID superblocks are out of sync and you can no longer init your RAID array.

One thing left: rewrite the RAID superblocks by `mkraid −−force`

To get this to work, you'll need to have an up to date `/etc/raidtab` − if it doesn't **EXACTLY** match devices and ordering of the original disks this won't work.

Look at the sylog produced by trying to start the array, you'll see the event count for each superblock; usually it's best to leave out the disk with the lowest event count, i.e the oldest one.

If you `mkraid` without `failed−disk`, the recovery thread will kick in immediately and start rebuilding the parity blocks − not necessarily what you want at that moment.

With `failed−disk` you can specify exactly which disks you want to be active and perhaps try different combinations for best results. BTW, only mount the filesystem read−only while trying this out... This has been successfully used by at least two guys I've been in contact with.

7.Performance

This section contains a number of benchmarks from a real−world system using software RAID.

Benchmarks are done with the `bonnie` program, and at all times on files twice− or more the size of the physical RAM in the machine.

The benchmarks here *only* measures input and output bandwidth on one large single file. This is a nice thing to know, if it's maximum I/O throughput for large reads/writes one is interested in. However, such numbers tell us little about what the performance would be if the array was used for a news spool, a web−server, etc. etc. Always keep in mind, that benchmarks numbers are the result of running a ``synthetic'' program. Few real−world programs do what `bonnie` does, and although these I/O numbers are nice to look at, they are not ultimate real−world−appliance performance indicators. Not even close.

For now, I only have results from my own machine. The setup is:

- Dual Pentium Pro 150 MHz
- 256 MB RAM (60 MHz EDO)
- Three IBM UltraStar 9ES 4.5 GB, SCSI U2W
- Adaptec 2940U2W
- One IBM UltraStar 9ES 4.5 GB, SCSI UW
- Adaptec 2940 UW
- Kernel 2.2.7 with RAID patches

The three U2W disks hang off the U2W controller, and the UW disk off the UW controller.

It seems to be impossible to push much more than 30 MB/s thru the SCSI busses on this system, using RAID or not. My guess is, that because the system is fairly old, the memory bandwidth sucks, and thus limits what can be sent thru the SCSI controllers.

7.1 RAID−0

Read is **Sequential block input**, and **Write** is **Sequential block output**. File size was 1GB in all tests. The tests where done in single−user mode. The SCSI driver was configured not to use tagged command queuing.

Chunk size	Block size	Read KB/s	Write KB/s

4k	1k	19712	18035
4k	4k	34048	27061
8k	1k	19301	18091
8k	4k	33920	27118
16k	1k	19330	18179
16k	2k	28161	23682
16k	4k	33990	27229
32k	1k	19251	18194
32k	4k	34071	26976

From this it seems that the RAID chunk−size doesn't make that much of a difference. However, the ext2fs block−size should be as large as possible, which is 4KB (eg. the page size) on IA−32.

7.2 RAID−0 with TCQ

This time, the SCSI driver was configured to use tagged command queuing, with a queue depth of 8. Otherwise, everything's the same as before.

Chunk size	Block size	Read KB/s	Write KB/s
32k	4k	33617	27215

No more tests where done. TCQ seemed to slightly increase write performance, but there really wasn't much of a difference at all.

7.3 RAID−5

The array was configured to run in RAID−5 mode, and similar tests where done.

Chunk size	Block size	Read KB/s	Write KB/s
8k	1k	11090	6874
8k	4k	13474	12229
32k	1k	11442	8291
32k	2k	16089	10926

32k	4k	18724	12627

Now, both the chunk−size and the block−size seems to actually make a difference.

7.4 RAID−10

RAID−10 is ``mirrored stripes", or, a RAID−1 array of two RAID−0 arrays. The chunk−size is the chunk sizes of both the RAID−1 array and the two RAID−0 arrays. I did not do test where those chunk−sizes differ, although that should be a perfectly valid setup.

Chunk size	Block size	Read KB/s	Write KB/s
32k	1k	13753	11580
32k	4k	23432	22249

No more tests where done. The file size was 900MB, because the four partitions involved where 500 MB each, which doesn't give room for a 1G file in this setup (RAID−1 on two 1000MB arrays).

8.Credits

The following people contributed to the creation of this documentation:

- Ingo Molnar
- Jim Warren
- Louis Mandelstam
- Allan Noah
- Yasunori Taniike
- Martin Bene
- Bennett Todd
- The Linux−RAID mailing list people
- The ones I forgot, sorry :)

Please submit corrections, suggestions etc. to the author. It's the only way this HOWTO can improve.
