Software−RAID HOWTO

Table of Contents

Software−RAID HOWTO.....1

Linas Vepstas, linas@linas.org.....1
1.Introduction.....1
2.Understanding RAID.....1
3.Setup & Installation Considerations.....2
4.Error Recovery.....2
5.Troubleshooting Install Problems.....2
6.Supported Hardware & Software.....2
7.Modifying an Existing Installation.....2
8.Performance, Tools & General Bone−headed Questions.....2
9.High Availability RAID.....2
10.Questions Waiting for Answers.....2
11.Wish List of Enhancements to MD and Related Software.....2
1.Introduction.....2
2.Understanding RAID.....5
3.Setup & Installation Considerations.....9
4.Error Recovery.....15
5.Troubleshooting Install Problems.....21
6.Supported Hardware & Software.....24
7.Modifying an Existing Installation.....26
8.Performance, Tools & General Bone−headed Questions.....28
9.High Availability RAID.....37
10.Questions Waiting for Answers.....38
11.Wish List of Enhancements to MD and Related Software.....38

Software−RAID HOWTO

Linas Vepstas, linas@linas.org

v0.54, 21 November 1998

RAID stands for "Redundant Array of Inexpensive Disks", and is meant to be a way of creating a fast and reliable disk−drive subsystem out of individual disks. RAID can guard against disk failure, and can also improve performance over that of a single disk drive. This document is a tutorial/HOWTO/FAQ for users of the Linux MD kernel extension, the associated tools, and their use. The MD extension implements RAID−0 (striping), RAID−1 (mirroring), RAID−4 and RAID−5 in software. That is, with MD, no special hardware or disk controllers are required to get many of the benefits of RAID.

Preamble

RAID, although designed to improve system reliability by adding redundancy, can also lead to a false sense of security and confidence when used improperly. This false confidence can lead to even greater disasters. In particular, note that RAID is designed to protect against *disk* failures, and not against *power* failures or *operator* mistakes. Power failures, buggy development kernels, or operator/admin errors can lead to damaged data that it is not recoverable! RAID is *not* a substitute for proper backup of your system. Know what you are doing, test, be knowledgeable and aware!

1.Introduction

2.Understanding RAID

3.Setup & Installation Considerations

4.Error Recovery

5.Troubleshooting Install Problems

6.Supported Hardware & Software

7.Modifying an Existing Installation

8.Performance, Tools & General Bone−headed Questions

9.High Availability RAID

10.Questions Waiting for Answers

11.Wish List of Enhancements to MD and Related Software

1.Introduction

1. **Q**: What is RAID?

   A: RAID stands for "Redundant Array of Inexpensive Disks", and is meant to be a way of creating a fast and reliable disk−drive subsystem out of individual disks. In the PC world, "I" has come to stand for "Independent", where marketing forces continue to differentiate IDE and SCSI. In it's original meaning, "I" meant "Inexpensive as compared to refrigerator−sized mainframe 3380 DASD", monster drives which made nice houses look cheap, and diamond rings look like trinkets.

2. **Q**: What is this document?

A: This document is a tutorial/HOWTO/FAQ for users of the Linux MD kernel extension, the associated tools, and their use. The MD extension implements RAID−0 (striping), RAID−1 (mirroring), RAID−4 and RAID−5 in software. That is, with MD, no special hardware or disk controllers are required to get many of the benefits of RAID.

This document is **NOT** an introduction to RAID; you must find this elsewhere.

3. **Q**: What levels of RAID does the Linux kernel implement?

   A: Striping (RAID−0) and linear concatenation are a part of the stock 2.x series of kernels. This code is of production quality; it is well understood and well maintained. It is being used in some very large USENET news servers.

   RAID−1, RAID−4 & RAID−5 are a part of the 2.1.63 and greater kernels. For earlier 2.0.x and 2.1.x kernels, patches exist that will provide this function. Don't feel obligated to upgrade to 2.1.63; upgrading the kernel is hard; it is *much* easier to patch an earlier kernel. Most of the RAID user community is running 2.0.x kernels, and that's where most of the historic RAID development has focused. The current snapshots should be considered near−production quality; that is, there are no known bugs but there are some rough edges and untested system setups. There are a large number of people using Software RAID in a production environment.

   RAID−1 hot reconstruction has been recently introduced (August 1997) and should be considered alpha quality. RAID−5 hot reconstruction will be alpha quality any day now.

   A word of caution about the 2.1.x development kernels: these are less than stable in a variety of ways. Some of the newer disk controllers (e.g. the Promise Ultra's) are supported only in the 2.1.x kernels. However, the 2.1.x kernels have seen frequent changes in the block device driver, in the DMA and interrupt code, in the PCI, IDE and SCSI code, and in the disk controller drivers. The combination of these factors, coupled to cheapo hard drives and/or low−quality ribbon cables can lead to considerable heartbreak. The `ckraid` tool, as well as `fsck` and `mount` put considerable stress on the RAID subsystem. This can lead to hard lockups during boot, where even the magic alt−SysReq key sequence won't save the day. Use caution with the 2.1.x kernels, and expect trouble. Or stick to the 2.0.34 kernel.

4. **Q**: I'm running an older kernel. Where do I get patches?

   A: Software RAID−0 and linear mode are a stock part of all current Linux kernels. Patches for Software RAID−1,4,5 are available from http://luthien.nuclecu.unam.mx/~miguel/raid. See also the quasi−mirror ftp://linux.kernel.org/pub/linux/daemons/raid/ for patches, tools and other goodies.

5. **Q**: Are there other Linux RAID references?

   A:

- Generic RAID overview: http://www.dpt.com/uraiddoc.html.
- General Linux RAID options: http://linas.org/linux/raid.html.
- Latest version of this document: http://linas.org/linux/Software−RAID/Software−RAID.html.
- Linux−RAID mailing list archive: http://www.linuxhq.com/lnxlists/.
- Linux Software RAID Home Page: http://luthien.nuclecu.unam.mx/~miguel/raid.
- Linux Software RAID tools: ftp://linux.kernel.org/pub/linux/daemons/raid/.
- How to setting up linear/stripped Software RAID: http://www.ssc.com/lg/issue17/raid.html.
- Bootable RAID mini−HOWTO: ftp://ftp.bizsystems.com/pub/raid/bootable−raid.
- Root RAID HOWTO: ftp://ftp.bizsystems.com/pub/raid/Root−RAID−HOWTO.
- Linux RAID−Geschichten: http://www.infodrom.north.de/~joey/Linux/raid/.

6. **Q**: Who do I blame for this document?

A: Linas Vepstas slapped this thing together. However, most of the information, and some of the words were supplied by

- Bradley Ward Allen < ulmo@Q.Net>
- Luca Berra < bluca@comedia.it>
- Brian Candler < B.Candler@pobox.com>
- Bohumil Chalupa < bochal@apollo.karlov.mff.cuni.cz>
- Rob Hagopian < hagopiar@vu.union.edu>
- Anton Hristozov < anton@intransco.com>
- Miguel de Icaza < miguel@luthien.nuclecu.unam.mx>
- Marco Meloni < tonno@stud.unipg.it>
- Ingo Molnar < mingo@pc7537.hil.siemens.at>
- Alvin Oga < alvin@planet.fef.com>
- Gadi Oxman < gadio@netvision.net.il>
- Vaughan Pratt < pratt@cs.Stanford.EDU>
- Steven A. Reisman < sar@pressenter.com>
- Michael Robinton < michael@bzs.org>
- Martin Schulze < joey@finlandia.infodrom.north.de>
- Geoff Thompson < geofft@cs.waikato.ac.nz>
- Edward Welbon < welbon@bga.com>
- Rod Wilkens < rwilkens@border.net>
- Johan Wiltink < j.m.wiltink@pi.net>
- Leonard N. Zubkoff < lnz@dandelion.com>
- Marc ZYNGIER < zyngier@ufr−info−p7.ibp.fr>

Copyrights

Thanks all for being there!

2.Understanding RAID

1. **Q**: What is RAID? Why would I ever use it?

   A: RAID is a way of combining multiple disk drives into a single entity to improve performance and/or reliability. There are a variety of different types and implementations of RAID, each with its own advantages and disadvantages. For example, by putting a copy of the same data on two disks (called **disk mirroring**, or RAID level 1), read performance can be improved by reading alternately from each disk in the mirror. On average, each disk is less busy, as it is handling only 1/2 the reads (for two disks), or 1/3 (for three disks), etc. In addition, a mirror can improve reliability: if one disk fails, the other disk(s) have a copy of the data. Different ways of combining the disks into one, referred to as **RAID levels**, can provide greater storage efficiency than simple mirroring, or can alter latency (access−time) performance, or throughput (transfer rate) performance, for reading or writing, while still retaining redundancy that is useful for guarding against failures.

   Although RAID can protect against disk failure, it does not protect against operator and administrator (human) error, or against loss due to programming bugs (possibly due to bugs in the RAID software itself). The net abounds with tragic tales of system administrators who have bungled a RAID installation, and have lost all of their data. RAID is not a substitute for frequent, regularly scheduled backup.

   RAID can be implemented in hardware, in the form of special disk controllers, or in software, as a kernel module that is layered in between the low−level disk driver, and the file system which sits above it. RAID hardware is always a "disk controller", that is, a device to which one can cable up the disk drives. Usually it comes in the form of an adapter card that will plug into a ISA/EISA/PCI/S−Bus/MicroChannel slot. However, some RAID controllers are in the form of a box that connects into the cable in between the usual system disk controller, and the disk drives. Small ones may fit into a drive bay; large ones may be built into a storage cabinet with its own drive bays and power supply. The latest RAID hardware used with the latest & fastest CPU will usually provide the best overall performance, although at a significant price. This is because most RAID controllers come with on−board DSP's and memory cache that can off−load a considerable amount of processing from the main CPU, as well as allow high transfer rates into the large controller cache. Old RAID hardware can act as a "de−accelerator" when used with newer CPU's: yesterday's fancy DSP and cache can act as a bottleneck, and it's performance is often beaten by pure−software RAID and new but otherwise plain, run−of−the−mill disk controllers. RAID hardware can offer an advantage over pure−software RAID, if it can makes use of disk−spindle synchronization and its knowledge of the disk−platter position with regard to the disk head, and the desired disk−block. However, most modern (low−cost) disk drives do not offer this information and level of control anyway, and thus, most RAID hardware does not take advantage of it. RAID hardware is usually not compatible across different brands, makes and models: if a RAID controller fails, it must be replaced by another controller of the same type. As of this writing (June 1998), a broad variety of hardware controllers will operate under Linux; however, none of them currently come with configuration and management utilities that run under Linux.

Software−RAID is a set of kernel modules, together with management utilities that implement RAID purely in software, and require no extraordinary hardware. The Linux RAID subsystem is implemented as a layer in the kernel that sits above the low−level disk drivers (for IDE, SCSI and Paraport drives), and the block−device interface. The filesystem, be it ext2fs, DOS−FAT, or other, sits above the block−device interface. Software−RAID, by its very software nature, tends to be more flexible than a hardware solution. The downside is that it of course requires more CPU cycles and power to run well than a comparable hardware system. Of course, the cost can't be beat. Software RAID has one further important distinguishing feature: it operates on a partition−by−partition basis, where a number of individual disk partitions are ganged together to create a RAID partition. This is in contrast to most hardware RAID solutions, which gang together entire disk drives into an array. With hardware, the fact that there is a RAID array is transparent to the operating system, which tends to simplify management. With software, there are far more configuration options and choices, tending to complicate matters.

As of this writing (June 1998), the administration of RAID under Linux is far from trivial, and is best attempted by experienced system administrators. The theory of operation is complex. The system tools require modification to startup scripts. And recovery from disk failure is non−trivial, and prone to human error. RAID is not for the novice, and any benefits it may bring to reliability and performance can be easily outweighed by the extra complexity. Indeed, modern disk drives are incredibly reliable and modern CPU's and controllers are quite powerful. You might more easily obtain the desired reliability and performance levels by purchasing higher−quality and/or faster hardware.

2. **Q**: What are RAID levels? Why so many? What distinguishes them?

   A: The different RAID levels have different performance, redundancy, storage capacity, reliability and cost characteristics. Most, but not all levels of RAID offer redundancy against disk failure. Of those that offer redundancy, RAID−1 and RAID−5 are the most popular. RAID−1 offers better performance, while RAID−5 provides for more efficient use of the available storage space. However, tuning for performance is an entirely different matter, as performance depends strongly on a large variety of factors, from the type of application, to the sizes of stripes, blocks, and files. The more difficult aspects of performance tuning are deferred to a later section of this HOWTO.

   The following describes the different RAID levels in the context of the Linux software RAID implementation.

   ◊ **RAID−linear** is a simple concatenation of partitions to create a larger virtual partition. It is handy if you have a number small drives, and wish to create a single, large partition. This concatenation offers no redundancy, and in fact decreases the overall reliability: if any one disk fails, the combined partition will fail.

◊ **RAID−1** is also referred to as "mirroring". Two (or more) partitions, all of the same size, each store an exact copy of all data, disk−block by disk−block. Mirroring gives strong protection against disk failure: if one disk fails, there is another with the an exact copy of the same data. Mirroring can also help improve performance in I/O−laden systems, as read requests can be divided up between several disks. Unfortunately, mirroring is also the least efficient in terms of storage: two mirrored partitions can store no more data than a single partition.

◊ **Striping** is the underlying concept behind all of the other RAID levels. A stripe is a contiguous sequence of disk blocks. A stripe may be as short as a single disk block, or may consist of thousands. The RAID drivers split up their component disk partitions into stripes; the different RAID levels differ in how they organize the stripes, and what data they put in them. The interplay between the size of the stripes, the typical size of files in the file system, and their location on the disk is what determines the overall performance of the RAID subsystem.

◊ **RAID−0** is much like RAID−linear, except that the component partitions are divided into stripes and then interleaved. Like RAID−linear, the result is a single larger virtual partition. Also like RAID−linear, it offers no redundancy, and therefore decreases overall reliability: a single disk failure will knock out the whole thing. RAID−0 is often claimed to improve performance over the simpler RAID−linear. However, this may or may not be true, depending on the characteristics to the file system, the typical size of the file as compared to the size of the stripe, and the type of workload. The `ext2fs` file system already scatters files throughout a partition, in an effort to minimize fragmentation. Thus, at the simplest level, any given access may go to one of several disks, and thus, the interleaving of stripes across multiple disks offers no apparent additional advantage. However, there are performance differences, and they are data, workload, and stripe−size dependent.

◊ **RAID−4** interleaves stripes like RAID−0, but it requires an additional partition to store parity information. The parity is used to offer redundancy: if any one of the disks fail, the data on the remaining disks can be used to reconstruct the data that was on the failed disk. Given N data disks, and one parity disk, the parity stripe is computed by taking one stripe from each of the data disks, and XOR'ing them together. Thus, the storage capacity of a an (N+1)−disk RAID−4 array is N, which is a lot better than mirroring (N+1) drives, and is almost as good as a RAID−0 setup for large N. Note that for N=1, where there is one data drive, and one parity drive, RAID−4 is a lot like mirroring, in that each of the two disks is a copy of each other.

However, RAID−4 does **NOT** offer the read−performance of mirroring, and offers considerably degraded write performance. In brief, this is because updating the parity requires a read of the old parity, before the new parity can be calculated and written out. In an environment with lots of writes, the parity disk can become a bottleneck, as each write must access the parity disk.

◊ **RAID−5** avoids the write−bottleneck of RAID−4 by alternately storing the parity stripe on each of the drives. However, write performance is still not as good as for mirroring, as the parity stripe must still be read and XOR'ed before it is written. Read performance is also not as good as it is for mirroring, as, after all, there is only one copy of the data, not two or more. RAID−5's principle advantage over mirroring is that it offers redundancy and protection against single−drive failure, while offering far more storage capacity when used with three or more drives.

◊ **RAID−2 and RAID−3** are seldom used anymore, and to some degree are have been made obsolete by modern disk technology. RAID−2 is similar to RAID−4, but stores ECC information instead of parity. Since all modern disk drives incorporate ECC under the covers, this offers little additional protection. RAID−2 can offer greater data consistency if power is lost during a write; however, battery backup and a clean shutdown can offer the same benefits. RAID−3 is similar to RAID−4, except that it uses the smallest possible stripe size. As a result, any given read will involve all disks, making overlapping I/O requests difficult/impossible. In order to avoid delay due to rotational latency, RAID−3 requires that all disk drive spindles be synchronized. Most modern disk drives lack spindle−synchronization ability, or, if capable of it, lack the needed connectors, cables, and manufacturer documentation. Neither RAID−2 nor RAID−3 are supported by the Linux Software−RAID drivers.

◊ **Other RAID levels** have been defined by various researchers and vendors. Many of these represent the layering of one type of raid on top of another. Some require special hardware, and others are protected by patent. There is no commonly accepted naming scheme for these other levels. Sometime the advantages of these other systems are minor, or at least not apparent until the system is highly stressed. Except for the layering of RAID−1 over RAID−0/linear, Linux Software RAID does not support any of the other variations.

3.Setup & Installation Considerations

1. **Q**: What is the best way to configure Software RAID?

   A: I keep rediscovering that file−system planning is one of the more difficult Unix configuration tasks. To answer your question, I can describe what we did. We planned the following setup:

   ◊ two EIDE disks, 2.1.gig each.

```
disk partition mount pt.  size    device
  1      1       /        300M   /dev/hda1
  1      2       swap      64M   /dev/hda2
  1      3       /home    800M   /dev/hda3
  1      4       /var     900M   /dev/hda4

  2      1       /root    300M   /dev/hdc1
  2      2       swap      64M   /dev/hdc2
  2      3       /home    800M   /dev/hdc3
  2      4       /var     900M   /dev/hdc4
```

2. Each disk is on a separate controller (& ribbon cable). The theory is that a controller failure and/or ribbon failure won't disable both disks. Also, we might possibly get a performance boost from parallel operations over two controllers/cables.
3. Install the Linux kernel on the root (`/`) partition `/dev/hda1`. Mark this partition as bootable.
4. `/dev/hdc1` will contain a ``cold'' copy of `/dev/hda1`. This is NOT a raid copy, just a plain old copy−copy. It's there just in case the first disk fails; we can use a rescue disk, mark `/dev/hdc1` as bootable, and use that to keep going without having to reinstall the system. You may even want to put `/dev/hdc1`'s copy of the kernel into LILO to simplify booting in case of failure. The theory here is that in case of severe failure, I can still boot the system without worrying about raid superblock−corruption or other raid failure modes & gotchas that I don't understand.
5. `/dev/hda3` and `/dev/hdc3` will be mirrors `/dev/md0`.
6. `/dev/hda4` and `/dev/hdc4` will be mirrors `/dev/md1`.
7. we picked `/var` and `/home` to be mirrored, and in separate partitions, using the following logic:

   ♦ `/` (the root partition) will contain relatively static, non−changing data: for all practical purposes, it will be read−only without actually being marked & mounted read−only.
   ♦ `/home` will contain "slowly" changing data.
   ♦ `/var` will contain rapidly changing data, including mail spools, database contents and web server logs.

   The idea behind using multiple, distinct partitions is that **if**, for some bizarre reason, whether it is human error, power loss, or an operating system gone wild, corruption is limited to one partition. In one typical case, power is lost while the system is writing to disk. This will almost certainly lead to a corrupted filesystem, which will be repaired by `fsck` during the next boot. Although `fsck` does it's best to make the repairs without creating additional damage during those repairs, it can be comforting to know that any such damage has been limited to one partition. In another typical case, the sysadmin makes a mistake during rescue operations, leading to erased or destroyed data. Partitions can help limit the repercussions of the operator's errors.

8. Other reasonable choices for partitions might be `/usr` or `/opt`. In fact, `/opt` and `/home` make great choices for RAID−5 partitions, if we had more disks. A word of caution: **DO NOT** put `/usr` in a RAID−5 partition. If a serious fault occurs, you may find that you cannot mount `/usr`, and that you want some of the tools on it (e.g. the networking tools, or the compiler.) With RAID−1, if a fault has occurred, and you can't get RAID to work, you can at least mount one of the two mirrors. You can't do this with any of the other RAID levels (RAID−5, striping, or linear append).

   So, to complete the answer to the question:

   - install the OS on disk 1, partition 1. do NOT mount any of the other partitions.
   - install RAID per instructions.
   - configure `md0` and `md1`.
   - convince yourself that you know what to do in case of a disk failure! Discover sysadmin mistakes now, and not during an actual crisis. Experiment! (we turned off power during disk activity this proved to be ugly but informative).
   - do some ugly mount/copy/unmount/rename/reboot scheme to move `/var` over to the `/dev/md1`. Done carefully, this is not dangerous.
   - enjoy!
9. **Q**: What is the difference between the `mdadd`, `mdrun`, *etc.* commands, and the `raidadd`, `raidrun` commands?

   A: The names of the tools have changed as of the 0.5 release of the raidtools package. The `md` naming convention was used in the 0.43 and older versions, while `raid` is used in 0.5 and newer versions.

10. **Q**: I want to run RAID−linear/RAID−0 in the stock 2.0.34 kernel. I don't want to apply the raid patches, since these are not needed for RAID−0/linear. Where can I get the raid−tools to manage this?

    A: This is a tough question, indeed, as the newest raid tools package needs to have the RAID−1,4,5 kernel patches installed in order to compile. I am not aware of any pre−compiled, binary version of the raid tools that is available at this time. However, experiments show that the raid−tools binaries, when compiled against kernel 2.1.100, seem to work just fine in creating a RAID−0/linear partition under 2.0.34. A brave soul has asked for these, and I've **temporarily** placed the binaries mdadd, mdcreate, etc. at http://linas.org/linux/Software−RAID/ You must get the man pages, etc. from the usual raid−tools package.

11. **Q**: Can I strip/mirror the root partition (`/`)? Why can't I boot Linux directly from the `md` disks?

    A: Both LILO and Loadlin need an non−stripped/mirrored partition to read the kernel image from. If you want to strip/mirror the root partition (`/`), then you'll want to create an unstriped/mirrored partition to hold the kernel(s). Typically, this partition is named `/boot`. Then you either use the initial ramdisk support (initrd), or patches from Harald Hoyer < HarryH@Royal.Net> that allow a stripped partition to be used as the root device. (These patches are now a standard part of recent 2.1.x kernels)

    There are several approaches that can be used. One approach is documented in detail

in the Bootable RAID mini−HOWTO:
ftp://ftp.bizsystems.com/pub/raid/bootable−raid.

Alternately, use `mkinitrd` to build the ramdisk image, see below.

Edward Welbon < welbon@bga.com> writes:

◊ ... all that is needed is a script to manage the boot setup. To mount an `md` filesystem as root, the main thing is to build an initial file system image that has the needed modules and md tools to start `md`. I have a simple script that does this.
◊ For boot media, I have a small **cheap** SCSI disk (170MB I got it used for $20). This disk runs on a AHA1452, but it could just as well be an inexpensive IDE disk on the native IDE. The disk need not be very fast since it is mainly for boot.
◊ This disk has a small file system which contains the kernel and the file system image for `initrd`. The initial file system image has just enough stuff to allow me to load the raid SCSI device driver module and start the raid partition that will become root. I then do an

```
echo 0x900 > /proc/sys/kernel/real-root-dev
```

(`0x900` is for `/dev/md0`) and exit `linuxrc`. The boot proceeds normally from there.

- I have built most support as a module except for the AHA1452 driver that brings in the `initrd` filesystem. So I have a fairly small kernel. The method is perfectly reliable, I have been doing this since before 2.1.26 and have never had a problem that I could not easily recover from. The file systems even survived several 2.1.4[45] hard crashes with no real problems.
- At one time I had partitioned the raid disks so that the initial cylinders of the first raid disk held the kernel and the initial cylinders of the second raid disk hold the initial file system image, instead I made the initial cylinders of the raid disks swap since they are the fastest cylinders (why waste them on boot?).
- The nice thing about having an inexpensive device dedicated to boot is that it is easy to boot from and can also serve as a rescue disk if necessary. If you are interested, you can take a look at the script that builds my initial ram disk image and then runs `LILO`.

```
http://www.realtime.net/~welbon/initrd.md.tar.gz
```

It is current enough to show the picture. It isn't especially pretty and it could certainly build a much smaller filesystem image for the initial ram disk. It would be easy to a make it more efficient. But it uses `LILO` as is. If you make any improvements, please forward a copy to me. 8−)

12. **Q**: I have heard that I can run mirroring over striping. Is this true? Can I run mirroring over the loopback device?

A: Yes, but not the reverse. That is, you can put a stripe over several disks, and then build a mirror on top of this. However, striping cannot be put on top of mirroring.

A brief technical explanation is that the linear and stripe personalities use the `ll_rw_blk` routine for access. The `ll_rw_blk` routine maps disk devices and sectors, not blocks. Block devices can be layered one on top of the other; but devices that do raw, low−level disk accesses, such as `ll_rw_blk`, cannot.

Currently (November 1997) RAID cannot be run over the loopback devices, although this should be fixed shortly.

13. **Q**: I have two small disks and three larger disks. Can I concatenate the two smaller disks with RAID−0, and then create a RAID−5 out of that and the larger disks?

    A: Currently (November 1997), for a RAID−5 array, no. Currently, one can do this only for a RAID−1 on top of the concatenated drives.

14. **Q**: What is the difference between RAID−1 and RAID−5 for a two−disk configuration (i.e. the difference between a RAID−1 array built out of two disks, and a RAID−5 array built out of two disks)?

    A: There is no difference in storage capacity. Nor can disks be added to either array to increase capacity (see the question below for details).

    RAID−1 offers a performance advantage for reads: the RAID−1 driver uses distributed−read technology to simultaneously read two sectors, one from each drive, thus doubling read performance.

    The RAID−5 driver, although it contains many optimizations, does not currently (September 1997) realize that the parity disk is actually a mirrored copy of the data disk. Thus, it serializes data reads.

15. **Q**: How can I guard against a two−disk failure?

    A: Some of the RAID algorithms do guard against multiple disk failures, but these are not currently implemented for Linux. However, the Linux Software RAID can guard against multiple disk failures by layering an array on top of an array. For example, nine disks can be used to create three raid−5 arrays. Then these three arrays can in turn be hooked together into a single RAID−5 array on top. In fact, this kind of a configuration will guard against a three−disk failure. Note that a large amount of disk space is "wasted" on the redundancy information.

```
For an NxN raid−5 array,
N=3, 5 out of 9 disks are used for parity (=55%)
N=4, 7 out of 16 disks
N=5, 9 out of 25 disks
...
N=9, 17 out of 81 disks (=~20%)
```

In general, an MxN array will use M+N−1 disks for parity. The least amount of space is "wasted" when M=N.

Another alternative is to create a RAID−1 array with three disks. Note that since all three disks contain identical data, that 2/3's of the space is "wasted".

16. **Q**: I'd like to understand how it'd be possible to have something like `fsck`: if the partition hasn't been cleanly unmounted, `fsck` runs and fixes the filesystem by itself more than 90% of the time. Since the machine is capable of fixing it by itself with `ckraid --fix`, why not make it automatic?

    A: This can be done by adding lines like the following to `/etc/rc.d/rc.sysinit`:

    ```
    mdadd /dev/md0 /dev/hda1 /dev/hdc1 || {
        ckraid --fix /etc/raid.usr.conf
        mdadd /dev/md0 /dev/hda1 /dev/hdc1
    }
    ```

    or

    ```
    mdrun −p1 /dev/md0
    if [ $? −gt 0 ] ; then
            ckraid −−fix /etc/raid1.conf
            mdrun −p1 /dev/md0
    fi
    ```

    Before presenting a more complete and reliable script, lets review the theory of operation. Gadi Oxman writes: In an unclean shutdown, Linux might be in one of the following states:

    ◊ The in−memory disk cache was in sync with the RAID set when the unclean shutdown occurred; no data was lost.
    ◊ The in−memory disk cache was newer than the RAID set contents when the crash occurred; this results in a corrupted filesystem and potentially in data loss. This state can be further divided to the following two states:

        · Linux was writing data when the unclean shutdown occurred.
        · Linux was not writing data when the crash occurred.

    Suppose we were using a RAID−1 array. In (2a), it might happen that before the crash, a small number of data blocks were successfully written only to some of the mirrors, so that on the next reboot, the mirrors will no longer contain the same data. If we were to ignore the mirror differences, the raidtools−0.36.3 read−balancing code might choose to read the above data blocks from any of the mirrors, which will result in inconsistent behavior (for example, the output of `e2fsck −n /dev/md0` can differ from run to run).

    Since RAID doesn't protect against unclean shutdowns, usually there isn't any "obviously correct" way to fix the mirror differences and the filesystem corruption.

For example, by default `ckraid --fix` will choose the first operational mirror and update the other mirrors with its contents. However, depending on the exact timing at the crash, the data on another mirror might be more recent, and we might want to use it as the source mirror instead, or perhaps use another method for recovery.

The following script provides one of the more robust boot−up sequences. In particular, it guards against long, repeated `ckraid`'s in the presence of uncooperative disks, controllers, or controller device drivers. Modify it to reflect your config, and copy it to `rc.raid.init`. Then invoke `rc.raid.init` after the root partition has been fsck'ed and mounted rw, but before the remaining partitions are fsck'ed. Make sure the current directory is in the search path.

```
mdadd /dev/md0 /dev/hda1 /dev/hdc1 || {
    rm -f /fastboot             # force an fsck to occur
    ckraid --fix /etc/raid.usr.conf
    mdadd /dev/md0 /dev/hda1 /dev/hdc1
}
# if a crash occurs later in the boot process,
# we at least want to leave this md in a clean state.
/sbin/mdstop /dev/md0

mdadd /dev/md1 /dev/hda2 /dev/hdc2 || {
    rm -f /fastboot             # force an fsck to occur
    ckraid --fix /etc/raid.home.conf
    mdadd /dev/md1 /dev/hda2 /dev/hdc2
}
# if a crash occurs later in the boot process,
# we at least want to leave this md in a clean state.
/sbin/mdstop /dev/md1

mdadd /dev/md0 /dev/hda1 /dev/hdc1
mdrun -p1 /dev/md0
if [ $? -gt 0 ] ; then
    rm -f /fastboot             # force an fsck to occur
    ckraid --fix /etc/raid.usr.conf
    mdrun -p1 /dev/md0
fi
# if a crash occurs later in the boot process,
# we at least want to leave this md in a clean state.
/sbin/mdstop /dev/md0

mdadd /dev/md1 /dev/hda2 /dev/hdc2
mdrun -p1 /dev/md1
if [ $? -gt 0 ] ; then
    rm -f /fastboot             # force an fsck to occur
    ckraid --fix /etc/raid.home.conf
    mdrun -p1 /dev/md1
fi
# if a crash occurs later in the boot process,
# we at least want to leave this md in a clean state.
/sbin/mdstop /dev/md1

# OK, just blast through the md commands now.  If there were
# errors, the above checks should have fixed things up.
/sbin/mdadd /dev/md0 /dev/hda1 /dev/hdc1
/sbin/mdrun -p1 /dev/md0

/sbin/mdadd /dev/md12 /dev/hda2 /dev/hdc2
```

```
/sbin/mdrun −p1 /dev/md1
```

In addition to the above, you'll want to create a `rc.raid.halt` which should look like the following:

```
/sbin/mdstop /dev/md0
/sbin/mdstop /dev/md1
```

Be sure to modify both `rc.sysinit` and `init.d/halt` to include this everywhere that filesystems get unmounted before a halt/reboot. (Note that `rc.sysinit` unmounts and reboots if `fsck` returned with an error.)

17. **Q**: Can I set up one−half of a RAID−1 mirror with the one disk I have now, and then later get the other disk and just drop it in?

    A: With the current tools, no, not in any easy way. In particular, you cannot just copy the contents of one disk onto another, and then pair them up. This is because the RAID drivers use glob of space at the end of the partition to store the superblock. This decreases the amount of space available to the file system slightly; if you just naively try to force a RAID−1 arrangement onto a partition with an existing filesystem, the raid superblock will overwrite a portion of the file system and mangle data. Since the ext2fs filesystem scatters files randomly throughput the partition (in order to avoid fragmentation), there is a very good chance that some file will land at the very end of a partition long before the disk is full.

    If you are clever, I suppose you can calculate how much room the RAID superblock will need, and make your filesystem slightly smaller, leaving room for it when you add it later. But then, if you are this clever, you should also be able to modify the tools to do this automatically for you. (The tools are not terribly complex).

    Note:A careful reader has pointed out that the following trick may work; I have not tried or verified this: Do the `mkraid` with `/dev/null` as one of the devices. Then `mdadd −r` with only the single, true disk (do not mdadd `/dev/null`). The `mkraid` should have successfully built the raid array, while the mdadd step just forces the system to run in "degraded" mode, as if one of the disks had failed.

4.Error Recovery

1. **Q**: I have a RAID−1 (mirroring) setup, and lost power while there was disk activity. Now what do I do?

   A: The redundancy of RAID levels is designed to protect against a **disk** failure, not against a **power** failure. There are several ways to recover from this situation.

   ◊ Method (1): Use the raid tools. These can be used to sync the raid arrays.

They do not fix file−system damage; after the raid arrays are sync'ed, then the file−system still has to be fixed with fsck. Raid arrays can be checked with `ckraid /etc/raid1.conf` (for RAID−1, else, `/etc/raid5.conf`, etc.) Calling `ckraid /etc/raid1.conf --fix` will pick one of the disks in the array (usually the first), and use that as the master copy, and copy its blocks to the others in the mirror. To designate which of the disks should be used as the master, you can use the `--force-source` flag: for example, `ckraid /etc/raid1.conf --fix --force-source /dev/hdc3` The ckraid command can be safely run without the `--fix` option to verify the inactive RAID array without making any changes. When you are comfortable with the proposed changes, supply the `--fix` option.

◊ Method (2): Paranoid, time−consuming, not much better than the first way. Lets assume a two−disk RAID−1 array, consisting of partitions `/dev/hda3` and `/dev/hdc3`. You can try the following:

1. `fsck /dev/hda3`
2. `fsck /dev/hdc3`
3. decide which of the two partitions had fewer errors, or were more easily recovered, or recovered the data that you wanted. Pick one, either one, to be your new ``master'' copy. Say you picked `/dev/hdc3`.
4. `dd if=/dev/hdc3 of=/dev/hda3`
5. `mkraid raid1.conf -f --only-superblock`

Instead of the last two steps, you can instead run `ckraid /etc/raid1.conf --fix --force-source /dev/hdc3` which should be a bit faster.

◊ Method (3): Lazy man's version of above. If you don't want to wait for long fsck's to complete, it is perfectly fine to skip the first three steps above, and move directly to the last two steps. Just be sure to run `fsck /dev/md0` after you are done. Method (3) is actually just method (1) in disguise.

In any case, the above steps will only sync up the raid arrays. The file system probably needs fixing as well: for this, fsck needs to be run on the active, unmounted md device.

With a three−disk RAID−1 array, there are more possibilities, such as using two disks to "vote" a majority answer. Tools to automate this do not currently (September 97) exist.

2. **Q**: I have a RAID−4 or a RAID−5 (parity) setup, and lost power while there was disk activity. Now what do I do?

A: The redundancy of RAID levels is designed to protect against a **disk** failure, not against a **power** failure. Since the disks in a RAID−4 or RAID−5 array do not contain a file system that fsck can read, there are fewer repair options. You cannot use fsck to do preliminary checking and/or repair; you must use ckraid first.

The `ckraid` command can be safely run without the `--fix` option to verify the inactive RAID array without making any changes. When you are comfortable with the proposed changes, supply the `--fix` option.

If you wish, you can try designating one of the disks as a "failed disk". Do this with the `--suggest-failed-disk-mask` flag.

Only one bit should be set in the flag: RAID−5 cannot recover two failed disks. The mask is a binary bit mask: thus:

```
0x1 == first disk
0x2 == second disk
0x4 == third disk
0x8 == fourth disk, etc.
```

Alternately, you can choose to modify the parity sectors, by using the `--suggest-fix-parity` flag. This will recompute the parity from the other sectors.

The flags `--suggest-failed-dsk-mask` and `--suggest-fix-parity` can be safely used for verification. No changes are made if the `--fix` flag is not specified. Thus, you can experiment with different possible repair schemes.

3. **Q**: My RAID−1 device, `/dev/md0` consists of two hard drive partitions: `/dev/hda3` and `/dev/hdc3`. Recently, the disk with `/dev/hdc3` failed, and was replaced with a new disk. My best friend, who doesn't understand RAID, said that the correct thing to do now is to "`dd if=/dev/hda3 of=/dev/hdc3`". I tried this, but things still don't work.

   A: You should keep your best friend away from you computer. Fortunately, no serious damage has been done. You can recover from this by running:

   ```
   mkraid raid1.conf -f --only-superblock
   ```

   By using `dd`, two identical copies of the partition were created. This is almost correct, except that the RAID−1 kernel extension expects the RAID superblocks to be different. Thus, when you try to reactivate RAID, the software will notice the problem, and deactivate one of the two partitions. By re−creating the superblock, you should have a fully usable system.
4. **Q**: My version of `mkraid` doesn't have a `--only-superblock` flag. What do I do?

   A: The newer tools drop support for this flag, replacing it with the `--force-resync` flag. It has been reported that the following sequence appears to work with the latest tools and software:

   ```
     umount /web (where /dev/md0 was mounted on)
   ```

```
raidstop /dev/md0
mkraid /dev/md0 --force-resync --really-force
raidstart /dev/md0
```

After doing this, a `cat /proc/mdstat` should report `resync in progress`, and one should be able to `mount /dev/md0` at this point.

5. **Q**: My RAID−1 device, `/dev/md0` consists of two hard drive partitions: `/dev/hda3` and `/dev/hdc3`. My best (girl?)friend, who doesn't understand RAID, ran `fsck` on `/dev/hda3` while I wasn't looking, and now the RAID won't work. What should I do?

   A: You should re−examine your concept of ``best friend''. In general, `fsck` should never be run on the individual partitions that compose a RAID array. Assuming that neither of the partitions are/were heavily damaged, no data loss has occurred, and the RAID−1 device can be recovered as follows:

   1. make a backup of the file system on `/dev/hda3`
   2. `dd if=/dev/hda3 of=/dev/hdc3`
   3. `mkraid raid1.conf -f --only-superblock`

   This should leave you with a working disk mirror.

6. **Q**: Why does the above work as a recovery procedure?

   A: Because each of the component partitions in a RAID−1 mirror is a perfectly valid copy of the file system. In a pinch, mirroring can be disabled, and one of the partitions can be mounted and safely run as an ordinary, non−RAID file system. When you are ready to restart using RAID−1, then unmount the partition, and follow the above instructions to restore the mirror. Note that the above works ONLY for RAID−1, and not for any of the other levels.

   It may make you feel more comfortable to reverse the direction of the copy above: copy **from** the disk that was untouched **to** the one that was. Just be sure to fsck the final md.

7. **Q**: I am confused by the above questions, but am not yet bailing out. Is it safe to run `fsck /dev/md0` ?

   A: Yes, it is safe to run `fsck` on the `md` devices. In fact, this is the **only** safe place to run `fsck`.

8. **Q**: If a disk is slowly failing, will it be obvious which one it is? I am concerned that it won't be, and this confusion could lead to some dangerous decisions by a sysadmin.

   A: Once a disk fails, an error code will be returned from the low level driver to the RAID driver. The RAID driver will mark it as ``bad'' in the RAID superblocks of the ``good'' disks (so we will later know which mirrors are good and which aren't), and continue RAID operation on the remaining operational mirrors.

   This, of course, assumes that the disk and the low level driver can detect a read/write error, and will not silently corrupt data, for example. This is true of current drives (error detection schemes are being used internally), and is the basis of RAID operation.

9. **Q**: What about hot−repair?

   A: Work is underway to complete ``hot reconstruction''. With this feature, one can add several ``spare'' disks to the RAID set (be it level 1 or 4/5), and once a disk fails, it will be reconstructed on one of the spare disks in run time, without ever needing to shut down the array.

   However, to use this feature, the spare disk must have been declared at boot time, or it must be hot−added, which requires the use of special cabinets and connectors that allow a disk to be added while the electrical power is on.

   As of October 97, there is a beta version of MD that allows:

   ◊ RAID 1 and 5 reconstruction on spare drives
   ◊ RAID−5 parity reconstruction after an unclean shutdown
   ◊ spare disk to be hot−added to an already running RAID 1 or 4/5 array

   By default, automatic reconstruction is (Dec 97) currently disabled by default, due to the preliminary nature of this work. It can be enabled by changing the value of `SUPPORT_RECONSTRUCTION` in `include/linux/md.h`.

   If spare drives were configured into the array when it was created and kernel−based reconstruction is enabled, the spare drive will already contain a RAID superblock (written by `mkraid`), and the kernel will reconstruct its contents automatically (without needing the usual `mdstop`, replace drive, `ckraid`, `mdrun` steps).

   If you are not running automatic reconstruction, and have not configured a hot−spare disk, the procedure described by Gadi Oxman < gadio@netvision.net.il> is recommended:

   ◊ Currently, once the first disk is removed, the RAID set will be running in degraded mode. To restore full operation mode, you need to:

      · stop the array (`mdstop /dev/md0`)
      · replace the failed drive
      · run `ckraid raid.conf` to reconstruct its contents
      · run the array again (`mdadd`, `mdrun`).

      At this point, the array will be running with all the drives, and again protects against a failure of a single drive.

   Currently, it is not possible to assign single hot−spare disk to several arrays. Each array requires it's own hot−spare.

10. **Q**: I would like to have an audible alarm for ``you schmuck, one disk in the mirror is down'', so that

the novice sysadmin knows that there is a problem.

> **A**: The kernel is logging the event with a ``KERN_ALERT'' priority in syslog. There are several software packages that will monitor the syslog files, and beep the PC speaker, call a pager, send e−mail, etc. automatically.

11. **Q**: How do I run RAID−5 in degraded mode (with one disk failed, and not yet replaced)?

    > **A**: Gadi Oxman < gadio@netvision.net.il> writes: Normally, to run a RAID−5 set of n drives you have to:

```
mdadd /dev/md0 /dev/disk1 ... /dev/disk(n)
mdrun -p5 /dev/md0
```

    Even if one of the disks has failed, you still have to `mdadd` it as you would in a normal setup. (?? try using /dev/null in place of the failed disk ??? watch out) Then, The array will be active in degraded mode with (n − 1) drives. If ``mdrun'' fails, the kernel has noticed an error (for example, several faulty drives, or an unclean shutdown). Use ``dmesg'' to display the kernel error messages from ``mdrun''. If the raid−5 set is corrupted due to a power loss, rather than a disk crash, one can try to recover by creating a new RAID superblock:

```
mkraid -f --only-superblock raid5.conf
```

    A RAID array doesn't provide protection against a power failure or a kernel crash, and can't guarantee correct recovery. Rebuilding the superblock will simply cause the system to ignore the condition by marking all the drives as ``OK'', as if nothing happened.
12. **Q**: How does RAID−5 work when a disk fails?

    > **A**: The typical operating scenario is as follows:
    >
    > ◊ A RAID−5 array is active.
    > ◊ One drive fails while the array is active.
    > ◊ The drive firmware and the low−level Linux disk/controller drivers detect the failure and report an error code to the MD driver.
    > ◊ The MD driver continues to provide an error−free `/dev/md0` device to the higher levels of the kernel (with a performance degradation) by using the remaining operational drives.
    > ◊ The sysadmin can `umount /dev/md0` and `mdstop /dev/md0` as usual.
    > ◊ If the failed drive is not replaced, the sysadmin can still start the array in degraded mode as usual, by running `mdadd` and `mdrun`.
13. **Q**:

    > **A**:
14. **Q**: Why is there no question 13?

A: If you are concerned about RAID, High Availability, and UPS, then its probably a good idea to be superstitious as well. It can't hurt, can it?

15. **Q**: I just replaced a failed disk in a RAID−5 array. After rebuilding the array, `fsck` is reporting many, many errors. Is this normal?

    A: No. And, unless you ran fsck in "verify only; do not update" mode, its quite possible that you have corrupted your data. Unfortunately, a not−uncommon scenario is one of accidentally changing the disk order in a RAID−5 array, after replacing a hard drive. Although the RAID superblock stores the proper order, not all tools use this information. In particular, the current version of `ckraid` will use the information specified with the `−f` flag (typically, the file `/etc/raid5.conf`) instead of the data in the superblock. If the specified order is incorrect, then the replaced disk will be reconstructed incorrectly. The symptom of this kind of mistake seems to be heavy & numerous `fsck` errors.

    And, in case you are wondering, **yes**, someone lost **all** of their data by making this mistake. Making a tape backup of **all** data before reconfiguring a RAID array is **strongly recommended**.

16. **Q**: The QuickStart says that `mdstop` is just to make sure that the disks are sync'ed. Is this REALLY necessary? Isn't unmounting the file systems enough?

    A: The command `mdstop /dev/md0` will:

    ◊ mark it "clean". This allows us to detect unclean shutdowns, for example due to a power failure or a kernel crash.
    ◊ sync the array. This is less important after unmounting a filesystem, but is important if the `/dev/md0` is accessed directly rather than through a filesystem (for example, by `e2fsck`).

5.Troubleshooting Install Problems

1. **Q**: What is the current best known−stable patch for RAID in the 2.0.x series kernels?

   A: As of 18 Sept 1997, it is "2.0.30 + pre−9 2.0.31 + Werner Fink's swapping patch + the alpha RAID patch". As of November 1997, it is 2.0.31 + ... !?

2. **Q**: The RAID patches will not install cleanly for me. What's wrong?

   A: Make sure that `/usr/include/linux` is a symbolic link to `/usr/src/linux/include/linux`. Make sure that the new files `raid5.c`, etc. have been copied to their correct locations. Sometimes the patch command will not create new files. Try the `−f` flag on `patch`.

3. **Q**: While compiling raidtools 0.42, compilation stops trying to include <pthread.h> but it doesn't

exist in my system. How do I fix this?

A: raidtools−0.42 requires linuxthreads−0.6 from: ftp://ftp.inria.fr/INRIA/Projects/cristal/Xavier.Leroy Alternately, use glibc v2.0.

4. **Q**: I get the message: `mdrun −a /dev/md0: Invalid argument`

   A: Use `mkraid` to initialize the RAID set prior to the first use. `mkraid` ensures that the RAID array is initially in a consistent state by erasing the RAID partitions. In addition, `mkraid` will create the RAID superblocks.

5. **Q**: I get the message: `mdrun −a /dev/md0: Invalid argument` The setup was:

   - raid build as a kernel module
   - normal install procedure followed ... mdcreate, mdadd, etc.
   - `cat /proc/mdstat` shows

   ```
   Personalities :
   read_ahead not set
   md0 : inactive sda1 sdb1 6313482 blocks
   md1 : inactive
   md2 : inactive
   md3 : inactive
   ```

   - `mdrun −a` generates the error message `/dev/md0: Invalid argument`

   A: Try `lsmod` (or, alternately, `cat /proc/modules`) to see if the raid modules are loaded. If they are not, you can load them explicitly with the `modprobe raid1` or `modprobe raid5` command. Alternately, if you are using the autoloader, and expected `kerneld` to load them and it didn't this is probably because your loader is missing the info to load the modules. Edit `/etc/conf.modules` and add the following lines:

   ```
   alias md−personality−3 raid1
   alias md−personality−4 raid5
   ```

6. **Q**: While doing `mdadd −a` I get the error: `/dev/md0: No such file or directory`. Indeed, there seems to be no `/dev/md0` anywhere. Now what do I do?

   A: The raid−tools package will create these devices when you run `make install` as root. Alternately, you can do the following:

   ```
   cd /dev
   ./MAKEDEV md
   ```

7. **Q**: After creating a raid array on `/dev/md0`, I try to mount it and get the following error: `mount: wrong fs type, bad option, bad superblock on /dev/md0, or too many mounted file systems`. What's wrong?

   A: You need to create a file system on `/dev/md0` before you can mount it. Use `mke2fs`.

8. **Q**: Truxton Fulton wrote:

   > On my Linux 2.0.30 system, while doing a `mkraid` for a RAID−1 device, during the clearing of the two individual partitions, I got `"Cannot allocate free page"` errors appearing on the console, and `"Unable to handle kernel paging request at virtual address ..."` errors in the system log. At this time, the system became quite unusable, but it appears to recover after a while. The operation appears to have completed with no other errors, and I am successfully using my RAID−1 device. The errors are disconcerting though. Any ideas?

   > **A**: This was a well−known bug in the 2.0.30 kernels. It is fixed in the 2.0.31 kernel; alternately, fall back to 2.0.29.
9. **Q**: I'm not able to `mdrun` a RAID−1, RAID−4 or RAID−5 device. If I try to `mdrun` a `mdadd`'ed device I get the message "`invalid raid superblock magic`".

   > **A**: Make sure that you've run the `mkraid` part of the install procedure.
10. **Q**: When I access `/dev/md0`, the kernel spits out a lot of errors like `md0: device not running, giving up !` and `I/O error...`. I've successfully added my devices to the virtual device.

    > **A**: To be usable, the device must be running. Use `mdrun -px /dev/md0` where x is l for linear, 0 for RAID−0 or 1 for RAID−1, etc.
11. **Q**: I've created a linear md−dev with 2 devices. `cat /proc/mdstat` shows the total size of the device, but `df` only shows the size of the first physical device.

    > **A**: You must `mkfs` your new md−dev before using it the first time, so that the filesystem will cover the whole device.
12. **Q**: I've set up `/etc/mdtab` using mdcreate, I've `mdadd`'ed, `mdrun` and `fsck`'ed my two `/dev/mdX` partitions. Everything looks okay before a reboot. As soon as I reboot, I get an `fsck` error on both partitions: `fsck.ext2: Attempt to read block from filesystem resulted in short read while trying too open /dev/md0`. Why?! How do I fix it?!

    > **A**: During the boot process, the RAID partitions must be started before they can be `fsck`'ed. This must be done in one of the boot scripts. For some distributions, `fsck` is called from `/etc/rc.d/rc.S`, for others, it is called from `/etc/rc.d/rc.sysinit`. Change this file to `mdadd -ar` *before* `fsck -A` is executed. Better yet, it is suggested that `ckraid` be run if `mdadd` returns with an error. How do do this is discussed in greater detail in question 14 of the section "Error Recovery".
13. **Q**: I get the message `invalid raid superblock magic` while trying to run an array which consists of partitions which are bigger than 4GB.

    > **A**: This bug is now fixed. (September 97) Make sure you have the latest raid code.

14. **Q**: I get the message `Warning: could not write 8 blocks in inode table starting at 2097175` while trying to run mke2fs on a partition which is larger than 2GB.

    A: This seems to be a problem with `mke2fs` (November 97). A temporary work−around is to get the mke2fs code, and add `#undef HAVE_LLSEEK` to `e2fsprogs−1.10/lib/ext2fs/llseek.c` just before the first `#ifdef HAVE_LLSEEK` and recompile mke2fs.

15. **Q**: `ckraid` currently isn't able to read `/etc/mdtab`

    A: The RAID0/linear configuration file format used in `/etc/mdtab` is obsolete, although it will be supported for a while more. The current, up−to−date config files are currently named `/etc/raid1.conf`, etc.

16. **Q**: The personality modules (`raid1.o`) are not loaded automatically; they have to be manually modprobe'd before mdrun. How can this be fixed?

    A: To autoload the modules, we can add the following to `/etc/conf.modules`:

    ```
    alias md−personality−3 raid1
    alias md−personality−4 raid5
    ```

17. **Q**: I've `mdadd`'ed 13 devices, and now I'm trying to `mdrun −p5 /dev/md0` and get the message: `/dev/md0: Invalid argument`

    A: The default configuration for software RAID is 8 real devices. Edit `linux/md.h`, change `#define MAX_REAL=8` to a larger number, and rebuild the kernel.

18. **Q**: I can't make `md` work with partitions on our latest SPARCstation 5. I suspect that this has something to do with disk−labels.

    A: Sun disk−labels sit in the first 1K of a partition. For RAID−1, the Sun disk−label is not an issue since `ext2fs` will skip the label on every mirror. For other raid levels (0, linear and 4/5), this appears to be a problem; it has not yet (Dec 97) been addressed.

6.Supported Hardware & Software

1. **Q**: I have SCSI adapter brand XYZ (with or without several channels), and disk brand(s) PQR and LMN, will these work with md to create a linear/stripped/mirrored personality?

    A: Yes! Software RAID will work with any disk controller (IDE or SCSI) and any disks. The disks do not have to be identical, nor do the controllers. For example, a RAID mirror can be created with one half the mirror being a SCSI disk, and the other an IDE disk. The disks do not even have to be the same size. There are no restrictions on the mixing & matching of disks and controllers.

This is because Software RAID works with disk partitions, not with the raw disks themselves. The only recommendation is that for RAID levels 1 and 5, the disk partitions that are used as part of the same set be the same size. If the partitions used to make up the RAID 1 or 5 array are not the same size, then the excess space in the larger partitions is wasted (not used).

2. **Q**: I have a twin channel BT−952, and the box states that it supports hardware RAID 0, 1 and 0+1. I have made a RAID set with two drives, the card apparently recognizes them when it's doing it's BIOS startup routine. I've been reading in the driver source code, but found no reference to the hardware RAID support. Anybody out there working on that?

   A: The Mylex/BusLogic FlashPoint boards with RAIDPlus are actually software RAID, not hardware RAID at all. RAIDPlus is only supported on Windows 95 and Windows NT, not on Netware or any of the Unix platforms. Aside from booting and configuration, the RAID support is actually in the OS drivers.

   While in theory Linux support for RAIDPlus is possible, the implementation of RAID−0/1/4/5 in the Linux kernel is much more flexible and should have superior performance, so there's little reason to support RAIDPlus directly.

3. **Q**: I want to run RAID with an SMP box. Is RAID SMP−safe?

   A: "I think so" is the best answer available at the time I write this (April 98). A number of users report that they have been using RAID with SMP for nearly a year, without problems. However, as of April 98 (circa kernel 2.1.9x), the following problems have been noted on the mailing list:

   - Adaptec AIC7xxx SCSI drivers are not SMP safe (General note: Adaptec adapters have a long & lengthly history of problems & flakiness in general. Although they seem to be the most easily available, widespread and cheapest SCSI adapters, they should be avoided. After factoring for time lost, frustration, and corrupted data, Adaptec's will prove to be the costliest mistake you'll ever make. That said, if you have SMP problems with 2.1.88, try the patch ftp://ftp.bero−online.ml.org/pub/linux/aic7xxx−5.0.7−linux21.tar.gz I am not sure if this patch has been pulled into later 2.1.x kernels. For further info, take a look at the mail archives for March 98 at http://www.linuxhq.com/lnxlists/linux−raid/lr_9803_01/ As usual, due to the rapidly changing nature of the latest experimental 2.1.x kernels, the problems described in these mailing lists may or may not have been fixed by the time your read this. Caveat Emptor.)
   - IO−APIC with RAID−0 on SMP has been reported to crash in 2.1.90

7.Modifying an Existing Installation

1. **Q**: Are linear MD's expandable? Can a new hard−drive/partition be added, and the size of the existing file system expanded?

   A: Miguel de Icaza < miguel@luthien.nuclecu.unam.mx> writes:

   I changed the ext2fs code to be aware of multiple−devices instead of the regular one device per file system assumption.

   So, when you want to extend a file system, you run a utility program that makes the appropriate changes on the new device (your extra partition) and then you just tell the system to extend the fs using the specified device.

   You can extend a file system with new devices at system operation time, no need to bring the system down (and whenever I get some extra time, you will be able to remove devices from the ext2 volume set, again without even having to go to single−user mode or any hack like that).

   You can get the patch for 2.1.x kernel from my web page:

   `http://www.nuclecu.unam.mx/~miguel/ext2−volume`
2. **Q**: Can I add disks to a RAID−5 array?

   A: Currently, (September 1997) no, not without erasing all data. A conversion utility to allow this does not yet exist. The problem is that the actual structure and layout of a RAID−5 array depends on the number of disks in the array. Of course, one can add drives by backing up the array to tape, deleting all data, creating a new array, and restoring from tape.
3. **Q**: What would happen to my RAID1/RAID0 sets if I shift one of the drives from being `/dev/hdb` to `/dev/hdc`? Because of cabling/case size/stupidity issues, I had to make my RAID sets on the same IDE controller (`/dev/hda` and `/dev/hdb`). Now that I've fixed some stuff, I want to move `/dev/hdb` to `/dev/hdc`. What would happen if I just change the `/etc/mdtab` and `/etc/raid1.conf` files to reflect the new location?

   A: For RAID−0/linear, one must be careful to specify the drives in exactly the same order. Thus, in the above example, if the original config is

   ```
   mdadd /dev/md0 /dev/hda /dev/hdb
   ```

   Then the new config *must* be

```
mdadd /dev/md0 /dev/hda /dev/hdc
```

For RAID−1/4/5, the drive's "RAID number" is stored in its RAID superblock, and therefore the order in which the disks are specified is not important. RAID−0/linear does not have a superblock due to it's older design, and the desire to maintain backwards compatibility with this older design.

4. **Q**: Can I convert a two−disk RAID−1 mirror to a three−disk RAID−5 array?

   A: Yes. Michael at BizSystems has come up with a clever, sneaky way of doing this. However, like virtually all manipulations of RAID arrays once they have data on them, it is dangerous and prone to human error. **Make a backup before you start**.

```
I will make the following assumptions:
---------------------------------------------
disks
original: hda - hdc
raid1 partitions hda3 - hdc3
array name /dev/md0

new hda - hdc - hdd
raid5 partitions hda3 - hdc3 - hdd3
array name: /dev/md1

You must substitute the appropriate disk and partition numbers for
you system configuration. This will hold true for all config file
examples.
--------------------------------------------
DO A BACKUP BEFORE YOU DO ANYTHING
1) recompile kernel to include both raid1 and raid5
2) install new kernel and verify that raid personalities are present
3) disable the redundant partition on the raid 1 array. If this is a
 root mounted partition (mine was) you must be more careful.

 Reboot the kernel without starting raid devices or boot from rescue
 system ( raid tools must be available )

 start non-redundant raid1
mdadd -r -p1 /dev/md0 /dev/hda3

4) configure raid5 but with 'funny' config file, note that there is
  no hda3 entry and hdc3 is repeated. This is needed since the
  raid tools don't want you to do this.
-------------------------------
# raid-5 configuration
raiddev                 /dev/md1
raid-level              5
nr-raid-disks           3
chunk-size              32

# Parity placement algorithm
parity-algorithm        left-symmetric

# Spare disks for hot reconstruction
nr-spare-disks          0

device                  /dev/hdc3
raid-disk               0

device                  /dev/hdc3
```

```
raid-disk               1

device                  /dev/hdd3
raid-disk               2
---------------------------------------
 mkraid /etc/raid5.conf
5) activate the raid5 array in non-redundant mode

mdadd -r -p5 -c32k /dev/md1 /dev/hdc3 /dev/hdd3

6) make a file system on the array

mke2fs -b {blocksize} /dev/md1

recommended blocksize by some is 4096 rather than the default 1024.
this improves the memory utilization for the kernel raid routines and
matches the blocksize to the page size. I compromised and used 2048
since I have a relatively high number of small files on my system.

7) mount the two raid devices somewhere

mount -t ext2 /dev/md0 mnt0
mount -t ext2 /dev/md1 mnt1

8) move the data

cp -a mnt0 mnt1

9) verify that the data sets are identical
10) stop both arrays
11) correct the information for the raid5.conf file
  change /dev/md1 to /dev/md0
  change the first disk to read /dev/hda3

12) upgrade the new array to full redundant status
 (THIS DESTROYS REMAINING raid1 INFORMATION)

ckraid --fix /etc/raid5.conf
```

8.Performance, Tools & General Bone−headed Questions

1. **Q**: I've created a RAID−0 device on `/dev/sda2` and `/dev/sda3`. The device is a lot slower than a single partition. Isn't md a pile of junk?

   A: To have a RAID−0 device running a full speed, you must have partitions from different disks. Besides, putting the two halves of the mirror on the same disk fails to give you any protection whatsoever against disk failure.

2. **Q**: What's the use of having RAID−linear when RAID−0 will do the same thing, but provide higher performance?

   A: It's not obvious that RAID−0 will always provide better performance; in fact, in some cases, it could make things worse. The ext2fs file system scatters files all over a partition, and it attempts to keep all of the blocks of a file contiguous, basically in an attempt to prevent fragmentation. Thus, ext2fs behaves "as if" there were a (variable−sized) stripe per file. If there are several disks concatenated into a single RAID−linear, this will result files being statistically distributed on each of the disks. Thus, at least for ext2fs, RAID−linear will behave a lot like RAID−0 with large stripe sizes. Conversely, RAID−0 with small stripe sizes can cause excessive disk activity leading to severely degraded performance if several large files are accessed simultaneously.

In many cases, RAID−0 can be an obvious win. For example, imagine a large database file. Since ext2fs attempts to cluster together all of the blocks of a file, chances are good that it will end up on only one drive if RAID−linear is used, but will get chopped into lots of stripes if RAID−0 is used. Now imagine a number of (kernel) threads all trying to random access to this database. Under RAID−linear, all accesses would go to one disk, which would not be as efficient as the parallel accesses that RAID−0 entails.

3. **Q**: How does RAID−0 handle a situation where the different stripe partitions are different sizes? Are the stripes uniformly distributed?

   A: To understand this, lets look at an example with three partitions; one that is 50MB, one 90MB and one 125MB. Lets call D0 the 50MB disk, D1 the 90MB disk and D2 the 125MB disk. When you start the device, the driver calculates 'strip zones'. In this case, it finds 3 zones, defined like this:

```
Z0 : (D0/D1/D2) 3 x 50 = 150MB  total in this zone
Z1 : (D1/D2)  2 x 40 = 80MB total in this zone
Z2 : (D2) 125-50-40 = 35MB total in this zone.
```

   You can see that the total size of the zones is the size of the virtual device, but, depending on the zone, the striping is different. Z2 is rather inefficient, since there's only one disk. Since `ext2fs` and most other Unix file systems distribute files all over the disk, you have a 35/265 = 13% chance that a fill will end up on Z2, and not get any of the benefits of striping. (DOS tries to fill a disk from beginning to end, and thus, the oldest files would end up on Z0. However, this strategy leads to severe filesystem fragmentation, which is why no one besides DOS does it this way.)

4. **Q**: I have some Brand X hard disks and a Brand Y controller. and am considering using `md`. Does it significantly increase the throughput? Is the performance really noticeable?

   A: The answer depends on the configuration that you use.

   Linux MD RAID−0 and RAID−linear performance:

   If the system is heavily loaded with lots of I/O, statistically, some of it will go to one disk, and some to the others. Thus, performance will improve over a single large disk. The actual improvement depends a lot on the actual data, stripe sizes, and other factors. In a system with low I/O usage, the performance is equal to that of a single disk.

Linux MD RAID−1 (mirroring) read performance:

MD implements read balancing. That is, the RAID−1 code will alternate between each of the (two or more) disks in the mirror, making alternate reads to each. In a low−I/O situation, this won't change performance at all: you will have to wait for one disk to complete the read. But, with two disks in a high−I/O environment, this could as much as double the read performance, since reads can be issued to each of the disks in parallel. For N disks in the mirror, this could improve performance N−fold.

Linux MD RAID−1 (mirroring) write performance:

Must wait for the write to occur to all of the disks in the mirror. This is because a copy of the data must be written to each of the disks in the mirror. Thus, performance will be roughly equal to the write performance to a single disk.

Linux MD RAID−4/5 read performance:

Statistically, a given block can be on any one of a number of disk drives, and thus RAID−4/5 read performance is a lot like that for RAID−0. It will depend on the data, the stripe size, and the application. It will not be as good as the read performance of a mirrored array.

Linux MD RAID−4/5 write performance:

This will in general be considerably slower than that for a single disk. This is because the parity must be written out to one drive as well as the data to another. However, in order to compute the new parity, the old parity and the old data must be read first. The old data, new data and old parity must all be XOR'ed together to determine the new parity: this requires considerable CPU cycles in addition to the numerous disk accesses.

5. **Q**: What RAID configuration should I use for optimal performance?

   A: Is the goal to maximize throughput, or to minimize latency? There is no easy answer, as there are many factors that affect performance:

   ◊ operating system − will one process/thread, or many be performing disk access?
   ◊ application − is it accessing data in a sequential fashion, or random access?
   ◊ file system − clusters files or spreads them out (the ext2fs clusters together the blocks of a file, and spreads out files)
   ◊ disk driver − number of blocks to read ahead (this is a tunable parameter)
   ◊ CEC hardware − one drive controller, or many?
   ◊ hd controller − able to queue multiple requests or not? Does it provide a cache?
   ◊ hard drive − buffer cache memory size −− is it big enough to handle the

write sizes and rate you want?
◊ physical platters − blocks per cylinder −− accessing blocks on different cylinders will lead to seeks.

6. **Q**: What is the optimal RAID−5 configuration for performance?

   A: Since RAID−5 experiences an I/O load that is equally distributed across several drives, the best performance will be obtained when the RAID set is balanced by using identical drives, identical controllers, and the same (low) number of drives on each controller. Note, however, that using identical components will raise the probability of multiple simultaneous failures, for example due to a sudden jolt or drop, overheating, or a power surge during an electrical storm. Mixing brands and models helps reduce this risk.

7. **Q**: What is the optimal block size for a RAID−4/5 array?

   A: When using the current (November 1997) RAID−4/5 implementation, it is strongly recommended that the file system be created with `mke2fs -b 4096` instead of the default 1024 byte filesystem block size.

   This is because the current RAID−5 implementation allocates one 4K memory page per disk block; if a disk block were just 1K in size, then 75% of the memory which RAID−5 is allocating for pending I/O would not be used. If the disk block size matches the memory page size, then the driver can (potentially) use all of the page. Thus, for a filesystem with a 4096 block size as opposed to a 1024 byte block size, the RAID driver will potentially queue 4 times as much pending I/O to the low level drivers without allocating additional memory.

   Note: the above remarks do NOT apply to Software RAID−0/1/linear driver.

   Note: the statements about 4K memory page size apply to the Intel x86 architecture. The page size on Alpha, Sparc, and other CPUS are different; I believe they're 8K on Alpha/Sparc (????). Adjust the above figures accordingly.

   Note: if your file system has a lot of small files (files less than 10KBytes in size), a considerable fraction of the disk space might be wasted. This is because the file system allocates disk space in multiples of the block size. Allocating large blocks for small files clearly results in a waste of disk space: thus, you may want to stick to small block sizes, get a larger effective storage capacity, and not worry about the "wasted" memory due to the block−size/page−size mismatch.

   Note: most "typical" systems do not have that many small files. That is, although there might be thousands of small files, this would lead to only some 10 to 100MB wasted space, which is probably an acceptable tradeoff for performance on a multi−gigabyte disk.

   However, for news servers, there might be tens or hundreds of thousands of small files. In such cases, the smaller block size, and thus the improved storage capacity, may be more important than the more efficient I/O scheduling.

Note: there exists an experimental file system for Linux which packs small files and file chunks onto a single block. It apparently has some very positive performance implications when the average file size is much smaller than the block size.

Note: Future versions may implement schemes that obsolete the above discussion. However, this is difficult to implement, since dynamic run−time allocation can lead to dead−locks; the current implementation performs a static pre−allocation.

8. **Q**: How does the chunk size (stripe size) influence the speed of my RAID−0, RAID−4 or RAID−5 device?

   A: The chunk size is the amount of data contiguous on the virtual device that is also contiguous on the physical device. In this HOWTO, "chunk" and "stripe" refer to the same thing: what is commonly called the "stripe" in other RAID documentation is called the "chunk" in the MD man pages. Stripes or chunks apply only to RAID 0, 4 and 5, since stripes are not used in mirroring (RAID−1) and simple concatenation (RAID−linear). The stripe size affects both read and write latency (delay), throughput (bandwidth), and contention between independent operations (ability to simultaneously service overlapping I/O requests).

   Assuming the use of the ext2fs file system, and the current kernel policies about read−ahead, large stripe sizes are almost always better than small stripe sizes, and stripe sizes from about a fourth to a full disk cylinder in size may be best. To understand this claim, let us consider the effects of large stripes on small files, and small stripes on large files. The stripe size does not affect the read performance of small files: For an array of N drives, the file has a 1/N probability of being entirely within one stripe on any one of the drives. Thus, both the read latency and bandwidth will be comparable to that of a single drive. Assuming that the small files are statistically well distributed around the filesystem, (and, with the ext2fs file system, they should be), roughly N times more overlapping, concurrent reads should be possible without significant collision between them. Conversely, if very small stripes are used, and a large file is read sequentially, then a read will issued to all of the disks in the array. For a the read of a single large file, the latency will almost double, as the probability of a block being 3/4'ths of a revolution or farther away will increase. Note, however, the trade−off: the bandwidth could improve almost N−fold for reading a single, large file, as N drives can be reading simultaneously (that is, if read−ahead is used so that all of the disks are kept active). But there is another, counter−acting trade−off: if all of the drives are already busy reading one file, then attempting to read a second or third file at the same time will cause significant contention, ruining performance as the disk ladder algorithms lead to seeks all over the platter. Thus, large stripes will almost always lead to the best performance. The sole exception is the case where one is streaming a single, large file at a time, and one requires the top possible bandwidth, and one is also using a good read−ahead algorithm, in which case small stripes are desired.

> Note that this HOWTO previously recommended small stripe sizes for news spools or other systems with lots of small files. This was bad advice, and here's why: news spools contain not only many small files, but also large summary files, as well as large directories. If the summary file is larger than the stripe size, reading it will cause many disks to be accessed, slowing things down as each disk performs a seek. Similarly, the current ext2fs file system searches directories in a linear, sequential fashion. Thus, to find a given file or inode, on average half of the directory will be read. If this directory is spread across several stripes (several disks), the directory read (e.g. due to the ls command) could get very slow. Thanks to Steven A. Reisman < sar@pressenter.com> for this correction. Steve also adds:
>
> I found that using a 256k stripe gives much better performance. I suspect that the optimum size would be the size of a disk cylinder (or maybe the size of the disk drive's sector cache). However, disks nowadays have recording zones with different sector counts (and sector caches vary among different disk models). There's no way to guarantee stripes won't cross a cylinder boundary.

The tools accept the stripe size specified in KBytes. You'll want to specify a multiple of if the page size for your CPU (4KB on the x86).

9. **Q**: What is the correct stride factor to use when creating the ext2fs file system on the RAID partition? By stride, I mean the −R flag on the `mke2fs` command:

```
mke2fs −b 4096 −R stride=nnn  ...
```

What should the value of nnn be?

> **A**: The `−R stride` flag is used to tell the file system about the size of the RAID stripes. Since only RAID−0,4 and 5 use stripes, and RAID−1 (mirroring) and RAID−linear do not, this flag is applicable only for RAID−0,4,5. Knowledge of the size of a stripe allows `mke2fs` to allocate the block and inode bitmaps so that they don't all end up on the same physical drive. An unknown contributor wrote:
>
> I noticed last spring that one drive in a pair always had a larger I/O count, and tracked it down to the these meta−data blocks. Ted added the `−R stride=` option in response to my explanation and request for a workaround.

For a 4KB block file system, with stripe size 256KB, one would use `−R stride=64`.

If you don't trust the `−R` flag, you can get a similar effect in a different way. Steven A. Reisman < sar@pressenter.com> writes:

> Another consideration is the filesystem used on the RAID−0 device. The ext2 filesystem allocates 8192 blocks per group. Each group has its own set of inodes. If there are 2, 4 or 8 drives, these inodes cluster on the first disk. I've distributed the inodes across all drives by telling mke2fs to allocate only 7932 blocks per group.

Some mke2fs pages do not describe the `[-g blocks-per-group]` flag used in this operation.

10. **Q**: Where can I put the `md` commands in the startup scripts, so that everything will start automatically at boot time?

    A: Rod Wilkens < rwilkens@border.net> writes:

    What I did is put ``mdadd -ar'' in the ``/etc/rc.d/rc.sysinit'' right after the kernel loads the modules, and before the ``fsck'' disk check. This way, you can put the ``/dev/md?'' device in the ``/etc/fstab''. Then I put the ``mdstop -a'' right after the ``umount -a'' unmounting the disks, in the ``/etc/rc.d/init.d/halt'' file.

    For raid−5, you will want to look at the return code for `mdadd`, and if it failed, do a

    ```
    ckraid --fix /etc/raid5.conf
    ```

    to repair any damage.
11. **Q**: I was wondering if it's possible to setup striping with more than 2 devices in `md0`? This is for a news server, and I have 9 drives... Needless to say I need much more than two. Is this possible?

    A: Yes. (describe how to do this)
12. **Q**: When is Software RAID superior to Hardware RAID?

    A: Normally, Hardware RAID is considered superior to Software RAID, because hardware controllers often have a large cache, and can do a better job of scheduling operations in parallel. However, integrated Software RAID can (and does) gain certain advantages from being close to the operating system.

    For example, ... ummm. Opaque description of caching of reconstructed blocks in buffer cache elided ...

    On a dual PPro SMP system, it has been reported that Software−RAID performance exceeds the performance of a well−known hardware−RAID board vendor by a factor of 2 to 5.

    Software RAID is also a very interesting option for high−availability redundant server systems. In such a configuration, two CPU's are attached to one set or SCSI disks. If one server crashes or fails to respond, then the other server can `mdadd`, `mdrun` and `mount` the software RAID array, and take over operations. This sort of dual−ended operation is not always possible with many hardware RAID controllers, because of the state configuration that the hardware controllers maintain.
13. **Q**: If I upgrade my version of raidtools, will it have trouble manipulating older raid arrays? In short, should I recreate my RAID arrays when upgrading the raid utilities?

A: No, not unless the major version number changes. An MD version x.y.z consists of three sub−versions:

```
x:      Major version.
y:      Minor version.
z:      Patchlevel version.
```

Version x1.y1.z1 of the RAID driver supports a RAID array with version x2.y2.z2 in case (x1 == x2) and (y1 >= y2). Different patchlevel (z) versions for the same (x.y) version are designed to be mostly compatible.

The minor version number is increased whenever the RAID array layout is changed in a way which is incompatible with older versions of the driver. New versions of the driver will maintain compatibility with older RAID arrays.

The major version number will be increased if it will no longer make sense to support old RAID arrays in the new kernel code.

For RAID−1, it's not likely that the disk layout nor the superblock structure will change anytime soon. Most all Any optimization and new features (reconstruction, multithreaded tools, hot−plug, etc.) doesn't affect the physical layout.

14. **Q**: The command `mdstop /dev/md0` says that the device is busy.

    A: There's a process that has a file open on `/dev/md0`, or `/dev/md0` is still mounted. Terminate the process or `umount /dev/md0`.
15. **Q**: Are there performance tools?

    A: There is also a new utility called `iotrace` in the `linux/iotrace` directory. It reads `/proc/io−trace` and analyses/plots it's output. If you feel your system's block IO performance is too low, just look at the iotrace output.
16. **Q**: I was reading the RAID source, and saw the value `SPEED_LIMIT` defined as 1024K/sec. What does this mean? Does this limit performance?

    A: `SPEED_LIMIT` is used to limit RAID reconstruction speed during automatic reconstruction. Basically, automatic reconstruction allows you to `e2fsck` and `mount` immediately after an unclean shutdown, without first running `ckraid`. Automatic reconstruction is also used after a failed hard drive has been replaced.

    In order to avoid overwhelming the system while reconstruction is occurring, the reconstruction thread monitors the reconstruction speed and slows it down if its too fast. The 1M/sec limit was arbitrarily chosen as a reasonable rate which allows the reconstruction to finish reasonably rapidly, while creating only a light load on the system so that other processes are not interfered with.
17. **Q**: What about "spindle synchronization" or "disk synchronization"?

    A: Spindle synchronization is used to keep multiple hard drives spinning at exactly the same speed, so that their disk platters are always perfectly aligned. This is used by some hardware controllers to better organize disk writes. However, for software RAID, this information is not used, and spindle synchronization might even hurt performance.

18. **Q**: How can I set up swap spaces using raid 0? Wouldn't striped swap ares over 4+ drives be really fast?

    A: Leonard N. Zubkoff replies: It is really fast, but you don't need to use MD to get striped swap. The kernel automatically stripes across equal priority swap spaces. For example, the following entries from `/etc/fstab` stripe swap space across five drives in three groups:

```
/dev/sdg1       swap    swap    pri=3
/dev/sdk1       swap    swap    pri=3
/dev/sdd1       swap    swap    pri=3
/dev/sdh1       swap    swap    pri=3
/dev/sdl1       swap    swap    pri=3
/dev/sdg2       swap    swap    pri=2
/dev/sdk2       swap    swap    pri=2
/dev/sdd2       swap    swap    pri=2
/dev/sdh2       swap    swap    pri=2
/dev/sdl2       swap    swap    pri=2
/dev/sdg3       swap    swap    pri=1
/dev/sdk3       swap    swap    pri=1
/dev/sdd3       swap    swap    pri=1
/dev/sdh3       swap    swap    pri=1
/dev/sdl3       swap    swap    pri=1
```

19. **Q**: I want to maximize performance. Should I use multiple controllers?

    A: In many cases, the answer is yes. Using several controllers to perform disk access in parallel will improve performance. However, the actual improvement depends on your actual configuration. For example, it has been reported (Vaughan Pratt, January 98) that a single 4.3GB Cheetah attached to an Adaptec 2940UW can achieve a rate of 14MB/sec (without using RAID). Installing two disks on one controller, and using a RAID−0 configuration results in a measured performance of 27 MB/sec.

    Note that the 2940UW controller is an "Ultra−Wide" SCSI controller, capable of a theoretical burst rate of 40MB/sec, and so the above measurements are not surprising. However, a slower controller attached to two fast disks would be the bottleneck. Note also, that most out−board SCSI enclosures (e.g. the kind with hot−pluggable trays) cannot be run at the 40MB/sec rate, due to cabling and electrical noise problems.

    If you are designing a multiple controller system, remember that most disks and controllers typically run at 70−85% of their rated max speeds.

    Note also that using one controller per disk can reduce the likelihood of system outage due to a controller or cable failure (In theory −− only if the device driver for the controller can gracefully handle a broken controller. Not all SCSI device drivers seem to be able to handle such a situation without panicking or otherwise locking up).

9.High Availability RAID

1. **Q**: RAID can help protect me against data loss. But how can I also ensure that the system is up as long as possible, and not prone to breakdown? Ideally, I want a system that is up 24 hours a day, 7 days a week, 365 days a year.

   A: High−Availability is difficult and expensive. The harder you try to make a system be fault tolerant, the harder and more expensive it gets. The following hints, tips, ideas and unsubstantiated rumors may help you with this quest.

   ◊ IDE disks can fail in such a way that the failed disk on an IDE ribbon can also prevent the good disk on the same ribbon from responding, thus making it look as if two disks have failed. Since RAID does not protect against two−disk failures, one should either put only one disk on an IDE cable, or if there are two disks, they should belong to different RAID sets.
   ◊ SCSI disks can fail in such a way that the failed disk on a SCSI chain can prevent any device on the chain from being accessed. The failure mode involves a short of the common (shared) device ready pin; since this pin is shared, no arbitration can occur until the short is removed. Thus, no two disks on the same SCSI chain should belong to the same RAID array.
   ◊ Similar remarks apply to the disk controllers. Don't load up the channels on one controller; use multiple controllers.
   ◊ Don't use the same brand or model number for all of the disks. It is not uncommon for severe electrical storms to take out two or more disks. (Yes, we all use surge suppressors, but these are not perfect either). Heat & poor ventilation of the disk enclosure are other disk killers. Cheap disks often run hot. Using different brands of disk & controller decreases the likelihood that whatever took out one disk (heat, physical shock, vibration, electrical surge) will also damage the others on the same date.
   ◊ To guard against controller or CPU failure, it should be possible to build a SCSI disk enclosure that is "twin−tailed": i.e. is connected to two computers. One computer will mount the file−systems read−write, while the second computer will mount them read−only, and act as a hot spare. When the hot−spare is able to determine that the master has failed (e.g. through a watchdog), it will cut the power to the master (to make sure that it's really off), and then fsck & remount read−write. If anyone gets this working, let me know.
   ◊ Always use an UPS, and perform clean shutdowns. Although an unclean shutdown may not damage the disks, running ckraid on even small−ish arrays is painfully slow. You want to avoid running ckraid as much as possible. Or you can hack on the kernel and get the hot−reconstruction code debugged ...
   ◊ SCSI cables are well−known to be very temperamental creatures, and prone to cause all sorts of problems. Use the highest quality cabling that you can find for sale. Use e.g. bubble−wrap to make sure that ribbon cables to not get too close to one another and cross−talk. Rigorously observe cable−length

restrictions.
◊ Take a look at SSI (Serial Storage Architecture). Although it is rather expensive, it is rumored to be less prone to the failure modes that SCSI exhibits.
◊ Enjoy yourself, its later than you think.

10.Questions Waiting for Answers

1. **Q**: If, for cost reasons, I try to mirror a slow disk with a fast disk, is the S/W smart enough to balance the reads accordingly or will it all slow down to the speed of the slowest?
2. **Q**: For testing the raw disk thru put... is there a character device for raw read/raw writes instead of `/dev/sdaxx` that we can use to measure performance on the raid drives?? is there a GUI based tool to use to watch the disk thru−put??

11.Wish List of Enhancements to MD and Related Software

Bradley Ward Allen < ulmo@Q.Net> wrote:

Ideas include:

♦ Boot−up parameters to tell the kernel which devices are to be MD devices (no more ``mdadd'')
♦ Making MD transparent to ``mount''/``umount'' such that there is no ``mdrun'' and ``mdstop''
♦ Integrating ``ckraid'' entirely into the kernel, and letting it run as needed

(So far, all I've done is suggest getting rid of the tools and putting them into the kernel; that's how I feel about it, this is a filesystem, not a toy.)

♦ Deal with arrays that can easily survive N disks going out simultaneously or at separate moments, where N is a whole number > 0 settable by the administrator
♦ Handle kernel freezes, power outages, and other abrupt shutdowns better
♦ Don't disable a whole disk if only parts of it have failed, e.g., if the sector errors are confined to less than 50% of access over the attempts of 20 dissimilar requests, then it continues just ignoring those sectors of that particular disk.
♦ Bad sectors:

◊ A mechanism for saving which sectors are bad, someplace onto the disk.
◊ If there is a generalized mechanism for marking degraded bad blocks that upper filesystem levels can recognize, use that. Program it if not.
◊ Perhaps alternatively a mechanism for telling the upper layer that the size of the disk got smaller, even arranging for the upper layer to move out stuff

from the areas being eliminated. This would help with a degraded blocks as well.

◊ Failing the above ideas, keeping a small (admin settable) amount of space aside for bad blocks (distributed evenly across disk?), and using them (nearby if possible) instead of the bad blocks when it does happen. Of course, this is inefficient. Furthermore, the kernel ought to log every time the RAID array starts each bad sector and what is being done about it with a ``crit'' level warning, just to get the administrator to realize that his disk has a piece of dust burrowing into it (or a head with platter sickness).

♦ Software−switchable disks:

``disable this disk''

would block until kernel has completed making sure there is no data on the disk being shut down that is needed (e.g., to complete an XOR/ECC/other error correction), then release the disk from use (so it could be removed, etc.);

``enable this disk''

would `mkraid` a new disk if appropriate and then start using it for ECC/whatever operations, enlarging the RAID5 array as it goes;

``resize array''

would respecify the total number of disks and the number of redundant disks, and the result would often be to resize the size of the array; where no data loss would result, doing this as needed would be nice, but I have a hard time figuring out how it would do that; in any case, a mode where it would block (for possibly hours (kernel ought to log something every ten seconds if so)) would be necessary;

``enable this disk while saving data''

which would save the data on a disk as−is and move it to the RAID5 system as needed, so that a horrific save and restore would not have to happen every time someone brings up a RAID5 system (instead, it may be simpler to only save one partition instead of two, it might fit onto the first as a gzip'd file even); finally,

``re−enable disk''

would be an operator's hint to the OS to try out a previously failed disk (it would simply call disable then enable, I suppose).

Other ideas off the net:

- finalrd analog to initrd, to simplify root raid.
- a read−only raid mode, to simplify the above
- Mark the RAID set as clean whenever there are no "half writes" done. −− That is, whenever there are no write transactions that were committed on one disk but still unfinished on another disk. Add a "write inactivity" timeout (to avoid frequent seeks to the RAID superblock when the RAID set is relatively busy).