Using the Software RAID Functionality of Linux

Kevin Carson
Solution Architect
Hewlett-Packard (Canada) Co.

What this presentation IS

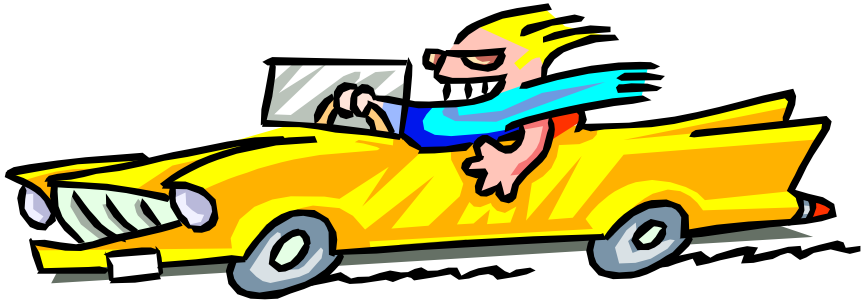
- Description of Linux Software RAID Functionality
- Examination of Booting from Software RAID
- Examination of Swapping failed Disks
- How to Layer RAID Levels
- How to mirror to Network Block Devices
- Issues in Choosing Between Software and Hardware RAID Configurations
- Based upon Debian 2.4.26 and 2.6.6/7 kernels

What this presentation IS NOT

- Introduction to RAID concepts
- RAID kernel code discussion
- A discussion of Linux Logical Volume Management
- Impromptu troubleshooting of any particular Linux Software RAID implementation

Linux Software RAID Description

“md” Driver



- “md” = multiple devices
- Various configurations are called “personalities” so the RAID levels supported each have a “personality” as well as special groupings of devices for such as a logical volume management and concatentation of devices.
- “md” is not loaded as a driver module, instead by each “personality”; one does need to load every possible personality.
- The kernel, some filesystems, and partition tables are able to recognize some/all personalities to leverage them for boot, file layout, auto-detection.

md “personalities”

Linear	Concatenation of devices into one large device.
RAID 0	Striped set of devices.
RAID 1	Mirrored set of devices.
RAID 5	Distributed parity with multiple choices for the distribution. Also implements RAID 4 (all parity on a single disk).
RAID 6	Kernel 2.6 only. Dual parity information.
Multipath	Failover between different connections to the same device.
HSM	Hierarchical Storage Management – Unused in kernel tree. Placeholder for external features?
Translucent	Meant for union/overlay filesystems but is unused in the kernel. Separate efforts trying to provide this facility.

Software RAID Concepts & Concerns

- chunksize is more commonly known as a “stripe”. It is the basic division of data spread out over the disks within an array. Size ranges from 1 page to 4MB in powers of 2.
- md devices must be explicitly stopped before shutdown.
- If a spare device is assigned to the array, it will take the place of a failed device. Later replacements of the failed physical device will become the new spare if they are added after the rebuild has finished.
- Spares are NOT scrubbed while idle. They are only known to be good when a rebuild action commences.
- Most RAID personalities base their per drive disk usage on the smallest partition/drive in the set.
- A md device will survive the physical shuffling of drives.

General kernel support of RAID

- Under the following conditions, the kernel will automatically start existing md arrays during boot only:
  - The RAID array consists of partitions whose type is “linux raid autodetect”
  - The partitions contain a persistent RAID superblock.
  - The partition table type is recognized by the kernel.
  - The appropriate RAID personality is not a dynamic module but compiled in to the kernel.
  - The driver for the IO channels to the disks is compiled in as well.
- Arrays not properly shutdown will need to be rebuilt upon activation but this will proceed in the background.
- When multiple arrays rely on the same physical device; only one rebuild action will be active.

General kernel support of RAID cont'd

- Status through /proc/mdstat

```
~# cat /proc/mdstat

Personalities : [linear] [raid0] [raid1] [raid5]
[multipath] [raid6]
md3 : active raid5 sdh[2] sdf[1] sde[0]
      17755392 blocks level 4, 64k chunk,
algorithm 0 [3/3] [UUU]

md2 : active raid1 md1[1] md0[0]
      17755328 blocks [2/2] [_U]
      [===================>..]  resync = 94.5%
(16780736/17755328) finish=4.0min speed=3996K/sec
…
unused devices: <none>
```

General kernel support of RAID cont'd

- Status through /proc/mdstat

```
Personalities : [linear] [raid0] [raid1] [raid5]
[multipath]
md3 : active raid5 sdf[1] sdh[3](F) sde[0]
      17755392 blocks level 4, 64k chunk, algorithm 0
[3/2] [UU_]
…
md1 : active raid1 sdd[2] sdc[0]
      8877696 blocks [2/1] [U_]
      [>....................]  recovery =  0.4%
(43392/8877696)
      finish=37.3min speed=3944K/sec
…
unused devices: <none>
```

General kernel support of RAID cont'd

- Control of rebuild through /proc/sys/dev/raid/ or sysctl. Speed_limit_min defines the floor (minimum rebuild speed) and speed_limit_max defines the ceiling. Usually the floor has to be raised as any activity on any disk will slow the rebuild dramatically.
- `/proc/sys/dev/raid/speed_limit_min`
  - `echo "10000" >speed_limit_min (default is 100 KiB/s)`
  - `sysctl -w dev.raid.speed_limit_min=10000`
- `/proc/sys/dev/raid/speed_limit_max`
  - `echo "100000" >speed_limit_max (default is 10 MB/s)`
  - `sysctl -w dev.raid.speed_limit_max=100000`

RAID 0 Personality

- Very little error checking on device state so system can hang with retries.
- Maximum IO transaction to a device will be chunksize.
- Attempts to merge adjancent IOs to a limit of chunksize
- If devices within RAID 0 set are varying size, remaining chunks will land on largest device

RAID 1 Personality

- chunksize is ignored; transactions are size of block of underlying device (typically 512 bytes). Lower block layer would normally merge successive requests.
- Read balancing (as of 2.4.20) based upon device with nearest last known position with respect to RAID device. If multiple partitions or mirrors active on the same device, then this strategy fails as the last known head position is tracked by mirror. For constant sequential reads, this strategy can cause the same device to be hit constantly.
- For 2.6, a better strategy is implemented for spreading sequential reads and tracking last known head position. Resynchronization proceeds in 64KiB transactions.
- One can specify multiple copies to get n-way mirroring.
- To split off a mirror (eg for backup), one would set the device as “faulty” and then remove the device from the RAID set. Adding back causes a resynchroniziaton.
- Need a minimum of two devices.

RAID 5 Personality

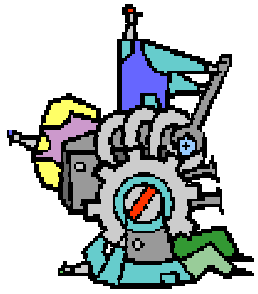
- Upon boot/module load, several different parity algorithms are tried and the best method chosen.
- Four choices on how parity chunks are distributed among the devices of the RAID set: Left Asymmetric, Right Asymmetric, Left Symmetric, Right Symmetric.
- Also handles RAID 4 functionality (dedicated parity disk instead of distributed parity chunks). The last active disk in the RAID set is chosen as the parity disk.
- RAID 4 shows up as a RAID 5 device /proc/mdstat but with “level 4” noted in device line.
- Need a minimum of 3 devices.

RAID 6 Personality

- Only available with 2.6.x kernels.
- Two sets of parity information so will survive failure of two devices.
- Specialized algorithms for parity calculation are only available for x86 or x86-64 MMX/SSE capable CPUs though generic algorithms perform well on IA64 (only ~24% decrease over RAID 5 calculation). For very generic CPUs, results can be as much as 5X slower over RAID5.
- Choices of parity distribution is identical to RAID5.
- Can only be manipulated with mdadm utility.
- Minimum of 4 devices.

Tools for Software RAID

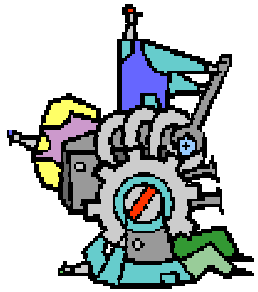
raidtools2



- Second version of original set of tools.
- Each different action has a different command associated (eg, lsraid, mkraid, raidhotadd, raidhotremove, etc.)
- Expects to have a valid configuration file describing the current (or about to be created) md devices. /etc/raidtab by default.
- Use “lsraid –R –p” to output a formatted raidtab
- Example /etc/raidtab entry:

```
# md device [dev 9, 0] /dev/md0 queried online
raiddev /dev/md0
      raid-level                1
      nr-raid-disks             2
      nr-spare-disks            0
      persistent-superblock     1
      chunk-size                0

      device           /dev/sda1
      raid-disk                 0
      device           /dev/sdb1
      raid-disk                 1
```

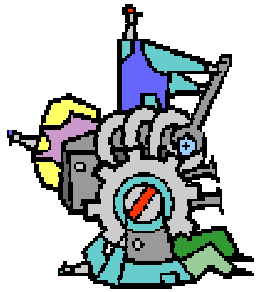
mdadm



- Provides all the same actions as the raidtools2 suite but with different command line options rather than separate commands.
- Most actions don't require a configuration file.
- Mdadm can also run in the background, monitoring the running md devices and detect changes. An event can trigger log events, mail notification, and the running of scripts.
- While mdadm is running as a monitor it can float a spare device between arrays to create a global spare. Without this functionality, spares are statically assigned per array only.
- For some operations, the monitoring action of mdadm may prevent changes to the array's state. Temporarily shut off the daemon.
- Use "mdadm –detail --scan" to output a formatted configuration file. Default configuration file is /etc/mdadm/mdadm.conf
- Example /etc/mdadm/mdadm.conf entry:

```
ARRAY /dev/md0 level=raid1 num-devices=2
UUID=40e8c451:71a182ed:23642c05:476cf390
 devices=/dev/sda1,/dev/sdb1
```

Filesystems' support of RAID



- EXT2/3 and XFS provide parameters to tune allocation of data to the stripe size of the RAID unit. These parameters are supplied when the filesystem is first created on the md device.
- Ext2/3:
  - use "-R stride=XXX" parameter to align data allocation. Stride should be set to chunk size / filesystem block size.
- XFS aligns allocations for both its log and data sections using:
  - "-d su=XXX" or "-d sunit=XXX"
  - "-l su=XXX" or "-l sunit=XXX"

  where "su=" is specified in bytes and "sunit=" is specified in 512 byte blocks.

Caveats:

- Its best to move journalling filesystems' log to a separate RAID 1 device.
- Ext2/3 allocates blocks in groups of 32768 by default. The start of each block group has the inode and block allocation bitmaps. If the group block size is an even multiple of stripe set size (chunk size * #devices) and the "-R stride=XXX" isn't used, then accesses for these bitmaps will happen on the same device. Don't forget the "-R stride=XXX" option!

Layering Software RAID Personalities

Layering md Devices

- Simple as using other md devices as the “raid disks”. Here is an example creating a RAID10 (striped mirrors) array:

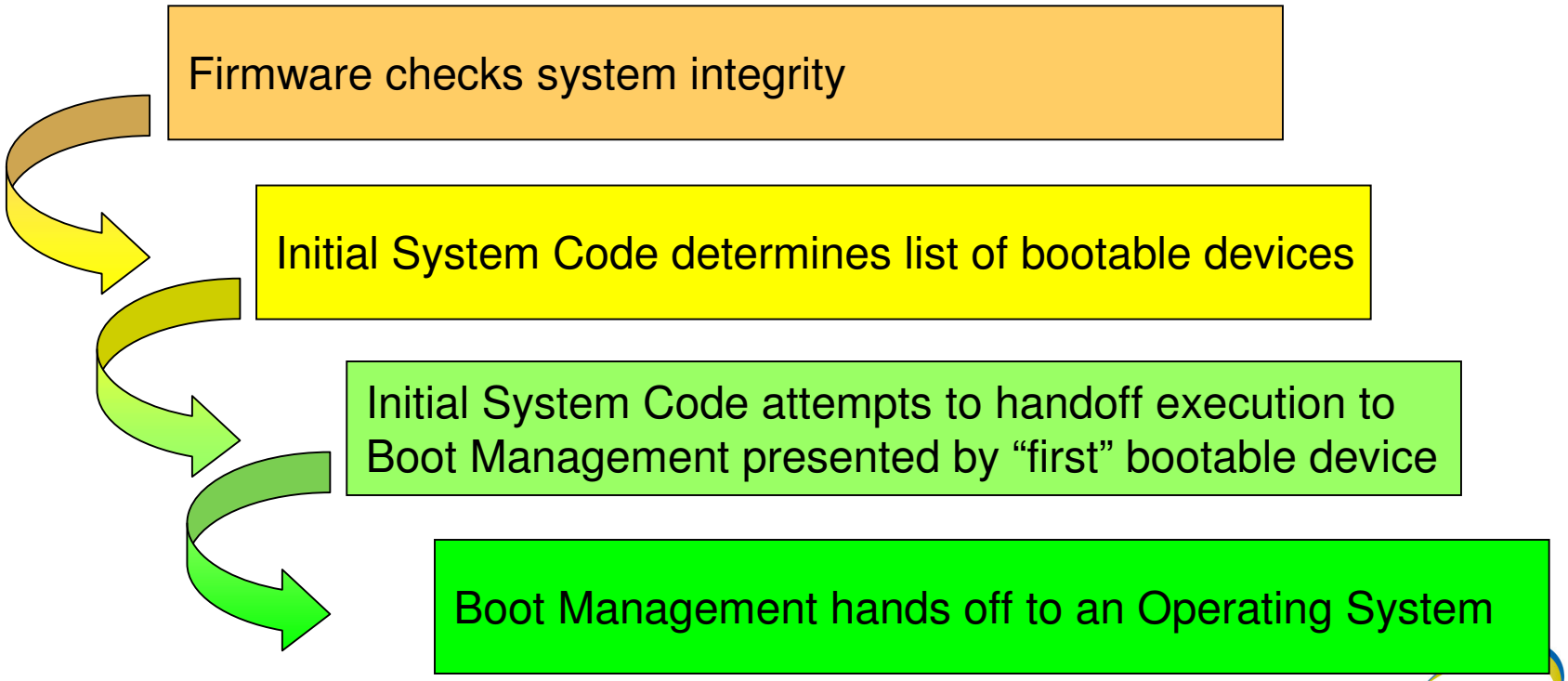
```
~# mdadm -C /dev/md0 -l 1 -n 2 -a /dev/sda1 -a /dev/sdb1
~# mdadm -C /dev/md1 -l 1 -n 2 -a /dev/sdc1 -a /dev/sdd1
~# mdadm -C /dev/md2 -l 1 -n 2 -a /dev/md0 -a /dev/md1

Personalities : [raid0] [raid1]
md2 : active raid0 md1[1] md0[0]
      17753728 blocks 64k chunks
md1 : active raid1 sdd1[1] sdc1[0]
      8876928 blocks [2/2] [UU]
      [>....................]  resync =  1.3% (120704/8876928) finish=54.4min speed=2678K/sec
md0 : active raid1 sdb1[1] sda1[0]
      8876928 blocks [2/2] [UU]
      [>....................]  resync =  1.5% (134336/8876928) finish=97.1min speed=1498K/sec
unused devices: <none>
```

- Raidtab order is important such that /dev/md0 and /dev/md1 entries must appear before /dev/md2’s entry.
- One could do something similar with nested RAID5 sets to get a RAID6 like protection in a 2.4 kernel.
- Using layers increases overhead in processing. This is a reason for implementing true RAID6 in the kernel.

Booting from Software RAID

Boot Process Conceptualized

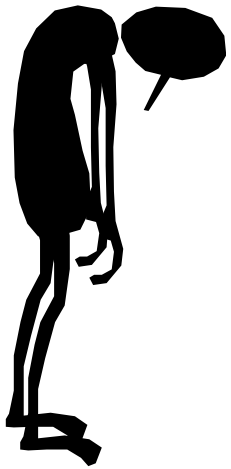
- Most platforms implement these steps with varying divisions of which firmware/bootloader does what:



Firmware Checks System Integrity

"I'm ALIVE!!!!"

Or not…



Initial System Code Finds Boot Devices

- Two extremes of implementation:
  - Can be a very restricted list of device types the ISC will recognize.
  - EFI allows adapter to install driver from adapter option ROM.
- Different ways of making the list:
  - BIOS expansion ROM presented by adapter adds entry if adapter has been told of/detected a candidate.
  - Firmware is configured with explicit list of devices.
- Depending on the Initial System Code's flexibility, one may have to mix device types (eg, a boot floppy and a hard drive) to get redundancy.
- Identical setups of later stage software on each.

ISC to Boot Management Hand-off

- Initial System Code attempts to handoff execution to Boot Management presented by “first” bootable device.
- Multiple methods of determining that a device has a boot loader installed:
  - Load sector 0 and look for magic bytes.
  - Device contains a partition marked as bootable with a specific filesystem.
- Boot Management may be multi stage.
- For redundancy, there must boot management on multiple devices.
- If there is a detectable failure, then the ISC may be able to try the next candidate. If not a detectable failure or a Boot Management misconfiguration/corruption then boot process stops.

Boot Management to OS Hand-off

- The boot partition contains last stages of boot management.
- The boot partition has to contain a kernel that will be able to recognize RAID partitions.
- The boot partition must be present on each redundant device.
- For root devices (mounted by the kernel from the boot partition), other RAID levels can be used.

BIOS Enhanced RAID of HP Proliant BL30p

- If enabled, the BIOS will take special steps with respect to any Linux Software RAID1 boot partitions.
- Blade server has two ATA drives on a single ATA channel.
- If the primary ATA disk has failed, it will boot off of the secondary ATA disk if it also has a bootable Linux Software RAID1 partition.
- If the software RAID superblocks are out of sync on the two disks, it will choose to boot from the one with the most recently updated superblock.

Lilo Support of RAID Boot

- Lilo (Linux Loader) only supports being installed on RAID 1 partitions for redundant boot.
- Using either –x command line option or “raid-extra-boot=“ lilo.conf entry, the boot management configuration is replicated on multiple devices.
- Variants from other platforms (palo on PA-RISC, elilo for EFI) don’t support the above handy feature.
- Lilo and variants are the only choice on several platforms.
- Partitions used as boot should be primary instead of extended. The Master Boot Record loader expects to find a primary partition marked as active/bootable.

Grub Support of RAID Boot

- Create **boot** partition as RAID 1 with type “Linux raid autodetect” (0xfd) on disks. The boot partition should be marked active (or bootable).
- Create RAID **root** partitions also with “0xfd” type.
- Install 1st stage bootloader on MBR on both disks with GRUB’s “setup” command.
- Make two boot entries in “menu.lst”. The two entries point to the same kernel and root parititon (/dev/md1) but specifies different boot devices:

```
# First entry is default entry to boot
default 0

# If the default is not available boot second entry
fallback 1

# Don’t wait for manual selection after 10 seconds
timeout 10

# Default boot entry
title Default
kernel hd(0,0)/vmlinuz root=/dev/md1

# Fallback boot entry
title Fallback
kernel hd(1,0)/vmlinuz root=/dev/md1
```

- (x86) Install the 1st stage bootloader in the MBR on both disks with GRUB’s setup command.

Retrofit of existing system

1. Set up first system disk as usual on Linux partition.
2. Set up partition of type “Linux raid auto-detect” on second drive.
3. Configure a RAID device where second disk is active and first disk is specified but failed.
4. Copy contents of system on first disk to second drive.
5. Set boot management (lilo/grub) to expect to boot md device. This may include a regeneration of initial ram disk image via mkinitrd.
6. Reboot system so now its booting off new md device.
7. Repartition first disk to match second disk and then add it to the md device. Resync will begin.
8. Add first drive as another boot device in boot management.
9. Wait until resync has finished before next reboot and voila!

Swapping Failed Disks

Issues for Device Hot Replacement

- Channel & Device has to support electrical isolation (avoiding electrostatic discharges and controlling inrush current)
- Device has to be rescanned by kernel
- Transfer partitions from old device to new device
- Old device has to be removed from md device
- New device has to be added back
- Device will be immediately accessible but not highly available until resync is finished.

SCSI Swap Example

- Capture disk partitions
  - sfdisk –d /dev/sdc >/tmp/sdc.parts
- Remove disk (or partition) from md device
  - raidsetfaulty /dev/md0 /dev/sdc
  - raidhotremove /dev/md0 /dev/sdc

OR

  - mdadm /dev/md0 –f /dev/sdc –r /dev/sdc
- Gather host adapter, channel, target, lun information
  - cat /proc/scsi/scsi

OR

  - sginfo -l
- Notify kernel's scsi subsystem of the removal of the device via /proc pseudo filesystem interface using host adapter, channel, target and lun numbers
  - echo "scsi remove-single-device 0 0 2 0" >/proc/scsi/scsi
- Physically swap the drive with an identical replacement
- Notify kernel's scsi subsystem of the addition of the device via /proc pseudo filesystem interface
  - echo "scsi add-single-device 0 0 2 0" >/proc/scsi/scsi
- Re-establish the partitioning
  - sfdisk /dev/sdc </tmp/sdc.parts
- Add the disk (or partition) back into the md device
  - raidhotadd /dev/md0 /dev/sdc

OR

  - mdadm /dev/md0 –a /dev/sdc

Swapping on Software RAID

How to do it

Just like any other block device:

```
mkswap /dev/md0

swapon /dev/md0
```

- But this a so-so idea
  - Swap partitions on a RAID device should work and limited testing has been done. However, no one is ready to commit that kernel code paths have been thoroughly checked after every update.

Workarounds

- If RAID 0 is of interest (ie, higher performance is required) then one can do this with just the regular swap driver. In the /etc/fstab file:

```
/dev/sda   none swap sw,pri=8   0     0
/dev/sdb   none swap sw,pri=8   0     0
/dev/sdc   none swap sw,pri=8   0     0
```

The matching values for priority in the option field (“pri=8”) means that pages will be striped between the three SCSI disks. This yields performance without loading another kernel driver. Priorities range from 0 to 32767 with 32767 being the highest. Higher priority devices will be used before lower priority devices.

Workarounds Cont'd

- Don't use swap. Swap is a side effect of installation but often not used in production. Out of control processes are better handled by the OOM (Out Of Memory) killer which often picks the errant process correctly.
- Use filesystem swap which also allows for better dynamic control of swap/disk resources. It uses slightly different code paths but the extra layers make it slower.
- Hardware RAID based swap.

Mirroring to Network Block Devices

Network Block Device with RAID

- NBD is standard in kernel. Presents a file or block device on a server to a different client computer across a TCP connection.
- Client can include local block device entry (eg, /dev/nbd0) in creation of a RAID entry.
- Device can be formatted with regular filesystem. Local caching is a side effect of the normal block cache mechanism (not specially designed like NFS).
- References are 64 bit but underlying platform may restrict devices to 2GB. Use multiple NBD devices to overcome this restriction.
- Implementations of the nbd server available for Windows and BSD as well.

What one might use NBD for….

- Disaster Recovery site like commercial solutions (eg, CA, SRDF, etc…) but any temporary network loss would cause a FULL resync. Enhanced NBD project (which ISN'T in the kernel) allows for delta synchronization.
- Have a highly available ram disk by mirroring a local /dev/ramXX device to /dev/nbdXX. ndbXX could be a file resident in cache or another ramdisk to yield good performance.
- Checkpointing in High Performance Computing. When checkpointing is all writes it makes less sense to use a protocol with a lot of caching algorithms like NFS.
- Diskless systems. Can't provide boot capability but can provide other partitions and swap devices. Copy on write option for NBD server allows sharing of read only block device among many.
- Stretch a full image backup window by mirroring over network but using software RAID's speed_limit_max throttling to control amount of bandwidth used during working hours.

Choosing Between Hardware and Software RAID

The Differences: Simplicity

Software RAID	Hardware RAID
Exposes the management of each individual device.	Most interaction is with logical block device representation.
Requires understanding of entire chain of reliances.	Usually has an “Express” setup capability.
Setup can invariably be scripted. Usually generically over many configurations.	If vendor doesn’t provide OS neutral or Linux compatible command line configuration then manual intervention required. Generic configuration often harder.

The Differences: Flexibility

Software RAID	Hardware RAID
Can use many controllers and hard disks. Including ones that have never been qualified together.	Tries to lock down configuration to only supported disk types. May even load disk firmware or do low level reformat.
Allows use or mixing of more block device types; ATA, ESDI, /dev/ram NBD, etc.	Usually constrained to one disk type (eg, SCSI). Mixed types and higher capabilities available in Enterprise gear.
Layering of RAID levels allow for many permutations. Better match to specific workloads.	Invariably a subset of RAID levels supported.
Tricky RAID level migration in place.	Easy RAID level migration in place.
RAID 6 only available with 2.6 kernels.	Higher availability choices available now on all platforms.

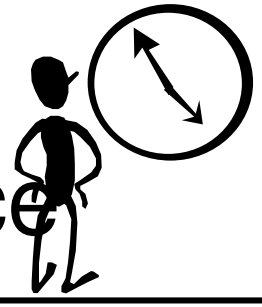
The Differences: Cost

Software RAID	Hardware RAID
Cheaper hardware due to huge range of choices.	Offload of I/O may allow lesser SPU to fufill same requirements. Potentially a price already paid if hardware RAID is standard.
More administrative cost due to more exposed complexity and testing migrated from vendor to user.	Less administrative cost for user but cost of vendor testing and support amortized over installed base of controllers.

The Differences: Supportability

Software RAID	Hardware RAID
Somewhat supported to the extent the user has done testing of complete configuration including corner failure cases.	High levels of support only on specific configurations of controllers and disks. Tests are exhaustive and include corner cases.
Device error detection rely on the underlying device and/or communication channels.	Hardware RAID can offer active spare, disk surface scrubbing, parity verification. Error checking goes beyond passive detection.
No assumptions are made about components to match the range of possible configurations.	Component choices are constrained so optimal use of device through specific capabilities
Raw disks and partitions are exposed so server firmware and boot management must be able to utilize these.	Logical block device is exposed that emulates common device geometry and behaviour.

The Differences: Performance



Software RAID	Hardware RAID
On benchmarks dedicated to measuring storage subsystem only, software RAID is usually faster.	Reduces number of context switches and interrupts flowing through system. Read-Update-Write parity cycle does not occupy system resources.
RAID 1 requires a transaction over every individual IO channel and device that has a copy of the data.	RAID 1 only requires a transaction to the controller which handles propagation to disks.
Either the devices are assumed to be able to commit remaining writes upon power failure or the volumes are mounted synchronously	Battery backed cache allows for quick returns from writes over PCI bus.
Optimal performance means choosing transfer sizes that are best suited to device behaviour first and workload second.	Onboard cache allows transaction size to suit workload. Controller handles mapping to device characteristics.
With respect to capital cost, can usually configure higher peak performance/$.	May yield better results in actual production for less administrative effort/$.

Linux Software RAID Resources

References

Linux Software RAID HowTo

http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html

Boot+Root+RAID+LILO mini-HOWTO

http://www.tldp.org/HOWTO/Boot+Root+Raid+LILO.html

Kernel Linux-RAID List

linux-raid at http://vger.kernel.org

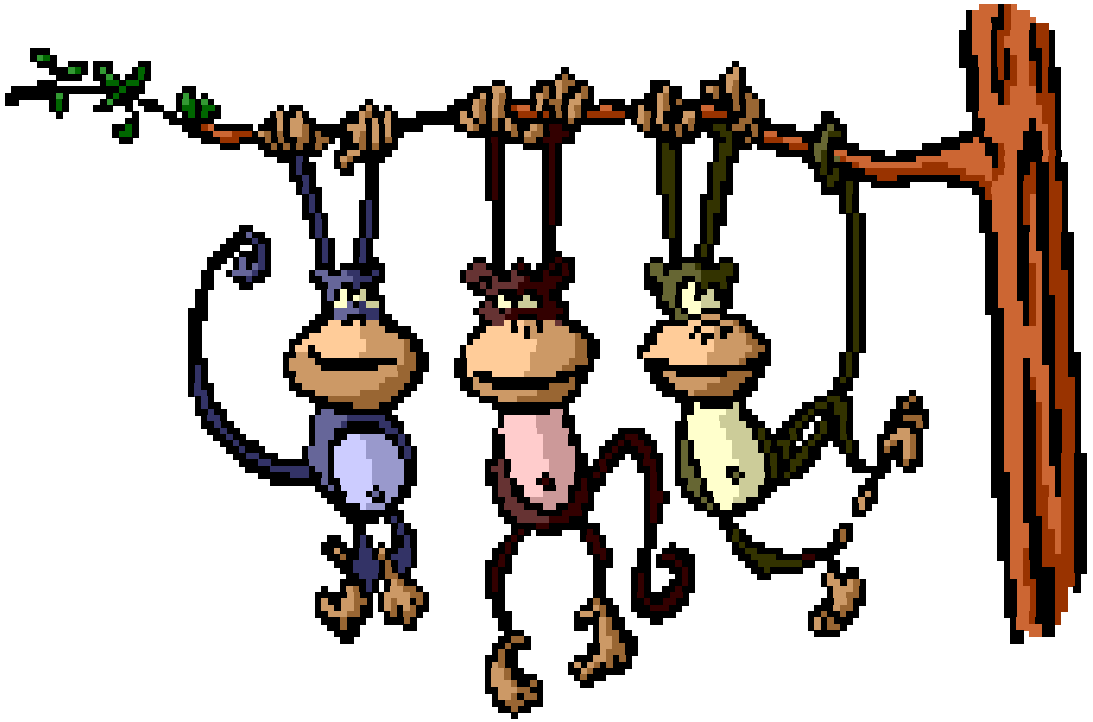
NBD and Enhanced NBD Project Sites

NBD: http://nbd.sourceforge.net/

ENBD: http://www.it.uc3m.es/~ptb/nbd/

Thanks to….

Dan Zink
Theodora Carson
Christie Melnychuk
Rick Stern
Lia Lindell
Yizhi Wang
Karthik Prabhakar



HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

RECOMMENDED TRAINING VENUE FOR THE

HP Certified Professional