Proceedings of the Linux Symposium

Volume One

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.

dmraid - device-mapper RAID tool

Supporting ATARAID devices via the generic Linux device-mapper

Heinz Mauelshagen
Red Hat Cluster and Storage Development
mauelshagen@redhat.com

Abstract

Device-mapper, the new Linux 2.6 kernel generic device-mapping facility, is capable of mapping block devices in various ways (e.g. linear, striped, mirrored). The mappings are implemented in runtime loadable plugins called mapping targets.

These mappings can be used to support arbitrary software RAID solutions on Linux 2.6, such as ATARAID, without the need to have a special low-level driver as it used to be with Linux 2.4. This avoids code-redundancy and reduces error rates.

Device-mapper runtime mappings (e.g. map sector N of a mapped device onto sector M of another device) are defined in mapping tables.

The dmraid application is capable of creating these for a variety of ATARAID solutions (e.g. Highpoint, NVidia, Promise, VIA). It uses an abstracted representation of RAID devices and RAID sets internally to keep properties such as paths, sizes, offsets into devices and layout types (e.g. RAID0). RAID sets can be of arbitrary hierarchical depth in order to reflect more complex RAID configurations such as RAID10.

Because the various vendor specific metadata formats stored onto ATA devices by the ATARAID BIOS are all different, metadata format handlers are used to translate between the ondisk representation and the internal abstracted format.

The mapping tables which need to be loaded into device-mapper managed devices are derived from the internal abstracted format.

My talk will give a device-mapper architecture/feature overview and elaborate on the dmraid architecture and how it uses the device-mapper features to enable access to ATARAID devices.

1 ATARAID

Various vendors (e.g. Highpoint, Silicon Image) ship ATARAID products to deploy Software RAID (Redundant Array Of Inexpensive Disks) on desktop and low-end server system. ATARAID essentially can be characterized as:

- 1-n P/SATA or SCSI interfaces
- a BIOS extension to store binary RAID set configuration metadata on drives attached
- a BIOS extension to map such RAID sets in early boot and access them as a single device so that booting off them is enabled

- an operating system driver (typically for Windows) which maps the RAID sets after boot and updates the vendor specific metadata on state changes (e.g. mirror failure)
- a management application to deal with configuration changes such as mirror failures and replacements

Such ATARAID functionality can either be provided via an additional ATARAID card or it can be integrated on the mainboard as with solutions from NVidia or VIA. It enables the user to setup various RAID layouts, boot off them and have the operating system support them as regular block devices via additional software (hence the need for Windows drivers). Most vendors do RAID0 and RAID1, some go beyond that by offering concatenation, stacked RAID sets (i.e. RAID10) or higher RAID levels (i.e. RAID3 and RAID5).

1.1 Some metadata background

The vendor on-disk metadata keeps information about:

- the size(s) of the areas mapped onto a disk
- the size of the RAID set
- the layout of the RAID set (e.g. RAID1)
- the number of drives making up the set
- a unique identifier (typically 1 or 2 32 bit numbers) for the set
- the state of the set (e.g. synchronized for RAID1) so that the driver can start a resynchronization if necessary

What it usually doesn't keep is a unique human readable name for the set which means, that dmraid needs to derive it from some available unique content and make a name up. The tradoff is, that names are somewhat recondite.

Some vendors (i.e. Intel) retrieve ATA or SCSI device serial numbers and store them in the metadata as RAID set identifiers. Others just make unique numbers using random number generators.

1.2 Support in Linux

Linux 2.4 supported a limited list of products via ATARAID specific low-level drivers.

Now that we have the very flexible device-mapper runtime in the Linux 2.6 kernel series, this approach is no longer senseful, because an application (i.e. dmraid) can translate between all the different on-disk metadata formats and the information device-mapper needs to activate access to ATRAID sets. This approach avoids the overhead of seperate low-level drivers for the different vendor solutions in the Linux kernel completely.

2 Device-Mapper

The device-mapper architecture can be delineated in terms of these userspace and kernel components:

- a core in the Linux kernel which maintains mapped devices (accessible as regular block devices) and their segmented mappings definable in tuples of offset, range, target, and target-specific parameters. Offset and ranges are in units of sectors of 512 bytes. Such tuples are called targets (see examples below). An arbitrary length list of targets defining segments in the logical address space of a mapped device make up a device mapping table.

- a growing list of kernel modules for plugable mapping targets (e.g. linear, striped, mirror, zero, error, snapshot, cluster mapping targets...) which are responsible for (re)mapping IOs to a sector address range in the mapped devices logical address space to underlying device(s) (e.g. to mirrors in a mirror set).
- an ioctl interface module in the kernel to communicate with userspace which exports functionality to create and destroy mapped devices, load and reload mapping tables, etc.
- a device-mapper userspace library (libdevmapper) which communicates with the kernel through the ioctl interface accessing the functions to create/destroy mapped devices and load/reload their ASCII formated mapping tables. This library is utilized by a couple applications such as LVM2 and dmraid.
- a dmsetup tool which uses libdevmapper to manage mapped devices with their mapping tables and show supported mapping targets, etc.

2.1 Mapping table examples

1. 0 1024 linear /dev/sda 0
2. 0 2048 striped 2 64 /dev/sda 1024 /dev/sdb 0
3. 0 4711 mirror core 2 64 nosync 2 /dev/sda 2048 /dev/sdb 1024
4. 0 3072 zero
   3072 1024 error

Example 1 maps an address range (segment) starting at sector 0, length 1024 sectors linearly (*linear* is a keyword selecting the linear mapping target) onto /dev/sda, offset 0. `/dev/sda 0` (a device path and an offset in sectors) are the 2 target-specific parameters required for the linear target.

Example 2 maps a segment starting at sector 0, length 2048 sectors striped (the *striped* keyword selects the striping target) onto /dev/sda, offset 1024 sectors and `/dev/sdb`, offset 0 sectors. The striped target needs to know the number of striped devices to map to (i.e. '2') and the stride size (i.e. '64' sectors) to use to split the IO requests.

Example 3 maps a segment starting at sector 0, length 4711 sectors mirrored (the *mirror* keyword selects the mirror target) onto `/dev/sda`, offset 2048 sectors (directly after the striped mapping from before) and `/dev/sdb`, offset 1024 sectors (after the striped mapping).

Example 4 maps a segment starting at sector 0, length 3072 sectors using the 'zero' target, which returns success on both reads and writes. On reads a zeroed buffer content is returned. A segment beginning at offset 3072, length 1024 gets mapped with the 'error' target, which always returns an error on reads and writes. Both segments map a device size of 4096 sectors.

As the last example shows, each target line in a mapping table is allowed to use a different mapping target. This makes the mapping capabilities of device-mapper very flexible and powerfull, because each segent can have IO optimized properties (e.g. more stripes than other segments).

Note: Activating the above mappings at once is just for the purpose of the example.

2.2 dmsetup usage examples

By putting arbitrary mapping tables like the above ones into files readable by the dmsetup tool (which can read mapping tables from standard input as well), mapped devices can be created or removed and their mappings can be loaded or reloaded.

1. dmsetup create ols filename
2. dmsetup reload ols another_filename
3. dmsetup rename ols OttawaLinuxSymposium
4. dmsetup remove OttawaLinuxSymposium

Example 1 creates a mapped device named 'ols' in the default device-mapper directory `/dev/mapper/`, loads the mapping table from 'filename' (e.g. '0 3072 zero') and activates it for access.

Example 2 loads another mapping table from 'another_filename' into 'ols' replacing any given previous one.

Example 3 renames mapped device 'ols' to 'OttawaLinuxSymposium'.

Example 4 deactivates 'OttawaLinuxSymposium', destroys its mapping table in the kernel, and removes the device node.

3 dmraid

The purpose of dmraid is to arbitrate between the ATARAID on-disk metadata and the device-mapper need to name mapped devices and define mapping tables for them.

Because the ATARAID metadata is vendor specific and the respective formats therefore all differ, an internal metadata format abtraction is necessary to translate into and derive the mapped device names and mapping tables content from.

The 'translators' between the vendor formats and the internal format are called 'metadata format handlers.' One of them is needed for any given format supported by dmraid.

An activation layer translates from there into mapping tables and does the libdevmapper calls to carry out device creation and table loads to gain access to RAID sets.

3.1 dmraid components

- the dmraid tool which parses the command line and calls into
- the dmraid library with:
  - a device access layer to read and write metadata from/to RAID devices and to retrieve ATA and SCSI serial numbers
  - a metadata layer for the internal metadata format abstraction with generic properties to describe RAID devices and RAID sets with their sizes and offsets into devices, RAID layouts (e.g. RAID1) including arbitrary stacks of sets (e.g. RAID10)
  - metadata format handlers for every vendor specific solution (e.g. Highpoint, NVidia, VIA, ...) translating between those formats and the internal generic one
  - an activation layer doing device-mapper library calls
  - a display layer to show properties of block devices, RAID devices and RAID sets
  - a logging layer which handles output for verbosity and debug levels
  - a memory management layer (mainly for debugging purposes)
  - a locking layer to prevent parallel dmraid runs messing with the metadata

3.2 Command line interface

The dmraid CLI comprehends options to:

- activate or deactivate ATARAID sets
- select metadata formats
- display properties of
  - block devices
  - RAID devices
  - RAID sets
  - vendor specific metadata
- display help (command synopsis)
- list supported metadata formats
- dump vendor metadata and locations into files
- display the dmraid, dmraid library and the device-mapper versions

The command synopsis looks like:

```
dmraid: Device-Mapper Software RAID tool

* = [-d|--debug]... [-v|--verbose]...

dmraid  {-a|--activate} {y|n|yes|no} *
        [-f|--format FORMAT]
        [-p|--no_partitions]
        [-t|--test]
        [RAID-set...]

dmraid  {-b|--block_devices} *
        [-c|--display_columns]...

dmraid  {-h|--help}

dmraid  {-l|--list_formats} *

dmraid  {-n|--native_log} *
        [-f|--format FORMAT]
        [device-path...]

dmraid  {-r|--raid_devices} *
        [-c|--display_columns]...
        [-D|--dump_metadata]
        [-f|--format FORMAT]
        [device-path...]

dmraid  {-r|--raid_devices} *
        {-E|--erase_metadata}
        [-f|--format FORMAT]
        [device-path...]

dmraid  {-s|--sets}...[a|i|active|inactive] *
        [-c|--display_columns]...
        [-f|--format FORMAT]
        [-g|--display_group]
        [RAID-set...]

dmraid  {-V/--version}
```

3.3 dmraid usage examples

List all available block devices:

```
# dmraid -b
/dev/sda:     72170879 total, "680631431K"
/dev/sdb:      8887200 total, "LG142316"
/dev/sdc:     72170879 total, "680620811K"
```

List all discovered RAID devices:

```
# dmraid -r
/dev/dm-14: hpt45x, "hpt45x_dbagefdi", \
 stripe, ok, 320172928 sectors, data@ 0
/dev/dm-18: hpt45x, "hpt45x_dbagefdi", \
 stripe, ok, 320172928 sectors, data@ 0
/dev/dm-22: hpt45x, "hpt45x_bhchfdeie", \
 mirror, ok, 320173045 sectors, data@ 0
/dev/dm-26: hpt45x, "hpt45x_bhchfdeie", \
 mirror, ok, 320173045 sectors, data@ 0
/dev/dm-30: hpt45x, "hpt45x_edieecfd", \
 linear, ok, 320173045 sectors, data@ 0
/dev/dm-34: hpt45x, "hpt45x_edieecfd", \
 linear, ok, 320173045 sectors, data@ 0
/dev/dm-38: hpt45x, "hpt45x_chidjhaiaa-0", \
 stripe, ok, 320172928 sectors, data@ 0
/dev/dm-42: hpt45x, "hpt45x_chidjhaiaa-0", \
 stripe, ok, 320172928 sectors, data@ 0
/dev/dm-46: hpt45x, "hpt45x_chidjhaiaa-1", \
 stripe, ok, 320172928 sectors, data@ 0
/dev/dm-50: hpt45x, "hpt45x_chidjhaiaa-1", \
 stripe, ok, 320172928 sectors, data@ 0
```

List all discovered RAID sets:

```
# dmraid -cs
hpt45x_dbagefdi
hpt45x_bhchfdeie
hpt45x_edieecfd
hpt45x_chidjhaiaa
```

Show mapped devices and mapping tables for RAID sets discovered:

```
# dmraid -tay
hpt45x_dbagefdi: 0 640345856 striped \
 2 128 /dev/dm-14 0 /dev/dm-18 0
hpt45x_bhchfdeie: 0 320173045 mirror \
 core 2 64 nosync 2 /dev/dm-22 0 /dev/dm-26 0
hpt45x_edieecfd: 0 320173045 \
 linear /dev/dm-30 0
hpt45x_edieecfd: 320173045 320173045 \
 linear /dev/dm-34 0
hpt45x_chidjhaiaa-0: 0 640345856 striped \
 2 128 /dev/dm-38 0 /dev/dm-42 0
hpt45x_chidjhaiaa-1: 0 640345856 striped \
 2 128 /dev/dm-46 0 /dev/dm-50 0
hpt45x_chidjhaiaa: 0 640345856 mirror \
 core 2 256 nosync 2 \
 /dev/mapper/hpt45x_chidjhaiaa-0 0 \
 /dev/mapper/hpt45x_chidjhaiaa-1 0
```

Activate particular discovered RAID sets:

```
# dmraid -ay hpt45x_dbagefdi hpt45x_bhchfdeie
```

3.4 Testbed

It is too costly to keep a broad range of ATARAID products in a test environment for regression tests. This would involve plenty of different ATARAID cards and ATARAID-equipped mainboards. Even worse, multiple of each of those would be needed in order to keep various configurations they support acessible for tests in parallel (e.g. Highpoint 47x type cards support RAID0, RAID1, RAID 10 and drive concatination). Not to mention the amount of disks needed to cover those *and* a couple of different sizes for each layout. For the formats already supported by dmraid, the costs easily sum up to a couple of USD 10K.

Because of that, the author created a testbed which utilizes device-mapper to 'fake' ATARAID devices via sparse mapped devices (that's why the examples above show `/dev/dm-*` device names). A sparse mapped device is a stack of a zero and a snapshot mapping on top. The snapshot redirects all writes to the underlying device and keeps track of those redirects while allowing all reads to not-redirected areas to hit the underlying device. In case of the zero mapping, success and a zeroed buffer will be returned to the application. The space where the snapshot redirects writes to (called exception store) can be way smaller than the size of the zero device. That in turn allows the creation of much larger sparse than available physical storage.

The testbed is a directory structure containing subdirectory hierarchies for every vendor, adaptor type and configuration (images of the metadata on each drive and drive size). The top directory holds setup and remove scripts to create and tear down all sparse mapped devices for the drives invloved in the configuration which get called from configuration directory scripts listing them.

A typical subdirectory (e.g. `/dmraid/ataraid.data/hpt/454/raid10`) looks like:

hde.size hdg.size hdi.size hdk.size
hde.dat hdg.dat hdi.dat hdk.dat
setup remove

`dmraid -rD` is able to create the .dat and .size files for supported formats for easy addition to the testbed. Users only need to tar those up and send them to the author on request.

4 dmraid status and futures

dmraid and device-mapper are included in various distributions such as Debian, Fedora, Novell/SuSE, and Red Hat.

Source is available at
`http://people.redhat.com/`

`heinzm/sw/dmraid`
for dmraid and
`http://sources.redhat.com/dm`
for device-mapper.

The mailing list for information exchange on ATARAID themes including dmraid is `ataraid-list@redhat.com`. If you'd like to subscribe to that list, please go to `https://www.redhat.com/mailman/listinfo/ataraid-list`.

Work is in progress to add Fedora installer support for dmraid and device-monitoring via an event daemon (dmeventd) and a libdevmapper interface extension to allow registration of mapped devices for event handling. dmeventd loads application specific dynamic shared objects (e.g. for dmraid or lvm2) and calls into those once an event on a registered device occurs. The DSO can carry out appropriate steps to change mapped device configurations (e.g. activate a spare and start resynchronization of the mirrored set).

Additional metadata format handlers will be added to dmraid including one for SNIA DDF.

The author is open for any proposals which other formats need supporting...