RAID arrays

Boot time assembly of RAID arrays

Tools that manage md devices can be found at
https://www.kernel.org/pub/linux/utils/raid/

You can boot with your md device with the following kernel command lines:

for old raid arrays without persistent superblocks:

```
md=<md device no.>,<raid level>,<chunk size factor>,<fault level>,dev0,dev1,...,devn
```

for raid arrays with persistent superblocks:

```
md=<md device no.>,dev0,dev1,...,devn
```

or, to assemble a partitionable array:

```
md=d<md device no.>,dev0,dev1,...,devn
```

`md device no.`

The number of the md device

`md device no.`	device
0	md0
1	md1
2	md2
3	md3
4	md4

`raid level`

level of the RAID array

`raid level`	level
-1	linear mode
0	striped mode

other modes are only supported with persistent super blocks

`chunk size factor`

(raid-0 and raid-1 only)

Set the chunk size as 4k << n.

`fault level`

Totally ignored

`dev0` to `devn`

e.g. `/dev/hda1`, `/dev/hdc1`, `/dev/sda1`, `/dev/sdb1`

A possible loadlin line (Harald Hoyer <HarryH@Royal.Net>) looks like this:

```
e:\loadlin\loadlin e:\zimage root=/dev/md0 md=0,0,4,0,/dev/hdb2,/dev/hdc3 ro
```

Boot time autodetection of RAID arrays

When md is compiled into the kernel (not as module), partitions of type 0xfd are scanned and automatically assembled into RAID arrays. This autodetection may be suppressed with the kernel parameter `raid=noautodetect`. As of kernel 2.6.9, only drives with a type 0 superblock can be autodetected and run at boot time.

The kernel parameter `raid=partitionable` (or `raid=part`) means that all auto-detected arrays are assembled as partitionable.

Boot time assembly of degraded/dirty arrays

If a raid5 or raid6 array is both dirty and degraded, it could have undetectable data corruption. This is because the fact that it is

`dirty` means that the parity cannot be trusted, and the fact that it is degraded means that some datablocks are missing and cannot reliably be reconstructed (due to no parity).

For this reason, md will normally refuse to start such an array. This requires the sysadmin to take action to explicitly start the array despite possible corruption. This is normally done with:

```
mdadm --assemble --force ....
```

This option is not really available if the array has the root filesystem on it. In order to support this booting from such an array, md supports a module parameter `start_dirty_degraded` which, when set to 1, bypassed the checks and will allows dirty degraded arrays to be started.

So, to boot with a root filesystem of a dirty degraded raid 5 or 6, use:

```
md-mod.start_dirty_degraded=1
```

Superblock formats

The md driver can support a variety of different superblock formats. Currently, it supports superblock formats `0.90.0` and the `md-1` format introduced in the 2.5 development series.

The kernel will autodetect which format superblock is being used.

Superblock format `0` is treated differently to others for legacy reasons - it is the original superblock format.

General Rules - apply for all superblock formats

An array is `created` by writing appropriate superblocks to all devices.

It is `assembled` by associating each of these devices with an particular md virtual device. Once it is completely assembled, it can be accessed.

An array should be created by a user-space tool. This will write superblocks to all devices. It will usually mark the array as `unclean`, or with some devices missing so that the kernel md driver can create appropriate redundancy (copying in raid 1, parity calculation in raid 4/5).

When an array is assembled, it is first initialized with the SET_ARRAY_INFO ioctl. This contains, in particular, a major and minor version number. The major version number selects which superblock format is to be used. The minor number might be used to tune handling of the format, such as suggesting where on each device to look for the superblock.

Then each device is added using the ADD_NEW_DISK ioctl. This provides, in particular, a major and minor number identifying the device to add.

The array is started with the RUN_ARRAY ioctl.

Once started, new devices can be added. They should have an appropriate superblock written to them, and then be passed in with ADD_NEW_DISK.

Devices that have failed or are not yet active can be detached from an array using HOT_REMOVE_DISK.

Specific Rules that apply to format-0 super block arrays, and arrays with no superblock (non-persistent)

An array can be `created` by describing the array (level, chunksize etc) in a SET_ARRAY_INFO ioctl. This must have `major_version==0` and `raid_disks != 0`.

Then uninitialized devices can be added with ADD_NEW_DISK. The structure passed to ADD_NEW_DISK must specify the state of the device and its role in the array.

Once started with RUN_ARRAY, uninitialized spares can be added with HOT_ADD_DISK.

MD devices in sysfs

md devices appear in sysfs (`/sys`) as regular block devices, e.g.:

```
/sys/block/md0
```

Each `md` device will contain a subdirectory called `md` which contains further md-specific information about the device.

All md devices contain:

level

> a text file indicating the `raid level`. e.g. raid0, raid1, raid5, linear, multipath, faulty. If no raid level has been set yet (array is still being assembled), the value will reflect whatever has been written to it, which may be a name like the above, or may be a number such as `0`, `5`, etc.

raid_disks

a text file with a simple number indicating the number of devices in a fully functional array. If this is not yet known, the file will be empty. If an array is being resized this will contain the new number of devices. Some raid levels allow this value to be set while the array is active. This will reconfigure the array. Otherwise it can only be set while assembling an array. A change to this attribute will not be permitted if it would reduce the size of the array. To reduce the number of drives in an e.g. raid5, the array size must first be reduced by setting the `array_size` attribute.

chunk_size

This is the size in bytes for `chunks` and is only relevant to raid levels that involve striping (0,4,5,6,10). The address space of the array is conceptually divided into chunks and consecutive chunks are striped onto neighbouring devices. The size should be at least PAGE_SIZE (4k) and should be a power of 2. This can only be set while assembling an array

layout

The `layout` for the array for the particular level. This is simply a number that is interpreted differently by different levels. It can be written while assembling an array.

array_size

This can be used to artificially constrain the available space in the array to be less than is actually available on the combined devices. Writing a number (in Kilobytes) which is less than the available size will set the size. Any reconfiguration of the array (e.g. adding devices) will not cause the size to change. Writing the word `default` will cause the effective size of the array to be whatever size is actually available based on `level`, `chunk_size` and `component_size`.

This can be used to reduce the size of the array before reducing the number of devices in a raid4/5/6, or to support external metadata formats which mandate such clipping.

reshape_position

This is either `none` or a sector number within the devices of the array where `reshape` is up to. If this is set, the three attributes mentioned above (raid_disks, chunk_size, layout) can potentially have 2 values, an old and a new value. If these values differ, reading the attribute returns:

```
new (old)
```

and writing will effect the `new` value, leaving the `old` unchanged.

component_size

For arrays with data redundancy (i.e. not raid0, linear, faulty, multipath), all components must be the same size - or at least there must a size that they all provide space for. This is a key part or the geometry of the array. It is measured in sectors and can be read from here. Writing to this value may resize the array if the personality supports it (raid1, raid5, raid6), and if the component drives are large enough.

metadata_version

This indicates the format that is being used to record metadata about the array. It can be 0.90 (traditional format), 1.0, 1.1, 1.2 (newer format in varying locations) or `none` indicating that the kernel isn't managing metadata at all. Alternately it can be `external:` followed by a string which is set by user-space. This indicates that metadata is managed by a user-space program. Any device failure or other event that requires a metadata update will cause array activity to be suspended until the event is acknowledged.

resync_start

The point at which resync should start. If no resync is needed, this will be a very large number (or `none` since 2.6.30-rc1). At array creation it will default to 0, though starting the array as `clean` will set it much larger.

new_dev

This file can be written but not read. The value written should be a block device number as major:minor. e.g. 8:0 This will cause that device to be attached to the array, if it is available. It will then appear at md/dev-XXX (depending on the name of the device) and further configuration is then possible.

safe_mode_delay

When an md array has seen no write requests for a certain period of time, it will be marked as `clean`. When another write request arrives, the array is marked as `dirty` before the write commences. This is known as `safe_mode`. The `certain period` is controlled by this file which stores the period as a number of seconds. The default is 200msec (0.200). Writing a value of 0 disables safemode.

array_state

This file contains a single word which describes the current state of the array. In many cases, the state can be set by writing the word for the desired state, however some states cannot be explicitly set, and some transitions are

not allowed.

Select/poll works on this file. All changes except between Active_idle and active (which can be frequent and are not very interesting) are notified. active->active_idle is reported if the metadata is externally managed.

clear

No devices, no size, no level

Writing is equivalent to STOP_ARRAY ioctl

inactive

May have some settings, but array is not active all IO results in error

When written, doesn't tear down array, but just stops it

suspended (not supported yet)

All IO requests will block. The array can be reconfigured.

Writing this, if accepted, will block until array is quiessent

readonly

no resync can happen. no superblocks get written.

Write requests fail

read-auto

like readonly, but behaves like `clean` on a write request.

clean

no pending writes, but otherwise active.

When written to inactive array, starts without resync

If a write request arrives then if metadata is known, mark `dirty` and switch to `active`. if not known, block and switch to write-pending

If written to an active array that has pending writes, then fails.

active

fully active: IO and resync can be happening. When written to inactive array, starts with resync

write-pending

clean, but writes are blocked waiting for `active` to be written.

active-idle

like active, but no writes have been seen for a while (safe_mode_delay).

bitmap/location

This indicates where the write-intent bitmap for the array is stored.

It can be one of `none`, `file` or `[+-]N`. `file` may later be extended to `file:/file/name` `[+-]N` means that many sectors from the start of the metadata.

This is replicated on all devices. For arrays with externally managed metadata, the offset is from the beginning of the device.

bitmap/chunksize

The size, in bytes, of the chunk which will be represented by a single bit. For RAID456, it is a portion of an individual device. For RAID10, it is a portion of the array. For RAID1, it is both (they come to the same thing).

bitmap/time_base

The time, in seconds, between looking for bits in the bitmap to be cleared. In the current implementation, a bit will be cleared between 2 and 3 times `time_base` after all the covered blocks are known to be in-sync.

bitmap/backlog

When write-mostly devices are active in a RAID1, write requests to those devices proceed in the background - the filesystem (or other user of the device) does not have to wait for them. `backlog` sets a limit on the number of concurrent background writes. If there are more than this, new writes will by synchronous.

bitmap/metadata

This can be either `internal` or `external`.

`internal`

is the default and means the metadata for the bitmap is stored in the first 256 bytes of the allocated

space and is managed by the md module.

`external`

means that bitmap metadata is managed externally to the kernel (i.e. by some userspace program)

bitmap/can_clear

This is either `true` or `false`. If `true`, then bits in the bitmap will be cleared when the corresponding blocks are thought to be in-sync. If `false`, bits will never be cleared. This is automatically set to `false` if a write happens on a degraded array, or if the array becomes degraded during a write. When metadata is managed externally, it should be set to true once the array becomes non-degraded, and this fact has been recorded in the metadata.

consistency_policy

This indicates how the array maintains consistency in case of unexpected shutdown. It can be:

none

Array has no redundancy information, e.g. raid0, linear.

resync

Full resync is performed and all redundancy is regenerated when the array is started after unclean shutdown.

bitmap

Resync assisted by a write-intent bitmap.

journal

For raid4/5/6, journal device is used to log transactions and replay after unclean shutdown.

ppl

For raid5 only, Partial Parity Log is used to close the write hole and eliminate resync.

The accepted values when writing to this file are `ppl` and `resync`, used to enable and disable PPL.

uuid

This indicates the UUID of the array in the following format: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

As component devices are added to an md array, they appear in the `md` directory as new directories named:

`dev-XXX`

where `XXX` is a name that the kernel knows for the device, e.g. hdb1. Each directory contains:

block

a symlink to the block device in /sys/block, e.g.:

```
/sys/block/md0/md/dev-hdb1/block -> ../../../../block/hdb/hdb1
```

super

A file containing an image of the superblock read from, or written to, that device.

state

A file recording the current state of the device in the array which can be a comma separated list of:

faulty

device has been kicked from active use due to a detected fault, or it has unacknowledged bad blocks

in_sync

device is a fully in-sync member of the array

writemostly

device will only be subject to read requests if there are no other options.

This applies only to raid1 arrays.

blocked

device has failed, and the failure hasn't been acknowledged yet by the metadata handler.

Writes that would write to this device if it were not faulty are blocked.

spare

device is working, but not a full member.

This includes spares that are in the process of being recovered to

write_error

device has ever seen a write error.

want_replacement

device is (mostly) working but probably should be replaced, either due to errors or due to user request.

replacement

device is a replacement for another active device with same raid_disk.

This list may grow in future.

This can be written to.

Writing `faulty` simulates a failure on the device.

Writing `remove` removes the device from the array.

Writing `writemostly` sets the writemostly flag.

Writing `-writemostly` clears the writemostly flag.

Writing `blocked` sets the `blocked` flag.

Writing `-blocked` clears the `blocked` flags and allows writes to complete and possibly simulates an error.

Writing `in_sync` sets the in_sync flag.

Writing `write_error` sets writeerrorseen flag.

Writing `-write_error` clears writeerrorseen flag.

Writing `want_replacement` is allowed at any time except to a replacement device or a spare. It sets the flag.

Writing `-want_replacement` is allowed at any time. It clears the flag.

Writing `replacement` or `-replacement` is only allowed before starting the array. It sets or clears the flag.

This file responds to select/poll. Any change to `faulty` or `blocked` causes an event.

errors

An approximate count of read errors that have been detected on this device but have not caused the device to be evicted from the array (either because they were corrected or because they happened while the array was read-only). When using version-1 metadata, this value persists across restarts of the array.

This value can be written while assembling an array thus providing an ongoing count for arrays with metadata managed by userspace.

slot

This gives the role that the device has in the array. It will either be `none` if the device is not active in the array (i.e. is a spare or has failed) or an integer less than the `raid_disks` number for the array indicating which position it currently fills. This can only be set while assembling an array. A device for which this is set is assumed to be working.

offset

This gives the location in the device (in sectors from the start) where data from the array will be stored. Any part of the device before this offset is not touched, unless it is used for storing metadata (Formats 1.1 and 1.2).

size

The amount of the device, after the offset, that can be used for storage of data. This will normally be the same as the component_size. This can be written while assembling an array. If a value less than the current component_size is written, it will be rejected.

recovery_start

When the device is not `in_sync`, this records the number of sectors from the start of the device which are known to be correct. This is normally zero, but during a recovery operation it will steadily increase, and if the recovery is interrupted, restoring this value can cause recovery to avoid repeating the earlier blocks. With v1.x metadata, this value is saved and restored automatically.

This can be set whenever the device is not an active member of the array, either before the array is activated, or before the `slot` is set.

Setting this to `none` is equivalent to setting `in_sync`. Setting to any other value also clears the `in_sync` flag.

bad_blocks

This gives the list of all known bad blocks in the form of start address and length (in sectors respectively). If output is too big to fit in a page, it will be truncated. Writing `sector length` to this file adds new acknowledged (i.e. recorded to disk safely) bad blocks.

unacknowledged_bad_blocks

This gives the list of known-but-not-yet-saved-to-disk bad blocks in the same form of `bad_blocks`. If output is too big to fit in a page, it will be truncated. Writing to this file adds bad blocks without acknowledging them. This is largely for testing.

ppl_sector, ppl_size

Location and size (in sectors) of the space used for Partial Parity Log on this device.

An active md device will also contain an entry for each active device in the array. These are named:

```
rdNN
```

where `NN` is the position in the array, starting from 0. So for a 3 drive array there will be rd0, rd1, rd2. These are symbolic links to the appropriate `dev-XXX` entry. Thus, for example:

```
cat /sys/block/md*/md/rd*/state
```

will show `in_sync` on every line.

Active md devices for levels that support data redundancy (1,4,5,6,10) also have

sync_action

a text file that can be used to monitor and control the rebuild process. It contains one word which can be one of:

resync
: redundancy is being recalculated after unclean shutdown or creation

recover
: a hot spare is being built to replace a failed/missing device

idle
: nothing is happening

check
: A full check of redundancy was requested and is happening. This reads all blocks and checks them. A repair may also happen for some raid levels.

repair
: A full check and repair is happening. This is similar to `resync`, but was requested by the user, and the write-intent bitmap is NOT used to optimise the process.

This file is writable, and each of the strings that could be read are meaningful for writing.

`idle` will stop an active resync/recovery etc. There is no guarantee that another resync/recovery may not be automatically started again, though some event will be needed to trigger this.

`resync` or `recovery` can be used to restart the corresponding operation if it was stopped with `idle`.

`check` and `repair` will start the appropriate process providing the current state is `idle`.

This file responds to select/poll. Any important change in the value triggers a poll event. Sometimes the value will briefly be `recover` if a recovery seems to be needed, but cannot be achieved. In that case, the transition to `recover` isn't notified, but the transition away is.

degraded

This contains a count of the number of devices by which the arrays is degraded. So an optimal array will show `0`. A single failed/missing drive will show `1`, etc.

This file responds to select/poll, any increase or decrease in the count of missing devices will trigger an event.

mismatch_count

When performing `check` and `repair`, and possibly when performing `resync`, md will count the number of errors that are found. The count in `mismatch_cnt` is the number of sectors that were re-written, or (for `check`) would have been re-written. As most raid levels work in units of pages rather than sectors, this may be larger than the number of actual errors by a factor of the number of sectors in a page.

bitmap_set_bits

If the array has a write-intent bitmap, then writing to this attribute can set bits in the bitmap, indicating that a resync would need to check the corresponding blocks. Either individual numbers or start-end pairs can be written. Multiple numbers can be separated by a space.

Note that the numbers are `bit` numbers, not `block` numbers. They should be scaled by the bitmap_chunksize.

sync_speed_min, sync_speed_max

This are similar to `/proc/sys/dev/raid/speed_limit_{min,max}` however they only apply to the particular array.

If no value has been written to these, or if the word `system` is written, then the system-wide value is used. If a value, in kibibytes-per-second is written, then it is used.

When the files are read, they show the currently active value followed by `(local)` or `(system)` depending on whether it is a locally set or system-wide value.

sync_completed

This shows the number of sectors that have been completed of whatever the current sync_action is, followed by the number of sectors in total that could need to be processed. The two numbers are separated by a `/` thus effectively showing one value, a fraction of the process that is complete.

A `select` on this attribute will return when resync completes, when it reaches the current sync_max (below) and possibly at other times.

sync_speed

This shows the current actual speed, in K/sec, of the current sync_action. It is averaged over the last 30 seconds.

suspend_lo, suspend_hi

The two values, given as numbers of sectors, indicate a range within the array where IO will be blocked. This is currently only supported for raid4/5/6.

sync_min, sync_max

The two values, given as numbers of sectors, indicate a range within the array where `check`/`repair` will operate. Must be a multiple of chunk_size. When it reaches `sync_max` it will pause, rather than complete. You can use `select` or `poll` on `sync_completed` to wait for that number to reach sync_max. Then you can either increase `sync_max`, or can write `idle` to `sync_action`.

The value of `max` for `sync_max` effectively disables the limit. When a resync is active, the value can only ever be increased, never decreased. The value of `0` is the minimum for `sync_min`.

Each active md device may also have attributes specific to the personality module that manages it. These are specific to the implementation of the module and could change substantially if the implementation changes.

These currently include:

stripe_cache_size (currently raid5 only)
: number of entries in the stripe cache. This is writable, but there are upper and lower limits (32768, 17). Default is 256.

strip_cache_active (currently raid5 only)
: number of active entries in the stripe cache

preread_bypass_threshold (currently raid5 only)
: number of times a stripe requiring preread will be bypassed by a stripe that does not require preread. For fairness defaults to 1. Setting this to 0 disables bypass accounting and requires preread stripes to wait until all full-width stripe- writes are complete. Valid values are 0 to stripe_cache_size.

journal_mode (currently raid5 only)
: The cache mode for raid5. raid5 could include an extra disk for caching. The mode can be "write-throuth" and "write-back". The default is "write-through".

ppl_write_hint
: NVMe stream ID to be set for each PPL write request.