

Manual Documentation Man Page

eltdarcl bocouaceltr

md - Multiple Device driver aka Linux Software Raid

Contents

- 1 SYNOPSIS
- 2 DESCRIPTION
- 3 MD SUPER BLOCK
- 4 ARRAYS WITHOUT SUPERBLOCKS
- 5 LINEAR
- 6 RAID0
- 7 RAID1
- 8 RAID4
- 9 RAID5
- 10 RAID6
- 11 RAID10
- 12 MUTIPATH
- 13 FAULTY
- 14 UNCLEAN SHUTDOWN
- 15 RECOVERY
- 16 BITMAP WRITE-INTENT LOGGING
- 17 WRITE-BEHIND
- 18 RESTRIPING
- 19 SYSFS INTERFACE
- 20 KERNEL PARAMETERS
 - 20.1 FILES
 - 20.2 RELATED
 - 20.3 CATEGORY

SYNOPSIS

```
/dev/mdn  
/dev/md/n
```

DESCRIPTION

The **md** driver provides virtual devices that are created from one or more independent underlying devices. This array of devices often contains redundancy, and hence the acronym RAID which stands for a Redundant Array of Independent Devices.

md supports RAID levels 1 (mirroring), 4 (striped array with parity device), 5 (striped array with distributed parity information), 6 (striped array with distributed dual redundancy information), and 10 (striped and mirrored). If some number of underlying devices fails while using one of these levels, the array will continue to function; this number is one for RAID levels 4 and 5, two for RAID level 6, and all but one (N-1) for RAID level 1, and dependant on configuration for level 10.

md also supports a number of pseudo RAID (non-redundant) configurations including RAID0 (striped array), LINEAR (catenated array), MULTIPATH (a set of different interfaces to the same device), and FAULTY (a layer over a single device into which errors can be injected).

MD SUPER BLOCK

Each device in an array may have a **superblock** which records information about the structure and state of the array. This allows the array to be reliably re-assembled after a shutdown.

From Linux kernel version 2.6.10, **md** provides support for two different formats of this superblock, and other formats can be added. Prior to this release, only one format is supported.

The common format - known as version 0.90 - has a superblock that is 4K long and is written into a 64K aligned block that starts at least 64K and less than 128K from the end of the device (i.e. to get the address of the superblock round the size of the device down to a multiple of 64K and then subtract 64K). The available size of each device is the amount of space before the super block, so between 64K and 128K is lost when a device is incorporated into an MD array. This superblock stores multi-byte fields in a processor-dependant manner, so arrays cannot easily be moved between computers with different processors.

The new format - known as version 1 - has a superblock that is normally 1K long, but can be longer. It is normally stored between 8K and 12K from the end of the device, on a 4K boundary, though variations can be stored at the start of the device (version 1.1) or 4K from the start of the device (version 1.2). This superblock format stores multibyte data in a processor-independent format and has supports up to hundreds of component devices (version 0.90 only supports 28).

The superblock contains, among other things:

LEVEL The manner in which the devices are arranged into the array (linear, raid0, raid1, raid4, raid5, raid10, multipath).

UUID a 128 bit Universally Unique Identifier that identifies the array that this device is part of.

When a version 0.90 array is being reshaped (e.g. adding extra devices to a RAID5), the version number is temporarily set to 0.91. This ensures that if the reshape process is stopped in the middle (e.g. by a system crash) and the machine boots into an older kernel that does not support reshaping, then the array will not be assembled (which would cause data corruption) but will be left untouched until a kernel that can complete the reshape processes is used.

ARRAYS WITHOUT SUPERBLOCKS

While it is usually best to create arrays with superblocks so that they can be assembled reliably, there are some circumstances where an array without superblocks is preferred. This include:

LEGACY ARRAYS

Early versions of the **md** driver only supported Linear and Raid0 configurations and did not use a superblock (which is less critical with these configurations). While such arrays should be rebuilt with superblocks if possible, **md** continues to support them.

FAULTY Being a largely transparent layer over a different device, the **FAULTY** personality doesn't gain anything from having a superblock.

MULTIPATH

It is often possible to detect devices which are different paths to the same storage directly rather than having a distinctive superblock written to the device and searched for on all paths. In this case, a **MULTIPATH** array with no superblock makes sense.

RAID1 In some configurations it might be desired to create a raid1 configuration that does use a superblock, and to maintain the state of the array elsewhere. While not encouraged for general use, it does have special-purpose uses and is supported.

LINEAR

A linear array simply catenates the available space on each drive together to form one large virtual drive.

One advantage of this arrangement over the more common RAID0 arrangement is that the array may be reconfigured at a later time with an extra drive and so the array is made bigger without disturbing the data that is on the array. However this cannot be done on a live array.

If a chunksize is given with a **LINEAR** array, the usable space on each device is rounded down to a multiple of this chunksize.

RAID0

A RAID0 array (which has zero redundancy) is also known as a striped array. A RAID0 array is configured at creation with a **Chunk Size** which must be a power of two, and at least 4 kibibytes.

The RAID0 driver assigns the first chunk of the array to the first device, the second chunk to the second device, and so on until all drives have been assigned one chunk. This collection of chunks forms a **stripe**. Further chunks are gathered into stripes in the same way which are assigned to the remaining space in the drives.

If devices in the array are not all the same size, then once the smallest device has been exhausted, the RAID0 driver starts collecting chunks into smaller stripes that only span the drives which still have remaining space.

RAID1

A RAID1 array is also known as a mirrored set (though mirrors tend to provide reflected images, which RAID1 does not) or a plex.

Once initialised, each device in a RAID1 array contains exactly the same data. Changes are written to all devices in parallel. Data is read from any one device. The driver attempts to distribute read requests across all devices to maximise performance.

All devices in a RAID1 array should be the same size. If they are not, then only the amount of space available on the smallest device is used. Any extra space on other devices is wasted.

RAID4

A RAID4 array is like a RAID0 array with an extra device for storing parity. This device is the last of the active devices in the array. Unlike RAID0, RAID4 also requires that all stripes span all drives, so extra space on devices that are larger than the smallest is wasted.

When any block in a RAID4 array is modified the parity block for that stripe (i.e. the block in the parity device at the same device offset as the stripe) is also modified so that the parity block always contains the "parity" for the whole stripe. i.e. its contents is equivalent to the result of performing an exclusive-or operation between all the data blocks in the stripe.

This allows the array to continue to function if one device fails. The data that was on that device can be calculated as needed from the parity block and the other data blocks.

RAID5

RAID5 is very similar to RAID4. The difference is that the parity blocks for each stripe, instead of being on a single device, are distributed across all devices. This allows more parallelism when writing as two different block updates will quite possibly affect parity blocks on different devices so there is less contention.

This also allows more parallelism when reading as read requests are distributed over all the devices in the array instead of all but one.

RAID6

RAID6 is similar to RAID5, but can handle the loss of any **two** devices without data loss. Accordingly, it requires N+2 drives to store N drives worth of data.

The performance for RAID6 is slightly lower but comparable to RAID5 in normal mode and single disk failure mode. It is very slow in dual disk failure mode, however.

RAID10

RAID10 provides a combination of RAID1 and RAID0, and sometimes known as RAID1+0. Every datablock is duplicated some number of times, and the resulting collection of datablocks are distributed over multiple drives.

When configuring a RAID10 array it is necessary to specify the number of replicas of each data block that are required (this will normally be 2) and whether the replicas should be 'near', 'offset' or 'far'. (Note that the 'offset' layout is only available from 2.6.18).

When 'near' replicas are chosen, the multiple copies of a given chunk are laid out consecutively across the stripes of the array, so the two copies of a datablock will likely be at the same offset on two adjacent devices.

When 'far' replicas are chosen, the multiple copies of a given chunk are laid out quite distant from each other. The first copy of all data blocks will be striped across the early part of all drives in RAID0 fashion, and then the next copy of all blocks will be striped across a later section of all drives, always ensuring that all copies of any given block are on different drives.

The 'far' arrangement can give sequential read performance equal to that of a RAID0 array, but at the cost of degraded write performance.

When 'offset' replicas are chosen, the multiple copies of a given chunk are laid out on consecutive drives and at consecutive offsets. Effectively each stripe is duplicated and the copies are offset by one device. This should give similar read characteristics to 'far' if a suitably large chunk size is used, but without as much seeking for writes.

It should be noted that the number of devices in a RAID10 array need not be a multiple of the number of replica of each data block, those there must be at least as many devices as replicas.

If, for example, an array is created with 5 devices and 2 replicas, then space equivalent to 2.5 of the devices will be available, and every block will be stored on two different devices.

Finally, it is possible to have an array with both 'near' and 'far' copies. If an array is configured with 2 near copies and 2 far copies, then there will be a total of 4 copies of each block, each on a different drive. This is an artifact of the implementation and is unlikely to be of real value.

MULTIPATH

MULTIPATH is not really a RAID at all as there is only one real device in a MULTIPATH md array. However there are multiple access points (paths) to this device, and one of these paths might fail, so there are some similarities.

A MULTIPATH array is composed of a number of logically different devices, often fibre channel interfaces, that all refer to the same real device. If one of these interfaces fails (e.g. due to cable problems), the multipath driver will attempt to redirect requests to another interface.

FAULTY

The FAULTY md module is provided for testing purposes. A faulty array has exactly one component device and is normally assembled without a superblock, so the md array created provides direct access to all of the data in the component device.

The FAULTY module may be requested to simulate faults to allow testing of other md levels or of filesystems. Faults can be chosen to trigger on read requests or write requests, and can be transient (a subsequent read/write at the address will probably succeed) or persistent (subsequent read/write of the same address will fail). Further, read faults can be "fixable" meaning that they persist until a write request at the same address.

Fault types can be requested with a period. In this case the fault will recur repeatedly after the given number of requests of the relevant type. For example if persistent read faults have a period of 100, then every 100th read request would generate a fault, and the faulty sector would be recorded so that subsequent reads on that sector would also fail.

There is a limit to the number of faulty sectors that are remembered. Faults generated after this limit is exhausted are treated as transient.

The list of faulty sectors can be flushed, and the active list of failure modes can be cleared.

UNCLEAN SHUTDOWN

When changes are made to a RAID1, RAID4, RAID5, RAID6, or RAID10 array there is a possibility of inconsistency for short periods of time as each update requires at least two blocks to be written to different devices, and these writes probably won't happen at exactly the same time. Thus if a system with one of these arrays is shut-down in the middle of a write operation (e.g. due to power failure), the array may not be consistent.

To handle this situation, the md driver marks an array as "dirty" before writing any data to it, and marks it as "clean" when the array is being disabled, e.g. at shutdown. If the md driver finds an array to be dirty at startup, it proceeds to correct any possible inconsistency. For RAID1, this involves copying the contents of the first drive onto all other drives. For RAID4, RAID5 and RAID6 this involves recalculating the parity for each stripe and making sure that the parity block has the correct data. For RAID10 it involves copying one of the replicas of each block onto all the others. This process, known as "resynchronising" or "resync" is performed in the background. The array can still be used, though possibly with reduced performance.

If a RAID4, RAID5 or RAID6 array is degraded (missing at least one drive) when it is restarted after an unclean shutdown, it cannot recalculate parity, and so it is possible that data might be undetectably corrupted. The 2.4 md driver **does not** alert the operator to this condition. The 2.6 md driver will fail to start an array in this condition without manual intervention, though this behaviour can be overridden by a kernel parameter.

RECOVERY

If the md driver detects a write error on a device in a RAID1, RAID4, RAID5, RAID6, or RAID10 array, it immediately disables that device (marking it as faulty) and continues operation on the remaining devices. If there is a spare drive, the driver will start recreating on one of the spare drives the data that was on that failed drive, either by copying a working drive in a RAID1 configuration, or by doing calculations with the parity block on RAID4, RAID5 or RAID6, or by finding a copying originals for RAID10.

In kernels prior to about 2.6.15, a read error would cause the same effect as a write error. In later kernels, a read-error will instead cause md to attempt a recovery by overwriting the bad block. i.e. it will find the correct data from elsewhere, write it over the block that failed, and then try to read it back again. If either the write or the re-read fail, md will treat the error the same way that a write error is treated and will fail the whole device.

While this recovery process is happening, the md driver will monitor accesses to the array and will slow down the rate of recovery if other activity is happening, so that normal access to the array will not be unduly affected. When no other activity is happening, the recovery process proceeds at full speed. The actual speed targets for the two different situations can be controlled by the `speed_limit_min` and `speed_limit_max` control files mentioned below.

BITMAP WRITE-INTENT LOGGING

From Linux 2.6.13, **md** supports a bitmap based write-intent log. If configured, the bitmap is used to record which blocks of the array may be out of sync. Before any write request is honoured, md will make sure that the corresponding bit in the log is set. After a period of time with no writes to an area of the array, the corresponding bit will be cleared.

This bitmap is used for two optimisations.

Firstly, after an unclear shutdown, the resync process will consult the bitmap and only resync those blocks that correspond to bits in the bitmap that are set. This can dramatically increase resync time.

Secondly, when a drive fails and is removed from the array, md stops clearing bits in the intent log. If that same drive is re-added to the array, md will notice and will only recover the sections of the drive that are covered by bits in the intent log that are set. This can allow a device to be temporarily removed and reinserted without causing an enormous recovery cost.

The intent log can be stored in a file on a separate device, or it can be stored near the superblocks of an array which has superblocks.

It is possible to add an intent log or an active array, or remove an intent log if one is present.

In 2.6.13, intent bitmaps are only supported with RAID1. Other levels with redundancy are supported from 2.6.15.

WRITE-BEHIND

From Linux 2.6.14, **md** supports WRITE-BEHIND on RAID1 arrays.

This allows certain devices in the array to be flagged as **write-mostly**. MD will only read from such devices if there is no other option.

If a write-intent bitmap is also provided, write requests to write-mostly devices will be treated as write-behind requests and md will not wait for writes to those requests to complete before reporting the write as complete to the filesystem.

This allows for a RAID1 with WRITE-BEHIND to be used to mirror data over a slow link to a remote computer (providing the link isn't too slow). The extra latency of the remote link will not slow down normal operations, but the remote system will still have a reasonably up-to-date copy of all data.

RESTRIPING

Restriping, also known as **Reshaping**, is the processes of re-arranging the data stored in each stripe into a new layout. This might involve changing the number of devices in the array (so the stripes are wider) changing the chunk size (so stripes are deeper or shallower), or changing the arrangement of data and parity, possibly changing the raid level (e.g. 1 to 5 or 5 to 6).

As of Linux 2.6.17, md can reshape a raid5 array to have more devices. Other possibilities may follow in future kernels.

During any stripe process there is a 'critical section' during which live data is being over-written on disk. For the operation of increasing the number of drives in a raid5, this critical section covers the first few stripes (the number being the product of the old and new number of devices). After this critical section is passed, data is only written to areas of the array which no longer hold live data - the live data has already been located away.

md is not able to ensure data preservation if there is a crash (e.g. power failure) during the critical section. If md is asked to start an array which failed during a critical section of restriping, it will fail to start the array.

To deal with this possibility, a user-space program must

• Disable writes to that section of the array (using the **sysfs** interface),

• Take a copy of the data somewhere (i.e. make a backup)

• Allow the process to continue and invalidate the backup and restore write access once the critical section is passed, and

• Provide for restoring the critical data before restarting the array after a system crash.

mdadm version 2.4 and later will do this for growing a RAID5 array.

For operations that do not change the size of the array, like simply increasing chunk size, or converting RAID5 to RAID6 with one extra device, the entire process is the critical section. In this case the restripe will need to progress in stages as a section is suspended, backed up, restriped, and released. This is not yet implemented.

SYSFS INTERFACE

All block devices appear as a directory in **sysfs** (usually mounted at **/sys**). For MD devices, this directory will contain a subdirectory called **md** which contains various files for providing access to information about the array.

This interface is documented more fully in the file **Documentation/md.txt** which is distributed with the kernel sources. That file should be consulted for full documentation. The following are just a selection of attribute files that are available.

md/sync_speed_min

This value, if set, overrides the system-wide setting in **/proc/sys/dev/raid/speed_limit_min** for this array only. Writing the value **system** to this file cause the system-wide setting to have effect.

md/sync_speed_max

This is the partner of **md/sync_speed_min** and overrides **/proc/sys/dev/raid/spool_limit_max** described below.

md/sync_action

This can be used to monitor and control the resync/recovery process of MD. In particular, writing "check" here will cause the array to read all data block and check that they are consistent (e.g. parity is correct, or all mirror replicas are the same). Any discrepancies found are **NOT** corrected.

A count of problems found will be stored in **md/mismatch_count**.

Alternately, "repair" can be written which will cause the same check to be performed, but any errors will be corrected.

Finally, "idle" can be written to stop the check/repair process.

md/stripe_cache_size

This is only available on RAID5 and RAID6. It records the size (in pages per device) of the stripe cache which is used for synchronising all read and write operations to the array. The default is 128. Increasing this number can increase performance in some situations, at some cost in system memory.

KERNEL PARAMETERS

The md driver recognised several different kernel parameters.

raid=noautodetect

This will disable the normal detection of md arrays that happens at boot time. If a drive is partitioned with MS-DOS style partitions, then if any of the 4 main partitions has a partition type of 0xFD, then that partition will normally be inspected to see if it is part of an MD array, and if any full arrays are found, they are started. This kernel parameter disables this behaviour.

raid=partitionable

raid=part

These are available in 2.6 and later kernels only. They indicate that autodetected MD arrays should be created as partitionable arrays, with a different major device number to the original non-partitionable md arrays. The device number is listed as **mdp** in **/proc/devices**.

md_mod.start_ro=1

This tells md to start all arrays in read-only mode. This is a soft read-only that will automatically switch to read-write on the first write request. However until that write request, nothing is written to any device by md, and in particular, no resync or recovery operation is started.

md_mod.start_dirty_degraded=1

As mentioned above, md will not normally start a RAID4, RAID5, or RAID6 that is both dirty and degraded as this situation can imply hidden data loss. This can be awkward if the root filesystem is affected. Using the module parameter allows such arrays to be started at boot time. It should be understood that there is a real (though small) risk of data corruption in this situation.

md=n, dev, dev, ...

md=d_n, dev, dev, ...

This tells the md driver to assemble **/dev/md n** from the listed devices. It is only necessary to start the device holding the root filesystem this way. Other arrays are best started once the system is booted.

In 2.6 kernels, the **d** immediately after the **=** indicates that a partitionable device (e.g. **/dev/md/d0**) should be created rather than the original non-partitionable device.

md=n, l, c, i, dev, ...

This tells the md driver to assemble a legacy RAID0 or LINEAR array without a superblock. **n** gives the md device number, **l** gives the level, 0 for RAID0 or -1 for LINEAR, **c** gives the chunk size as a base-2 logarithm offset by twelve, so 0 means 4K, 1 means 8K. **i** is ignored (legacy support).

FILES

/proc/mdstat

Contains information about the status of currently running array.

/proc/sys/dev/raid/speed_limit_min

A readable and writable file that reflects the current goal rebuild speed for times when non-rebuild activity is current on an array. The speed is in Kibibytes per second, and is a per-device rate, not a per-array rate (which means that an array with more disc will shuffle more data for a given speed). The default is 100.

/proc/sys/dev/raid/speed_limit_max

A readable and writable file that reflects the current goal rebuild speed for times when no non-rebuild activity is current on an array. The default is 100,000.

RELATED

mdadm(8), mkraid(8).

CATEGORY