# 5 Tips To Speed Up Linux Software Raid Rebuilding And Re-syncing

It is no secret that I am a pretty big fan of excellent Linux Software RAID. Creating, assembling and rebuilding small array is fine. But, things started to get nasty when you try to rebuild or re-sync large size array. You may get frustrated when you see it is going to take 22 hours to rebuild the array. You can always increase the speed of Linux Software RAID 0/1/5/6 reconstruction using the following five tips.

## Why speed up Linux software RAID rebuilding and re-syncing?

Recently, I build a small NAS server running Linux for one my client with 5 x 2TB disks in RAID 6 configuration for all in one backup server for Linux, Mac OS X, and Windows XP/Vista/7/10 client computers. Next, I type the command `cat /proc/mdstat` and it reported that md0 is active and recovery is in progress. The recovery speed was around 4000K/sec and will complete in approximately in 22 hours. I wanted to finish this early.

## A note about lazy initialization and ext4 file system

When creating an ext4 file system, the Linux kernel uses lazy initialization. This feature allows the faster creatation of a file system. A process called “ext4lazyinit” runs in the background to create rest of all inode tables. As a result, your RAID rebuild is going to operate at minimal speed. This only affects if you have just created an ext4 filesystem. There is an option to enable or disable this feature while running `mkfs.ext4` command:

1. If enabled and the uninit_bg feature is enabled, the inode table will not be fully initialized by mke2fs. This speeds up filesystem initialization noticeably, but it requires the kernel to finish initializing the filesystem in the background when the filesystem is first mounted. If the option value is omitted, it defaults to 1 to enable lazy inode table zeroing.

```
lazy_itable_init[= <0 to disable, 1 to enable>]
```

2. If enabled, the journal inode will not be fully zeroed out by mke2fs. This speeds up filesystem initialization noticeably, but carries some small risk if the system crashes before the journal has been overwritten entirely one time. If the option value is omitted, it defaults to 1 to enable lazy journal inode zeroing.

```
lazy_journal_init[= <0 to disable, 1 to enable>]
```

# Tip #1:

# `/proc/sys/dev/raid/{speed_limit_max,speed_limit_min} kernel variables`

The `/proc/sys/dev/raid/speed_limit_min` is config 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 (1 kibibyte = 210 bytes = 1024 bytes), and is a per-device rate, not a per-array rate . The default is 1000.

The `/proc/sys/dev/raid/speed_limit_max` is config 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.

To see current limits, enter:

```
sysctl dev.raid.speed_limit_min
sysctl dev.raid.speed_limit_max
```

Sample outputs:

```
dev.raid.speed_limit_min = 10000
dev.raid.speed_limit_max = 20000
```

> NOTE: The following hacks are used for recovering Linux software raid, and to increase the speed of RAID rebuilds. Options are good for tweaking rebuilt process and may increase overall system load, high cpu and memory usage.

To increase speed, enter:

```
echo value > /proc/sys/dev/raid/speed_limit_min
```

OR

```
sysctl -w dev.raid.speed_limit_min=value
```

In this example, set it to 50000 K/Sec, enter:

```
echo 50000 > /proc/sys/dev/raid/speed_limit_min
```

OR

```
sysctl -w dev.raid.speed_limit_min=50000
```

If you want to override the defaults you could add these two lines to /etc/sysctl.conf:

```
#################NOTE ###############
##  You are limited by CPU and memory too #
#########################################
dev.raid.speed_limit_min = 50000
## good for 4-5 disks based array ##
dev.raid.speed_limit_max = 2000000
## good for large 6-12 disks based array ###
dev.raid.speed_limit_max = 5000000
```

# Tip #2: Set read-ahead option

Set readahead (in 512-byte sectors) per raid device. The syntax is:

```
blockdev --setra 65536 /dev/mdX
## Set read-ahead to 32 MiB ##
blockdev --setra 65536 /dev/md0
blockdev --setra 65536 /dev/md1
```

# Tip #3: Set stripe-cache_size for RAID5 or RAID 6

This is only available on **RAID5 and RAID6** and boost sync performance by 3-6 times. It records the size (in pages per device) of the stripe cache which is used for synchronising all write operations to the array and all read operations if the array is degraded. The default is 256. Valid

values are 17 to 32768. Increasing this number can increase performance in some situations, at some cost in system memory. Note, setting this value too high can result in an “out of memory” condition for the system. Use the following formula:

```
memory_consumed = system_page_size * nr_disks * stripe_cache_size
```

To `set stripe_cache_size` to 16 MiB for `/dev/md0`, type:

```
echo 16384 > /sys/block/md0/md/stripe_cache_size
```

To `set stripe_cache_size` to 32 MiB for `/dev/md3`, type:

```
echo 32768 > /sys/block/md3/md/stripe_cache_size
```

# Tip #4: Disable NCQ on all disks

The following will disable NCQ on `/dev/sda`, `/dev/sdb`,.., `/dev/sde` using bash for loop

```
## sample for loop ##
for i in sd[abcde]
do
  echo 1 > /sys/block/$i/device/queue_depth
done
```

# Tip #5: Bitmap Option

Bitmaps optimize rebuild time after a crash, or after removing and re-adding a device. Turn it on by typing the following command:

```
mdadm --grow --bitmap=internal /dev/md0
```

Once array rebuild or fully synced, disable bitmaps:

```
mdadm --grow --bitmap=none /dev/md0
```

# Results

My speed went from 4k to 51k:

```
cat /proc/mdstat
```

Sample outputs:

```
Personalities : [linear] [raid0] [raid1] [raid10] [raid6] [raid5] [raid4] [multipath]
md5 : active raid1 sde2[2](S) sdd2[3](S) sdc2[4](S) sdb2[1] sda2[0]
      530048 blocks [2/2] [UU]

md0 : active raid6 sde3[4] sdd3[3] sdc3[2] sdb3[1] sda3[0]
      5855836800 blocks level 6, 64k chunk, algorithm 2 [5/5] [UUUUU]
      [============>........]  resync = 61.7% (1205475036/1951945600) finish=242.9min
speed=51204K/sec
```

# Monitoring raid rebuilding/recovery process like a pro

You `cat /proc/mdstat` file. This read-only file contains information about the status of currently running array and shows rebuilding speed:

```
cat /proc/mdstat
```

Alternatively use the watch command to display /proc/mdstat output on screen repeatedly, type:

```
watch -n1 cat /proc/mdstat
```

Sample outputs:
1cat-etc-mdstat-command.webp
Fig.01: Performance optimization for Linux raid6 for `/dev/md2`

The following command provide details about `/dev/md2` raid arrray including status and health report:

```
mdadm --detail /dev/md2
```

Sample outputs:
2mdadm-detail-command.webp
Fig.02: Finding out information about md2 raid array

You can filter out info using the grep command or egrep command as follows:

```
mdadm --detail /dev/md3 | grep 'info-you-want'
```

Another option is to see what is actually happening by typing the following iostat command to see disk utilization:

```
watch iostat -k 1 2
watch -n1 iostat -k 1 2
```

Sample outputs:
3raid-iostat-command.webp
Fig.03: Find out CPU statistics and input/output statistics for devices and partitions

Feel free to use the df command or `du` command to get info about the disk space usage on Linux. For example:

```
df -hT /raid1
du -csh /raid1
```

# Conclusion

We learned how to optimize speed for Linux software RAID devices.

See man pages – mdadm(8) using the man command:

```
man 4 md
man 8 mdadm
man 5 proc
```

Also look into `/etc/cron.d/mdadm` and `/usr/share/mdadm/checkarray` on Debian/Ubuntu Linux

Original Article

---

Revision #1
Created 2023-11-10 05:25:18 UTC by Ryan
Updated 2025-02-12 01:12:50 UTC by Ryan