Bad block HOWTO for smartmontools

Bruce Allen <smartmontools-support@lists.sourceforge.net>
Douglas Gilbert <smartmontools-support@lists.sourceforge.net>

```
2007-01-23
Revision History
Revision 1.1  2007-01-23     dpg
add sections on ReiserFS and partition table damage
Revision 1.0  2006-11-14     dpg
merge BadBlockHowTo.txt and BadBlockSCSIHowTo.txt
```

Abstract

This article describes what actions might be taken when smartmontools detects a bad block on a disk. It demonstrates how to identify the file associated with an unreadable disk sector, and how to force that sector to reallocate.

Table of Contents

Introduction

Handling bad blocks is a difficult problem as it often involves decisions about losing information. Modern storage devices tend to handle the simple cases automatically, for example by writing a disk sector that was read with difficulty to another area on the media. Even though such a remapping can be done by a disk drive transparently, there is still a lingering worry about media deterioration and the disk running out of spare sectors to remap.

Can smartmontools help? As the SMART acronym [1] suggests, the smartctl command and the smartd daemon concentrate on monitoring and analysis. So apart from changing some reporting settings, smartmontools will not modify the raw data in a device. Also smartmontools only works with physical devices, it does not know about partitions and file systems. So other tools are needed. The job of smartmontools is to alert the user that something is wrong and user intervention may be required.

When a bad block is reported one approach is to work out the mapping between the logical block address used by a storage device and a file or some other component of a file system using that device. Note that there may not be such a mapping reflecting that a bad block has been found at a location not currently used by the file system. A user may want to do this analysis to localize and minimize the number of replacement files that are retrieved from some backup store. This approach requires knowledge of the file system involved and this document uses the Linux ext2/ext3 and ReiserFS file systems for examples. Also the type of content may come into play. For example if an area storing video has a corrupted sector, it may be easiest to accept that a frame or two might be corrupted and instruct the disk not to retry as that may have the visual effect of causing a momentary blank into a 1 second pause (while the disk retries the faulty sector, often accompanied by a telltale clicking sound).

Another approach is to ignore the upper level consequences (e.g. corrupting a file or worse damage to a file system) and use the facilities offered by a storage device to repair the damage. The SCSI disk command set is used elaborate on this low level approach.

Repairs in a file system

This section contains examples of what to do at the file system level when smartmontools reports a bad block. These examples assume the Linux operating system and either the ext2/ext3 or ReiserFS file system. The various Linux commands shown have man pages and the reader is encouraged to examine these. Of note is the dd command which is often used in repair work [2] and has a unique command line syntax.

The authors would like to thank Sergey Vlasov, Theodore Ts'o, Michael Bendzick, and others for explaining this approach. The authors would like to add text showing how to do this for other file systems, in particular XFS, and JFS: please email if you can provide this information.

ext2/ext3 first example

In this example, the disk is failing self-tests at Logical Block Address LBA = 0x016561e9 = 23421417. The LBA counts sectors in units of 512 bytes, and starts at zero.

```
root]# smartctl -l selftest /dev/hda:

SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Extended offline    Completed: read failure       90%       217         0x016561e9
```

Note that other signs that there is a bad sector on the disk can be found in the non-zero value of the Current Pending Sector count:

```
root]# smartctl -A /dev/hda
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       0
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       1
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       1
```

First Step: We need to locate the partition on which this sector of the disk lives:

```
root]# fdisk -lu /dev/hda

Disk /dev/hda: 123.5 GB, 123522416640 bytes
255 heads, 63 sectors/track, 15017 cylinders, total 241254720 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *        63   4209029   2104483+  83  Linux
/dev/hda2       4209030   5269319    530145   82  Linux swap
/dev/hda3       5269320 238227884 116479282+  83  Linux
/dev/hda4     238227885 241248104   1510110   83  Linux
```

The partition /dev/hda3 starts at LBA 5269320 and extends past the 'problem' LBA. The 'problem' LBA is offset 23421417 - 5269320 = 18152097 sectors into the partition /dev/hda3.

To verify the type of the file system and the mount point, look in /etc/fstab:

```
root]# grep hda3 /etc/fstab
/dev/hda3 /data ext2 defaults 1 2
```

You can see that this is an ext2 file system, mounted at /data.

Second Step: we need to find the block size of the file system (normally 4096 bytes for ext2):

```
root]# tune2fs -l /dev/hda3 | grep Block
Block count:              29119820
Block size:               4096
```

In this case the block size is 4096 bytes. Third Step: we need to determine which File System Block contains this LBA. The formula is:

```
  b = (int)((L-S)*512/B)
```

where:

```
b = File System block number
B = File system block size in bytes
L = LBA of bad sector
S = Starting sector of partition as shown by fdisk -lu
and (int) denotes the integer part.
```

In our example, L=23421417, S=5269320, and B=4096. Hence the 'problem' LBA is in block number

```
   b = (int)18152097*512/4096 = (int)2269012.125
so b=2269012.
```

Note: the fractional part of 0.125 indicates that this problem LBA is actually the second of the eight sectors that make up this file system block.

Fourth Step: we use debugfs to locate the inode stored in this block, and the file that contains that inode:

```
root]# debugfs
debugfs 1.32 (09-Nov-2002)
debugfs:  open /dev/hda3
debugfs:  testb 2269012
Block 2269012 not in use
```

If the block is not in use, as in the above example, then you can skip the rest of this step and go ahead to Step Five.

If, on the other hand, the block is in use, we want to identify the file that uses it:

```
debugfs:  testb 2269012
Block 2269012 marked in use
debugfs:  icheck 2269012
Block   Inode number
2269012 41032
debugfs:  ncheck 41032
Inode   Pathname
41032   /S1/R/H/714197568-714203359/H-R-714202192-16.gwf
```

In this example, you can see that the problematic file (with the mount point included in the path) is:
/data/S1/R/H/714197568-714203359/H-R-714202192-16.gwf

When we are working with an ext3 file system, it may happen that the affected file is the journal itself. Generally, if this is the case, the inode number will be very small. In any case, debugfs will not be able to get the file name:

```
debugfs:  testb 2269012
Block 2269012 marked in use
debugfs:  icheck 2269012
Block   Inode number
2269012 8
debugfs:  ncheck 8
Inode   Pathname
debugfs:
```

To get around this situation, we can remove the journal altogether:

```
tune2fs -O ^has_journal /dev/hda3
```

and then start again with Step Four: we should see this time that the wrong block is not in use any more. If we removed the journal file, at the end of the whole procedure we should remember to rebuild it:

```
tune2fs -j /dev/hda3
```

Fifth Step NOTE: This last step will permanently and irretrievably destroy the contents of the file system block that is damaged: if the block was allocated to a file, some of the data that is in this file is going to be overwritten with zeros. You will not be able to recover that data unless you can replace the file with a fresh or correct version.

To force the disk to reallocate this bad block we'll write zeros to the bad block, and sync the disk:

```
root]# dd if=/dev/zero of=/dev/hda3 bs=4096 count=1 seek=2269012
root]# sync
```

Now everything is back to normal: the sector has been reallocated. Compare the output just below to similar output near the top of this article:

```
root]# smartctl -A /dev/hda
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       1
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       1
```

```
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       1
```

Note: for some disks it may be necessary to update the SMART Attribute values by using smartctl -t offline /dev/hda

We have corrected the first errored block. If more than one blocks were errored, we should repeat all the steps for the subsequent ones. After we do that, the disk will pass its self-tests again:

```
root]# smartctl -t long /dev/hda  [wait until test completes, then]
root]# smartctl -l selftest /dev/hda

SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Extended offline    Completed without error       00%       239         -
# 2  Extended offline    Completed: read failure       90%       217         0x016561e9
# 3  Extended offline    Completed: read failure       90%       212         0x016561e9
# 4  Extended offline    Completed: read failure       90%       181         0x016561e9
# 5  Extended offline    Completed without error       00%        14         -
# 6  Extended offline    Completed without error       00%         4         -
```

and no longer shows any offline uncorrectable sectors:

```
root]# smartctl -A /dev/hda
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       1
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       1
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       0
```

ext2/ext3 second example

On this drive, the first sign of trouble was this email from smartd:

```
    To: ballen
    Subject: SMART error (selftest) detected on host: medusa-slave166.medusa.phys.uwm.edu

    This email was generated by the smartd daemon running on host:
    medusa-slave166.medusa.phys.uwm.edu in the domain: master001-nis

    The following warning/error was logged by the smartd daemon:
    Device: /dev/hda, Self-Test Log error count increased from 0 to 1
```

Running smartctl -a /dev/hda confirmed the problem:

```
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Extended offline    Completed: read failure       80%       682         0x021d9f44

Note that the failing LBA reported is 0x021d9f44 (base 16) = 35495748 (base 10)

ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       0
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       3
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       3
```

and one can see above that there are 3 sectors on the list of pending sectors that the disk can't read but would like to reallocate.

The device also shows errors in the SMART error log:

```
Error 212 occurred at disk power-on lifetime: 690 hours
  After command completion occurred, registers were:
  ER ST SC SN CL CH DH
  -- -- -- -- -- -- -
  40 51 12 46 9f 1d e2  Error: UNC 18 sectors at LBA = 0x021d9f46 = 35495750

  Commands leading to the command that caused the error were:
  CR FR SC SN CL CH DH DC   Timestamp  Command/Feature_Name
  -- -- -- -- -- -- -- --   ---------  --------------------
  25 00 12 46 9f 1d e0 00 2485545.000  READ DMA EXT
```

Signs of trouble at this LBA may also be found in SYSLOG:

```
[root]# grep LBA /var/log/messages | awk '{print $12}' | sort | uniq
 LBAsect=35495748
 LBAsect=35495750
```

So I decide to do a quick check to see how many bad sectors there really are. Using the bash shell I check 70 sectors around the trouble area:

```
[root]# export i=35495730
[root]# while [ $i -lt 35495800 ]
        > do echo $i
        > dd if=/dev/hda of=/dev/null bs=512 count=1 skip=$i
        > let i+=1
        > done

<SNIP>

35495734
1+0 records in
1+0 records out
35495735
dd: reading `/dev/hda': Input/output error
0+0 records in
0+0 records out

<SNIP>

35495751
dd: reading `/dev/hda': Input/output error
0+0 records in
0+0 records out
35495752
1+0 records in
1+0 records out

<SNIP>
```

which shows that the seventeen sectors 35495735-35495751 (inclusive) are not readable.

Next, we identify the files at those locations. The partitioning information on this disk is identical to the first example above, and as in that case the problem sectors are on the third partition /dev/hda3. So we have:

```
     L=35495735 to 35495751
     S=5269320
     B=4096
```

so that b=3778301 to 3778303 are the three bad blocks in the file system.

```
[root]# debugfs
debugfs 1.32 (09-Nov-2002)
debugfs:  open /dev/hda3
debugfs:  icheck 3778301
Block   Inode number
3778301 45192
debugfs:  icheck 3778302
Block   Inode number
3778302 45192
debugfs:  icheck 3778303
Block   Inode number
3778303 45192
debugfs:  ncheck 45192
Inode   Pathname
45192   /S1/R/H/714979488-714985279/H-R-714979984-16.gwf
debugfs:  quit
```

Note that the first few steps of this procedure could also be done with a single command, which is very helpful if there are many bad blocks (thanks to Danie Marais for pointing this out):

```
debugfs: icheck 3778301 3778302 3778303
```

And finally, just to confirm that this is really the damaged file:

```
[root]# md5sum /data/S1/R/H/714979488-714985279/H-R-714979984-16.gwf
md5sum: /data/S1/R/H/714979488-714985279/H-R-714979984-16.gwf: Input/output error
```

Finally we force the disk to reallocate the three bad blocks:

```
[root]# dd if=/dev/zero of=/dev/hda3 bs=4096 count=3 seek=3778301
[root]# sync
```

We could also probably use:

```
[root]# dd if=/dev/zero of=/dev/hda bs=512 count=17 seek=35495735
```

At this point we now have:

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   005    Pre-fail  Always       -       0
196 Reallocated_Event_Count 0x0032   100   100   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0022   100   100   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0008   100   100   000    Old_age   Offline      -       0
```

which is encouraging, since the pending sectors count is now zero. Note that the drive reallocation count has not yet increased: the drive may now have confidence in these sectors and have decided not to reallocate them..

A device self test:

```
  [root#] smartctl -t long /dev/hda
 (then wait about an hour) shows no unreadable sectors or errors:

Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Extended offline    Completed without error       00%       692         -
# 2  Extended offline    Completed: read failure       80%       682         0x021d9f44
```

Unassigned sectors

This section was written by Kay Diederichs. Even though this section assumes Linux and the ext2/ext3 file system, the strategy should be more generally applicable.

I read your badblocks-howto at and greatly benefited from it. One thing that's (maybe) missing is that often the **smartctl -t long** scan finds a bad sector which is not assigned to any file. In that case it does not help to run debugfs, or rather debugfs reports the fact that no file owns that sector. Furthermore, it is somewhat laborious to come up with the correct numbers for debugfs, and debugfs is slow ...

So what I suggest in the case of presence of Current_Pending_Sector/Offline_Uncorrectable errors is to create a huge file on that file system.

```
  dd if=/dev/zero of=/some/mount/point bs=4k
```

creates the file. Leave it running until the partition/file system is full. This will make the disk reallocate those sectors which do not belong to a file. Check the smartctl -a output after that and make sure that the sectors are reallocated. If any remain, use the debugfs method. Of course the usual caveats apply - back it up first, and so on.

ReiserFS example

This section was written by Joachim Jautz with additions from Manfred Schwarb.

The following problems were reported during a scheduled test:

```
smartd[575]: Device: /dev/hda, starting scheduled Offline Immediate Test.
[... 1 hour later …]
smartd[575]: Device: /dev/hda, 1 Currently unreadable (pending) sectors
smartd[575]: Device: /dev/hda, 1 Offline uncorrectable sectors
```

[Step 0] The SMART selftest/error log (see smartctl -l selftest) indicated there was a problem with block address (i.e. the 512 byte sector at) 58656333. The partition table (e.g. see sfdisk -luS /dev/hda or fdisk -ul /dev/hda) indicated that this block was in the /dev/hda3 partition which contained a ReiserFS file system. That partition started at block address 54781650.

While doing the initial analysis it may also be useful to take a copy of the disk attributes returned by smartctl -A /dev/hda. Specifically the values associated with the "Reallocated_Sector_Ct" and "Reallocated_Event_Count" attributes (for ATA disks, the grown list (GLIST) length for SCSI disks). If these are incremented at the end of the procedure it indicates that the disk has re-allocated one or more sectors.

[Step 1] Get the file system's block size:

```
# debugreiserfs /dev/hda3 | grep '^Blocksize'
Blocksize: 4096
```

[Step 2] Calculate the block number:

```
# echo "(58656333-54781650)*512/4096" | bc -l
484335.37500000000000000000
```

It is re-assuring that the calculated 4 KB damaged block address in /dev/hda3 is less than "Count of blocks on the device" shown in the output of debugreiserfs shown above.

[Step 3] Try to get more info about this block => reading the block fails as expected but at least we see now that it seems to be unused. If we do not get the `Cannot read the block' error we should check if our calculation in [Step 2] was correct ;)

```
# debugreiserfs -1 484335 /dev/hda3
debugreiserfs 3.6.19 (2003 http://www.namesys.com)

484335 is free in ondisk bitmap
The problem has occurred looks like a hardware problem.
```

If you have bad blocks, we advise you to get a new hard drive, because once you get one bad block that the disk drive internals cannot hide from your sight, the chances of getting more are generally said to become much higher (precise statistics are unknown to us), and this disk drive is probably not expensive enough for you to risk your time and data on it. If you don't want to follow that advice then if you have just a few bad blocks, try writing to the bad blocks and see if the drive remaps the bad blocks (that means it takes a block it has in reserve and allocates it for use for of that block number). If it cannot remap the block, use badblock option (-B) with reiserfs utils to handle this block correctly.

```
bread: Cannot read the block (484335): (Input/output error).

Aborted
```

So it looks like we have the right (i.e. faulty) block address.

[Step 4] Try then to find the affected file [3]:

```
tar -cO /mydir | cat >/dev/null
```

If you do not find any unreadable files, then the block may be free or located in some metadata of the file system.

[Step 5] Try your luck: bang the affected block with badblocks -n (non-destructive read-write mode, do unmount first), if you are very lucky the failure is transient and you can provoke reallocation [4]:

```
# badblocks -b 4096 -p 3 -s -v -n /dev/hda3 `expr 484335 + 100` `expr 484335 - 100`
```

[5]

check success with **debugreiserfs -1 484335 /dev/hda3**. Otherwise:

[Step 6] Perform this step only if Step 5 has failed to fix the problem: overwrite that block to force reallocation:

```
# dd if=/dev/zero of=/dev/hda3 count=1 bs=4096 seek=484335
1+0 records in
1+0 records out
4096 bytes transferred in 0.007770 seconds (527153 bytes/sec)
```

[Step 7] If you can't rule out the bad block being in metadata, do a file system check:

```
reiserfsck --check
```

This could take a long time so you probably better go for lunch ...

[Step 8] Proceed as stated earlier. For example, sync disk and run a long selftest that should succeed now.

Repairs at the disk level

This section first looks at a damaged partition table. Then it ignores the upper level impact of a bad block and just repairs the underlying sector so that defective sector will not cause problems in the future.

Partition table problems

Some software failures can lead to zeroes or random data being written on the first block of a disk. For disks that use a DOS-based partitioning scheme this will overwrite the partition table which is found at the end of the first block. This is a single point of failure so after the damage tools like fdisk have no alternate data to use so they report no partitions or a damaged partition table.

One utility that may help is testdisk which can scan a disk looking for partitions and recreate a partition table if requested. [6]

Programs that create DOS partitions often place the first partition at logical block address 63. In Linux a loop back mount can be attempted at the appropriate offset of a disk with a damaged partition table. This approach may involve placing the disk with the damaged partition table in a working computer or perhaps an external USB enclosure. Assuming the disk with the damaged partition is /dev/hdb. Then the following read-only loop back mount could be tried:

```
# mount -r /dev/hdb -o loop,offset=32256 /mnt
```

The offset is in bytes so the number given is (63 * 512). If the file system cannot be identified then a '-t <fs_type>' may be needed (although this is not a good sign). If this mount is successful, a backup procedure is advised.

Only the primary DOS partitions are recorded in the first block of a disk. The extended DOS partition table is placed elsewhere on a disk. Again there is only one copy of it so it represents another single point of failure. All DOS partition information can be read in a form that can be used to recreate the tables with the sfdisk command. Obviously this needs to be done beforehand and the file put on other media. Here is how to fetch the partition table information:

```
# sfdisk -dx /dev/hda > my_disk_partition_info.txt
```

Then my_disk_partition_info.txt should be placed on other media. If disaster strikes, then the disk with the damaged partition table(s) can be placed in a working system, let us say the damaged disk is now at /dev/hdc, and the following command restores the partition table(s):

```
# sfdisk -x -O part_block_prior.img /dev/hdc < my_disk_partition_info.txt
```

Since the above command is potentially destructive it takes a copy of the block(s) holding the partition table(s) and puts it in part_block_prior.img prior to any changes. Then it changes the partition tables as indicated by my_disk_partition_info.txt. For what it is worth the author did test this on his system! [7]

For creating, destroying, resizing, checking and copying partitions, and the file systems on them, GNU's parted is worth examining. The Large Disk HOWTO is also a useful resource.

LVM repairs

This section was written by Frederic BOITEUX. It was titled: "HOW TO LOCATE AND REPAIR BAD BLOCKS ON AN LVM VOLUME".

Smartd reports an error in a short test :

```
# smartctl -a /dev/hdb

SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Short offline       Completed: read failure       90%        66         37383668
```

So the disk has a bad block located in LBA block 37383668

In which physical partition is the bad block ?

```
# sfdisk -luS /dev/hdb  # or 'fdisk -ul /dev/hdb'

Disk /dev/hdb: 9729 cylinders, 255 heads, 63 sectors/track
Units = sectors of 512 bytes, counting from 0

   Device Boot    Start       End   #sectors  Id  System
/dev/hdb1            63    996029     995967  82  Linux swap / Solaris
/dev/hdb2   *    996030   1188809     192780  83  Linux
/dev/hdb3       1188810 156296384  155107575  8e  Linux LVM
/dev/hdb4             0         -          0   0  Empty
```

It's in the /dev/hdb3 partition, a LVM2 partition. From the LVM2 partition beginning, the bad block has an offset of

```
(37383668 - 1188810) = 36194858
```

We have to find in which LVM2 logical partition the block belongs to.

In which logical partition is the bad block ?

IMPORTANT : LVM2 can use different schemes dividing its physical partitions to logical ones : linear, striped, contiguous or not... The following example assumes that allocation is linear !

The physical partition used by LVM2 is divided in PE (Physical Extent) units of the same size, starting at pe_start' 512 bytes blocks from the beginning of the physical partition.

The 'pvdisplay' command gives the size of the PE (in KB) of the LVM partition :

```
#  part=/dev/hdb3 ; pvdisplay -c $part | awk -F: '{print $8}'
4096
```

To get its size in LBA block size (512 bytes or 0.5 KB), we multiply this number by 2 : 4096 * 2 = 8192 blocks for each PE.

To find the offset from the beginning of the physical partition is a bit more difficult : if you have a recent LVM2 version, try :

```
# pvs -o+pe_start $part
```

Either, you can look in /etc/lvm/backup :

```
# grep pe_start $(grep -l $part /etc/lvm/backup/*)
                        pe_start = 384
```

Then, we search in which PE is the badblock, calculating the PE rank in which the faulty block of the partition is : physical partition's bad block number / sizeof(PE) =

```
36194858 / 8192 = 4418.3176
```

So we have to find in which LVM2 logical partition is used the PE number 4418 (count starts from 0) :

```
# lvdisplay --maps |egrep 'Physical|LV Name|Type'
  LV Name                /dev/WDC80Go/racine
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    0 to 127
  LV Name                /dev/WDC80Go/usr
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    128 to 1407
  LV Name                /dev/WDC80Go/var
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    1408 to 1663
  LV Name                /dev/WDC80Go/tmp
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    1664 to 1791
  LV Name                /dev/WDC80Go/home
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    1792 to 3071
  LV Name                /dev/WDC80Go/ext1
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    3072 to 10751
  LV Name                /dev/WDC80Go/ext2
    Type                linear
    Physical volume     /dev/hdb3
    Physical extents    10752 to 18932
```

So the PE #4418 is in the /dev/WDC80Go/ext1 LVM logical partition.

Size of logical block of file system on /dev/WDC80Go/ext1 :

It's a ext3 fs, so I get it like this :

```
# dumpe2fs /dev/WDC80Go/ext1 | grep 'Block size'
dumpe2fs 1.37 (21-Mar-2005)
Block size:               4096
```

bad block number for the file system :

The logical partition begins on PE 3072 :

```
 (# PE's start of partition * sizeof(PE)) + parttion offset[pe_start] =
 (3072 * 8192) + 384 = 25166208
```

512b block of the physical partition, so the bad block number for the file system is :

```
(36194858 - 25166208) / (sizeof(fs block) / 512)
= 11028650 / (4096 / 512)  = 1378581.25
```

Test of the fs bad block :

```
dd if=/dev/WDC80Go/ext1 of=block1378581 bs=4096 count=1 skip=1378581
```

If this dd command succeeds, without any error message in console or syslog, then the block number calculation is probably wrong ! ***Don't*** go further, re-check it and if you don't find the error, please renounce !

Search / correction follows the same scheme as for simple partitions :
- find possible impacted files with debugfs (icheck <fs block nb>, then ncheck <icheck nb>).
- reallocate bad block writing zeros in it, *using the fs block size* :

```
dd if=/dev/zero of=/dev/WDC80Go/ext1 count=1 bs=4096 seek=1378581
```

Et voilà !

Bad block reassignment

The SCSI disk command set and associated disk architecture are assumed in this section. SCSI disks have their own logical to physical mapping allowing a damaged sector (usually carrying 512 bytes of data) to be remapped irrespective of the operating system, file system or software RAID being used.

The terms block and sector are used interchangeably, although block tends to get used in higher level or more abstract contexts such as a logical block.

When a SCSI disk is formatted, defective sectors identified during the manufacturing process (the so called primary list: PLIST), those found during the format itself (the certification list: CLIST), those given explicitly to the format command (the DLIST) and optionally the previous grown list (GLIST) are not used in the logical block map. The number (and low level addresses) of the unmapped sectors can be found with the READ DEFECT DATA SCSI command.

SCSI disks tend to be divided into zones which have spare sectors and perhaps spare tracks, to support the logical block address mapping process. The idea is that if a logical block is remapped, the heads do not have to move a long way to access the replacement sector. Note that spare sectors are a scarce resource.

Once a SCSI disk format has completed successfully, other problems may appear over time. These fall into two categories:
- recoverable: the Error Correction Codes (ECC) detect a problem but it is small enough to be corrected. Optionally other strategies such as retrying the access may retrieve the data.
- unrecoverable: try as it may, the disk logic and ECC algorithms cannot recover the data. This is often reported as a medium error.

Other things can go wrong, typically associated with the transport and they will be reported using a term other than medium error. For example a disk may decide a read operation was successful but a computer's host bus adapter (HBA) checking the incoming data detects a CRC error due to a bad cable or termination.

Depending on the disk vendor, recoverable errors can be ignored. After all, some disks have up to 68 bytes of ECC above the payload size of 512 bytes so why use up spare sectors which are limited in number [8] ? If the disk can recover the data and does decide to re-allocate (reassign) a sector, then first it checks the settings of the ARRE and AWRE bits in the read-write error recovery mode page. Usually these bits are set [9] enabling automatic (read or write) re-allocation. The automatic re-allocation may also fail if the zone (or disk) has run out of spare sectors.

Another consideration with RAIDs, and applications that require a high data rate without pauses, is that the controller logic may not want a disk to spend too long trying to recover an error.

Unrecoverable errors will cause a medium error sense key, perhaps with some useful additional sense information. If the extended background self test includes a full disk read scan, one would expect the self test log to list the bad block, as shown in the the section called “Repairs in a file system”. Recent SCSI disks with a periodic background scan should also list unrecoverable read errors (and some recoverable errors as well). The advantage of the background scan is that it runs to completion while self tests will often terminate at the first serious error.

SCSI disks expect unrecoverable errors to be fixed manually using the REASSIGN BLOCKS SCSI command since loss of data is involved. It is possible that an operating system or a file system could issue the REASSIGN BLOCKS command itself but the authors are unaware of any examples. The REASSIGN BLOCKS command will reassign one or more blocks, attempting to (partially ?) recover the data (a forlorn hope at this stage), fetch an unused spare sector from the current zone while adding the damaged old sector to the GLIST (hence the name "grown" list). The contents of the GLIST may not be that interesting but smartctl prints out the number of entries in the grown list and if that number grows quickly, the disk may be approaching the end of its useful life.

Here is an alternate brute force technique to consider: if the data on the SCSI or ATA disk has all been backed up (e.g. is held on the other disks in a RAID 5 enclosure), then simply reformatting the disk may be the least cumbersome approach.

Example

Given a "bad block", it still may be useful to look at the fdisk command (if the disk has multiple partitions) to find out which partition is involved, then use debugfs (or a similar tool for the file system in question) to find out which, if any, file or other part of the file system may have been damaged. This is discussed in the the section called “Repairs in a file system”.

Then a program that can execute the REASSIGN BLOCKS SCSI command is required. In Linux (2.4 and 2.6 series), FreeBSD, Tru64(OSF) and Windows the author's sg_reassign utility in the sg3_utils package can be used. Also found in that package is sg_verify which can be used to check that a block is readable.

Assume that logical block address 1193046 (which is 123456 in hex) is corrupt [10] on the disk at /dev/sdb. A long selftest command like smartctl -t long /dev/sdb may result in log results like this:

```
# smartctl -l selftest /dev/sdb
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce Allen
Home page is http://smartmontools.sourceforge.net/

SMART Self-test log
Num  Test              Status                 segment  LifeTime  LBA_first_err [SK ASC ASQ]
     Description                              number   (hours)
# 1  Background long   Failed in segment      -        354           1193046 [0x3 0x11 0x0]
# 2  Background short  Completed              -        323                 - [-   -    -]
# 3  Background short  Completed              -        194                 - [-   -    -]
```

The sg_verify utility can be used to confirm that there is a problem at that address:

```
# sg_verify --lba=1193046 /dev/sdb
verify (10):  Fixed format, current;  Sense key: Medium Error
 Additional sense: Unrecovered read error
  Info fld=0x123456 [1193046]
  Field replaceable unit code: 228
  Actual retry count: 0x008b
medium or hardware error, reported lba=0x123456
```

Now the GLIST length is checked before the block reassignment:

```
# sg_reassign --grown /dev/sdb
>> Elements in grown defect list: 0
```

And now for the actual reassignment followed by another check of the GLIST length:

```
# sg_reassign --address=1193046 /dev/sdb

# sg_reassign --grown /dev/sdb
>> Elements in grown defect list: 1
```

The GLIST length has grown by one as expected. If the disk was unable to recover any data, then the "new" block at lba 0x123456 has vendor specific data in it. The sg_reassign utility can also do bulk reassigns, see man sg_reassign for more information.

The dd command could be used to read the contents of the "new" block:

```
# dd if=/dev/sdb iflag=direct skip=1193046 of=blk.img bs=512 count=1
```

and a hex editor [11] used to view and potentially change the blk.img file. An altered blk.img file (or /dev/zero) could be written back with:

```
# dd if=blk.img of=/dev/sdb seek=1193046 oflag=direct bs=512 count=1
```

More work may be needed at the file system level, especially if the reassigned block held critical file system information such as a superblock or a directory.

Even if a full backup of the disk is available, or the disk has been "ejected" from a RAID, it may still be worthwhile to reassign the bad block(s) that caused the problem (or simply format the disk (see sg_format in the sg3_utils package)) and re-use the disk later (not unlike the way a replacement disk from a manufacturer might be used).

$Id: badblockhowto.xml 2873 2009-08-11 21:46:20Z dipohl $

[1] Self-Monitoring, Analysis and Reporting Technology -> SMART

[2] Starting with GNU coreutils release 5.3.0, the dd command in Linux includes the options 'iflag=direct' and 'oflag=direct'. Using these with the dd commands should be helpful, because adding these flags should avoid any interaction with the block buffering IO layer in Linux and permit direct reads/writes from the raw device. Use dd --help to see if your version of dd supports these options. If not, the latest code for dd can be found at alpha.gnu.org/gnu/coreutils.

[3] Do not use tar -c -f /dev/null or tar -cO /mydir >/dev/null. GNU tar does not actually read the files if /dev/null is used as archive path or as standard output, see info tar.

[4] Important: set blocksize range is arbitrary, but do not only test a single block, as bad blocks are often social. Not too large as this test probably has not 0% risk.

[5] The rather awkward `expr 484335 + 100` (note the back quotes) can be replaced with $((484335+100)) if the bash shell is being used. Similarly the last argument can become $((484335-100)) .

[6] testdisk scans the media for the beginning of file systems that it recognizes. It can be tricked by data that looks like the beginning of a file system or an old file system from a previous partitioning of the media (disk). So care should be taken. Note that file systems should not overlap apart from the fact that extended partitions lie wholly within a extended partition table allocation. Also if the root partition of a Linux/Unix installation can be found then the /etc/fstab file is a useful resource for finding the partition numbers of other partitions.

[7] Thanks to Manfred Schwarb for the information about storing partition table(s) beforehand.

[8] Detecting and fixing an error with ECC "on the fly" and not going the further step and reassigning the block in question may explain why some disks have large numbers in their read error counter log. Various worried users have reported large numbers in the "errors corrected without substantial delay" counter field which is in the "Errors corrected by ECC fast" column in the smartctl -l error output.

[9] Often disks inside a hardware RAID have the ARRE and AWRE bits cleared (disabled) so the RAID controller can do things manually or flag the disk for replacement.

[10] In this case the corruption was manufactured by using the WRITE LONG SCSI command. See sg_write_long in sg3_utils.

[11] Most window managers have a handy calculator that will do hex to decimal conversions. More work may be needed at the file system level,