How to recover from a broken RAID5

How GNU/Linux saved our data

Edmundo Carmona

In this article I will describe an experience I had that began with the failure of some RAID5 disks at the Hospital of Pediatric Especialties, where I work. While I wouldn't wish such an event on my worst enemy, it was something that made me learn about the power of knowledge—a deep knowledge, which is so important in the hacking culture.

Friday, April 29, 2005

A 5-disk (18GB each) RAID5 was mounted on a HP Netserver Rack Storage/12. Due to a power outage yesterday, it would no longer recognize the RAID. As a matter of fact, there were two more RAIDs on the rack that were recovered... but this one (holding about 60GB of data) just wouldn't work.

The IT manager decided to call in some "gurus" to try to get the data back on-line. I (the only GNU/Linux user at the IT department) thought that something could be done with GNU/Linux. My first thought was: "If I get images of the separate disks, maybe I can start a software RAID on GNU/Linux. All I need is enough disk space to handle all of the images". I told my crazy (so far) idea to the IT manager and he decided to give it a try... but only once the gurus gave up.

Monday, May 2, 2005

The gurus are still trying to get the data back on-line.

Tuesday, May 3, 2005

The gurus are still trying to get the data back on-line.

Wednesday, May 4, 2005

These guys are stubborn, aren't they?

Thursday, May 5, 2005

The IT manager called me late in the afternoon. I was given the chance to *Save the Republic*. One of the disks of the array had been removed. I put the disks on a computer as separate disks (no RAID), booted with Knoppix (the environment of the IT department is Windows based, apart for my desktop, which has the XP that came with the HP box and Mandriva, which is where the computer normally stays) and made the four images of the four disks left from the original five:

```
# for i in a b c d; \
  do dd if=/dev/sd$i of=image$i.dat bs=4k; done
```

I got all the files in a single HD and left the office.

Friday, May 6, 2005

I wanted to start a software RAID, fooling the kernel into thinking that the files where HDs. Just having the images was not enough to bring the RAID on-line. RAID5

has a number of options: algorithm (left/right parity, synchronous/asynchronous), chunk (strip) size, but most important: the order of the images in the RAID. I had to tell the kernel how the RAID controller had mounted them so it could replicate the RAID.

I had already been given the hint that the chunks were 64KB long. By the end of the day, the software RAID idea hadn't worked at all. I started thinking about rebuilding the data the "hard" way: Making a single image of the RAID from the separate images.

Weekend, May 7 and May 8, 2005

I did some research during the weekend, plus a little study of the images. The images didn't look encrypted at all. The first "chunk" of the four images looked like garbage, but one of the disks showed a Partition Table right on the second chunk and the other chunks appeared to have other kind of data:

```
# fdisk -lu discoa1
You must set cylinders.
You can do this from the extra functions menu.

Disk discoa1: 0 MB, 0 bytes
255 heads, 63 sectors/track, 0 cylinders,
  total 0 sectors
Units = sectors of 1 * 512 = 512 bytes

  Device Boot   Start     End   Blocks  Id System
discoa1p1       63  142175249  71087593+  7 HPFS/NTFS
Partition 1 has different physical/logical endings:
   phys=(1023, 254, 63) logical=(8849, 254, 63)
```

`fdisk` was complaining because it was a 64KB file, not the expected 72GB one (written in the partition table). I studied the images and noticed that the data chunks and the parity chunks were distinguishable from each other, and that they seemed to follow a plain RAID5 distribution and algorithm... I was *hopeful*.

I made a java class that could rebuild the RAID content from the separate images (Had I used C/C++, I would still be coding!). It was all about placing the right chunk from the right disk (image of disk) at the right place of the final image. I was missing one image, but it could be calculated with the help of the parity chunks spread all over the disks (see Textbox 1). The class was no big deal: selecting the right chunks from the disks, and using XORs to calculate the missing chunks. I guess it took about three or four hours at most to code it. I was finally ready to give it a try. The problem I hit was that while testing the software RAID at home I had damaged the images. So, I have to wait until Monday to test the class with the images of the RAID.

Table 1 - RAID5's chunk disposition (in a 5-disk array)

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
1	2	3	4	P
5	6	7	P	8
9	10	P	11	12
13	P	14	15	16
P	17	18	19	20
21	22	23	24	P
25	26	27	P	28

RAID stands for *Redundant Array of Independent Disks*. All it does is make a number of disks "look" like they are one to improve throughput or fault-tolerance. There are a number of ways to put them together. Some of them are:

Mirroring: in this case, each disk has exactly the same content. Size of the array: the size of the smallest disk. Redundancy: There must be at least a disk working for the data to remain intact.

Linear: one disk *follows* the other. The size of each disk doesn't matter at all. Size of the array: the sum of the size of the separate disks. Redundancy: If you remove one disk, you will lose the information on that disk and potentially all of the data in the array.

RAID5: The information is spread in all of the available devices in a manner different from linear. Size of the array: the size of the smallest disk multiplied by the number of the disks minus one. Redundancy: At most one disk can be removed/replaced from the array without data loss. Instead of having disks that follow each other, the information is written in "chunks" of data, one disk at a time (see Table 1). In Table 1, the numbers represent the order in which the chunks are written on the disks (in this example, it's left parity, asynchronous). There is a parity chunk per every n – 1 chunks of data. That is done for redundancy.

It works like this: parity is calculated by XORing the n – 1 chunks of data in a row. This logical operator has a very interesting property for redundancy. If you remove one of the data chunks and use the *parity* chunk instead for the XOR operation, you will get the missing chunk of data:

```
a xor b xor c xor d xor e = p
```

If you remove c, then:

```
a xor b xor d xor e xor p = c
```

What does this mean for the RAID? It means that if you remove a whole disk from the array, the RAID can still work... though with a little overhead to calculate the missing chunks. Furthermore, if you replace a missing disk with a new one, the data that was in the removed disk can be rewritten to the new disk. There will be *no* loss of data (provided that no more than a single disk is missing at any given moment).

The process of making a RAID image wasn't complicated. I started the Java class by telling it the conditions of the run: algorithm, images, order of the disks, chunk size, skipped chunks (remember there were 64KB of garbage at the beginning of every image), and output file.

Monday, May 9 2005

I made some attempts at rebuilding the RAID content. Each try took roughly two or three hours. After a run, I had a RAID.dat file (about 72GB in size) that was the "supposed" image of a HD, just like doing:

```
# dd if=/dev/hda of=ata.dat
```

Please notice the *lack* of partition number in the input file (a raw HD block device).

Then I had to use that image as a hard drive. First, I had to use fdisk to know the "partitioning" of the hard drive (it had no problem handling the file at all). At that point, just as I had thought, I discovered that the file was the image of a HD and I could see a partition starting from sector 63. I was more than happy! There were no complaints from fdisk this time. Unfortunately, I can't give you console output from now on, because the files have already been erased. Instead, I'll show the commands that were involved:

```
# fdisk -lu RAID.dat
```

Then mounting. How could I make the kernel think that this file was a hard drive? Well... it took me some more research to learn that *losetup* is used to link *loop devices* to files. It felt like the solution was at hand! I had to *link* the file to a loop device starting from byte 32256 (I had to skip the first 63 sectors 512 bytes each, according to fdisk):

```
#losetup -o 32256 /dev/loop0 RAID.dat
```

It linked, no problem! Then mounted:

```
#mount -t ntfs /dev/loop0 /mnt/tmp
```

There was no complaint when mounting. All of the pieces were fitting together after all.

I just forgot to take into consideration one very important factor in the IT world: Murphy's Law. The RAID was not going to give itself away so easily after all.

When I ran `ls`, in the mount point, I could see *a few* of the directories, but the information wasn't usable. I couldn't `cd` to those directories and `dmesg` said there were problems with the NTFS indexes. I guessed I must have made a mistake ordering the disks... or used the wrong algorithm. I tried twice (with different options), but failed.

Tuesday, May 10 2005

I had left another attempt working when I left the office. That one failed too. I was getting frustrated at the time. Three of the developers at the IT department offered their help and started analysing the whole thing with me. I made another class that rebuilt the missing image, which I felt would help us in the analysis—no matter what the algorithm, order or strip size, according to *RAID theory*, the missing image's content would always be the same.

We noticed that I had indeed made a mistake when ordering the disks! (Hey, I can't always be right, can I?) We studied the images a little further to make sure, and started the whole thing again. It was already getting late, so we had to wait until the next morning to see the results.

Wednesday, May 11 2005

First thing in the morning (and I didn't sleep very well because of the wait), I did a `ls` and...

Eureka! It worked.

All of the directories were there (otherwise, I wouldn't have written this article in the first place, right?). I tried to work with some of the files in the partition... and it was perfect.

I suddenly became the spoiled kid of the IT department! I got a big chocolate cake—that's what I call a bargain!

Even better, the experience caused some of the guys from the IT department to install GNU/Linux on their own personal computers. That's quite an achievement!

Conclusion

I want to finish saying that I did nothing miraculous... but definitely clever! I certainly used the resources I had at hand... plus Knoppix. I also got a lot of help from the GNU/Linux community (through www.linuxquestions.org (http://www.linuxquestions.org) mostly). Thank you people!

It's very important that you make sure backups be made on a regular basis to avoid this kind of situation. I don't think it's likely you will find yourself in the same situation we got ourselves into. But, if you *do* find yourself in the same boat, I hope this information allows you to not lose the data the Microsoft way (just format the disk, and forget about your data). Don't freak out, get a Knoppix CD (if you can get a GNU/Linux guru along with it, all the better!), and with a little programming you will most likely solve the problem.

Thanks

I'd like to thank Simon Carreno, Heberto Ramos and Javier Machado, for their help in analysing the way the images of the RAID had to be put together. I'd also like to thank the IT crew as a whole for their support.

Copyright information

About the author

Edmundo is a Venezuelan Computer Engineer. He is a Java developer at the Hospital of Pediatric Especialties (http://www.espediatricas.com) in Maracaibo, Venezuela. He has also been a GNU/Linux user and consultant for several years.