Data Integrity

Finding Truth in a World of Guesses and Lies

DOUG HUGHES

Doug Hughes is the manager for the infrastructure team at D. E. Shaw Research, LLC, in Manhattan. He is a past LOPSA board member and was the LISA '11 conference co-chair. Doug fell into system administration accidentally, after acquiring a BE in Computer Engineering, and decided that it suited him.
doug@will.to

At LISA '11 in Boston, as I was sitting in a talk [1] on GPFS by Veera Deenadhayalan of IBM, I saw something that I instinctively knew was incorrect. It wasn't anything fundamental to his talk. To the contrary, the work that IBM is doing on GPFS is quite impressive and one of the reasons we had them come to give this talk. There are two main, salutary features of upcoming GPFS versions coming out of the IBM Almaden Research Center. The first is the de-clustering of RAID stripes from full disks, which, to be brief, allows very fast rebuilding of stripes of data across 100,000+ disk systems, where the expectation is that a RAID rebuild will be happening every eight hours or so. The second, and the focus of my article here, is the integration of superior integrity checks built into file systems.

This is nothing new, right? ZFS has been doing this for years. This is definitely worthwhile work, and I'm extremely glad to see this being integrated into other file systems, natively. The thing that struck me was about 33 minutes into the talk on slide 26. It was a reference to the paper "Evaluating the Impact of Undetected Disk Errors in RAID Systems" [2] published in 2009 in the IEEE International Conference on Dependable Systems. This publication clearly nailed the problem, but models indicated that a 1000-disk system would experience an undetected corruption every five years. This is where my mind jumped the rails a little bit, but not for the reasons you might think. I have practical experience that shows that this is extremely optimistic, and my own recorded failure rate is *much* higher than this! I mentioned this to Veera and he requested that I publish these results; I agreed.

The Problem

But first, it may be necessary to take a step back. What is an undetected error, how does one catch it, and why is this a problem? You are probably more familiar with the lengths to which most vendors will go to tell you how safe the data is that they are writing to disk. They will quote MTBF or MTTDL [3] numbers, call your attention to scrubbers for bad data, etc. But, from a data integrity perspective, the question is this: how do you know that the data that you are reading back is what you wrote? That's the crux of the matter. Verifying correct writes at the time of the write is obviously important, but equally so, or perhaps even more important in some cases, is ensuring that you are reading back the correct data.

Disks Lie

Do you trust that your disks are returning to you what you wrote? When you read back a file, there are a panoply of things intermediating between you and your data. First, you have to get the data off the disk. Disks are incredibly complicated bits of near black-magic. It's amazing that they even work at all. They have little electromagnetic heads floating on air cushion mere microns above a rapidly spinning surface of mirror-polished rust—a 7200 RPM 3.5″ disk is moving at about 100 km/h at the outer edge. They have to be at exactly the right place at the right time to read off little chunks of data that are probabilistically encoded and decoded via ultra-fine magnetic fields at multiple depths using quantum effects [4]. The controller collects all of this analog data, analyzes it on the fly, accounts for surface expansion and contraction because of thermal effects, aligns the heads precisely, and uses complex error correction codes to reconstruct the data and turn it into streams of 0's and 1's. What could possibly go wrong!? When your disks lie to you, and they probably have already, do you know?

There are many things that could happen. You could write the data but, because of head alignment or a firmware issue, the data that you wrote might turn into a big chunk of 0's. This is one of the reasons that GPFS writes the data and the checksum to totally different disks. If the data and checksum were written to the same place, and these blocks erroneously became 0's, when you read it back the checksum would match! The checksum or parity would be correct as far as what is on the disk, but it would be wrong with respect to what you intended to be recorded (unless, of course, you are in the habit of storing 0's).

Another possibility is that the magnetic field degrades to such a point that the algorithms used to reconstruct the data guess incorrectly. After all, the retrieved data is an evaluation based upon mathematical best guesses based upon congruent magnetic fields. Add to this the possibility of firmware bugs and you've got a lot of potential for something to happen.

But disks are not the only ant in the colony. Connecting the disk controller to the rest of the system is a cable, which connects to a board with integrated circuits, which passes it over a bus, which passes the data through memory (usually) on its way through several levels of cache through the CPU to the operating system, which has a driver for the disk and a file system to aggregate multiple disks and which uses its own memory and usually passes through the CPU/memory systems several times on its way to a user program, which usually resides on multiple systems (whew). Sometimes there is also a RAID controller card with an ASIC, FPGA, or CPU involved. It just so happens, out of all of these, the disk is complicated and techno-magical enough that it is the most frequent source of errors.

What about client machines over the network? Some file systems, such as GPFS, include network clients natively as an option. Others use the more common NFS or AFS for remote access. These are beyond the scope of this article, for various reasons.

How ZFS helps

In the remainder of this article, I'm going to illustrate the problem using ZFS. ZFS has the advantage of telling you about prevarication up to and through the file system. ZFS verifies the integrity of everything that it touches, including the CPU, memory, cables, down to the disks. Since ZFS reports these problems in the form of easily accessible checksum errors, I can easily share them with you. ZFS is freely available. ZFS uses very strong checksums and verifies every single checksum on read, so you know when there is a phantom flip in any subsystem. It may not know exactly where the flip occurred. That would be hard. But it does associate the error with the disk holding that block, even though it may not be the disk that is at fault. We'll get to that in a little bit.

But, you ask, if ZFS detects it, how can it be undetected? It is because the disk (which I'll use as shorthand for all of the various components connecting the disk to the file system) did not detect that it was bad. It thinks that the data that it is returning is perfectly fine. This is where "undetected" comes from. It is the hardware's inability to realize that the equivalent of mischievous gremlins have been hopping through the data fields kicking over the bit grains without triggering any errors. This last part is important. There may be no other error! The head is fine, the disk platters may be totally fine. There are no timeouts, no bad blocks, etc. These physical media errors have been around for a very long time and systems already know how to deal with these fairly well. (Yet, somehow the exact failure states of disks, firmware issues, bus timeouts, and other ephemera still manage to torpedo us even after all of these years.)

I have been able to collect checksum failure data on a population of about 1000 disks over the course of a couple of years, and hopefully you will gain some appreciation that the scope of the problem is worse than the IBM people thought when they designed GPFS. Serendipitously, my population of ~1000 disks matches the prediction pool from the research paper mentioned earlier. A good detection and correction strategy is a shield from the bit gremlins.

Interpreting the Data

A normal zpool status output for the generic pool zpool1 made of a two-way redundant stripe (raidz2 in ZFS parlance, equivalent to RAID6) looks like this:

```
# zpool status zpool1
pool: zpool1
state: ONLINE
scrub: none requested
config:

NAME       STATE    READ    WRITE   CKSUM
zpool1     ONLINE   0       0       0
raidz2-0   ONLINE   0       0       0
c1t0d0s0   ONLINE   0       0       0
c2t0d0s0   ONLINE   0       0       0
c3t0d0s0   ONLINE   0       0       0
c5t7d0s0   ONLINE   0       0       0
c4t0d0s0   ONLINE   0       0       0
```

There are no read errors, no write errors, and no checksum errors. The checksum error is conveniently stored in the last column for each disk in the pool and all of the errors are tracked separately. Read and write errors are what you might expect: head alignment, media error, controller failure, bus timeout, etc. Most of the read and write errors are also visible to the system via normal error reporting: iostat –E, /var/adm/messages, and the Solaris (if using that) event manager. Checksum errors are as described earlier; the data read back from the disk does not match the checksum that was stored when the data was written. This could be because the checksum block is bad or because the data block is bad, but either way, ZFS flags this as an error in the CKSUM column and automatically corrects the data, if possible, by reconstructing from parity. This is the part that makes ZFS so integral to integrity. For the sake of space, I'll be stripping the headers and other extraneous rows for the following illustrations.

APR 7, 2011

```
        c5t3d0 ONLINE 0  0  1
```

We had one checksum error on c5t3d0. Was it the disk? Was it an undetected ECC flip? Was it a bit flip on a bus somewhere? We don't really know. We wait for further flips and keep the disk under observation. But, had we not had ZFS here, bad data would have been returned to the program—that is certain.

OCT 8, 2010

```
            c0t6d0   ONLINE      0     0     2  5K resilvered
```

Two checksum errors! We can reasonably conclude that this is the disk, because in a many-disk system other subsystem errors would be randomly spread around to other disks. Not only that, but ZFS informs us that it has resilvered the bad data to other disks for us. It's very polite and helpful. Resilvering is the name for the process by which all of the proper data is reconstructed onto the new disk to repair the raidz2 stripe, an homage to repairing old glass mirrors.

SEP 2, 2010

```
        raidz2                 DEGRADED    0    0       0
            c5t5d0s0           ONLINE      0    0       0
            c0t6d0s0           ONLINE      0    0       0
            spare              DEGRADED    0    0   1.21M
              replacing        DEGRADED    0    0       0
                c1t6d0s0/old   FAULTED     0    0       0    corrupted data
                c1t6d0         ONLINE      0    0       0    2.95T resilvered
              c4t7d0           ONLINE      0    0       0    2.95T resilvered
            c2t6d0s0           ONLINE      0    0      20
            c3t6d0s0           ONLINE      0    0      20
```

We have a faulted disk, c1t6d0, that has been replaced, and a spare disk, c4t7d0, that has been swapped in. c1t6d0s0/old represents the original failed disk. c1t6d0 and c4t7d0 are mirrored during the sparing process. We also see 20 checksum errors each on c2t6d0s0 and c3t6d0s0. This is odd, particularly the exact equivalence, and most likely correlated with transient controller issues when the original disk failed, but ZFS was able to correct them on the fly. We chose to ignore these errors and clear them to see if there were any other issues after the resilvering was complete. There were none.

APR 5, 2010

```
        c7t0d0s0         ONLINE     0     0       0
          spare          ONLINE   220     0     212
            c7t1d0s0     ONLINE    25     2     212
            c7t7d0s0     ONLINE     0     0     432    373G resilvered
```

That's a lot of checksum errors, above (April 5, 2010)! But also read (25) and write (2) errors. You can see that c7t1d0s0 was erroring all over the place and likely had fairly severe media defects, perhaps a head dip, a particle of dust, or a random manufacturing defect. c7t7d0s0 was hot-swapped into place and experienced 432 checksum errors while resilvering. /var/adm/messages was quite popular that day. Fortunately, the errors were corrected on the fly. That would have been a data retrieval nightmare if they had been passed to the user program. It turns out that this resilver got stuck and that the replacement disk had an issue. We replaced the replacement and things worked much better.

DEC 15, 2011

```
        c1t4d0     ONLINE   0         0         1
```

Another single corrected error while I write this article. This disk is under observation for further errors.

Conclusion

Of the events recorded above, we have five separate events in an 18-month period. I think there are probably a few others that I was not able to find. Based upon rough memory, it seems that we have a corrected checksum error every six months or so. It would not be an over-stretch to call this 10 otherwise undetected bit flip events in five years, if we aggregate the co-temporal events. However, if we use the raw numbers, the actual number of bit/block corruptions is much higher than this (432 in one event alone)! The more conservative number is 10x the study projection, and I would consider our overall disk reliability to have been pretty good over the last four years.

Hopefully, I haven't frightened you too much with my tales of doom and gloom. There are other ways that you can protect your data without ZFS, such as keeping md5 checksums [5] on every file in the system. One thing that you should do is demand that your storage vendor, who is implementing RAID6 [6], check all reads and verify the parity on every block. This is relatively easy for them to do. It's slightly more overhead to read the blocks off the two parity disks and calculate the codes, but it helps to verify the data is correct: ASICs and CPUs are fast and inexpensive. Parity isn't as good as either the GPFS or ZFS checksum, which is verified on the main CPU, but it's better than nothing.

Many vendors will argue that they have strong guarantees that the data is written correctly. This is not enough! However, in some cases (images and videos come to mind), the bit flip phenomenon is inconsequential. Who cares if a pixel changes color in a movie in a frame? You can make your own judgment about your tolerance for data integrity and pick a solution that is appropriate for your enterprise.

References

[1] IBM talk on improvements to GPFS: http://www.youtube.com/watch?v=2g5rx4gP6yU (or the LISA '11 Web site).

[2] Rozier et al., "Evaluating the Impact of Undetected Disk Errors in RAID Systems": https://www.perform.csl.illinois.edu/Papers/USAN_papers/09ROZ01.pdf.

[3] Richard Elling, "A Story of Two MTTDL Models": http://blogs.oracle.com/relling/entry/a_story_of_two_mttdl.

[4] http://en.wikipedia.org/wiki/Giant_magnetoresistance.

[5] Andrew Hume, "How's Your OS These Days?" ;login:, vol. 30, no. 3, June 2005, USENIX: https://www.usenix.org/publications/login/june-2005-volume-30-number-3/hows-your-os-these-days.

[6] Robin Harris, "Why RAID5 Stops Working in 2009," July 2007: http://www.zdnet.com/blog/storage/why-raid-5-stops-working-in-2009/162.