When Disasters Collide

A Many-Fanged Tale of Woe, Coincidence, Patience, and Stress

DOUG HUGHES AND NATHAN OLLA

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

Nathan Olla is a system administrator at D. E. Shaw Research, LLC., where his focus is on data availability and backup solutions. Nathan found system administration by volunteering over a decade ago to support something called "Linux" while working at Dell.

Never in my career have I experienced as many things go wrong as I did back in late January/early February of 2012. Strangely, not one of them was in any way related to any of the others; they just happened at the same time! Harrowing doesn't begin to cover our feelings in the midst of this maelstrom. I rode out hurricane Opal, as it ripped through Alabama, with less stress. We had four unrelated near-disasters in the span of about four days, and nearly lost 800 TB of data. This is that story, plus a few semi-interesting lessons.

Issue 1

Our problems started with the network. We have monitors in place (e.g., SmokePing [1]) that monitor latency on our wide area network. Our nominal latency is about 11 ms between our primary office site and our primary datacenter site. The network between them is an OC-12 which is an optical, leased line of about 640 Mbps traversing several carriers. We also have a 100 Mbps Internet link which can act as a backup connection, via IPSEC VPN, when the OC-12 is down. Occasionally, one of the OC-12 WAN providers has a maintenance or a minor service-impacting event, and that latency will jump to around 60 ms as the traffic takes an alternate path through the provider's networks, maintaining the same bandwidth. When this happens, the latency consistency (jitter) is relatively consistent, meaning that all packets have a round trip time (RTT) tightly clustered around 60 ms.

An example of a major event would be equipment failure on either end; this takes down the OC-12 connection entirely. On the day of the incident, we saw latency in the 20 ms range but very jittery, which is more characteristic of our backup link on 100 Mbps service. This did not appear to be a standard provider-internal path reroute. We managed to confirm this theory fairly quickly by running iperf [2] between the two sites, yielding a paltry 50 Mbps instead of our more normal 300+ Mbps. We opened a ticket with the primary provider, the one to whom we send the monthly check.

You may notice that the rerouted OC-12 path has much higher latency than the 100 megabit VPN path. We believe this is because the rerouted path is being directed through a distant state before coming back to NYC, although it is difficult to say for certain. There are several carriers who service the end-to-end circuit as we know it. We've seen cases where the local loop can add a significant amount of latency under pathological conditions. The high latency during provider maintenances is puzzling and under active investigation.

A further troubleshooting difficulty with our WAN situation is that we don't actually manage the routers containing the OC-12 linecards. We have a gigabit handoff with a partner organization who manages them for us. This means that when we have a non-trivial difficulty we need to engage this partner organization and the carrier organizations in a coordinated fashion. Such was this case. The carrier checked and re-checked the lines and could not find anything wrong. We've had instances where the OC-12 failback took more than 24 hours, so at first we assumed that this was one of those cases; a particular sub-carrier has been known to close tickets claiming nothing is wrong many times previously. In turn, we've repeatedly had to reopen and insist that "No, 60 milliseconds really isn't normal, there' s definitely something wrong. Please check again!" For this failure, after three days without OC-12, we were starting to worry. The worry was less directed than it might have been because of three other incidents that all happened while this one was occurring, but more on that later.

After many exchanges between carrier and partner, using loopback tests on both sides to prove the circuit was up, we isolated the problem to the datacenter side of the circuit. Upon further examination, we found that the link between the carrier-provided OC-12 equipment and the partner-provided OC-12 router, only about 15 meters of fiber, was the likely culprit. On the carrier side was a long range (LR) single-mode fiber-optical transceiver, and on the partner side was an intermediate range (IR) fiber-optic transceiver. Going by the specifications, the dB losses for both were mismatched. Also, IR transceivers have a range of about 15 km and LR have a range of about 40 km. It might be considered good luck that it managed to work for four years! We suggested that both sides adopt short range (SR) transceivers for the 15 meter distance to avoid overpowering the optical receivers on the other end. Neither organization had SR optics on hand. Both would have had to order the parts, leading to 2–3 more weeks of running on the 100 megabit backup link, which was clearly unacceptable. The local carrier, however, did have an IR transceiver on hand. So, after four days and replacing the LR transceiver we were up and running again on the primary link!

As a side note, the users were relatively unaffected by all of this because of some appliances we use on both ends of our WAN link. Among other useful features, the SilverPeak appliances that we use provide dynamic compression and optimization, network memory, and QoS:

- **Network memory** is the capacity to collect streams of data, store the patterns that haven't been seen before on disk buffers on both ends (keyed with a strong checksum algorithm), and, when that pattern is recognized on the sender side, send the checksum key instead of the entire data stream. The remote side, seeing the key on the WAN interface, pulls the data pattern corresponding to the key from the local disk and feeds it to the requester over the LAN interface. This works in both directions.
- **Compression and optimization** encompasses both TCP header compression and a standard off-the-shelf data compression algorithm to compress the data portion of the packets. It also performs optimization on TCP patterns such as window and buffer sizes, retransmit rates, etc., to get better usage of the available bandwidth. Obviously, compressibility of the data in question is an important factor, but can result in significant bandwidth reductions.

- **QoS** allows one to prioritize:
  - What streams get priority over others, by protocol (ssh, http, nfs, general TCP traffic, etc.)
  - What minimum bandwidth a given protocol is guaranteed in the face of contention
  - What maximum bandwidth rate a protocol can consume
  - Whether a stream is compressed or not (e.g., it makes no sense to try to compress ssh or https, nor to use network memory for either. If it were repeatable data, it wouldn't be secure.).

For us, QoS was the most important feature for running over the lower bandwidth connection, followed by network memory, which meant that patterns that had been requested before and were able to be fed to the requester at gigabit speeds. We prioritize ssh and VNC above all others, so interactive sessions were only mildly impacted. Not a single complaint ticket was originated.

Ponderables:

- Make sure that your backup links work; do periodic testing.
- Having QoS is very useful.
- Make sure that your optical transceivers are appropriately matched and appropriate for the distance of the fiber run.
- Your carrier(s) is/are likely to have escalation protocols. Make sure you know what these are for emergencies.

Issue 2

The second issue struck in the first day of the OC-12 outage. One of our 160 TB (raw) backup storage servers running ZFS lost knowledge of a group of about eight disks. We configured these servers with 6 by 10 disk Raidz2 stripes. This meant that two of the stripes were down two disks. A third disk failure in either of the Raidz2 sets (down two disks already) would mean a very large amount of lost backup data, because ZFS stripes blocks across all six of the Raidz2 stripe sets.

We've experienced peculiarities with the RAID controllers in this system before. Each controller puts a label on every disk to indicate what logical unit (LUN) the disk is a part of. This logical unit can be a simple pass-through, or one of several different hardware RAID configurations. A pass-through is the same as taking a disk that sits behind the controller, stamping a label on it, and passing it through as a logical unit to the host OS (1:1); RAID LUNs are 1:n. The logical unit also includes information such as whether read or write caching should be enabled, among other things. This was the first time a controller had lost its capricious little memory of eight disks at once. Suspiciously, they were all adjacent disks in the chassis, but we could find no subsequent hardware problems, and there were other disks on the same controller that had no issues.

So we did what any normal organization would do under such circumstances. We tried pulling and reseating the disks. It didn't help. We tried checking the SAS cables, which also didn't help. We tried power cycling the entire server. That didn't appear to help either. It turns out that the disks were physically there in the RAID controller view, but had disappeared from the OS view. We ended up having to re-add the RAID controller logical unit (LUN) labels onto the disks to make them show up to the OS. Once we did that, they were visible again in zpool status, but in a very odd way. If you remember my previous article covering ZFS (undetected

bit flips), you'll be familiar with the way that zpool status looks. This is how it was showing up to us on that day (as representative of a single Raidz2 stripe with two disks from the amnesia set):

```
c1t24d0    ONLINE   0    0     0
c1t25d0    ONLINE   0    0     0
c1t26d0    ONLINE   0    0     0
11422982192057997398    FAULTED    0    0    0    was /dev/dsk/c2t0d0s0
3080189289823161725     FAULTED    0    0    0    was /dev/dsk/c2t1d0s0
c2t12d0    ONLINE   0    0     0
```

Instead of a controller number, there's a big integer. Okay, this isn't a huge problem. We can also see that the right column contains information about what the disk was. Hey, that's handy! Or is it? No amount of zpool replace, add, or remove was able to deal with these disks. The zpool commands for removing and replacing disks told us the disks didn't exist, while the commands for adding told us that they were already a member of the pool! We even tried a 14+ hour scrub of the entire zpool, but that didn't recover anything any better.

It turns out that each of the labels for what the disk used to be (right column) also existed elsewhere in the status output for the 82 disks in the system. In other words, the disk mapping was FUBAR when the disks were lost. Unfortunately, we didn't discover this until after I had tried some creative means of fixing these disks. We thought about the best way to recover these disks and were using the name from the right column before realizing that it was a conflict. I used dd to overwrite the disk label to, hopefully, make ZFS forget about the disks so that they could be replaced. This is a bit dangerous, but at the time we felt we had nothing to lose since the disks were already unused by, and unusable to, ZFS. But this made ZFS forget about eight other legitimate, functional, and otherwise healthy disks in the system! Several stripes were now down three disks (below critical) and one was down five disks! Hello, worst case…

Fortunately, we were able to revert this situation by carefully reconstructing which disks were what. After many hours of trying things, we removed the eight offending disks from the system (remember, they were adjacent) and rebooted. The file system was still unrecoverable in this state. We re-inserted the disks and used the Solaris format –e command followed by label to force an EFI label onto the accidentally demolished disks. At this point, ZFS managed to detect the correct info from the remaining disks and we could bring the zpool online. It reconstructed the broken label on the dd'd disks and we were back to where we had been the day before. After running another zpool scrub we determined that dd had blown away data blocks in two legitimate files that were contained in snapshots. In other words, we didn't lose anything of primary value, just a copy of snapshotted copies of two old versions of files. It could have been a lot worse.

The problem remained that the original eight missing disks still had a broken ZFS label on them. We had to very carefully identify them since they did not have a usable /dev/dsk identification that could be used accurately at the system level. Using the long integer was also a failure; ZFS did not like that. Remember, we could neither remove them from ZFS knowledge, because they were not part of the current pool, nor could we add them to the pool, because their label said that they were part of an existing pool. Even using zpool commands with –f would not work.

We had to find a disk that wasn't already thought to be somewhere else in the system. Using zpool information correlated with ls /dev/dsk and the MegaCLI (which manages LUNs) command to map the single disk LUNs to disk numbers in the system, we were able to find a single, spare disk that was not being mapped to a LUN. This was painstaking work. Using this disk, we were able to replace one of the disks with the large integer identifier. Then we deleted that logical unit, re-created it, ran devfsadm to map it to a cXtXdX designation, added it to the zpool as a spare, and were able to use it to replace the next drive, and so on, one after another, 2 TB at a time, until we were done. Only by fixing the labels one at a time were we able to repair the system. Other than losing two old versions of files in snapshots, we made a full recovery, but we were perilously close to losing all of the backups on this system.

Along the way, we had some hilarious exchanges via Jabber, like this classic:

```
|2012-01-30T14:31:34|1|to|N---|that's curious -- grub prompt
|2012-01-30T14:31:48|1|from|N---|hrm.. let's reset again
|2012-01-30T14:32:58|1|from|N---|the 4 flash drives are there
|2012-01-30T14:33:13|1|from|N---|(or 1, divided into 4 chunks, anyway)
|2012-01-30T14:33:15|1|to|N---|they were before as well
|2012-01-30T14:33:40|1|to|N---|great, my console shrunk down to a teeny
window and can't be resized
|2012-01-30T14:34:43|1|from|N---|heh
|2012-01-30T14:34:50|1|from|N---|well, it's at grub again
…
|2012-01-30T18:08:17|1|to|N---|zpool import -Fn + slaughter a chicken?
|2012-01-30T18:08:27|1|to|N---|i guess that won't help
```

Ponderables:

- When you see "was" in ZFS output, make sure that it doesn't match something that is already there.
- You can use dd to wipe out a disk label, and ZFS will still be able to repair it under many circumstances. (This saved our bacon!)
- Use RAID cards that do not require creating an explicit single disk logical unit. RAID cards that do implicit pass-through of disks that are not part of hardware RAID set logical units seem to work much better and be less confusing overall. Certainly they are less trouble to set up on an 80-disk system.
- Check your sanity by double-checking your mapping for apparently missing disks. They may not be what you think they are.
- Locate lights are helpful. Most RAID CLI software supports this.
- If you can't replace a disk with itself because of some intermediate mapping SNAFU, try replacing it with a spare and working step-wise through the disks to get back to normal.

Issue 3

Sometime on the second day of the network outage, we were dismayed to find that an NFS monitor had triggered on one of our two primary application servers (which are exact copies of each other). These servers are not very sizeable machines, but they have enough memory to keep the ZFS adaptive replacement cache (ARC [3]) for all of the actively served NFS files in memory. They also have a level-2 ARC (L2ARC) MLC (multi-level cell) SSD disk for holding more than twice

RAM for any overflow. They also have 10-gigabit network interface cards (NIC) tuned to reduce the number of interrupts when something like a Python application launches and requests 10,000 accesses to NFS paths. Even when un-cached on the client, the server can feed the requested files (or lack thereof) back in ones or tens of microseconds instead of the typical ~10 milliseconds required if they had to be fetched from spinning media.

One of these two servers had gone unreachable. Unfortunately, this meant that roughly half of our cluster machines were stuck on new NFS application requests, owing to two factors:

1. Linux does not support failover to an alternate server for read-only NFS shares. Our NFS clients are all Linux.
2. Independent servers without shared storage cannot be set up as an NFS cluster in Solaris 10.

We started diagnostics by grabbing the serial console and checking for anything egregious. Unfortunately, it was non-responsive. So we issued a remote power cycle command through the lights out management (LOM) interface. It displayed the BIOS flash screen, passed the memory self-test, and as soon as it reached the part where it interacts with the RAID card BIOS, the server got stuck. We tried another reset and it got stuck in the same place. Clearly, something was wrong there.

Deductively, we suspected that the RAID card had gone bad, so we started to swap it with a RAID card from another machine of the same type. Both machines are engineered well enough that this is an operation that can be done without removing the servers from the rack. This took about 20 minutes, because we had to make sure that the other machine was in a suitable state of disuse.

Unfortunately, after the cards were swapped, and we powered the stuck server on, it got stuck in the same place! Unusual...What else could it be? We theorized that perhaps one of the SSD disks that we use for boot and ZFS intent log (ZIL) had gone bad in a particular way; perhaps it was hanging the SATA bus or controller with resets? So we removed the first of the two mirrored SSDs and did another reset. It got stuck in the same spot. We tried again, after pushing the first SSD back in and removing the second. Eureka! Somehow one of the two disks had failed in such a way that it *had* hung the bus, and removing it allowed the machine to boot normally.

Unfortunately, this had resulted in about 2–3 hours of downtime for quite a number of machines during primary work hours.

Ponderables:

- Make sure you have spares readily on hand, even for RAID cards.
- Having machines that allow for swapping RAID cards, CPUs, RAM, etc., without total removal from the rack is useful for important application servers. Had we not had this, it would have easily added another 30 minutes to the procedures.
- SSDs are still relatively new technology and can have unusual failure modes. Try swapping them before the RAID card.

Issue 4

This was the big one. We nearly lost 640 TB of primary storage! Unfortunately, this article has already gotten somewhat long, so we'll leave this as a tantalizing cliffhanger for the next issue!

References

[1] SmokePing: http://oss.oetiker.ch/smokeping/.

[2] Iperf: http://iperf.sourceforge.net.

[3] ARC: http://en.wikipedia.org/wiki/Adaptive_replacement_cache.