Sunday, March 31. 2013

No, ZFS still doesn't need a fsck. Really!

Friday was a day that i called once 10k day. More 10.000 visitors to my blog in one day. Saturday was similar. This surge was create by an link on news.ycombinator.com article i wrote roughly four years ago about ZFS: No, ZFS really doesn't need a fsck.

Just wanted to express that four years later and a lot more experience with ZFS later, 12 years after ZFS saw the light of the word, i'm more of the opinion that ZFS doesn't need a fsck than ever.

I think the key to understand ZFS is to understand that one of the key principles leading development was the concept of not trusting the rotating rust: Checksumming everything, doing copy on write, having redundant metadata even on a single disk, a lot of other small and large ideas. That said, my experience is that ZFS survives an unbelievable amount of abuse before even getting into the state that you need the recovery import feature. Thus many problems that leads to the need for an fsck tool in some other filesystem are just non-existent on ZFS. For example the idea of being always consistent: The filesystem has either the old state or the new state, but never something in between. Helps to counter a lot of problems.

When i discuss with people about the missing fsck, most often they quickly believe that we are protecting the data to counter outside effects like power-offs, unreliability of rotating rust and so on. But the next argument is often "There could be bugs in ZFS. ZFS needs fsck to repair such problems". And, yup â€¦ of course there are bugs in it. Because there must be bugs in it as here is no such thing as a bug free piece of code, at least when it's significantly longer than print "hello world" (that said from programming classes I gave in the past I know that people can even put bugs in such a short piece of code). And no, ZFS still doesn't need a fsck tool. In oder to solve bug i would consider a fsck even counterproductive. My reasoning against fsck is pretty simple:

Why should fsck address a bug that is obviously unknown before, because otherwise it would have been fixed?
When there is a bug in the code that writes or reads the on-disk state, why should the bug be addressed by the fsck code in order to do a successful repair that is more than just forcing the filesystem in a mountable structure? This would assume, that you know of the bug beforehand, but then you could better fix the bug in the code that writes or reads it.

Why should bugs or problems only addressed at fsck time
When there is a bug in the on-disk-state it should be addressed by the code that reads the data and should be repaired by it by correcting it on the fly. This shouldn't be done by a fsck tool, that you just have started when something has gone bad and you have rebooted the system. Or just every thirty reboots.

Why should i repair bugs in a generic manner?
The correction of a bug in the on-disk-state should be done on the basis of the exact knowledge about the bug by a piece of code tailored to solve the bug and not by a generic check tool, that forces the structure on-disk into a structure required by the filesystem.

How do i know without analysis if the bug is in read or write code
The next question interesting in this case: Is the bug in the read part or in the write part of the code. If it's in the read part you would perhaps correct perfectly correct data. The question is: Is it correct after the repair? Or still available?

How do i know if the attempt to repair is correct?
Repair is always based on assumptions. Those assumption canbe correct or incorrect. Thus a repair can be correct and incorrect. The more you know about the problem that led to the repair-worthy state, the more probable the assumptions are correct. Especially when you have to trust the repair for delivering a correct result.

Can i trust the repaired filesystem?
When a ZFS filesystem is that defunct that neither the integrated checks mechanisms and redundancies nor the transaction group rollback can revive it, i would question the integrity of the data altogether and go to my tapes and not trying to force it into function . Especially as the problem has broken obviously a lot of protection layers that would have send many other filesystems to the tapes a significant time before that moment. Because at the end the mountability of a files system doesn't matter. All that matters is the correctness of my data. The problem: How can you guarantee correctness of the data in a filesystem after a repair. Especially with filesystem that can't even guarantee correctness of

data when everything is working fine.

Doing things differently
The recovery of an unmountable filesystem works differently in ZFS. It doesn't need to force an filesystem into a readable state. There are up to 127 readable states on-disk, thus if you want to say it this way, there are up 127 mountable filesystems on your dataset. It sounds more sensible to fall back to the last known correct and consistent state of metadata and data, based on the on-disk-state represented by the pointer structure of the ueberblock with the highest transaction group commit number with a correct checksum . I don't have to repair after a crash, i just take the last intact state.The Transaction Group rollback at mount does exactly this.

I think that is one of the basic points behind the discussion, that many people wanting a fsck doesn't think about the implications of the COW-ness of ZFS. This COW-ness is key to many mechanisms that allows ZFS to do things differently. They make tricks like the PSARC 2009/479 or this script possible.

Conclusion
That said, when you really, really think that you should scratch the data from the disk, you can always use zdb. To end this article: In this discussion, there is just one argument in this discussion i would accept in favour of fsck: ZFS is a blatant layering violation again and what people call fsck is just part of the normal write and read work the filesystem is doing.

```
 Posted by Joerg Moellenkamp in English, Operating Systems, Solaris at 18:21
```

... but ZFS does need a defragmentation tool
Anonymous on Mar 31 2013, 18:47

is bp_rewrite cancelled or will it appear in ZFS for Solaris12?
Anonymous on Mar 31 2013, 19:44

"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair"

Douglas Adams
Anonymous on Mar 31 2013, 19:47

You should mention, though, that ZFS at least has the scrub and resilver command sets which at least in part do what fsck does for other FSs... (and if that job is only to verify the integrity and give the admin a warm fuzzy feeling...)
Anonymous on Apr 1 2013, 00:17

Yes, i feed up doing zfs send | zfs receive to defragment (offline) the zpool
Anonymous on Apr 1 2013, 03:06

ZFS scrubs compute checksums and compare them with result stored on-disk. As most other filesystems doesn't have those checksums with their data, it can't be a part of their checksums. The same is resilvering.
Anonymous on Apr 1 2013, 07:18

I absolutely love ZFS. I've been using it on my home server to safeguard family photos and videos, etc. for years (since OpenSolaris). I use it at work extensively (Solaris 11.1 zone/zfs integration is incredible).

That said, it does lack one critical tool for all, but those who can afford an enterprise backup solution: a zfsdump/zfsrestore utility.

So I've got the most advanced filesystem/volume-management-system/etc. ever released, and I'm using the most ancient software to back it up. I'm using tar for backups, since I can't use zfs send (which I'd actually prefer) due to the fact that my tapes are LTO3 and can only hold 400GB each, while my data is massively larger than that.
The zfs send command (for those who don't know) is not capable of handling "end of tape", so it won't prompt you to change tapes, it just dies with an error when it gets there.
The only way I could use zfs send would be if I were to create many separate zfs filesystems, artificially separating the pictures and videos into groups under 400GB each (which would break the way I have them organized).

I'm amazed it still lacks this feature - as when I search I find postings with people begging for it going back to 2005!

Also: Love the blog - glad you're still doing it after all these years.
Anonymous on Apr 2 2013, 01:53

Stop Using Tapes.

There are two correct ways to back up a zpool:

1. rsync the contents to a separate pool, on separate hardware, preferably in a geographically separate facility. take a snapshot on the remote zpool after each rsync job ends.

2. take frequent snapshots (I prefer hourly). zfs send -RI | zfs receive to a separate pool, on separate hardware, preferably in a geographically separate facility. you don't need to worry about taking snapshots on your remote pool, because your remote pool will have all the snapshots your local pool does.

There are pros and cons to each approach. The big pro for zfs send | zfs receive is that it's RIDICULOUSLY more efficient for large datasets, as it doesn't need to stat files or tokenize data before squirting the delta over the network. The big pro for rsync is that potential bugs in the underlying filesystem can't be transmitted by rsync, whereas they potentially could with zfs send | zfs receive since you're actually moving metadata, not just data.

IME it's actually cheaper to back up to hard drives than it is to tapes anyway. Short life span and everything else wrong with tapes just makes them a complete lose IMO.
Anonymous on Apr 6 2013, 00:07