Wednesday, November 4. 2009

No, ZFS really doesn't need a fsck

There is a discussion at osnews.com about a simple question: "Should ZFS Have a fsck Tool?". The answer is simple: No. I could stop now, as this answer is pretty obvious when you work a while with ZFS, but i want to explain my position. And i want to ask a different question at the end.

I wrote this explanation already in the comment section of osnews.com, so the following text will be a reworked version of my comment at that site.
(Update: There is a new article from March 2013 about this topic - No, ZFS still doesn't need a fsck. Really!)

Filesystems, checks and promisesYes, there are situations that put a zfs pool in a state rendering it unimportable, thus "crashing" the filesystem. And yes, i know ... we say, ZFS is crash-proof and always consistent and so on. Both is true.

We talk about a situation that occurs, when HW components aren't telling the truth about what they do and how they do it. That is in now way the normal use case of any filesystem. And still the statement "consistent and crash-proof" is true, too. Some of the characteristics of ZFS allows you recover into a consistent state even under this circumstances, it just need help to do so. But it doesn't need the help of a fsck tool. It doesn't need such a tool because of difference in its internals when compared to other combinations of LVM and filesystems.

The plague of sub-sub-substandard componentsTo allow ZFS to be crash proof (in the sense of automatically reacting to such a situation), there must be certain really basic mechanisms implemented in a way, that adheres to specifications and standards.

For example: FLUSH CACHE should only return, when the cache is flushed. But there are dirt cheap converter chips that sends the FLUSH CACHE to disk, but returns a successful FLUSH CACHE in the same moment back to the OS (of course without having NVRAM on disk or in a controller as this would allow to ignore CACHE FLUSH). Or interface converters reordering commands in really funny ways. By such reordering it may happen, that the uberblock is written to disk, before the rest of the structure has been written to disk.

Filesystems are hardened against many failures today, but in regard of devices that doesn't even adhere to basic standards, you have no chance. This can happen with any filesystem. Storage components not adhering to the most basic standards are really a plague in IT and i don't have an idea, why companies get away with it. Perhaps because everybody points to the universal scapegoat "Windows" in this case instead of the real root of the problems.

But perhaps it's just because of the fact, that there has to be a special situation to see this problem: You have to stop writing the data before all data has been written to disk, for example by unplugging power or USB from the disk. Because just in this case the point that the new uberblock has been written before the rest of the data is harmful. Normally the new uberblock is written as the last block of any transaction. As long this new uberblock isn't written to disk, the valid on-disk-state is still the old one. Until the new uberblock is on disk, the new blocks are just rubbish after a crash and not part of the on-disk-state.

But now think about the situation, that the uberblock is written before the data because of a converter reordering commands at it's own will. This uberblock points to a state that isn't complete. But as it is the highest ueberblock with a correct checksum, the import of the filesystem starts here. Gotcha - unimportable state when the disk power fails exactly in this moment.

There was another question on osnews.com why NTFS or ext3 aren't complaining about that situation, too. Well ... they have the same problem, but it has different effects, that aren't as visible as in ZFS. ZFS can end with a uberblock pointing to nowhere (that's easily repairable), other filesystems end with an inconsistent state of the data or storage for a database not adhering to the necessities of ACID ("Ladies and Gentleman, start the tapes"). And by the way: With NTFS or ext3 you can't know if your data is consistent and unharmed. You have no way to check it. There is no scrub for this filesystems, as there are no checksums. You just can say, that your filesystem metadata is consistent.

And even more important: Linux has similar problems with problematic hard- and software like components not honoring write barriers. Interestingly one of this problematic components is the LVM. Support for write-barriers in linear modes

isn't that old and only in really fresh versions of Linux the LVM honors write-barriers in more complex RAID levels. Thus in Linux you had to trade in the integrity of your data for the availability or you had to sacrify performance by switching of write caching on the disks. But that's a different story.

At the end all operating systems suffer from components because of bugs or cutting corners in firmware implementations because of a shorter time to market. And we talk about those problems here. Not about the normal "power failure while writting" stuff. Most filesystems are hardended against this problem reasonably well. And ZFS has several protections against the normal dangers in filesystem operation: For example copies of metadata blocks, the described method of a always-consistent on-disk state, checksums on metadata, autorepair functionalities and so on. No ... we talk about a greater and deeper problem.

Why does ZFS need no fsck?With ZFS you can tackle the problem from a different perspective. You do not repair it, you jump to a consistent state slightly before the crash. How does this works? At first you have to keep two things in mind (sorry, simplifications ahead): ZFS works with transaction groups and ZFS is copy-on-write. Furthermore you have to know that there isn't a single uberblock, there are 128 of them, (transaction group number modulo 128 is the exact uberblock used for a certain transaction group).

Given this points, there is a good chance, that you have a consistent state of your filesystem shortly before the crash and that it hasn't overwritten since due to the COW mechanism in ZFS. An effect of the transactional behaviour is the point, that you have older consistent states on your disks as well. At the end the on-disk state is just the consequence of all the transactions that took place from transaction group 0 to the last transaction group. Sound obvious, but this is the key to solution.

ZFS doesn't overwrite live data, so it's perfectly possible that you have several perfectly mountable and perfectly consistent older states of the data in your pool on the disks. In normal situation you don't need this effect, but things are different at the moment: We have the problem, that the last state was corrupted by functionally challenged disks subsystems.

ZFS supports this recovery by another behaviour of it's implementation: As far as i understand the inner workings of ZFS, freed blocks aren't even used again immediately, so getting back to a importable state is almost guaranteed. The reuse of blocks was defered with the putpack of 2009/479. (Okay, at most you can have 127 consistent old states, because there are 128 old versions of the uberblock, but it sounds unlikely that the normal write operations doesn't touch a single freed block while writing back the last 128 transactions group commits and the system doesn't defere reuse that long)

So you just have to rollback the transaction-groups until you have a state that can be scrubbed without errors. Resulting out of this steps you have a recovered state that is consistent and with validated integrity - metadata as well as normal data. You just lost the last few transactions. So, you don't need a fsck tool, you need a tool for a transaction rollback of your filesystem. And tools to check, if you recovered to a consistent state.

Anyway: You do not repair the state last state of the data. And in my opinion: You should not try to repair it ... at least not by automatic means. Such a repair would be risky in any case. Do you really know what the disk has done in which sequence? When it or components on the way to the disks doesn't even adhere to the basics? You have to keep in mind, that we got into this situation because of disks you can't really trust. In this situation i would just take my money from the table and call it a day. You may lose the last few changes, but your tapes are older.

PSARC 2009/479Anyway ... the outcome of PSARC 2009/479 is such a tool mentioned above. You can use those zpool commands to import a unimportable pool, by rolling back transactions made to your filesystem to get back to an importable state, thus other means of correcting the filesystem can work (like redundancies, self-healing et al).

Some may call the results of PSARC 2009/479 something like an fsck tool, but it isn't. It just leverages the transactional behaviour of ZFS to enable other tools to do their work. It just activates another uberblock as the current one.

Of course it would be easy now just to add a functionality to search for the first importable state, but the decision to do a recovery should be an sentient one, not one made by a machine. Perhaps you want to make a disk image of your disk before you try to recover it ... or you want to see a short summary of the stuff the recovery would do without doing the recovery actually.

Surely it took a little long to get such a functionality into the ZFS toolset, but I can just assume, that it wasn't clear, how commonplace such functionally challenged components are.

Just an idea: Rolling Recovery SnapshotsBy the way ... just had an idea: Rolling Recovery Snapshots. The last 10 or 100 transaction groups commits are automatically available as snapshots. As blocks used in snapshots will not used again until the snapshots gets deleted by the automatism, it's ensured, that you have a recovery state in the filesystem that wasn't harmed by the write operations of the interupted transaction group commit in the case you've freed blocks in that shortly before the last transaction group commit. I think, i will write an RfE for it ...

You can't do this with a fsckAll the stuff above isn't done by a fsck tool. You can't rollback transactions with such tool. You can't guarantee the the integrity of the data after the system reported back to you that the filesystem has been recovered. After a successful check of filesystem, you have exactly this: A recovered filesystem. Nothing less, but nothing more as well. At the end, the filesystem is just a tool to enable you to get data from you disk without writing down the sector numbers to get the data back via dd. And it just takes care of it's own problems, not the one of the data. It checks the filesystem, but not the data. It's called fsck and not datack for a reason. So we end of with a mountable filesystem, but the data in it ... that's a different story.

Conclusion and a provocation?ZFS doesn't need a fsck tool, because it doesn't solve the real problem. ZFS needs something better and with all the features of ZFS in conjunction with PSARC 2009/479 it obviously delivers something better.

At the end the solution has to start somewhere else: At first you should to throw the sub-sub-substandard hardware in the next available trash bin after copying the the data to a storage subsystem of better quality and wiping the old disks.

Perhaps after all the correct question should be: How long can other filesystems get away just with a filesystem check but without a data check?

```
Posted by Joerg Moellenkamp in English, Oracle, Solaris at 22:33
```

ZFS doesn't need a fsck tool that checks the entire filesystem... well it has one (IMO) "zpool scrub" However, it should have a way to help repair a pool that is damaged or for some reason won't mount, enough to make it attachable, to salvage intact data, even if only in a read-only state, for example.

For those who say ZFS doesn't need repair tools, I point you to:
http://opensolaris.org/jive/thread.jspa?messageID=289537

And many other hundreds of cases, where people have lost their entire pool, for one reason or another.

Most of the data should be intact, but the drives ignored CACHE FLUSH (or something such as that), and a crash occured, ZFS panics or won't re-attach the pool due to inconsistencies.

The inconsistency may be the user's fault, but ZFS is not off the hook. There should be easy to use recovery tools just like there are with other filesystems.
Anonymous on Nov 4 2009, 23:29

Yes, that is all well and good, but what about memory corruption (to which ECC memory is not immune) and bugs in ZFS, drivers or other parts of the kernel that cause memory corruption, or more specifically, corruption of metadata?

That is something that can pass undetected and be safely written to disk (with correct checksum and all) and be very long-lived, to the point where you can't rollback to a previous uberblock.

In fact this has already happened before, both memory corruption caused by hardware and metadata corruption caused by bugs in ZFS.

That is something that an fsck-like tool could fix and would prevent from having to restore from tape.
Anonymous on Nov 4 2009, 23:31

jimmy in the thread you linked to, you'll notice the final solution is exactly what was discussed in this blog entry. the pool was rolled back to the last usable uberblock and only the last few transactions were lost.
Anonymous on Nov 5 2009, 00:43

Have you actually read Joerg blog entry you are commenting on? He specifically addresses the issue which has been fixed in snv_126.

If you would read the link you provided more carefully you would learn that a user was able to import his pool and access its data thanks to help from Victor. The new functionality implemented in zfs in build 126 basically provides what Victory did manually - importing a pool by using a previous uberblock. For details please re-read the post you are commenting to...
Anonymous on Nov 5 2009, 00:58

I'm sorry but what you are describing is utter nonsense. No fsck tool in any filesystem will be able to fix your data which was already corrupted in a memory before writing it to disk.

And by the nature of the way fsck works in most filesystems it doesn;t even try to fix your data rather it is trying to guess and fix the correct metadata with mixed results. ZFS does much more than that.
Anonymous on Nov 5 2009, 01:05

Jimmy, I think the solution that you ask for has already been put back, and is on the way.

The repair method that Jeff, Mike, and others mention on the discussion thread that you linked to is the same idea that is now the pool recovery option in PSARC 2009/479.

-cheers, CSB
Anonymous on Nov 5 2009, 01:13

Please show me the filesystem and OS that can handle silent memory corruption, lying I/O controllers, meteor strike, &c.

ZFS does everything that it can to preserve integrity, however no software can be clever enough to get around the fundamental problem of "garbage in, garbage out".

Did the hardware lie? Then GIGO applies, whether it's a bad disk controller, or an uncorrected, undetected memory fault.

As for "bugs in ZFS", please be more specific as to which bugs you think are corrupting data. Same goes for bugs in drivers, and the kernel -- give CR numbers, please.

ZFS tries to be robust in the face of the most likely threats. Now that it's been around for a while, the implementors are working on some less likely stuff. Seems like the right priority to me.
Anonymous on Nov 5 2009, 01:23

I think anonymous thinks of not yet known bugs, which may be in zfs. And if a memory corruption happens, he likes to get the most of the data left.

This seems to be reasonable to me, but you can't predict, what a hidden error looks like.

What the zfs team could do, is a tool, which gives you the files on disk, without the path, if this is possible.

Doing predictions on what can happen is pretty useless, like making precise plans for "The Big One", which may hit earth in the future. You don't know when and where, so it's nice to talk about it while having a beer, but there is no urge to start working in the near future.
Anonymous on Nov 5 2009, 11:42

No, it is not "utter nonsense".

Read my post again. I didn't say fsck fixes data corruption, I said it fixes corruption of metadata (yes, metadata can be fixed/cleared and even reconstructed by fsck, as in the case of bitmap/spacemap corruption).
Anonymous on Nov 5 2009, 15:14

I think you don't know how an fsck tool works.

A fsck tool works by traversing the entire filesystem metadata and trying to spot inconsistencies. Once it spots an inconsistency (at least for ext3/4's fsck), it tries to fix it in the way that makes more sense, by assigning a score to each piece of information that is inconsistent, trying to see which one is more likely to be correct.

Of course, that is an heuristic and is not guaranteed to fix it in the correct way, but at least it brings the filesystem metadata back to a consistent state, with (in most cases) a minimal amount of lost data.

Many problems are very easy to fix, for example, orphaned files, bitmap/spacemap corruption (e.g. doubly allocated/crosslinked blocks, leaked blocks, etc), etc, there are tons of things fsck fixes.

All of these problems can also happen with ZFS, it is not immune against hardware memory corruption and bugs. And at least with ext3/ext4, you can bring your filesystem back to a consistent state (read: such a state that the implementation doesn't refuse to import/mount and that doesn't cause panics).

See CR 6458218 for a bug in ZFS that caused metadata corruption, feel free to search for others (which I know there are), because I have more interesting things to do :p
Anonymous on Nov 5 2009, 15:35

Sorry, I actually meant CR 6634517.
Anonymous on Nov 5 2009, 15:39

You can get the files ... you can do this with zdb ... nevertheless it isn't an easy task ...
Anonymous on Nov 5 2009, 22:47

Anonymous,

Yes, fsck can return an inconsistent filesystem to a consistent state. But what do those consistent sectors contain now? Is the data the same as before? Would the FS know the difference?

fsck cannot always guarantee that the consistent FS retains the data integrity that it had before. So fsck can succeed, and still leave you with missing or corrupted data. I've seen it a few times over the years, broken fragments of data in lost+found.

On the other hand -- given non-lying hardware -- ZFS is not vulnerable to this same variety of inconsistency. Is it free from any vulnerability or bug? No, of course not. But it is designed to avoid this type of inconsistency.

The idea with this recovery utility is that you can roll back a few transaction groups, and find something guaranteed to be consistent, both the FS structure and the referenced data.

So I disagree with your statement that ZFS is vulnerable to the same sort of corruption that can be fixed with fsck on other filesystems. Again if you have lying hardware, then all bets are off, regardless of OS or FS.

No FS or fsck in the world can guess as to what a bit ought to have been, if it was written as the result of an undetected memory error, or a broken I/O controller. Adding fsck to ZFS wouldn't help this, either.

As for CR 6634517, this is a dup of 6458218, which was reported back in 2006, and then fixed. Nobody should be affected by it these days.

If you're having trouble with this one, then perhaps you're using a very old OpenSolaris build? I would suggest that it's time to patch or upgrade, like you would with any OS or FS.

At this point, I think that everyone here has done well to explain the difference between ZFS and other filesystems, when it comes to maintaining consistency and data integrity.
Anonymous on Nov 5 2009, 23:50

You're missing the point completely.

Fsck can succeed without leaving you missing or corrupt data.

Corrupted free bitmaps/spacemaps can be fully reconstructed, for example.

ZFS is not immune to spacemap corruption (I pointed you to one of the bugs). Yes, the bug is fixed now, but there were people who had to restore from tape because of that bug. It could have been prevented if ZFS had an fsck tool.

The exact same thing can happen even if ZFS didn't have bugs. Hardware can corrupt a spacemap in memory. Fsck could fix it and prevent users from restoring from tape. I'm not going to repeat this again.

Also, even if fsck would cause you to lose some data, at least you wouldn't lose the entire pool. And in many cases you could easily find out which files had been corrupted or not. In fact, fsck even tells you which files had corrupted metadata.

In fact, if you think a bit, you'd realize that typically when metadata is corrupted (for whatever reason), the damage is very localized and doesn't even affect 99% of the filesystem.

Also, zip files/tarballs have embedded checksums, for example.
git and mercurial have a command to verify the consistency of the workspaces, with the help of their checksums.
I suppose databases could do the same (if they don't already).

What I'm trying to say is that fsck gives you choices that you don't have without it. Yes, it doesn't guarantee that all the data is 100% intact, but if you are fully aware of its limitations, you could use it to your advantage and prevent a lot of lost time.

It doesn't mean that you have to use it, but at least give others a chance to not lose their time.

And guess what, even with ZFS you don't know whether your data is corrupted or not. Again, hardware memory corruption and bugs in ZFS/drivers/the OS can cause your data to be corrupted, it could be a 1-in-a-billion event and you wouldn't know it. Hardware is not perfect, software is not perfect, ZFS is not magical and chances are that none of these will never change.
Anonymous on Nov 6 2009, 12:08

Friend, I tried to think of another way to present the differences between a transactional COW filesystem and other (more traditional) types of FS, but words fail me, so I'll stop now.

I'll leave you with this constructive thought -- you seem passionate about the possibilities of fsck, and about the shortcomings of ZFS. That's just fine, as fortunately for us all, ZFS is open-source.

That being the case, I encourage you to put together a prototype of a ZFS fsck tool as you describe. This would be something fundamentally different than txg rollback.

If you can make ZFS better, i.e. allow it to maintain or recover data integrity whereas the current methods cannot, then I'm sure that the ZFS team would gladly welcome your contribution, and give due credit.

Beyond that, I think you will eventually discover that there is no programmatic way to get around the universal phenomenon of GIGO. It can never be eliminated entirely.

Thanks all... see you next topic. =-) -cheers, CSB
Anonymous on Nov 6 2009, 19:44

This whole debate seems rather political to me, with the fsck/bazaar style mentality opposed to the zfs-integrity/cathedral one.
I'm no fs designer, but, as far as I can understand, one side says "shit happens, so you'd better be prepared to recover for a bad situation, whatever it is, trying to salvage as much data as you can , and leave it up to the user to sort out if the recovered stuff is

useful". The other side says "we build so much checking into the system that data is always good, and in the extremely unlikely chance it is bad, we do not want to spend time trying to fix it, but rather go back to a known state where it was good".

When data loss occurred to me, it was because of cheap hardware and feeding to it too much current from the wrong adapter - after the disk onboard chips were fried the only option I had left (not being willing to pay 500 plus eur for professional data recovery) was recovery of data from another disk where it had been previously - voluntarily - deleted from. I think in such situation the 'recover as much as you can' line of thinking would bear more fruit, but undelete is a different thing altogether from fsck... would zfs be better/worse/equal in such a case?
Anonymous on Nov 7 2009, 12:21

Well ... i don't think of both opinions of an cathedral/bazar distinction. It's more about data availability vs. data integrity thinking. When you look at Linux it leans forward to availability, especiallly when you see performance as a property of availability. Solaris leans to data integrity, even when it costs performance. Linux doesn't use write barriers per default, thus it sacrifies integrity for availability, whereas Solaris switches of the write cache with UFS for example, to ensure that no power failure can harm the data integrity, even when this slows down the system.

I think fsck vs. rollback is the same. With fsck you essentially doesn't know in what state you are getting the data back. With rollback you exactly know that you have a consistent state at the moment designated by the timestamp.

Regarding the undelete: In this case i assume you are better with ZFS: At first practically unlimited number of snapshots (i'm doing one each evening on my fileserver) and at second it's relatively easy to get back a file with zdb (as long as you know a little bit about the on-disk structure) and with COPY on Write and the deferred reuse of blocks, it should be even possible to recover an older version of a file, you have accidentially overwritten (like echo "blah" > wrongfile) when you react immediatly)
Anonymous on Nov 7 2009, 20:57

Simple Question (but slightly offtopic): How can i tell sub-sub-substandard hardware from standard hardware?
Anonymous on Nov 12 2009, 18:23

As no vendor would say, that their devices are substandard and there is no way to check it by a program, the only way it tripping the wire by simulating such an failure ...
Anonymous on Nov 12 2009, 22:23

True, but there are ways to tell some shortcuts have been taken. If things tell you they have worked much faster than is possible for instance you can assume they are lying. It should be possible to test for many common shortcuts and give reports on it.
Anonymous on Jun 29 2010, 04:55

Honestly, for old FS lags like me you could've summarized that as "ZFS has a virtual log structure" and I wouldn't need the rest. A huge oversimplification I know but it expresses the kernel of the reasoning.
Anonymous on Mar 29 2013, 21:55