Checksumming in filesystems, and why ZFS is doing it right

oshogbo.com/blog/73

Feb. 10, 2020, 5:40 p.m.

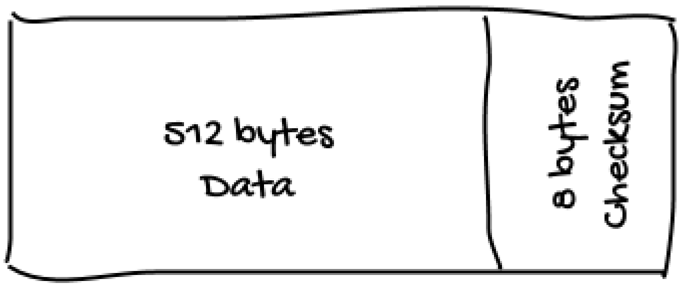
One of the best aspects of ZFS is its reliability. This can be accomplished using a few features like copy-on-write approach and checksumming. Today we will look at how ZFS does checksumming and why it does it the proper way.

Most of the file systems don’t provide any integrity checking and fail in several scenarios:

- *Data bit flips* - when the data that we wanted to store are bit flipped by the hard drives, or cables, and the wrong data is stored on the hard drive.
- *Misdirected writes* - when the CPU/cable/hard drive will bit flip a block to which the data should be written.
- *Misdirected read* - when we miss reading the block when a bit flip occurred.
- *Phantom writes* - when the write operation never made it to the disk. For example, a disk or kernel may have some bug that it will return success even if the hard drive never made the write. This problem can also occur when data is kept only in the hard drive cache.

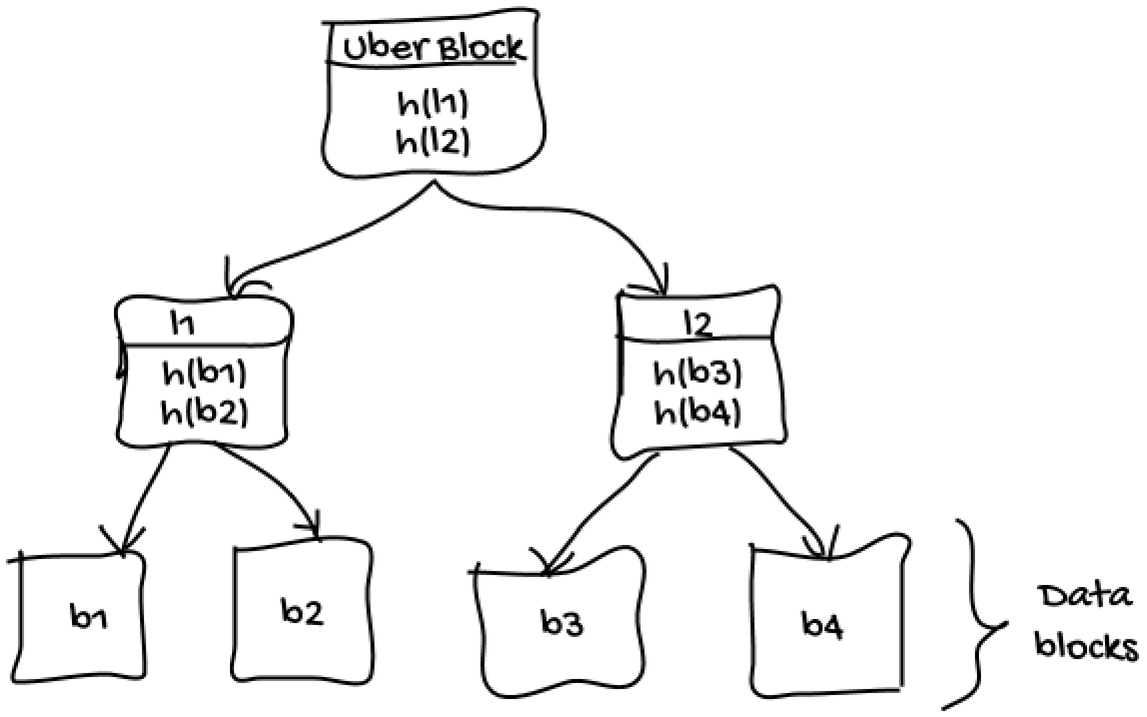
Checksumming may help us detect errors in a few of those situations.

Before jumping into ZFS let’s look at how checksumming could be implemented in a simpler manner, and what problems it addresses and which it doesn't. As an example, we will use NetApp the Write-Anywhere-File-Layout (WAFL) and it’s feature called block checksum (BCS). In BCS the hard drives were formatted with a 520 byte per sector, instead of the standard 512 bytes. Those additional 8 bytes were used for checksumming.



The operating system still saw the disk as 512 byte per sector disk, but when the data from the sector was read, WAFL verified it checksum. This approach helps to detect data bit flips. When the wrong data is read or written to the disk the checksum allows us to verify it. What about the other problems? Unfortunately, this approach doesn't solve them. If the sector was read from a different place then the checksum also was read from that place, and because of that we are unable to detect misdirected reads. If data was misdirected, or there was a phantom write (hard drive will only pretend that data data was stored) there is no way to detect such behavior in this structure. The BCS approach is better than not checksumming at all but it solves only one issue: data bit flips.

ZFS, like always, decided to go with a different approach. The structure that ZFS is using is called a merkle tree. Each node has a checksum of all of its child nodes.



The image above is a little bit simplified but should give you an idea of how data is organized in ZFS. The checksum of the block is not stored with the block data and is instead stored in the node above. It goes up to the uber block which keeps the checksum of it leaves (this is quite simplified because the uber block keeps only one checksum for the object set which keeps the checksum for the rest of the dnodes). Thanks to that we can detect the data bit flips; the checksum will not match the stored data. Also, we can detect misdirected reads: if there was a bit flip on offset the checksum will not match the read data, because the data was read from the wrong offset but the checksum was from the right one, or the other way around. If there was a partial or phantom read, the checksum will not match the stored data. If there was a misdirected write, the data will not match the checksum.

We can see that using a merkle tree is a very safe way to store data on the disk, as it allows us to detect a lot of problems in the system. It is also worth noting that if ZFS is configured in the redundant way, it will detect broken data, return correct data (of course if possible) and also will repair them. This feature is called self-healing.

Bibliography

⇐ Windows Debugging loop in C EKCD - Encrypted Crash Dumps in FreeBSD ⇒