SEE TEXT ONLY

ZFS 2018 and Onward

BY ALLAN JUDE

2018 is going to be a big year for not only FreeBSD, but for OpenZFS as well. OpenZFS is the collaboration of developers from IllumOS, FreeBSD, Linux, Mac OS X, many industry vendors, and a number of other projects to maintain and advance the open-source version of Sun's ZFS filesystem.

Improved Pool Import • 2018Q1

A new patch set changes the way pools are imported, such that they depend less on configuration data from the system. Instead, the possibly outdated configuration from the system is used to find the pool, and partially open it read-only. Then the correct on-disk configuration is loaded. This reduces the number of errors that may prevent a pool from importing. The patch also improves the error messages to better explain what has gone wrong when the pool could not be imported. Additionally, it adds a long-sought-after feature, the ability to import a pool with a missing top-level device. Such imports are read-only since too much data is missing for the pool to be used normally, but some data may be recoverable.

ZFS Device Evacuation • 2018Q1

The ability to remove excess VDEVs from a pool. Data residing on stripe (non-redundant) or mirror VDEVs can be "evacuated", and moved to devices that will remain, allowing the selected VDEV to be removed from the pool. This is accomplished using an indirection table, where ranges of blocks from the removed device are remapped to another device. This feature does not work with RAID-Z VDEVs, and support is not currently planned.

ZFS RAID-Z Expansion • 2018Q4

The ability to add an additional disk to an existing RAID-Z VDEV to grow its size without changing its redundancy level. For example, a RAID-Z2 consisting of 6 disks could be expanded to contain 7 disks, increasing the available space. This is accomplished by reflowing the data across the disks, so as to not change an individual block's offset from the start of the VDEV. The new disk will appear as a new column on the right, and the data will be relocated to maintain the offset within the VDEV. Similar to reflowing a paragraph by making it wider, without changing the order of the words. In the end this means all of the new space shows up at the end of the disk, without any fragmentation. Sadly, existing fragmentation within the VDEV is maintained. In the future it may be possible to use a similar scheme to increase the redundancy level (convert a RAID-Z1 to a RAID-Z2 by adding an additional disk), but that is not part of the current project specification.

ZFS ZStandard Compression • 2018Q2

This project will incorporate Facebook's new Zstandard compression algorithm into ZFS as an optional transparent compression algorithm. This new algorithm is designed to provide compression ratios as good or better than gzip, but many times faster. Developed by Yann Collet, the author of LZ4, the algorithm that has been the default in OpenZFS for many years, ZSTD provides a more enticing trade-off between compression and performance. While not as fast as LZ4, it can achieve much greater compression, and still saturate many spinning disks with just a few CPU cores. It also offers greater control;

with 19 levels of compression to choose from, each dataset can be configured with the optimal amount of compression.

ZFS Adaptive Compression • Under Investigation

The adaptive compression feature is still undergoing preliminary investigation, but, if implemented, would see ZFS automatically adjust the compression ratio up and down as the data is being written, depending on the amount of dirty data waiting to be compressed and written. When the system is not busy, additional CPU time can be allocated to compressing data, but when the throughput of the compression is not able to keep up with the demand of writes, the compression level will be lowered to avoid becoming a bottleneck.

FreeBSD ZFS Spare and Fault Management • 2019Q1

Enhancements to the way FreeBSD boots from ZFS, to bring it more in line with the original design of ZFS. These changes will obviate the need for a freebsd-boot partition and allow the creation of the partition table to be handled by ZFS instead of GEOM, and, therefore, make it possible for zfsd and the ZFS fault management framework to automatically attach replacement devices to the pool and begin the resilver operation without requiring administrator action. Currently this is only possible if the entire device is dedicated to ZFS and does not have a partition table. To be able to boot from the device, a freebsd-boot or EFI ESP partition must exist for the system to boot from. In the original design of ZFS, there is an area dedicated to storing legacy BIOS bootcode, but it is only used under FreeBSD if the disk is partitioned MBR. In other implementations of ZFS, if the entire disk is used for ZFS, a basic partition table is created that marks the entire device as a single ZFS partition. When ZFS was ported to FreeBSD, the 'whole_disk' flag was taken literally, and the raw disk is used by ZFS. With some enhancements that have been proposed upstream to create an EFI partition as part of the ZFS whole_disk layout, with some minor changes FreeBSD could work the same way. This means that hot spares and disks that are swapped after a failure could automatically be labeled, partitioned, and made active in the pool, without requiring a human as they often do now.

ZFS Persistent L2ARC • 2018Q3

The ZFS L2ARC provides a second-level cache, allowing a high-speed device like an SSD or NVMe to provide another tier of cache between the ARC (in RAM) and the main storage pool. This can be extremely important in order to get the required level of performance where it is not economical, or possible, to have an amount of main memory larger than the working set. The L2ARC relies on headers in the ARC to point to the data stored in the L2ARC. The ARC is not persistent, since it is stored in main memory, so when the system is rebooted, the L2ARC contents are orphaned. At boot the L2ARC is considered empty and is refilled as the cache heats up. This can result in a significant loss of performance until the cache has rewarmed. This is somewhat mitigated by configuration options that increase the fill speed of the L2ARC when it is cold. The new Persistent L2ARC feature keeps log records on the L2ARC that can be reloaded after the system boots. Once the system is online, it asynchronously recreates the ARC headers that point to the data on the L2ARC, allowing its contents to persist through a reboot. While they are not available immediately, the system reaches a hot cache state a lot more quickly than without this feature, and with less wear on the L2ARC devices.

ZFS Sequential Resilver • Pending Integration

One of the advantages of ZFS is that because the filesystem and the volume manager are combined, ZFS is aware of which bytes on the disk are in use, and which are not. A disk that is only 1/3 full will only need to resilver that 1/3 of the data, rather than the entire contents of the disk like with a typical hardware RAID. However, to affect this advantage, ZFS scans the content in the order the objects appear in its metadata. This can result in a very large number of random reads and writes, which perform much worse than sequential reads and writes. This enhancement to the resilvering process will scan the metadata and build a range tree of blocks that will need to be resilvered. Once this tree reaches a configured size limit, the largest contiguous range of blocks in the tree are resilvered, and then the metadata scan continues, until the range tree is full again. This approach ensures that the resilver I/Os are done

in large contiguous ranges which will provide much better performance.

ZFS Resilver Prefetch Improvements • Design Review

This set of improvements aims to increase the resilver performance in a different way that complements the sequential resilver work. The new prefetcher is closer to a depth-first search, rather than only working ahead of the scrub and stalling at the end of each sub-tree. In the new system, a demand read completion triggers the next batch of prefetch operations, keeping the I/O queue full. Configuration prevents more than two scrub prefetch I/Os outstanding at once, preventing the prefetch from delaying actual scrub operations.

ZFS Ashift Policy • Design Phase

The goal of this work is to support time-variable geometric. Allowing older 512-byte sector disks to be replaced with newer 4096 byte sector disks without the performance penalty of doing sub-sector writes. Even now, many disks are 4Kn (4k Native), and will refuse to perform sub-sector I/Os. This feature allows an allocation policy to be set that all future allocations will be at least 4k and avoid the performance hit.

ZFS Spacemap Log • Design Review

ZFS uses a data structure called a spacemap to track which space on the disks is free for future allocations. There is both an in-memory and an on-disk representation of the spacemap. Usually a small number of spacemaps are kept loaded at a time, and allocations are made from those maps. Under heavy fragmentation, the system can spend a lot of time trying to find free space to allocate from. The existing spacemap histogram feature helps mitigate this. The spacemap log feature will improve allocation performance by writing allocations and frees to an append-only log, which will then be coalesced into the traditional spacemap data structure once the log exceeds a configured size. In the event of a crash, the last spacemap is loaded, and then the changes specified in the spacemap log and replayed to bring it up to date.

Native Data and Metadata Encryption • Code Review

The native encryption support is based on a design similar to, but not compatible with, that used in the Oracle proprietary version of ZFS. The implementation allows a number of useful features, including authenticated encryption, meaning that not only is the privacy of the data protected, but the integrity as well. The OpenZFS implementation allows each dataset to be encrypted with a different key, or to inherit the key from its parent dataset. Keys can be loaded and unloaded as needed, so data can actually be put at rest, where encryption provides more meaningful protections, by unmounting the dataset and unloading its encryption key. Another useful feature is the fact that the checksum is split between the traditional checksum of the plaintext and the authentication data from the encryption cipher, which protects the ciphertext. This means that ZFS can detect corruption or modification of either form, but it also means that scrub and resilver operations can proceed even when the encryption keys are not loaded. And it means that encrypted datasets can be replicated in their encrypted form, making it impossible to read them on the receiving side without the correct keys.

Windows Port • Early Preview

This last item is mostly just for fun. Some of the people behind the OpenZFS-on-OS-X project wondered how much work it would take to get OpenZFS running on Microsoft Windows. As it turns out, it is not as impossible as you might think. You can check out the GitHub page https://github.com/openzfsonwindows/ZFSin if you want to learn more.

• Conclusions

OpenZFS has come a long way since its split with OpenSolaris in 2010, and the official formation of the OpenZFS organization in 2013. More than 50% of the code that exists in OpenZFS today is either new or replaced from the original OpenSolaris code. OpenZFS is continuing to lead the way in filesystem development, and the pace is only increasing as more projects and vendors join the effort.

You can find out more about many of these and other recently added and upcoming features of OpenZFS from the OpenZFS Developers Summit 2017 wiki page, which includes slides and video from each of the presentations: http://open-zfs.org/wiki/OpenZFS_Developer_Summit_2017. ●

ALLAN JUDE is VP of operations at ScaleEngine Inc., a video streaming content distribution network, where he makes extensive use of ZFS on FreeBSD. Allan is a FreeBSD src and doc committer, and was elected to the FreeBSD core team in summer 2016. He is also the host of the weekly video podcast BSDNow.tv (with Benedict Reuschling), and coauthor of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* with Michael W Lucas.