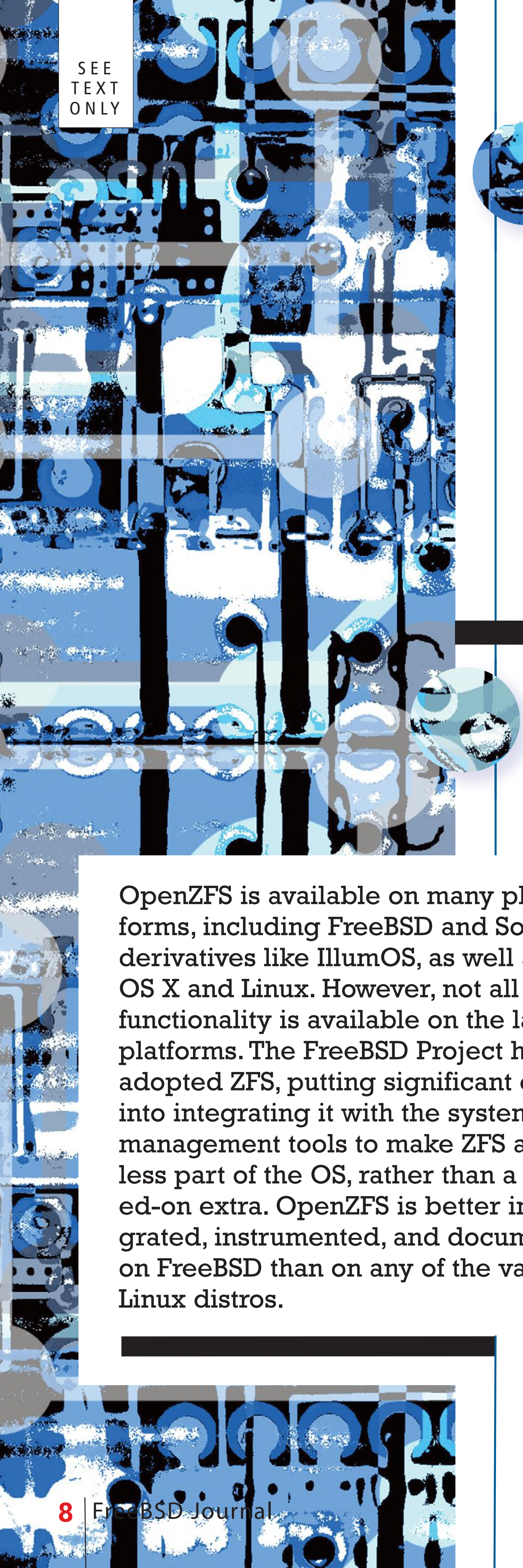

# FreeBSD vs. Linux: ZFS

BY ALLAN JUDE

OpenZFS is available on many platforms, including FreeBSD and Solaris derivatives like IllumOS, as well as Mac OS X and Linux. However, not all of the functionality is available on the latter platforms. The FreeBSD Project has fully adopted ZFS, putting significant effort into integrating it with the system and management tools to make ZFS a seamless part of the OS, rather than a bolted-on extra. OpenZFS is better integrated, instrumented, and documented on FreeBSD than on any of the various Linux distros.

## Why Use ZFS?

It is not that other filesystems are bad; they just make the mistake of trusting your storage hardware to return your data when you ask for it. As it turns out, hard drives are pretty good at that, but pretty good is often not good enough. ZFS is the only open-source, production-quality filesystem that can not only detect but correct the errors when a disk returns incorrect data. By combining the roles of filesystem and volume manager, ZFS is also able to ensure your data is safe, even in the absence of one or more disks, depending on your configuration. ZFS doesn't trust your hardware; it verifies that the correct data was returned from each read from the disk.

The primary design consideration for ZFS is the safety of the data. Every block that is written to the filesystem is accompanied by a checksum of the data, stored with the other metadata. That metadata block also has a checksum, as does its parent, all the way up to the top-level block, called the uber block. When the ZFS filesystem is mounted, it examines the available array of uber blocks and selects the newest one with a valid checksum. When combined with the copy-on-write feature, this means that in the event of a power failure or system crash, ZFS will still have a consistent view of the filesystem; any operations that were in progress, but did not complete, are rolled back to keep the filesystem in pristine shape. This means no need for a long filesystem check after an unexpected shutdown.

Every time a block of a file is read from a ZFS filesystem, the data returned by the disk is checksummed, and that checksum is verified against the one stored in the metadata. If the results differ, this means the disk has returned incorrect data. ZFS will detect this, keeping a count of such errors for each disk, as this may be a sign of impending disk failure. If your ZFS filesystem is configured with redundancy, like mirrors or RAID-Z, this parity information will be used to reconstruct the incorrect block, and write the repaired data back to the disk. If there is no redundancy and the data cannot be recovered, an error will be returned. This allows the operating system to stop an application from using invalid data, which may cause it to crash or do the wrong thing.

ZFS provides a great many other features, including transparent compression, advanced tiered caching, snapshots, deduplication, replication, delegation, boot environments, and, soon, even "at-rest" encryption. ZFS was designed to make the storage administrator's life easier by making adding capacity as easy as possible. ZFS also simplified the management of quotas and reservations, the management of network filesharing, the delegation of permissions, and all of the other common tasks that make up a storage admin's day. All of this is managed from an intuitive command line interface, designed to be used by both human hands and automated with scripts.

## Boot Environments

While the biggest feature of OpenZFS is the data-integrity guarantees it provides, the next most useful is the concept of boot environments. This feature allows a device to have multiple concurrent OS installations and easily flip between them at boot time. With the copy-on-write nature of ZFS, these additional images often consume very little storage space. All that is required is to have the root filesystem located on ZFS, and some integration with the boot loader. These features have been available in FreeBSD for years, and are constantly being refined and improved.

On a mobile computing platform, this flexibility and stability guarantee can be exceptionally useful. Imagine just a few minutes before your big presentation, you discover last week's update has caused some problem with the presentation software. Reboot, select last week's boot environment; now the OS and applications have been rolled back to the known working state, but your data files, home directory, etc., are still the most current version. Another reboot and you are back to the most up-to-date system image. The entire system is now effectively under a rudimentary type of version control. Fork your system and try an experiment, safe in the knowledge you can always flip back to the working system with a quick reboot.

Your separate system images do not need to be copy-on-write clones of each other. They can be entirely separate stand-alone images. Switch between the stable release and a developmental snapshot with ease, all while sharing your user data files. Need to test your latest work on every supported release of FreeBSD? No sweat. With some extra work, it is even possible to multiboot other operating systems that support OpenZFS.

At ScaleEngine we use this mechanism to distribute customized golden images of FreeBSD. After building and tweaking FreeBSD just as we like it, a replication stream is generated with the zfs send command, and redirected to a file. Production servers then fetch that image over HTTPS, and feed it into zfs recv, creating a new boot environment. Then zfsbootcfg configures that new boot environment to be used for the next boot (only). If there is something wrong with the image, such that it does not boot properly, or crashes, a power cycle will see the system come back up on the original system image. A script running in the new system image can make that image the new default if the system remains up for 10 minutes and the network connectivity requirements are met. This procedure allows us to safely upgrade remote systems all over the world with little chance of things requiring additional human intervention. If the new image is not satisfactory, a single command changes the default to one of the previous images and then a reboot puts the system back into operation. This all makes it extremely easy to test developmental images on a small fraction of our production fleet without any special handling.

Another feature that is coming soon to OpenZFS will see repeated system failures result in the system booting into a special debugging/rescue image. In environments like Amazon's EC2, where there is no console support, if the system does not come up fully, due to a boot failure or repeated panic, there is no easy way to repair the system, short of mounting its drive in a different EC2 instance. With this new feature, a counter will be incremented at each boot, and only if the system remains up long enough, will a script in the image reset this counter to zero. If the counter exceeds the threshold, the new boot will see it load the rescue image instead and summon an operator to debug and resolve the issue.

## Disk Encryption

FreeBSD has a high-performance, full disk encryption system called GELI. It has long been used in combination with ZFS to create fully encrypted pools. This required special handling of the boot process, since the loader needed to load the kernel from unencrypted storage. In early 2016, I integrated GELI support into the bootstrap and loader to allow booting from a zpool with no part of the filesystem unencrypted.

The closed-source version of ZFS, now from Oracle, has supported encryption for some time, but this feature has never been available in open-source ZFS.

## SSD TRIM

FreeBSD is currently the only OpenZFS implementation that has support for TRIM. TRIM support was integrated into FreeBSD in 2012 to improve the performance of pools consisting of SSDs and other flash-based devices. TRIM support provides feedback to the FTL (Flash Translation Layer) about blocks that are no longer in use and can be recycled, allowing the SSDs firmware to better manage wear leveling and garbage collection.

The OpenZFS project has work in progress to integrate a different, universal TRIM/UNMAP feature across all supported platforms; however, it is not expected to be completed until 2018. This new implementation is more advanced and is expected to further improve performance, but in the meantime, ZFS users on all other platforms are left with no TRIM support at all.

## Jails

Solaris and its derivatives have zones, and advanced container technology based on the concepts pioneered by FreeBSD jails. This means that, from the beginning, ZFS had advanced integration with zones, and when ZFS was ported to FreeBSD, those features came along and were adapted to work with FreeBSD jails. Linux does not have a direct analogue to jails or zones, and so at this time has no support for ZFS in containers.

In FreeBSD, a dataset (ZFS filesystem) can be marked as "jailed." When this flag is set, the filesystem is no longer able to be mounted on the host system (Global Zone in Solaris parlance). This is a security feature. Datasets that have been delegated to a container have mount points relative to the root of the container. A jailed dataset with a mount point of /etc that became mounted on the host system could change the root password and other configuration, which would allow a user from the container to control the contents of files that may end up on the host.

Once a filesystem has the jailed property set, the 'zfs jail' command can be used to delegate a dataset and all its children to a specific jail. Provided the allow.mount_zfs parameter is activated, root inside the container has complete control over the dataset and its children, except for the jailed property and limits, such as quotas. This allows root in the container to create and manage new datasets, snapshots, and all other features of ZFS. Root in the jail can even take advantage of the regular ZFS delegation feature, and further delegate access to commands and properties to regular users in that jail. This flexibility allows high-concurrency multi-tenancy with relative ease.

At ScaleEngine, we use these features to allow customers SSH access to their video storage via a jail so that they are isolated into an untrusted container with access to only their own files. User delegation and container delegation allow us to have remote replication of customer data without requiring any privileged access, and to isolate our customers while still providing them unfettered access to their own files.

## Licensing[1]

When Sun Microsystems released ZFS as part of OpenSolaris, they did so under the CDDL (Common Development and Distribution License, version 1.0). The CDDL is derived from the Mozilla Public License (MPL) and attempts to hold a middle ground between the GPL (viral) and the BSD license (permissive). Code licensed under the GPL must always remain so, and any derived or combined work, in source, binary, or other form, must be licensed under the GPL. The BSD license places no restrictions on how the licensed code is used, only that the copyright notices must not be removed from the source and must be reproduced in other formats. In contrast, source code released under the CDDL must remain under the CDDL in its source form, but binaries produced from it may be licensed in any way the creator chooses, as long as the modified source is still available under the CDDL. Files licensed under the CDDL may be combined with files licensed under other licenses, whether open source or proprietary. Sun saw this as giving businesses more flexibility in how they licensed their end product, while ensuring that all ZFS features remained open source. Combining OpenZFS with FreeBSD under its liberal license means not having to fret about hidden compliance problems.

The Free Software Foundation (FSF) issued a statement on April 11, 2016, clarifying that distributing CDDL-licensed software, ZFS specifically, as part of a GPL-covered work (the Linux Kernel), is a violation of the GPL license[2]. "It is not enough to require that the combined program be free software somehow. It must be released, as a whole, under the original copyleft license (GPL)." So, while under the CDDL license, you can combine the code with GPL-licensed code, and produce a binary module licensed under the GPL, this is still not compatible with the GPL, because the combined source code must also be released under the GPL, a condition not allowed by the CDDL. Luckily, these restrictions apply to redistribution, not to private use of the code. "The GNU GPL has no substantive requirements about what you do in private; the GPL conditions apply when you make the work available to others." So, you may use ZFS, but you cannot distribute it as a complete product. The Ubuntu project sees it differently[3]: "zfs.ko, as a self-contained filesystem module, is clearly not a derivative work of the Linux kernel, but rather quite obviously a derivative work of OpenZFS and OpenSolaris." The Software Freedom Conservancy (SFC) disagrees[4]. It would seem to be prudent to avoid this legal quagmire, and just use FreeBSD, with its simple two-clause license.

**The primary design consideration for ZFS is the safety of the data. Every block that is written to the filesystem is accompanied by a checksum of the data, stored with the other metadata. That metadata block also has a checksum, as does its parent, all the way up to the top-level block, called the uber block.**

## Conclusions

If the integrity of your data is important, OpenZFS is the only open-source solution you can trust. The best platform for running OpenZFS is FreeBSD. Take advantage of the entire ecosystem of features, utilities, applications, and solutions that make FreeBSD a leading solution for servers and storage.

If you would like to learn more about ZFS and how to apply its advanced features to your storage challenges, pick up copies of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* at your favorite retailer or by visiting www.zfsbook.com. ●

**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.

1 This article is not legal advice and you cannot and should not rely on it as such.
2 https://www.fsf.org/licensing/zfs-and-linux
3 https://insights.ubuntu.com/2016/02/18/zfs-licensing-and-linux/
4 https://sfconservancy.org/blog/2016/feb/25/zfs-and-linux/