ZFS

ZFS (previously: Zettabyte file system) combines a file system with a volume manager. It began as part of the Sun Microsystems Solaris operating system in 2001. Large parts of Solaris – including ZFS – were published under an open source license as OpenSolaris for around 5 years from 2005, before being placed under a closed source license when Oracle Corporation acquired Sun in 2009/2010. During 2005 to 2010, the open source version of ZFS was ported to Linux, Mac OS X (continued as MacZFS) and FreeBSD. In 2010, the illumos project forked a recent version of OpenSolaris, to continue its development as an open source project, including ZFS. In 2013, OpenZFS was founded to coordinate the development of open source ZFS.[3][4][5] OpenZFS maintains and manages the core ZFS code, while organizations using ZFS maintain the specific code and validation processes required for ZFS to integrate within their systems. OpenZFS is widely used in Unix-like systems.[6][7][8]

Contents

ZFS

Developer	Sun Microsystems (acquired by Oracle Corporation in 2009)
Written in	C, C++
OS family	Unix (System V Release 4)
Working state	Current
Source model	Mixed open-source / closed-source
Initial release	June 2006 Solaris 10 6/06 ("U2")
Latest release	11.4[1] / August 28, 2018
Marketing target	Workstation, server
Platforms	SPARC, x86-64, IA-32 (except Solaris 11), PowerPC (Solaris 2.5.1 only)
License	Various
Official website	www.oracle.com/solaris (https://www.oracle.com/solaris)

OpenZFS

OpenZFS

Initial release	Ported to various systems between 2006 and 2010. Forked from OpenSolaris August 2010
Repository	github.com/openzfs/zfs (https://github.com/openzfs/zfs)
Written in	C
Operating system	OpenSolaris, illumos distributions, OpenIndiana, FreeBSD, Mac OS X Server 10.5 (only read-only support), NetBSD, Linux via third-party kernel module ("ZFS on Linux")[2] or ZFS-FUSE, OSv
License	open source CDDL
Website	open-zfs.org (http://open-zfs.org)

Overview

The management of stored data generally involves two aspects: the physical volume management of one or more block storage devices such as hard drives and SD cards and their organization into logical block devices as seen by the operating system (often involving a volume manager, RAID controller, array manager, or suitable device driver), and the management of data and files that are stored on these logical block devices (a file system or other data storage).

> Example: A RAID array of 2 hard drives and an SSD caching disk is controlled by Intel's RST system, part of the chipset and firmware built into a desktop computer. The Windows user sees this as a single volume, containing an NTFS-formatted drive of their data, and NTFS is not necessarily aware of the manipulations that may be required (such as reading from/writing to the cache drive or rebuilding the RAID array if a disk fails). The management of the individual devices and their presentation as a single device is distinct from the management of the files held on that apparent device.

ZFS is unusual because, unlike most other storage systems, it unifies both of these roles and *acts as both the volume manager and the file system*. Therefore, it has complete knowledge of both the physical disks and volumes (including their condition and status, their logical

arrangement into volumes), and also of all the files stored on them. ZFS is designed to ensure (subject to suitable hardware) that data stored on disks cannot be lost due to physical errors or misprocessing by the hardware or operating system, or bit rot events and data corruption which may happen over time, and its complete control of the storage system is used to ensure that every step, whether related to file management or disk management, is verified, confirmed, corrected if needed, and optimized, in a way that storage controller cards and separate volume and file managers cannot achieve.

ZFS also includes a mechanism for dataset and pool-level snapshots and replication, including snapshot cloning which is described by the FreeBSD documentation as one of its "most powerful features", having features that "even other file systems with snapshot functionality lack".[9] Very large numbers of snapshots can be taken, without degrading performance, allowing snapshots to be used prior to risky system operations and software changes, or an entire production ("live") file system to be fully snapshotted several times an hour, in order to mitigate data loss due to user error or malicious activity. Snapshots can be rolled back "live" or previous file system states can be viewed, even on very large file systems, leading to savings in comparison to formal backup and restore processes.[9] Snapshots can also be cloned to form new independent file systems. A pool level snapshot (known as a "checkpoint") is available which allows rollback of operations that may affect the entire pool's structure, or which add or remove entire datasets.

History

Sun Microsystems (to 2010)

In 1987, AT&T Corporation and Sun announced that they were collaborating on a project to merge the most popular Unix variants on the market at that time: Berkeley Software Distribution, UNIX System V, and Xenix. This became Unix System V Release 4 (SVR4).[10] The project was released under the name Solaris, which became the successor to SunOS 4 (although SunOS 4.1.*x* micro releases were retroactively named *Solaris 1*).[11]

ZFS was designed and implemented by a team at Sun led by Jeff Bonwick, Bill Moore[12] and Matthew Ahrens. It was announced on September 14, 2004,[13] but development started in 2001.[14] Source code for ZFS was integrated into the main trunk of Solaris development on October 31, 2005,[15] and released for developers as part of build 27 of OpenSolaris on November 16, 2005. In June 2006, Sun announced that ZFS was included in the mainstream 6/06 update to Solaris 10.[16]

Historically, Solaris was developed as proprietary software. Sun Microsystems was a strong proponent of open source software. In June 2005, Sun released most of the codebase under the CDDL license, and founded the OpenSolaris open-source project.[17] Sun was an early proponent of open source software, and with OpenSolaris, Sun wanted to build a developer and user community around the software. In Solaris 10 6/06 ("U2"), Sun added the ZFS file system. During the next 5 years (2006 to 2010), Sun frequently updated ZFS with new features, and ZFS was ported to Linux, Mac OS X (continued as MacZFS) and FreeBSD, under this open source license.

The name at one point was said to stand for "Zettabyte File System",[18] but by 2006, the name was no longer considered to be an abbreviation.[19] A ZFS file system can store up to 256 quadrillion zettabytes (ZB).

In September 2007, NetApp sued Sun claiming that ZFS infringed some of NetApp's patents on Write Anywhere File Layout. Sun counter-sued in October the same year claiming the opposite. The lawsuits were ended in 2010 with an undisclosed settlement.[20]

Later development

Ported versions of ZFS began to appear in 2005. After the Sun acquisition by Oracle in 2010, Oracle's version of ZFS became closed source and development of open-source versions proceeded independently, coordinated by OpenZFS from 2013.

Features

Summary

Examples of features specific to ZFS include:

- Designed for long-term storage of data, and indefinitely scaled datastore sizes with zero data loss, and high configurability.
- Hierarchical checksumming of all data and metadata, ensuring that the entire storage system can be verified on use, and confirmed to be correctly stored, or remedied if corrupt. Checksums are stored with a block's parent block, rather than with the block itself. This contrasts with many file systems where checksums (if held) are stored with the data so that if the data is lost or corrupt, the checksum is also likely to be lost or incorrect.
- Can store a user-specified number of copies of data or metadata, or selected types of data, to improve the ability to recover from data corruption of important files and structures.
- Automatic rollback of recent changes to the file system and data, in some circumstances, in the event of an error or inconsistency.
- Automated and (usually) silent self-healing of data inconsistencies and write failure when detected, for all errors where the data is capable of reconstruction. Data can be reconstructed using all of the following: error detection and correction checksums stored in each block's parent block; multiple copies of data (including checksums) held on the disk; write intentions logged on the SLOG (ZIL) for writes that should have occurred but did not occur (after a power failure); parity data from RAID/RAID-Z disks and volumes; copies of data from mirrored disks and volumes.
- Native handling of standard RAID levels and additional ZFS RAID layouts ("RAID-Z"). The RAID-Z levels stripe data across only the disks required, for efficiency (many RAID systems stripe indiscriminately across all devices), and checksumming allows rebuilding of inconsistent or corrupted data to be minimized to those blocks with defects;
- Native handling of tiered storage and caching devices, which is usually a volume related task. Because ZFS also understands the file system, it can use file-related knowledge to inform, integrate and optimize its tiered storage handling which a separate device cannot;
- Native handling of snapshots and backup/replication which can be made efficient by integrating the volume and file handling. Relevant tools are provided at a low level and require external scripts and software for utilization.
- Native data compression and deduplication, although the latter is largely handled in RAM and is memory hungry.
- Efficient rebuilding of RAID arrays—a RAID controller often has to rebuild an entire disk, but ZFS can combine disk and file knowledge to limit any rebuilding to data which is actually missing or corrupt, greatly speeding up rebuilding;
- Unaffected by RAID hardware changes which affect many other systems. On many systems, if self-contained RAID hardware such as a RAID card fails, or the data is moved to another RAID system, the file system will lack information that was on the original RAID hardware, which is needed to manage data on the RAID array. This can lead to a total loss of data unless near-identical hardware can be acquired and used as a "stepping stone". Since ZFS manages RAID itself, a ZFS pool can be migrated to other hardware,

or the operating system can be reinstalled, and the RAID-Z structures and data will be recognized and immediately accessible by ZFS again.
- Ability to identify data that would have been found in a cache but has been discarded recently instead; this allows ZFS to reassess its caching decisions in light of later use and facilitates very high cache-hit levels (ZFS cache hit rates are typically over 80%);
- Alternative caching strategies can be used for data that would otherwise cause delays in data handling. For example, synchronous writes which are capable of slowing down the storage system can be converted to asynchronous writes by being written to a fast separate caching device, known as the SLOG (sometimes called the ZIL – ZFS Intent Log).
- Highly tunable—many internal parameters can be configured for optimal functionality.
- Can be used for high availability clusters and computing, although not fully designed for this use.

Data integrity

One major feature that distinguishes ZFS from other file systems is that it is designed with a focus on data integrity by protecting the user's data on disk against silent data corruption caused by data degradation, power surges (voltage spikes), bugs in disk firmware, phantom writes (the previous write did not make it to disk), misdirected reads/writes (the disk accesses the wrong block), DMA parity errors between the array and server memory or from the driver (since the checksum validates data inside the array), driver errors (data winds up in the wrong buffer inside the kernel), accidental overwrites (such as swapping to a live file system), etc.

A 1999 study showed that neither any of the then-major and widespread filesystems (such as UFS, Ext,[21] XFS, JFS, or NTFS), nor hardware RAID (which has some issues with data integrity) provided sufficient protection against data corruption problems.[22][23][24][25] Initial research indicates that ZFS protects data better than earlier efforts.[26][27] It is also faster than UFS[28][29] and can be seen as its replacement.

Within ZFS, data integrity is achieved by using a Fletcher-based checksum or a SHA-256 hash throughout the file system tree.[30] Each block of data is checksummed and the checksum value is then saved in the pointer to that block—rather than at the actual block itself. Next, the block pointer is checksummed, with the value being saved at *its* pointer. This checksumming continues all the way up the file system's data hierarchy to the root node, which is also checksummed, thus creating a Merkle tree.[30] In-flight data corruption or phantom reads/writes (the data written/read checksums correctly but is actually wrong) are undetectable by most filesystems as they store the checksum with the data. ZFS stores the checksum of each block in its parent block pointer so the entire pool self-validates.[30]

When a block is accessed, regardless of whether it is data or meta-data, its checksum is calculated and compared with the stored checksum value of what it "should" be. If the checksums match, the data are passed up the programming stack to the process that asked for it; if the values do not match, then ZFS can heal the data if the storage pool provides data redundancy (such as with internal mirroring), assuming that the copy of data is undamaged and with matching checksums.[31] It is optionally possible to provide additional in-pool redundancy by specifying `copies=2` (or `copies=3` or more), which means that data will be stored twice (or three times) on the disk, effectively halving (or, for `copies=3`, reducing to one third) the storage capacity of the disk.[32] Additionally some kinds of data used by ZFS to manage the pool are stored multiple times by default for safety, even with the default copies=1 setting.

If other copies of the damaged data exist or can be reconstructed from checksums and parity data, ZFS will use a copy of the data (or recreate it via a RAID recovery mechanism), and recalculate the checksum—ideally resulting in the reproduction of the originally expected value. If the data passes this integrity check, the system can then update all faulty copies with known-good data and redundancy will be restored.

Consistency of data held in memory, such as cached data in the ARC, is not checked by default, as ZFS is expected to run on enterprise-quality hardware with error correcting RAM, but the capability to check in-memory data exists and can be enabled using "debug flags".[33]

RAID ("RAID-Z")

For ZFS to be able to guarantee data integrity, it needs multiple copies of the data, usually spread across multiple disks. Typically this is achieved by using either a RAID controller or so-called "soft" RAID (built into a file system).

Avoidance of hardware RAID controllers

While ZFS *can* work with hardware RAID devices, ZFS will usually work more efficiently and with greater protection of data, if it has raw access to all storage devices, and disks are not connected to the system using a hardware, firmware or other "soft" RAID, or any other controller which modifies the usual ZFS-to-disk I/O path. This is because ZFS relies on the disk for an honest view, to determine the moment data is confirmed as safely written, and it has numerous algorithms designed to optimize its use of caching, cache flushing, and disk handling.

If a third-party device performs caching or presents drives to ZFS as a single system, or without the low level view ZFS relies upon, there is a much greater chance that the system will perform *less* optimally, and that a failure will not be preventable by ZFS or as quickly or fully recovered by ZFS. For example, if a hardware RAID card is used, ZFS may not be able to determine the condition of disks or whether the RAID array is degraded or rebuilding, it may not know of all data corruption, and it cannot place data optimally across the disks, make selective repairs only, control how repairs are balanced with ongoing use, and may not be able to make repairs even if it could usually do so, as the hardware RAID card will interfere. RAID controllers also usually add controller-dependent data to the drives which prevents software RAID from accessing the user data. While it is possible to read the data with a compatible hardware RAID controller, this isn't always possible, and if the controller card develops a fault then a replacement may not be available, and other cards may not understand the manufacturer's custom data which is needed to manage and restore an array on a new card.

Therefore, unlike most other systems, where RAID cards or similar are used to offload resources and processing and enhance performance and reliability, with ZFS it is strongly recommended these methods *not* be used as they typically *reduce* the system's performance and reliability.

If disks must be connected through a RAID or other controller, it is recommended to use a plain HBA (host adapter) or fanout card, or configure the card in JBOD mode (i.e. turn off RAID and caching functions), to allow devices to be attached but the ZFS-to-disk I/O pathway to be unchanged. A RAID card in JBOD mode may still interfere, if it has a cache or depending upon its design, and may detach drives that do not respond in time (as has been seen with many energy-efficient consumer-grade hard drives), and as such, may require Time-Limited Error Recovery (TLER)/CCTL/ERC-enabled drives to prevent drive dropouts, so not all cards are suitable even with RAID functions disabled.[34]

ZFS's approach: RAID-Z and mirroring

Instead of hardware RAID, ZFS employs "soft" RAID, offering *RAID-Z* (parity based like RAID 5 and similar) and disk mirroring (similar to RAID 1). The schemes are highly flexible.

RAID-Z is a data/parity distribution scheme like RAID-5, but uses dynamic stripe width: every block is its own RAID stripe, regardless of blocksize, resulting in every RAID-Z write being a full-stripe write. This, when combined with the copy-on-write transactional semantics of ZFS, eliminates the write hole error. RAID-Z is also faster than traditional RAID 5 because it does not need to perform the usual read-modify-write sequence.[35]

As all stripes are of different sizes, RAID-Z reconstruction has to traverse the filesystem metadata to determine the actual RAID-Z geometry. This would be impossible if the filesystem and the RAID array were separate products, whereas it becomes feasible when there is an integrated view of the logical and physical structure of the data. Going through the metadata means that ZFS can validate every block against its 256-bit checksum as it goes, whereas traditional RAID products usually cannot do this.[35]

In addition to handling whole-disk failures, RAID-Z can also detect and correct silent data corruption, offering "self-healing data": when reading a RAID-Z block, ZFS compares it against its checksum, and if the data disks did not return the right answer, ZFS reads the parity and then figures out which disk returned bad data. Then, it repairs the damaged data and returns good data to the requestor.[35]

RAID-Z and mirroring do not require any special hardware: they do not need NVRAM for reliability, and they do not need write buffering for good performance or data protection. With RAID-Z, ZFS provides fast, reliable storage using cheap, commodity disks.[35]

There are five different RAID-Z modes: *striping* (similar to RAID 0, offers no redundancy), *RAID-Z1* (similar to RAID 5, allows one disk to fail), *RAID-Z2* (similar to RAID 6, allows two disks to fail), *RAID-Z3* (a RAID 7[a] configuration, allows three disks to fail), and mirroring (similar to RAID 1, allows all but one disk to fail).[37]

The need for RAID-Z3 arose in the early 2000s as multi-terabyte capacity drives became more common. This increase in capacity—without a corresponding increase in throughput speeds—meant that rebuilding an array due to a failed drive could take "weeks or even months" to complete.[36] During this time, the older disks in the array will be stressed by the additional workload, which could result data corruption or drive failure. By increasing parity, RAID-Z3 reduces the chance of data loss by simply increasing redundancy.[38]

Resilvering and scrub (array syncing and integrity checking)

ZFS has no tool equivalent to fsck (the standard Unix and Linux data checking and repair tool for file systems).[39] Instead, ZFS has a built-in scrub function which regularly examines all data and repairs silent corruption and other problems. Some differences are:

- fsck must be run on an offline filesystem, which means the filesystem must be unmounted and is not usable while being repaired, while scrub is designed to be used on a mounted, live filesystem, and does not need the ZFS filesystem to be taken offline.
- fsck usually only checks metadata (such as the journal log) but never checks the data itself. This means, after an fsck, the data might still not match the original data as stored.
- fsck cannot always validate and repair data when checksums are stored with data (often the case in many file systems), because the checksums may also be corrupted or unreadable. ZFS always stores checksums separately from the data they verify, improving reliability and the ability of scrub to repair the volume. ZFS also stores multiple copies of data—metadata, in particular, may have upwards of 4 or 6 copies (multiple copies per disk and multiple disk mirrors per volume), greatly improving the ability of scrub to detect and repair extensive damage to the volume, compared to fsck.
- scrub checks everything, including metadata and the data. The effect can be observed by comparing fsck to scrub times—sometimes a fsck on a large RAID completes in a few minutes,

which means only the metadata was checked. Traversing all metadata and data on a large RAID takes many hours, which is exactly what scrub does.

The official recommendation from Sun/Oracle is to scrub enterprise-level disks once a month, and cheaper commodity disks once a week.[40][41]

Capacity

ZFS is a 128-bit file system,[42][15] so it can address 1.84×10^{19} times more data than 64-bit systems such as Btrfs. The maximum limits of ZFS are designed to be so large that they should never be encountered in practice. For instance, fully populating a single zpool with 2^{128} bits of data would require $3{\times}10^{24}$ TB hard disk drives.[43]

Some theoretical limits in ZFS are:

- 16 exbibytes (2^{64} bytes): maximum size of a single file
- 2^{48}: number of entries in any individual directory[44]
- 16 exbibytes: maximum size of any attribute
- 2^{56}: number of attributes of a file (actually constrained to 2^{48} for the number of files in a directory)
- 256 quadrillion zebibytes (2^{128} bytes): maximum size of any zpool
- 2^{64}: number of devices in any zpool
- 2^{64}: number of file systems in a zpool
- 2^{64}: number of zpools in a system

Encryption

With Oracle Solaris, the encryption capability in ZFS[45] is embedded into the I/O pipeline. During writes, a block may be compressed, encrypted, checksummed and then deduplicated, in that order. The policy for encryption is set at the dataset level when datasets (file systems or ZVOLs) are created. The wrapping keys provided by the user/administrator can be changed at any time without taking the file system offline. The default behaviour is for the wrapping key to be inherited by any child data sets. The data encryption keys are randomly generated at dataset creation time. Only descendant datasets (snapshots and clones) share data encryption keys.[46] A command to switch to a new data encryption key for the clone or at any time is provided—this does not re-encrypt already existing data, instead utilising an encrypted master-key mechanism.

As of 2019 the encryption feature is also fully integrated into OpenZFS 0.8.0 available for Debian and Ubuntu Linux distributions.[47]

Read/write efficiency

ZFS will automatically allocate data storage across all vdevs in a pool (and all devices in each vdev) in a way that generally maximises the performance of the pool. ZFS will also update its write strategy to take account of new disks added to a pool, when they are added.

As a general rule, ZFS allocates writes across vdevs based on the free space in each vdev. This ensures that vdevs which have proportionately less data already, are given more writes when new data is to be stored. This helps to ensure that as the pool becomes more used, the situation does not develop that some vdevs become

full, forcing writes to occur on a limited number of devices. It also means that when data is read (and reads are much more frequent than writes in most uses), different parts of the data can be read from as many disks as possible at the same time, giving much higher read performance. Therefore, as a general rule, pools and vdevs should be managed and new storage added, so that the situation does not arise that some vdevs in a pool are almost full and others almost empty, as this will make the pool less efficient.

Other features

Storage devices, spares, and quotas

Pools can have hot spares to compensate for failing disks. When mirroring, block devices can be grouped according to physical chassis, so that the filesystem can continue in the case of the failure of an entire chassis.

Storage pool composition is not limited to similar devices, but can consist of ad-hoc, heterogeneous collections of devices, which ZFS seamlessly pools together, subsequently doling out space to diverse filesystems as needed. Arbitrary storage device types can be added to existing pools to expand their size.[48]

The storage capacity of all vdevs is available to all of the file system instances in the zpool. A quota can be set to limit the amount of space a file system instance can occupy, and a reservation can be set to guarantee that space will be available to a file system instance.

Caching mechanisms: ARC, L2ARC, Transaction groups, ZIL, SLOG, Special VDEV

ZFS uses different layers of disk cache to speed up read and write operations. Ideally, all data should be stored in RAM, but that is usually too expensive. Therefore, data is automatically cached in a hierarchy to optimize performance versus cost;[49] these are often called "hybrid storage pools".[50] Frequently accessed data will be stored in RAM, and less frequently accessed data can be stored on slower media, such as solid state drives (SSDs). Data that is not often accessed is not cached and left on the slow hard drives. If old data is suddenly read a lot, ZFS will automatically move it to SSDs or to RAM.

ZFS caching mechanisms include one each for reads and writes, and in each case, two levels of caching can exist, one in computer memory (RAM) and one on fast storage (usually solid state drives (SSDs)), for a total of four caches.

	Where stored	Read cache	Write cache
First level cache	In RAM	Known as **ARC**, due to its use of a variant of the adaptive replacement cache (ARC) algorithm. RAM will always be used for caching, thus this level is always present. The efficiency of the ARC algorithm means that disks will often not need to be accessed, provided the ARC size is sufficiently large. If RAM is too small there will hardly be any ARC at all; in this case, ZFS always needs to access the underlying disks, which impacts performance, considerably.	Handled by means of **"transaction groups"** – writes are collated over a short period (typically 5 – 30 seconds) up to a given limit, with each group being written to disk ideally while the next group is being collated. This allows writes to be organized more efficiently for the underlying disks at the risk of minor data loss of the most recent transactions upon power interruption or hardware fault. In practice the power loss risk is avoided by ZFS write journaling and by the SLOG/ZIL second tier write cache pool (see below), so writes will only be lost if a write failure happens at the same time as a total loss of the second tier SLOG pool, and then only when settings related to synchronous writing and SLOG use are set in a way that would allow such a situation to arise. If data is received faster than it can be written, data receipt is paused until the disks can catch up.
Second level cache	On fast storage devices (which can be added or removed from a "live" system without disruption in current versions of ZFS, although not always in older versions)	Known as **L2ARC** ("Level 2 ARC"), optional. ZFS will cache as much data in L2ARC as it can, which can be tens or hundreds of gigabytes in many cases. L2ARC will also considerably speed up deduplication if the entire deduplication table can be cached in L2ARC. It can take several hours to fully populate the L2ARC from empty (before ZFS has decided which data are "hot" and should be cached). If the L2ARC device is lost, all reads will go out to the disks which slows down performance, but nothing else will happen (no data will be lost).	Known as **SLOG** or **ZIL** ("ZFS Intent Log") – the terms are often used incorrectly. A SLOG (secondary log device) is an optional dedicated cache on a separate device, for recording writes, in the event of a system issue. If an SLOG device exists, it will be used for the ZFS Intent Log as a second level log, and if no separate cache device is provided, the ZIL will be created on the main storage devices instead. The SLOG thus, technically, refers to the dedicated disk to which the ZIL is offloaded, in order to speed up the pool. Strictly speaking, ZFS does not use the SLOG device to cache its disk writes. Rather, it uses SLOG to ensure writes are captured to a permanent storage medium as quickly as possible, so that in the event of power loss or write failure, no data which was acknowledged as written, will be lost. The SLOG device allows ZFS to speedily store writes and quickly report them as written, even for storage devices such as HDDs that are much slower. In the normal course of activity, the SLOG is never referred to or read, and it does not act as a cache; its purpose is to safeguard data in flight during the few seconds taken for collation and "writing out", in case the eventual write were to fail. If all goes well, then the storage pool will be updated at some point within the next 5 to 60 seconds, when the current transaction group is written out to disk (see above), at which point the saved writes on the SLOG will simply be ignored and overwritten. If the write eventually fails, or the system suffers a crash or fault preventing its writing, then ZFS can identify all the writes that it has confirmed were written, by reading back the SLOG (the only time it is read from), and use this to completely repair the data loss. This becomes crucial if a large number of synchronous writes take place (such as with ESXi, NFS and some databases),[51] where the client requires confirmation of successful writing before continuing its activity; the SLOG allows ZFS to confirm writing is successful much more quickly than if it had to write to the main store every time,

			without the risk involved in misleading the client as to the state of data storage. If there is no SLOG device then part of the main data pool will be used for the same purpose, although this is slower. If the log device itself is lost, it is possible to lose the latest writes, therefore the log device should be mirrored. In earlier versions of ZFS, loss of the log device could result in loss of the entire zpool, although this is no longer the case. Therefore, one should upgrade ZFS if planning to use a separate log device.

A number of other caches, cache divisions, and queues also exist within ZFS. For example, each VDEV has its own data cache, and the ARC cache is divided between data stored by the user and metadata used by ZFS, with control over the balance between these.

Special VDEV Class

In ZFS 0.8 and later, it is possible to configure a Special VDEV class to preferentially store filesystem metadata, and optionally the Data Deduplication Table (DDT), and small filesystem blocks. This allows, for example, to create a Special VDEV on fast solid-state storage to store the metadata, while the regular file data is stored on spinning disks. This speeds up metadata-intensive operations such as filesystem traversal, scrub, and resilver, without the expense of storing the entire filesystem on solid-state storage.

Copy-on-write transactional model

ZFS uses a copy-on-write transactional object model. All block pointers within the filesystem contain a 256-bit checksum or 256-bit hash (currently a choice between Fletcher-2, Fletcher-4, or SHA-256)[52] of the target block, which is verified when the block is read. Blocks containing active data are never overwritten in place; instead, a new block is allocated, modified data is written to it, then any metadata blocks referencing it are similarly read, reallocated, and written. To reduce the overhead of this process, multiple updates are grouped into transaction groups, and ZIL (intent log) write cache is used when synchronous write semantics are required. The blocks are arranged in a tree, as are their checksums (see Merkle signature scheme).

Snapshots and clones

An advantage of copy-on-write is that, when ZFS writes new data, the blocks containing the old data can be retained, allowing a snapshot version of the file system to be maintained. ZFS snapshots are consistent (they reflect the entire data as it existed at a single point in time), and can be created extremely quickly, since all the data composing the snapshot is already stored, with the entire storage pool often snapshotted several times per hour. They are also space efficient, since any unchanged data is shared among the file system and its snapshots. Snapshots are inherently read-only, ensuring they will not be modified after creation, although they should not be relied on as a sole means of backup. Entire snapshots can be restored and also files and directories within snapshots.

Writeable snapshots ("clones") can also be created, resulting in two independent file systems that share a set of blocks. As changes are made to any of the clone file systems, new data blocks are created to reflect those changes, but any unchanged blocks continue to be shared, no matter how many clones exist. This is an implementation of the Copy-on-write principle.

Sending and receiving snapshots

ZFS file systems can be moved to other pools, also on remote hosts over the network, as the send command creates a stream representation of the file system's state. This stream can either describe complete contents of the file system at a given snapshot, or it can be a delta between snapshots. Computing the delta stream is very efficient, and its size depends on the number of blocks changed between the snapshots. This provides an efficient strategy, e.g., for synchronizing offsite backups or high availability mirrors of a pool.

Dynamic striping

Dynamic striping across all devices to maximize throughput means that as additional devices are added to the zpool, the stripe width automatically expands to include them; thus, all disks in a pool are used, which balances the write load across them.

Variable block sizes

ZFS uses variable-sized blocks, with 128 KB as the default size. Available features allow the administrator to tune the maximum block size which is used, as certain workloads do not perform well with large blocks. If data compression is enabled, variable block sizes are used. If a block can be compressed to fit into a smaller block size, the smaller size is used on the disk to use less storage and improve IO throughput (though at the cost of increased CPU use for the compression and decompression operations).[53]

Lightweight filesystem creation

In ZFS, filesystem manipulation within a storage pool is easier than volume manipulation within a traditional filesystem; the time and effort required to create or expand a ZFS filesystem is closer to that of making a new directory than it is to volume manipulation in some other systems.

Adaptive endianness

Pools and their associated ZFS file systems can be moved between different platform architectures, including systems implementing different byte orders. The ZFS block pointer format stores filesystem metadata in an endian-adaptive way; individual metadata blocks are written with the native byte order of the system writing the block. When reading, if the stored endianness does not match the endianness of the system, the metadata is byte-swapped in memory.

This does not affect the stored data; as is usual in POSIX systems, files appear to applications as simple arrays of bytes, so applications creating and reading data remain responsible for doing so in a way independent of the underlying system's endianness.

Deduplication

Data deduplication capabilities were added to the ZFS source repository at the end of October 2009,[54] and relevant OpenSolaris ZFS development packages have been available since December 3, 2009 (build 128).

Effective use of deduplication may require large RAM capacity; recommendations range between 1 and 5 GB of RAM for every TB of storage.[55][56][57] An accurate assessment of the memory required for deduplication is made by referring to the number of unique blocks in the pool, and the number of bytes on disk and in RAM ("core") required to store each record—these figures are reported by inbuilt commands such as `zpool` and `zdb`. Insufficient physical memory or lack of ZFS cache can result in virtual memory thrashing when using

deduplication, which can cause performance to plummet, or result in complete memory starvation. Because deduplication occurs at write-time, it is also very CPU-intensive and this can also significantly slow down a system.

Other storage vendors use modified versions of ZFS to achieve very high data compression ratios. Two examples in 2012 were GreenBytes[58] and Tegile.[59] In May 2014, Oracle bought GreenBytes for its ZFS deduplication and replication technology.[60]

As described above, deduplication is usually *not* recommended due to its heavy resource requirements (especially RAM) and impact on performance (especially when writing), other than in specific circumstances where the system and data are well-suited to this space-saving technique.

Additional capabilities

- Explicit I/O priority with deadline scheduling.
- Claimed globally optimal I/O sorting and aggregation.
- Multiple independent prefetch streams with automatic length and stride detection.
- Parallel, constant-time directory operations.
- End-to-end checksumming, using a kind of "Data Integrity Field", allowing data corruption detection (and recovery if you have redundancy in the pool). A choice of 3 hashes can be used, optimized for speed (fletcher), standardization and security (SHA256) and salted hashes (Skein).[61]
- Transparent filesystem compression. Supports LZJB, gzip[62] and LZ4.
- Intelligent scrubbing and resilvering (resyncing).[63]
- Load and space usage sharing among disks in the pool.[64]
- Ditto blocks: Configurable data replication per filesystem, with zero, one or two extra copies requested per write for user data, and with that same base number of copies plus one or two for metadata (according to metadata importance).[65] If the pool has several devices, ZFS tries to replicate over different devices. Ditto blocks are primarily an additional protection against corrupted sectors, not against total disk failure.[66]
- ZFS design (copy-on-write + superblocks) is safe when using disks with write cache enabled, if they honor the write barriers. This feature provides safety and a performance boost compared with some other filesystems.
- On Solaris, when entire disks are added to a ZFS pool, ZFS automatically enables their write cache. This is not done when ZFS only manages discrete slices of the disk, since it does not know if other slices are managed by non-write-cache safe filesystems, like UFS. The FreeBSD implementation can handle disk flushes for partitions thanks to its GEOM framework, and therefore does not suffer from this limitation.
- Per-user, per-group, per-project, and per-dataset quota limits.[67]
- Filesystem encryption since Solaris 11 Express[68] (on some other systems ZFS can utilize encrypted disks for a similar effect; GELI on FreeBSD can be used this way to create fully encrypted ZFS storage).
- Pools can be imported in read-only mode.
- It is possible to recover data by rolling back entire transactions at the time of importing the zpool.
- ZFS is not a clustered filesystem; however, clustered ZFS is available from third parties.
- Snapshots can be taken manually or automatically. The older versions of the stored data that they contain can be exposed as full read-only file systems. They can also be exposed as historic versions of files and folders when used with CIFS (also known as SMB, Samba or file

shares); this is known as "Previous versions", "VSS shadow copies", or "File history" on Windows, or AFP and "Apple Time Machine" on Apple devices.[69]
- Disks can be marked as 'spare'. A data pool can be set to automatically and transparently handle disk faults by activating a spare disk and beginning to resilver the data that was on the suspect disk onto it, when needed.

Limitations

There are several limitations of the ZFS filesystem.

Limitations in preventing data corruption

The authors of a 2010 study that examined the ability of file systems to detect and prevent data corruption, with particular focus on ZFS, observed that ZFS itself is effective in detecting and correcting data errors on storage devices, but that it assumes data in RAM is "safe", and not prone to error. The study comments that "a single bit flip in memory causes a small but non-negligible percentage of runs to experience a failure, with the probability of committing bad data to disk varying from 0% to 3.6% (according to the workload)", and that when ZFS caches pages or stores copies of metadata in RAM, or holds data in its "dirty" cache for writing to disk, no test is made whether the checksums still match the data at the point of use.[70] Much of this risk can be mitigated in one of two ways:

- According to the authors, by using ECC RAM; however, the authors considered that adding error detection related to the page cache and heap would allow ZFS to handle certain classes of error more robustly.[70]
- One of the main architects of ZFS, Matt Ahrens, explains there is an option to enable checksumming of data in memory by using the ZFS_DEBUG_MODIFY flag (zfs_flags=0x10) which addresses these concerns.[71]

Other limitations specific to ZFS

- Capacity expansion is normally achieved by adding groups of disks as a top-level vdev: simple device, RAID-Z, RAID Z2, RAID Z3, or mirrored. Newly written data will dynamically start to use all available vdevs. It is also possible to expand the array by iteratively swapping each drive in the array with a bigger drive and waiting for ZFS to self-heal; the heal time will depend on the amount of stored information, not the disk size.
- As of Solaris 10 Update 11 and Solaris 11.2, it was neither possible to reduce the number of top-level vdevs in a pool, nor to otherwise reduce pool capacity.[72] This functionality was said to be in development in 2007.[73] Enhancements to allow reduction of vdevs is under development in OpenZFS.[74]
- As of 2008 it was not possible to add a disk as a column to a RAID Z, RAID Z2 or RAID Z3 vdev. However, a new RAID Z vdev can be created instead and added to the zpool.[75]
- Some traditional nested RAID configurations, such as RAID 51 (a mirror of RAID 5 groups), are not configurable in ZFS. Vdevs can only be composed of raw disks or files, not other vdevs. However, a ZFS pool effectively creates a stripe (RAID 0) across its vdevs, so the equivalent of a RAID 50 or RAID 60 is common.
- Reconfiguring the number of devices in a top-level vdev requires copying data offline, destroying the pool, and recreating the pool with the new top-level vdev configuration, except for adding extra redundancy to an existing mirror, which can be done at any time or if all top level vdevs are mirrors with sufficient redundancy the zpool split[76] command can be used to remove a vdev from each top level vdev in the pool, creating a 2nd pool with identical data.

- IOPS performance of a ZFS storage pool can suffer if the ZFS raid is not appropriately configured. This applies to all types of RAID, in one way or another. If the zpool consists of only one group of disks configured as, say, eight disks in RAID Z2, then the IOPS performance will be that of a single disk (write speed will be equivalent to 6 disks, but random read speed will be similar to a single disk). However, there are ways to mitigate this IOPS performance problem, for instance add SSDs as L2ARC cache—which can boost IOPS into 100.000s.[77] In short, a zpool should consist of several groups of vdevs, each vdev consisting of 8–12 disks, if using RAID Z. It is not recommended to create a zpool with a single large vdev, say 20 disks, because IOPS performance will be that of a single disk, which also means that resilver time will be very long (possibly weeks with future large drives).
- Online shrink `zpool remove` was not supported until Solaris 11.4 released in August 2018[78]
- Resilver (repair) of a crashed disk in a ZFS RAID can take a long time which is not unique to ZFS, it applies to all types of RAID, in one way or another. This means that very large volumes can take several days to repair or to being back to full redundancy after severe data corruption or failure, and during this time a second disk failure may occur, especially as the repair puts additional stress on the system as a whole. In turn this means that configurations that only allow for recovery of a single disk failure, such as RAID Z1 (similar to RAID 5) should be avoided. Therefore, with large disks, one should use RAID Z2 (allow two disks to crash) or RAID Z3 (allow three disks to crash).[79] ZFS RAID differs from conventional RAID by only reconstructing live data and metadata when replacing a disk, not the entirety of the disk including blank and garbage blocks, which means that replacing a member disk on a ZFS pool that is only partially full will take proportionally less time compared to conventional RAID.[63]

Data recovery

Historically, ZFS has not shipped with tools such as fsck to repair damaged file systems, because the file system itself was designed to self-repair, so long as it had been built with sufficient attention to the design of storage and redundancy of data. If the pool was compromised because of poor hardware, inadequate design or redundancy, or unfortunate mishap, to the point that ZFS was unable to mount the pool, traditionally there were no tools which allowed an end-user to attempt partial salvage of the stored data. This led to threads in online forums where ZFS developers sometimes tried to provide ad-hoc help to home and other small scale users, facing loss of data due to their inadequate design or poor system management.[80]

Modern ZFS has improved considerably on this situation over time, and continues to do so:

- Removal or abrupt failure of caching devices no longer causes pool loss. (At worst, loss of the ZIL may lose very recent transactions, but the ZIL does not usually store more than a few seconds' worth of recent transactions. Loss of the L2ARC cache does not affect data.)
- If the pool is unmountable, modern versions of ZFS will attempt to identify the most recent consistent point at which the pool which can be recovered, at the cost of losing some of the most recent changes to the contents. Copy on write means that older versions of data, including top-level records and metadata, may still exist even though they are superseded, and if so, the pool can be wound back to a consistent state based on them. The older the data, the more likely it is that at least some blocks have been overwritten and that some data will be irrecoverable, so there is a limit at some point, on the ability of the pool to be wound back.
- Informally, tools exist to probe the reason why ZFS is unable to mount a pool, and guide the user or a developer as to manual changes required to force the pool to mount. These include using *zdb* (ZFS debug) to find a valid importable point in the pool, using dtrace or

similar to identify the issue causing mount failure, or manually bypassing health checks that cause the mount process to abort, and allow mounting of the damaged pool.

- As of March 2018, a range of significantly enhanced methods are gradually being rolled out within OpenZFS. These include:[80]

    - Code refactoring, and more detailed diagnostic and debug information on mount failures, to simplify diagnosis and fixing of corrupt pool issues;
    - The ability to trust or distrust the stored pool configuration. This is particularly powerful, as it allows a pool to be mounted even when top-level vdevs are missing or faulty, when top level data is suspect, and also to rewind *beyond* a pool configuration change if that change was connected to the problem. Once the corrupt pool is mounted, readable files can be copied for safety, and it may turn out that data can be rebuilt even for missing vdevs, by using copies stored elsewhere in the pool.
    - The ability to fix the situation where a disk needed in one pool, was accidentally removed and added to a different pool, causing it to lose metadata related to the first pool, which becomes unreadable.

OpenZFS and ZFS

Oracle Corporation ceased the public development of both ZFS and OpenSolaris after the acquisition of Sun in 2010. Some developers forked the last public release of OpenSolaris as the Illumos project. Because of the significant advantages present in ZFS, it has been ported to several different platforms with different features and commands. For coordinating the development efforts and to avoid fragmentation, OpenZFS was founded in 2013.

According to Matt Ahrens, one of the main architects of ZFS, over 50% of the original OpenSolaris ZFS code has been replaced in OpenZFS with community contributions as of 2019, making “Oracle ZFS” and “OpenZFS” politically and technologically incompatible.[81]

Commercial and open source products

- 2008: Sun shipped a line of ZFS-based 7000-series storage appliances.[82]
- 2013: Oracle shipped ZS3 series of ZFS-based filers and seized first place in the SPC-2 benchmark with one of them.[83]
- 2013: iXsystems ships ZFS-based NAS devices called FreeNAS, (now named TrueNAS CORE), for SOHO and TrueNAS for the enterprise.[84][85]
- 2014: Netgear ships a line of ZFS-based NAS devices called ReadyDATA, designed to be used in the enterprise.[86]
- 2015: rsync.net announces a cloud storage platform that allows customers to provision their own zpool and import and export data using zfs send and zfs receive.[87][88]
- 2020: iXsystems Begins development of a ZFS-based hyperconverged software called TrueNAS SCALE, for SOHO and TrueNAS for the enterprise.[85]

Oracle Corporation, closed source, and forking (from 2010)

In January 2010, Oracle Corporation acquired Sun Microsystems, and quickly discontinued the OpenSolaris distribution and the open source development model.[89][90] In August 2010, Oracle discontinued providing public updates to the source code of the Solaris kernel, effectively turning Solaris 11 back into a closed source proprietary operating system.[91]

In response to the changing landscape of Solaris and OpenSolaris, the illumos project was launched via webinar[92] on Thursday, August 3, 2010, as a community effort of some core Solaris engineers to continue developing the open source version of Solaris, and complete the open sourcing of those parts not already open sourced by Sun.[93] illumos was founded as a Foundation, the illumos Foundation, incorporated in the State of California as a 501(c)6 trade association. The original plan explicitly stated that illumos would not be a distribution or a fork. However, after Oracle announced discontinuing OpenSolaris, plans were made to fork the final version of the Solaris ON kernel allowing illumos to evolve into a kernel of its own.[94] As part of OpenSolaris, an open source version of ZFS was therefore integral within illumos.

ZFS was widely used within numerous platforms, as well as Solaris. Therefore, in 2013, the co-ordination of development work on the open source version of ZFS was passed to an umbrella project, *OpenZFS*. The OpenZFS framework allows any interested parties to collaboratively develop the core ZFS codebase in common, while individually maintaining any specific extra code which ZFS requires to function and integrate within their own systems.

Version history

Legend:

Old release
Latest FOSS stable release

ZFS Filesystem Version Number	Release date	Significant changes
1	OpenSolaris Nevada[95] build 36	First release
2	OpenSolaris Nevada b69	Enhanced directory entries. In particular, directory entries now store the object type. For example, file, directory, named pipe, and so on, in addition to the object number.
3	OpenSolaris Nevada b77	Support for sharing ZFS file systems over SMB. Case insensitivity support. System attribute support. Integrated anti-virus support.
4	OpenSolaris Nevada b114	Properties: userquota, groupquota, userused and groupused
5	OpenSolaris Nevada b137	System attributes; symlinks now their own object type

ZFS Pool Version Number	Release date	Significant changes
1	OpenSolaris Nevada[95] b36	First release
2	OpenSolaris Nevada b38	Ditto Blocks
3	OpenSolaris Nevada b42	Hot spares, double-parity RAID-Z (raidz2), improved RAID-Z accounting
4	OpenSolaris Nevada b62	zpool history
5	OpenSolaris Nevada b62	gzip compression for ZFS datasets
6	OpenSolaris Nevada b62	"bootfs" pool property
7	OpenSolaris Nevada b68	ZIL: adds the capability to specify a separate Intent Log device or devices
8	OpenSolaris Nevada b69	ability to delegate zfs(1M) administrative tasks to ordinary users
9	OpenSolaris Nevada b77	CIFS server support, dataset quotas
10	OpenSolaris Nevada b77	Devices can be added to a storage pool as "cache devices"
11	OpenSolaris Nevada b94	Improved zpool scrub / resilver performance
12	OpenSolaris Nevada b96	Snapshot properties
13	OpenSolaris Nevada b98	Properties: usedbysnapshots, usedbychildren, usedbyrefreservation, and usedbydataset
14	OpenSolaris Nevada b103	passthrough-x aclinherit property support
15	OpenSolaris Nevada b114	Properties: userquota, groupquota, usuerused and groupused; also required FS v4
16	OpenSolaris Nevada b116	STMF property support
17	OpenSolaris Nevada b120	triple-parity RAID-Z
18	OpenSolaris Nevada b121	ZFS snapshot holds
19	OpenSolaris Nevada b125	ZFS log device removal
20	OpenSolaris Nevada b128	zle compression algorithm that is needed to support the ZFS deduplication properties in ZFS pool version 21, which were released concurrently
21	OpenSolaris Nevada b128	Deduplication
22	OpenSolaris Nevada b128	zfs receive properties
23	OpenSolaris Nevada b135	slim ZIL
24	OpenSolaris Nevada b137	System attributes. Symlinks now their own object type. Also requires FS v5.
25	OpenSolaris Nevada b140	Improved pool scrubbing and resilvering statistics
26	OpenSolaris Nevada b141	Improved snapshot deletion performance
27	OpenSolaris Nevada b145	Improved snapshot creation performance (particularly recursive snapshots)
28	OpenSolaris Nevada b147	Multiple virtual device replacements

Note: The Solaris version under development by Sun since the release of Solaris 10 in 2005 was codenamed 'Nevada', and was derived from what was the OpenSolaris codebase. 'Solaris Nevada' is the codename for the next-generation Solaris OS to eventually succeed Solaris 10 and this new code was then pulled successively

into new OpenSolaris 'Nevada' snapshot builds.[95] *OpenSolaris is now discontinued and OpenIndiana forked from it.*[96][97] A final build (b134) of OpenSolaris was published by Oracle (2010-Nov-12) as an upgrade path to Solaris 11 Express.

List of operating systems supporting ZFS

List of Operating Systems, distributions and add-ons that support ZFS, the zpool version it supports, and the Solaris build they are based on (if any):

OS	Zpool version	Sun/Oracle Build #	Comments
Oracle Solaris 11.3	37	0.5.11-0.175.3.1.0.5.0	
Oracle Solaris 10 1/13 (U11)	32		
Oracle Solaris 11.2	35	0.5.11-0.175.2.0.0.42.0	
Oracle Solaris 11 2011.11	34	b175	
Oracle Solaris Express 11 2010.11	31	b151a	licensed for testing only
OpenSolaris 2009.06	14	b111b	
OpenSolaris (last dev)	22	b134	
OpenIndiana	5000	b147	distribution based on illumos; creates a name clash naming their build code 'b151a'
Nexenta Core 3.0.1	26	b134+	GNU userland
NexentaStor Community 3.0.1	26	b134+	up to 18 TB, web admin
NexentaStor Community 3.1.0	28	b134+	GNU userland
NexentaStor Community 4.0	5000	b134+	up to 18 TB, web admin
NexentaStor Enterprise	28	b134 +	not free, web admin
GNU/kFreeBSD "Squeeze" (as of 1/31/2013)	14		Requires package "zfsutils"
GNU/kFreeBSD "Wheezy-9" (as of 2/21/2013)	28		Requires package "zfsutils"
FreeBSD	5000		
zfs-fuse 0.7.2	23		suffered from performance issues; defunct
ZFS on Linux 0.6.5.8	5000		0.6.0 release candidate has POSIX layer
KQ Infotech's ZFS on Linux	28		defunct; code integrated into LLNL-supported ZFS on Linux
BeleniX 0.8b1	14	b111	small-size live-CD distribution; once based on OpenSolaris
Schillix 0.7.2	28	b147	small-size live-CD distribution; as SchilliX-ON (https://sourceforge.net/projects/schillix-on/) 0.8.0 based on OpenSolaris
StormOS "hail"			distribution once based on Nexenta Core 2.0+, Debian Linux; superseded by Dyson OS
Jaris			*Ja*panese Sola*ris* distribution; once based on OpenSolaris

MilaX 0.5	20	b128a	small-size live-CD distribution; once based on OpenSolaris
FreeNAS 8.0.2 / 8.2	15		
FreeNAS 8.3.0	28		based on FreeBSD 8.3
FreeNAS 9.1.0	5000		based on FreeBSD 9.1
XigmaNAS 11.4.0.4/12.2.0.4	5000		based on FreeBSD 11.4/12.2
Korona 4.5.0	22	b134	KDE
EON NAS (v0.6)	22	b130	embedded NAS
EON NAS (v1.0beta)	28	b151a	embedded NAS
napp-it (https://www.napp-it.org/)	28/5000	Illumos/Solaris	Storage appliance; OpenIndiana (Hipster), OmniOS, Solaris 11, Linux (ZFS management)
OmniOS CE (http://www.omniosce.org/)	28/5000	illumos-OmniOS branch	minimal stable/LTS storage server distribution based on Illumos, community driven
SmartOS	28/5000	Illumos b151+	minimal live distribution based on Illumos (USB/CD boot); cloud and hypervisor use (KVM)
macOS 10.5, 10.6, 10.7, 10.8, 10.9	5000		via MacZFS; superseded (https://www.geeklan.co.uk/?p=1556) by OpenZFS on OS X (https://openzfsonosx.org)
macOS 10.6, 10.7, 10.8	28		via ZEVO; superseded (https://www.geeklan.co.uk/?p=1556) by OpenZFS on OS X (https://openzfsonosx.org)
NetBSD	22		
MidnightBSD	6		
Proxmox VE	5000		native support since 2014 [1] (https://pve.proxmox.com/wiki/Roadmap#Proxmox_VE_3.2), pve.proxmox.com/wiki/ZFS_on_Linux (https://pve.proxmox.com/wiki/ZFS_on_Linux)
Ubuntu Linux 16.04 LTS, 18.04 LTS, 18.10, 19.10, 20.04 LTS	5000		native support via installable binary module (https://arstechnica.com/gadgets/2016/02/zfs-filesystem-will-be-built-into-ubuntu-16-04-lts-by-default/), wiki.ubuntu.com/ZFS (https://wiki.ubuntu.com/ZFS)
ZFSGuru 10.1.100	5000		

See also

- Comparison of file systems
- List of file systems
- Versioning file system – List of versioning file systems

Notes

a. While RAID 7 is not a standard RAID level, it has been proposed as a catch-all term for any >3 parity RAID configuration[36]

References

1. "Oracle Solaris 11.4 Released for General Availability" (https://blogs.oracle.com/solaris/oracle-solaris-114-released-for-general-availability). August 28, 2018. Archived (https://web.archive.org/web/20180829035408/https://blogs.oracle.com/solaris/oracle-solaris-114-released-for-general-availability) from the original on August 29, 2018. Retrieved September 19, 2020.
2. "1.1 What about the licensing issue?" (http://zfsonlinux.org/faq.html#WhatAboutTheLicensingIssue). Archived (https://web.archive.org/web/20100926104451/http://zfsonlinux.org/faq.html#WhatAboutTheLicensingIssue) from the original on September 26, 2010. Retrieved November 18, 2010.
3. "The OpenZFS project launches" (https://lwn.net/Articles/567090/). LWN.net. September 17, 2013. Archived (https://web.archive.org/web/20131004215341/http://lwn.net/Articles/567090/) from the original on October 4, 2013. Retrieved October 1, 2013.
4. "OpenZFS Announcement" (http://open-zfs.org/wiki/Announcement). OpenZFS. September 17, 2013. Archived (https://web.archive.org/web/20180402091425/http://open-zfs.org/wiki/Announcement) from the original on April 2, 2018. Retrieved September 19, 2013.
5. open-zfs.org /History (http://open-zfs.org/wiki/History) Archived (https://web.archive.org/web/20131224105247/http://open-zfs.org/wiki/History) December 24, 2013, at the Wayback Machine *"OpenZFS is the truly open source successor to the ZFS project [...] Effects of the fork (2010 to date)"*
6. Sean Michael Kerner (September 18, 2013). "LinuxCon: OpenZFS moves Open Source Storage Forward" (http://www.infostor.com/storage-management/linuxcon-openzfs-moves-open-source-storage-forward.html). infostor.com. Archived (https://web.archive.org/web/20140314145457/http://www.infostor.com/storage-management/linuxcon-openzfs-moves-open-source-storage-forward.html) from the original on March 14, 2014. Retrieved October 9, 2013.
7. "The OpenZFS project launches" (https://lwn.net/Articles/567090/). LWN.net. September 17, 2013. Archived (https://web.archive.org/web/20161011141200/https://lwn.net/Articles/567090/) from the original on October 11, 2016. Retrieved October 1, 2013.
8. "OpenZFS – Communities co-operating on ZFS code and features" (http://www.freebsdnews.net/2013/09/23/openzfs-communities-co-operating-on-zfs-code-and-features/). freebsdnews.net. September 23, 2013. Archived (https://web.archive.org/web/20131014000145/http://www.freebsdnews.net/2013/09/23/openzfs-communities-co-operating-on-zfs-code-and-features/) from the original on October 14, 2013. Retrieved March 14, 2014.
9. "19.4. zfs Administration" (https://www.freebsd.org/doc/handbook/zfs-zfs.html). *www.freebsd.org*. Archived (https://web.archive.org/web/20170223045940/https://www.freebsd.org/doc/handbook/zfs-zfs.html) from the original on February 23, 2017. Retrieved February 22, 2017.
10. Salus, Peter (1994). *A Quarter Century of Unix*. Addison-Wesley. pp. 199–200. ISBN 0-201-54777-5.
11. "What are SunOS and Solaris?" (http://kb.iu.edu/data/agjq.html). *Knowledge Base*. Indiana University Technology Services. May 20, 2013. Retrieved November 10, 2014.
12. Brown, David. "A Conversation with Jeff Bonwick and Bill Moore" (http://queue.acm.org/detail.cfm?id=1317400). *ACM Queue*. Association for Computing Machinery. Archived (https://web.archive.org/web/20110716221142/http://queue.acm.org/detail.cfm?id=1317400) from the original on July 16, 2011. Retrieved November 17, 2015.
13. "ZFS: the last word in file systems" (https://web.archive.org/web/20060428092023/http://www.sun.com/2004-0914/feature/). Sun Microsystems. September 14, 2004. Archived from the original (http://www.sun.com/2004-0914/feature/) on April 28, 2006. Retrieved April 30, 2006.
14. Matthew Ahrens (November 1, 2011). "ZFS 10 year anniversary" (https://web.archive.org/web/20160628084029/http://blog.delphix.com/matt/2011/11/01/zfs-10-year-anniversary/). Archived from the original (http://blog.delphix.com/matt/2011/11/01/zfs-10-year-anniversary/) on June 28, 2016. Retrieved July 24, 2012.

15. Bonwick, Jeff (October 31, 2005). "ZFS: The Last Word in Filesystems" (https://web.archive.org/web/20130619165135/https://blogs.oracle.com/bonwick/en_US/entry/zfs_the_last_word_in). *blogs.oracle.com*. Archived from the original (https://blogs.oracle.com/bonwick/en_US/entry/zfs_the_last_word_in) on June 19, 2013. Retrieved June 22, 2013.
16. "Sun Celebrates Successful One-Year Anniversary of OpenSolaris" (http://www.sun.com/smi/Press/sunflash/2006-06/sunflash.20060620.1.xml). Sun Microsystems. June 20, 2006. Archived (https://web.archive.org/web/20080928001733/http://www.sun.com/smi/Press/sunflash/2006-06/sunflash.20060620.1.xml) from the original on September 28, 2008. Retrieved April 30, 2018.
17. Michael Singer (January 25, 2005). "Sun Cracks Open Solaris" (http://www.internetnews.com/dev-news/article.php/3463621). InternetNews.com. Retrieved April 12, 2010.
18. "ZFS FAQ at OpenSolaris.org" (https://web.archive.org/web/20110515061128/http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq/). Sun Microsystems. Archived from the original (http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq#HWhatdoesZFSstandfor) on May 15, 2011. Retrieved May 18, 2011. "The largest SI prefix we liked was 'zetta' ('yotta' was out of the question)"
19. Jeff Bonwick (May 3, 2006). "You say zeta, I say zetta" (https://web.archive.org/web/20170223222515/https://blogs.oracle.com/bonwick/en_US/entry/you_say_zeta_i_say). *Jeff Bonwick's Blog*. Archived from the original (https://blogs.oracle.com/bonwick/en_US/entry/you_say_zeta_i_say) on February 23, 2017. Retrieved April 21, 2017. "So we finally decided to unpimp the name back to ZFS, which doesn't stand for anything."
20. "Oracle and NetApp dismiss ZFS lawsuits" (https://www.theregister.co.uk/2010/09/09/oracle_netapp_zfs_dismiss/). theregister.co.uk. September 9, 2010. Archived (https://web.archive.org/web/20170909065736/http://www.theregister.co.uk/2010/09/09/oracle_netapp_zfs_dismiss/) from the original on September 9, 2017. Retrieved December 24, 2013.
21. The Extended file system (Ext) has metadata structure copied from UFS. "Rémy Card (Interview, April 1998)" (https://web.archive.org/web/20120204082557/http://www.april.org/groupes/entretiens/remy_card.html). April Association. April 19, 1999. Archived from the original (http://www.april.org/groupes/entretiens/remy_card.html) on February 4, 2012. Retrieved February 8, 2012. (In French)
22. Vijayan Prabhakaran (2006). "IRON FILE SYSTEMS" (http://pages.cs.wisc.edu/~vijayan/vijayan-thesis.pdf) (PDF). *Doctor of Philosophy in Computer Sciences*. University of Wisconsin-Madison. Archived (https://web.archive.org/web/20110429011617/http://pages.cs.wisc.edu/~vijayan/vijayan-thesis.pdf) (PDF) from the original on April 29, 2011. Retrieved June 9, 2012.
23. "Parity Lost and Parity Regained" (http://www.cs.wisc.edu/adsl/Publications/parity-fast08.html). Archived (https://web.archive.org/web/20100615101314/http://www.cs.wisc.edu/adsl/Publications/parity-fast08.html) from the original on June 15, 2010. Retrieved November 29, 2010.
24. "An Analysis of Data Corruption in the Storage Stack" (http://www.cs.wisc.edu/adsl/Publications/corruption-fast08.pdf) (PDF). Archived (https://web.archive.org/web/20100615111630/http://www.cs.wisc.edu/adsl/Publications/corruption-fast08.pdf) (PDF) from the original on June 15, 2010. Retrieved November 29, 2010.
25. "Impact of Disk Corruption on Open-Source DBMS" (http://www.cs.wisc.edu/adsl/Publications/corrupt-mysql-icde10.pdf) (PDF). Archived (https://web.archive.org/web/20100615090935/http://www.cs.wisc.edu/adsl/Publications/corrupt-mysql-icde10.pdf) (PDF) from the original on June 15, 2010. Retrieved November 29, 2010.
26. Kadav, Asim; Rajimwale, Abhishek. "Reliability Analysis of ZFS" (http://pages.cs.wisc.edu/~kadav/zfs/zfsrel.pdf) (PDF). Archived (https://web.archive.org/web/20130921054610/http://pages.cs.wisc.edu/~kadav/zfs/zfsrel.pdf) (PDF) from the original on September 21, 2013. Retrieved September 19, 2013.

27. Yupu Zhang; Abhishek Rajimwale; Andrea C. Arpaci-Dusseau; Remzi H. Arpaci-Dusseau. "End-to-end Data Integrity for File Systems: A ZFS Case Study" (http://www.cs.wisc.edu/wind/Publications/zfs-corruption-fast10.pdf) (PDF). Madison: Computer Sciences Department, University of Wisconsin. p. 14. Archived (https://web.archive.org/web/20110626130632/http://www.cs.wisc.edu/wind/Publications/zfs-corruption-fast10.pdf) (PDF) from the original on June 26, 2011. Retrieved December 6, 2010.
28. Larabel, Michael. "Benchmarking ZFS and UFS On FreeBSD vs. EXT4 & Btrfs On Linux" (https://www.phoronix.com/scan.php?page=article&item=zfs_ext4_btrfs&num=2). Phoronix Media 2012. Archived (https://web.archive.org/web/20161129093628/https://www.phoronix.com/scan.php?page=article&item=zfs_ext4_btrfs&num=2) from the original on November 29, 2016. Retrieved November 21, 2012.
29. Larabel, Michael. "Can DragonFlyBSD's HAMMER Compete With Btrfs, ZFS?" (https://www.phoronix.com/scan.php?page=article&item=dragonfly_hammer&num=3). Phoronix Media 2012. Archived (https://web.archive.org/web/20161129033518/https://www.phoronix.com/scan.php?page=article&item=dragonfly_hammer&num=3) from the original on November 29, 2016. Retrieved November 21, 2012.
30. Bonwick, Jeff (December 8, 2005). "ZFS End-to-End Data Integrity" (https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data). *blogs.oracle.com*. Archived (https://web.archive.org/web/20120403015447/https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data) from the original on April 3, 2012. Retrieved September 19, 2013.
31. Cook, Tim (November 16, 2009). "Demonstrating ZFS Self-Healing" (https://blogs.oracle.com/timc/entry/demonstrating_zfs_self_healing). *blogs.oracle.com*. Archived (https://web.archive.org/web/20110812031213/http://blogs.oracle.com/timc/entry/demonstrating_zfs_self_healing) from the original on August 12, 2011. Retrieved February 1, 2015.
32. Ranch, Richard (May 4, 2007). "ZFS, copies, and data protection" (https://web.archive.org/web/20160818143115/https://blogs.oracle.com/relling/entry/zfs_copies_and_data_protection). *blogs.oracle.com*. Archived from the original (https://blogs.oracle.com/relling/entry/zfs_copies_and_data_protection) on August 18, 2016. Retrieved February 2, 2015.
33. "ZFS Without Tears: Using ZFS without ECC memory" (https://www.csparks.com/ZFS%20Without%20Tears.md#toc_using-zfs-without-ecc-memory). *www.csparks.com*. December 2015. Archived (https://web.archive.org/web/20210113202506/https://www.csparks.com/ZFS%20Without%20Tears.md#toc_using-zfs-without-ecc-memory) from the original on January 13, 2021. Retrieved June 16, 2020.
34. wdc.custhelp.com. "Difference between Desktop edition and RAID (Enterprise) edition drives" (http://wdc.custhelp.com/app/answers/detail/a_id/1397/~/difference-between-desktop-edition-and-raid-%28enterprise%29-edition-drives). Archived (https://web.archive.org/web/20150105040018/http://wdc.custhelp.com/app/answers/detail/a_id/1397/~/difference-between-desktop-edition-and-raid-(enterprise)-edition-drives) from the original on January 5, 2015. Retrieved September 8, 2011.
35. Bonwick, Jeff (November 17, 2005). "RAID-Z" (https://web.archive.org/web/20141216015058/https://blogs.oracle.com/bonwick/en_US/entry/raid_z). *Jeff Bonwick's Blog*. Oracle Blogs. Archived from the original (https://blogs.oracle.com/bonwick/en_US/entry/raid_z) on December 16, 2014. Retrieved February 1, 2015.
36. Leventhal, Adam (December 17, 2009). "Triple-Parity RAID and Beyond" (https://queue.acm.org/detail.cfm?id=1670144). *Queue*. **7** (11): 30. doi:10.1145/1661785.1670144 (https://doi.org/10.1145%2F1661785.1670144). Archived (https://web.archive.org/web/20190315183030/https://queue.acm.org/detail.cfm?id=1670144) from the original on March 15, 2019. Retrieved April 12, 2019.
37. "ZFS Raidz Performance, Capacity and integrity" (https://web.archive.org/web/20171127225445/https://calomel.org/zfs_raid_speed_capacity.html). *calomel.org*. Archived from the original (https://calomel.org/zfs_raid_speed_capacity.html) on November 27, 2017. Retrieved June 23, 2017.

38. "Why RAID 6 stops working in 2019" (https://www.zdnet.com/blog/storage/why-raid-6-stops-working-in-2019/805). *ZDNet*. February 22, 2010. Archived (https://web.archive.org/web/20141031164950/http://www.zdnet.com/blog/storage/why-raid-6-stops-working-in-2019/805) from the original on October 31, 2014. Retrieved October 26, 2014.
39. "No fsck utility equivalent exists for ZFS. This utility has traditionally served two purposes, those of file system repair and file system validation." "Checking ZFS File System Integrity" (http://docs.oracle.com/cd/E23823_01/html/819-5461/gbbwa.html). Oracle. Archived (https://web.archive.org/web/20130131040337/http://docs.oracle.com/cd/E23823_01/html/819-5461/gbbwa.html) from the original on January 31, 2013. Retrieved November 25, 2012.
40. "ZFS Scrubs" (https://web.archive.org/web/20121127160745/http://doc.freenas.org/index.php/ZFS_Scrubs). freenas.org. Archived from the original (http://doc.freenas.org/index.php/ZFS_Scrubs) on November 27, 2012. Retrieved November 25, 2012.
41. "You should also run a scrub prior to replacing devices or temporarily reducing a pool's redundancy to ensure that all devices are currently operational." "ZFS Best Practices Guide" (https://web.archive.org/web/20150905142644/http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide). solarisinternals.com. Archived from the original (http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide) on September 5, 2015. Retrieved 25 November 2012.
42. Jeff Bonwick. "128-bit storage: are you high?" (https://blogs.oracle.com/bonwick/entry/128_bit_storage_are_you). *oracle.com*. Archived (https://web.archive.org/web/20150529160107/https://blogs.oracle.com/bonwick/entry/128_bit_storage_are_you) from the original on May 29, 2015. Retrieved May 29, 2015.
43. "ZFS: Boils the Ocean, Consumes the Moon (Dave Brillhart's Blog)" (https://web.archive.org/web/20151208192725/https://blogs.oracle.com/dcb/entry/zfs_boils_the_ocean_consumes). Archived from the original (https://blogs.oracle.com/dcb/entry/zfs_boils_the_ocean_consumes) on December 8, 2015. Retrieved December 19, 2015.
44. "Solaris ZFS Administration Guide" (http://download.oracle.com/docs/cd/E19253-01/819-5461/zfsover-2/index.html). Oracle Corporation. Archived (https://web.archive.org/web/20210113202613/https://docs.oracle.com/cd/E19253-01/819-5461/zfsover-2/index.html) from the original on January 13, 2021. Retrieved February 11, 2011.
45. "Encrypting ZFS File Systems" (http://download.oracle.com/docs/cd/E19963-01/html/821-1448/gkkih.html). Archived (https://web.archive.org/web/20110623190612/http://download.oracle.com/docs/cd/E19963-01/html/821-1448/gkkih.html) from the original on June 23, 2011. Retrieved May 2, 2011.
46. "Having my secured cake and Cloning it too (aka Encryption + Dedup with ZFS)" (https://blogs.oracle.com/darren/entry/compress_encrypt_checksum_deduplicate_with). Archived (https://web.archive.org/web/20130529061709/https://blogs.oracle.com/darren/entry/compress_encrypt_checksum_deduplicate_with) from the original on May 29, 2013. Retrieved October 9, 2012.
47. "ZFS – Debian Wiki" (https://wiki.debian.org/ZFS#Encryption). *wiki.debian.org*. Archived (https://web.archive.org/web/20190908104724/https://wiki.debian.org/ZFS#Encryption) from the original on September 8, 2019. Retrieved December 10, 2019.
48. "Solaris ZFS Enables Hybrid Storage Pools—Shatters Economic and Performance Barriers" (http://download.intel.com/design/flash/nand/SolarisZFS_SolutionBrief.pdf) (PDF). Sun.com. September 7, 2010. Archived (https://web.archive.org/web/20111017204544/http://download.intel.com/design/flash/nand/SolarisZFS_SolutionBrief.pdf) (PDF) from the original on October 17, 2011. Retrieved November 4, 2011.
49. Gregg, Brendan. "ZFS L2ARC" (http://dtrace.org/blogs/brendan/2008/07/22/zfs-l2arc/). *Brendan's blog*. Dtrace.org. Archived (https://web.archive.org/web/20111106031228/http://dtrace.org/blogs/brendan/2008/07/22/zfs-l2arc/) from the original on November 6, 2011. Retrieved October 5, 2012.

50. Gregg, Brendan (October 8, 2009). "Hybrid Storage Pool: Top Speeds" (http://dtrace.org/blogs/brendan/2009/10/08/hybrid-storage-pool-top-speeds/). *Brendan's blog*. Dtrace.org. Archived (https://web.archive.org/web/20160405120351/http://dtrace.org/blogs/brendan/2009/10/08/hybrid-storage-pool-top-speeds/) from the original on April 5, 2016. Retrieved August 15, 2017.
51. "Solaris ZFS Performance Tuning: Synchronous Writes and the ZIL" (http://constantin.glez.de/blog/2010/07/solaris-zfs-synchronous-writes-and-zil-explained). Constantin.glez.de. July 20, 2010. Archived (https://web.archive.org/web/20120623100347/http://constantin.glez.de/blog/2010/07/solaris-zfs-synchronous-writes-and-zil-explained) from the original on June 23, 2012. Retrieved October 5, 2012.
52. "ZFS On-Disk Specification" (https://web.archive.org/web/20081230170058/http://www.opensolaris.org/os/community/zfs/docs/ondiskformat0822.pdf) (PDF). Sun Microsystems, Inc. 2006. Archived from the original (http://opensolaris.org/os/community/zfs/docs/ondiskformat0822.pdf) (PDF) on December 30, 2008. See section 2.4.
53. Eric Sproul (May 21, 2009). "ZFS Nuts and Bolts" (http://www.slideshare.net/esproul/zfs-nuts-and-bolts). slideshare.net. pp. 30–31. Archived (https://web.archive.org/web/20140622215818/http://www.slideshare.net/esproul/zfs-nuts-and-bolts) from the original on June 22, 2014. Retrieved June 8, 2014.
54. "ZFS Deduplication" (https://blogs.oracle.com/bonwick/zfs-deduplication-v2). *blogs.oracle.com*. Archived (https://web.archive.org/web/20191224020451/https://blogs.oracle.com/bonwick/zfs-deduplication-v2) from the original on December 24, 2019. Retrieved November 25, 2019.
55. Gary Sims (January 4, 2012). "Building ZFS Based Network Attached Storage Using FreeNAS 8" (https://web.archive.org/web/20120507220120/http://www.trainsignal.com/blog/zfs-nas-setup-guide). *TrainSignal Training*. TrainSignal, Inc. Archived from the original (http://www.trainsignal.com/blog/zfs-nas-setup-guide) (Blog) on May 7, 2012. Retrieved June 9, 2012.
56. Ray Van Dolson (May 2011). "[zfs-discuss] Summary: Deduplication Memory Requirements" (https://web.archive.org/web/20120425142508/http://mail.opensolaris.org/pipermail/zfs-discuss/2011-May/048159.html). zfs-discuss mailing list. Archived from the original (http://mail.opensolaris.org/pipermail/zfs-discuss/2011-May/048159.html) on April 25, 2012.
57. "ZFSTuningGuide" (http://wiki.freebsd.org/ZFSTuningGuide#Deduplication). Archived (https://web.archive.org/web/20120116113648/http://wiki.freebsd.org/ZFSTuningGuide#Deduplication) from the original on January 16, 2012. Retrieved January 3, 2012.
58. Chris Mellor (October 12, 2012). "GreenBytes brandishes full-fat clone VDI pumper" (https://www.theregister.co.uk/2012/10/12/greenbytes_chairman/). *The Register*. Archived (https://web.archive.org/web/20130324085407/http://www.theregister.co.uk/2012/10/12/greenbytes_chairman/) from the original on March 24, 2013. Retrieved August 29, 2013.
59. Chris Mellor (June 1, 2012). "Newcomer gets out its box, plans to sell it cheaply to all comers" (https://www.theregister.co.uk/2012/06/01/tegile_zebi/). *The Register*. Archived (https://web.archive.org/web/20130812033031/http://www.theregister.co.uk/2012/06/01/tegile_zebi/) from the original on August 12, 2013. Retrieved August 29, 2013.
60. Chris Mellor (December 11, 2014). "Dedupe, dedupe... dedupe, dedupe, dedupe: Oracle polishes ZFS diamond" (https://www.theregister.co.uk/2014/12/11/oracle_improving_zfs_dedupe/). The Register. Archived (https://web.archive.org/web/20170707155821/https://www.theregister.co.uk/2014/12/11/oracle_improving_zfs_dedupe/) from the original on July 7, 2017. Retrieved December 17, 2014.
61. "Checksums and Their Use in ZFS" (https://github.com/zfsonlinux/zfs/wiki/Checksums). *github.com*. September 2, 2018. Archived (https://web.archive.org/web/20190719225739/https://github.com/zfsonlinux/zfs/wiki/Checksums) from the original on July 19, 2019. Retrieved July 11, 2019.
62. "Solaris ZFS Administration Guide" (https://web.archive.org/web/20110205111337/http://download.oracle.com/docs/cd/E19963-01/821-1448/gavwq/index.html). *Chapter 6 Managing ZFS File Systems*. Archived from the original (http://download.oracle.com/docs/cd/E19963-01/821-1448/gavwq/index.html) on February 5, 2011. Retrieved March 17, 2009.

63. "Smokin' Mirrors" (https://blogs.oracle.com/bonwick/entry/smokin_mirrors). *blogs.oracle.com*. May 2, 2006. Archived (https://web.archive.org/web/20111216163425/http://blogs.oracle.com/bonwick/entry/smokin_mirrors) from the original on December 16, 2011. Retrieved February 13, 2012.
64. "ZFS Block Allocation" (https://blogs.oracle.com/bonwick/entry/zfs_block_allocation). *Jeff Bonwick's Weblog*. November 4, 2006. Archived (https://web.archive.org/web/20121102073644/https://blogs.oracle.com/bonwick/entry/zfs_block_allocation) from the original on November 2, 2012. Retrieved February 23, 2007.
65. "Ditto Blocks — The Amazing Tape Repellent" (https://web.archive.org/web/20130526084314/https://blogs.oracle.com/bill/entry/ditto_blocks_the_amazing_tape). *Flippin' off bits Weblog*. May 12, 2006. Archived from the original (https://blogs.oracle.com/bill/entry/ditto_blocks_the_amazing_tape) on May 26, 2013. Retrieved March 1, 2007.
66. "Adding new disks and ditto block behaviour" (https://web.archive.org/web/20110823190119/http://opensolaris.org/jive/thread.jspa?messageID=417776). Archived from the original (http://opensolaris.org/jive/thread.jspa?messageID=417776) on August 23, 2011. Retrieved October 19, 2009.
67. "OpenSolaris.org" (https://web.archive.org/web/20090508081240/http://www.opensolaris.org/os/community/zfs/version/15/). Sun Microsystems. Archived from the original (http://www.opensolaris.org/os/community/zfs/version/15/) on May 8, 2009. Retrieved May 22, 2009.
68. "What's new in Solaris 11 Express 2010.11" (http://www.oracle.com/technetwork/server-storage/solaris11/documentation/solaris-express-whatsnew-201011-175308.pdf) (PDF). Oracle. Archived (https://web.archive.org/web/20101116073641/http://www.oracle.com/technetwork/server-storage/solaris11/documentation/solaris-express-whatsnew-201011-175308.pdf) (PDF) from the original on November 16, 2010. Retrieved November 17, 2010.
69. "10. Sharing — FreeNAS User Guide 9.3 Table of Contents" (http://doc.freenas.org/9.3/freenas_sharing.html). *doc.freenas.org*. Archived (https://web.archive.org/web/20170107211538/http://doc.freenas.org/9.3/freenas_sharing.html) from the original on January 7, 2017. Retrieved February 23, 2017.
70. Zhang, Yupu; Rajimwale, Abhishek; Arpaci-Dusseau, Andrea C.; Arpaci-Dusseau, Remzi H. (January 2, 2018). "End-to-end Data Integrity for File Systems: A ZFS Case Study" (http://dl.acm.org/citation.cfm?id=1855511.1855514). USENIX Association. p. 3. Archived (https://web.archive.org/web/20210113202508/https://dl.acm.org/doi/10.5555/1855511.1855514) from the original on January 13, 2021. Retrieved June 7, 2020 – via ACM Digital Library.
71. "Ars walkthrough: Using the ZFS next-gen filesystem on Linux" (https://arstechnica.com/civis/viewtopic.php?f=2&t=1235679&p=26303271#p26303271). *arstechnica.com*. Archived (https://web.archive.org/web/20170210103308/https://arstechnica.com/civis/viewtopic.php?f=2&t=1235679&p=26303271#p26303271) from the original on February 10, 2017. Retrieved June 19, 2017.
72. "Bug ID 4852783: reduce pool capacity" (https://web.archive.org/web/20090629081219/http://bugs.opensolaris.org/view_bug.do?bug_id=4852783). OpenSolaris Project. Archived from the original (http://bugs.opensolaris.org/view_bug.do?bug_id=4852783) on June 29, 2009. Retrieved March 28, 2009.
73. Goebbels, Mario (April 19, 2007). "Permanently removing vdevs from a pool" (http://mail.opensolaris.org/pipermail/zfs-discuss/2007-April/010356.html). *zfs-discuss* (Mailing list). archive link (https://marc.info/?l=zfs-discuss&m=122362857630617&w=1) Archived (https://web.archive.org/web/20210113202545/https://marc.info/?l=zfs-discuss&m=122362857630617&w=1) January 13, 2021, at the Wayback Machine
74. Chris Siebenmann Information on future vdev removal (https://utcc.utoronto.ca/~cks/space/blog/solaris/ZFSPoolShrinkingIsComing) Archived (https://web.archive.org/web/20160811202352/https://utcc.utoronto.ca/~cks/space/blog/solaris/ZFSPoolShrinkingIsComing) August 11, 2016, at the Wayback Machine, Univ Toronto, blog, quote: informal Twitter announcement by Alex Reece (https://twitter.com/awreece/status/555533793700765696) Archived (https://web.archive.org/web/20160811220752/https://twitter.com/awreece/status/555533793700765696) August 11, 2016, at the Wayback Machine

75. "Expand-O-Matic RAID Z" (https://blogs.oracle.com/ahl/entry/expand_o_matic_raid_z). Adam Leventhal. April 7, 2008. Archived (https://web.archive.org/web/20111228072550/http://blogs.oracle.com/ahl/entry/expand_o_matic_raid_z) from the original on December 28, 2011. Retrieved April 16, 2012.
76. "zpool(1M)" (http://download.oracle.com/docs/cd/E19253-01/816-5166/zpool-1m/?l=en&n=1&a=view). Download.oracle.com. June 11, 2010. Archived (https://web.archive.org/web/20210113202512/https://docs.oracle.com/cd/E19253-01/816-5166/zpool-1m/?l=en&n=1&a=view) from the original on January 13, 2021. Retrieved November 4, 2011.
77. brendan (December 2, 2008). "A quarter million NFS IOPS" (https://web.archive.org/web/20111217073127/http://blogs.oracle.com/brendan/entry/a_quarter_million_nfs_iops). Oracle Sun. Archived from the original (https://blogs.oracle.com/brendan/entry/a_quarter_million_nfs_iops) on December 17, 2011. Retrieved January 28, 2012.
78. "Data Management Features – What's New in Oracle® Solaris 11.4" (https://docs.oracle.com/cd/E37838_01/html/E60974/dmgmt.html#scrolltoc). Archived (https://web.archive.org/web/20190924101556/https://docs.oracle.com/cd/E37838_01/html/E60974/dmgmt.html#scrolltoc) from the original on September 24, 2019. Retrieved October 9, 2019.
79. Leventhal, Adam. "Triple-Parity RAID Z" (http://dtrace.org/blogs/ahl/2009/07/21/triple-parity-raid-z). *Adam Leventhal's blog*. Archived (https://web.archive.org/web/20110416063604/http://dtrace.org/blogs/ahl/2009/07/21/triple-parity-raid-z/) from the original on April 16, 2011. Retrieved December 19, 2013.
80. "Turbocharging ZFS Data Recovery" (https://www.delphix.com/blog/openzfs-pool-import-recovery). Archived (https://web.archive.org/web/20181129054344/https://www.delphix.com/blog/openzfs-pool-import-recovery) from the original on November 29, 2018. Retrieved November 29, 2018.
81. "ZFS and OpenZFS" (https://www.ixsystems.com/blog/zfs-vs-openzfs/). iXSystems. Retrieved May 18, 2020.
82. "Sun rolls out its own storage appliances" (http://www.techworld.com.au/article/266682/sun_rolls_its_own_storage_appliances/). techworld.com.au. November 11, 2008. Archived (https://web.archive.org/web/20131113194325/http://www.techworld.com.au/article/266682/sun_rolls_its_own_storage_appliances/) from the original on November 13, 2013. Retrieved November 13, 2013.
83. Chris Mellor (October 2, 2013). "Oracle muscles way into seat atop the benchmark with hefty ZFS filer" (https://www.theregister.co.uk/2013/10/02/oracle_zs3/). theregister.co.uk. Archived (https://web.archive.org/web/20170707160152/https://www.theregister.co.uk/2013/10/02/oracle_zs3/) from the original on July 7, 2017. Retrieved July 7, 2014.
84. "Unified ZFS Storage Appliance built in Silicon Valley by iXsystem" (http://www.ixsystems.com/storage/truenas/). ixsystems.com. Archived (https://web.archive.org/web/20140703151518/http://www.ixsystems.com/storage/truenas/) from the original on July 3, 2014. Retrieved July 7, 2014.
85. "TrueNAS 12 & TrueNAS SCALE are officially here!" (https://www.ixsystems.com/blog/truenas-12-truenas-scale-are-officially-here-issue-87/). ixsystems.com. Retrieved January 2, 2021.
86. "ReadyDATA 516 – Unified Network Storage" (http://www.netgear.com/images/pdf/ReadyDATA_516_DS.pdf) (PDF). netgear.com. Archived (https://web.archive.org/web/20140715002129/http://www.netgear.com/images/pdf/ReadyDATA_516_DS.pdf) (PDF) from the original on July 15, 2014. Retrieved July 7, 2014.
87. Jim Salter (December 17, 2015). "rsync.net: ZFS Replication to the cloud is finally here—and it's fast" (https://arstechnica.com/information-technology/2015/12/rsync-net-zfs-replication-to-the-cloud-is-finally-here-and-its-fast/). arstechnica.com. Archived (https://web.archive.org/web/20170822052447/https://arstechnica.com/information-technology/2015/12/rsync-net-zfs-replication-to-the-cloud-is-finally-here-and-its-fast/) from the original on August 22, 2017. Retrieved August 21, 2017.

88. rsync.net, Inc. "Cloud Storage with ZFS send and receive over SSH" (http://www.rsync.net/products/zfsintro.html). rsync.net. Archived (https://web.archive.org/web/20170721090348/http://www.rsync.net/products/zfsintro.html) from the original on July 21, 2017. Retrieved August 21, 2017.
89. Steven Stallion / Oracle (August 13, 2010). "Update on SXCE" (http://sstallion.blogspot.com/2010/08/opensolaris-is-dead.html). Iconoclastic Tendencies.
90. Alasdair Lumsden. "OpenSolaris cancelled, to be replaced with Solaris 11 Express" (https://web.archive.org/web/20100816225601/http://mail.opensolaris.org/pipermail/opensolaris-discuss/2010-August/059310.html). *osol-discuss* (Mailing list). Archived from the original (http://mail.opensolaris.org/pipermail/opensolaris-discuss/2010-August/059310.html) on August 16, 2010. Retrieved November 24, 2014.
91. Solaris still sorta open, but OpenSolaris distro is dead (https://arstechnica.com/information-technology/2010/08/solaris-still-sorta-open-but-opensolaris-distro-is-dead/) Archived (https://web.archive.org/web/20170905030542/https://arstechnica.com/information-technology/2010/08/solaris-still-sorta-open-but-opensolaris-distro-is-dead/) September 5, 2017, at the Wayback Machine on Ars Technica by Ryan Paul (Aug 16, 2010)
92. Garrett D'Amore (August 3, 2010). "Illumos - Hope and Light Springs Anew - Presented by Garrett D'Amore" (http://www.illumos.org/attachments/download/3/illumos.pdf) (PDF). illumos.org. Retrieved August 3, 2010.
93. "Whither OpenSolaris? Illumos Takes Up the Mantle" (https://web.archive.org/web/20150926053916/http://www.linuxinsider.com/story/76669.html). Archived from the original (http://www.linuxinsider.com/story/76669.html) on September 26, 2015.
94. Garrett D'Amore (August 13, 2010). "The Hand May Be Forced" (http://gdamore.blogspot.com/2010/08/hand-may-be-forced.html). Retrieved November 14, 2013.
95. "While under Sun Microsystems' control, there were bi-weekly snapshots of Solaris Nevada (the codename for the next-generation Solaris OS to eventually succeed Solaris 10) and this new code was then pulled into new OpenSolaris preview snapshots available at Genunix.org. The stable releases of OpenSolaris are based off of [*sic*] these Nevada builds." Larabel, Michael. "It Looks Like Oracle Will Stand Behind OpenSolaris" (https://www.phoronix.com/scan.php?page=news_item&px=ODQyOQ). Phoronix Media. Archived (https://web.archive.org/web/20161129011453/https://www.phoronix.com/scan.php?page=news_item&px=ODQyOQ) from the original on November 29, 2016. Retrieved November 21, 2012.
96. Ljubuncic, Igor (May 23, 2011). "OpenIndiana — there's still hope" (http://distrowatch.com/weekly.php?issue=20110523#feature). DistroWatch. Archived (https://web.archive.org/web/20121027220918/http://distrowatch.com/weekly.php?issue=20110523#feature) from the original on October 27, 2012. Retrieved November 21, 2012.
97. "Welcome to Project OpenIndiana!" (http://openindiana.org/). Project OpenIndiana. September 10, 2010. Archived (https://web.archive.org/web/20121127012553/http://openindiana.org/) from the original on November 27, 2012. Retrieved September 14, 2010.

Bibliography

- Watanabe, Scott (November 23, 2009). *Solaris ZFS Essentials* (https://web.archive.org/web/20121001091103/http://www.informit.com/store/product.aspx?isbn=0137000103) (1st ed.). Prentice Hall. p. 256. ISBN 978-0-13-700010-4. Archived from the original (http://www.informit.com/store/product.aspx?isbn=0137000103) on October 1, 2012.

External links

- Fork Yeah! The Rise and Development of illumos (https://www.slideshare.net/bcantrill/fork-yeah-the-rise-and-development-of-illumos) - slide show covering much of the history of Solaris, the

decision to open source by Sun, the creation of ZFS, and the events causing it to be closed sourced and forked after Oracle's acquisition.

- The best cloud File System was created before the cloud existed (https://web.archive.org/web/20181215225125/https://mauteam.org/blog/infrastructure/40-the-best-cloud-file-system-was-created-before-the-cloud-existed/) (archived on Dec. 15, 2018)
- Comparison of SVM mirroring and ZFS mirroring (http://www.i-justblog.com/2009/08/zfs-tip-comparison-of-svm-mirroring-and.html)
- EON ZFS Storage (NAS) distribution (https://sites.google.com/site/eonstorage/)
- End-to-end Data Integrity for File Systems: A ZFS Case Study (http://www.usenix.org/events/fast10/tech/full_papers/zhang.pdf)
- ZFS – The Zettabyte File System (https://web.archive.org/web/20130228192209/http://academy.inseptra.com/featured/zfs-the-zettabyte-file-system) (archived on Feb. 28, 2013)
- ZFS and RAID-Z: The Über-FS? (http://pages.cs.wisc.edu/~remzi/Classes/736/Fall2007/Projects/BrianKynan/paper.pdf)
- ZFS: The Last Word In File Systems (https://web.archive.org/web/20170829215812/https://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf), by Jeff Bonwick and Bill Moore (archived on Aug. 29, 2017)
- Visualizing the ZFS intent log (ZIL) (https://pthree.org/2013/04/19/zfs-administration-appendix-a-visualizing-the-zfs-intent-log/), April 2013, by Aaron Toponce
- Features of illumos (https://illumos.org/docs/about/features/) including OpenZFS
    - Previous wiki page with more links: Getting Started with ZFS (https://web.archive.org/web/20181106212909/https://wiki.illumos.org/display/illumos/ZFS), Sep. 15, 2014 (archived on Dec. 30, 2018), part of the illumos documentation

---