PETER BAER GALVIN

Pete's all things Sun (PATS): the state of ZFS

Peter Baer Galvin (www.galvin.info) is the Chief Technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is coauthor of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide.

pbg@cptech.com

WE ARE IN THE MIDST OF A FILE SYSTEM revolution, and it is called ZFS. File system revolutions do not happen very often, so when they do, excitement ensues—maybe not as much excitement as during a political revolution, but file system revolutions are certainly exciting for geeks. What are the signs that we are in a revolution? By my definition, a revolution starts when the peasants (we sysadmins) are unhappy with the status quo, some group comes up with a better idea, and the idea spreads beyond that group and takes on a life of its own. Of course, in a successful revolution the new idea actually takes hold and does improve the peasant's lot.

;login: has had two previous articles about ZFS. The first, by Tom Haynes, provided an overview of ZFS in the context of building a home file server (*;login:*, vol. 31, no. 3). In the second, Dawidek and McKusick (*;login:*, vol. 32, no. 3) discuss ZFS's fundamental features, as well as the porting of ZFS to FreeBSD. This month I won't repeat those efforts, but, rather, continue on from that ZFS coverage to complete the list of ZFS features, discuss field experiences and the ZFS adoption status, and try to see into the future of ZFS. The revolution started in November 2005 when ZFS was made available for download. Now let's check in with the revolution and see how it is progressing.

The Current Feature List

This detailed summary of all of the current ZFS features can serve as a checklist to determine whether ZFS can do what is needed in a given environment. The following feature list is accurate as of April 2008. All of the features are included in the current commercial Solaris release (Update 4, also known as 11/07).

- Disks or slices are allocated to storage "pools" in RAID 0, 1, 0+1, 1+0, 5 (RAID Z), and 6 (RAID Z2) formats. (Note that RAID Z and Z2 are optimized over the standard RAID levels to remove the RAID 5 "write hole.")
- File systems live within a pool and grow and shrink automatically within that pool as needed.

- File systems can contain other file systems. (Think of ZFS file systems as being more like directories, with many new attributes.)
- File system attributes include compressed, NFS exported, iSCSI exported, owned by a container (a "dataset"), mount point, and user-definable.
- Copy-on-write allocation, data, and meta-data are always consistent on disk; no "fsck" is needed.
- There is end-to-end data and meta-data integrity checking via a Merkel tree structure; important blocks are automatically "dittoed," giving data protection far beyond other solutions.
- The system is "self-healing": If corrupt data or meta-data is found and a noncorrupt copy exists, the corrupt version is replaced with the good version.
- Highly efficient snapshots and clones (read-write snapshots) can be made.
- One can roll back a file system to a given snapshot and promote a clone to replace its parent file system.
- There are quotas to limit the size of a file system and reservations to guarantee space to a file system.
- One can make full and incremental backups and restores to a file or between two systems (replication) via `send` and `receive` commands.
- There is support for multiple block sizes, pipelined I/O, dynamic striping, and intelligent prefetch for performance.
- Fast re-silvering (re-mirroring) is allowed.
- ACLs are in NFS V4/NTFS style.
- Adaptive "endian-ness" allows import and export of ZFS pools between varying-architecture systems; new data writes are in the native format of the current system.
- Requestable pool integrity checks (scrubs) to search for corruption in the background can be made.
- Configuration data is stored with the data (e.g., disks know what RAID set they were a part of).
- The system can make use of hot spares, shareable between pools, with automatic RAID rebuild upon disk failure detection
- ZFS is implemented in two major commands (with lots of subcommands).
- Very, very large data structures (up to 128 bits) are allowed, with no arbitrary limits (files per directory, file systems, file size, disk per pool, snapshots, clones, and so on).
- ZFS is open source and free.
- It has been ported to FreeBSD, FUSE, and Mac OS X Leopard (read-only).

There are many articles about how to use ZFS and take advantage of these features, which, again, I won't repeat here [1].

ZFS Status

File system revolutions, as opposed to political revolutions, happen much more slowly and tend to be bloodless (although losing files can be very painful). A file system gradually gains trust as direct and shared experiences gradually build into an "it works" or "it loses files" general consensus. At this point in the life of ZFS it has passed that test for many people. The testing performed during its development and continuing every day is rather awe-inspiring, as described in Bill Moore's blog [2]. Reading through the posts at the ZFS forum [3] suggests that ZFS is being used a lot and at many sites, mostly very successfully. There is quite a lot of discussion of current and future features, as well as a few "something bad happened" discussions. Those posts, while revealing occasional problems, show in summary that ZFS is rock-solid, especially for such a new, innovative, core piece of software.

The next step in adopting new technology is support by other software products, such as backup/restore tools, clustering, and even general-purpose applications such as databases. Other vendors' products might work fine, but without a stamp of approval, commercial sites are very unlikely to use the new file system and risk being off of the support matrix. At first, of course there was zero non-Sun support for ZFS, but that situation has improved greatly. All major backup products support ZFS, and it is also now supported by Veritas and Sun cluster. Most applications are independent of the underlying file system, but those that do care, such as Oracle, are generally supporting ZFS.

Before a new technology can be put into top-priority environments (such as production OLTP database servers), it must perform as well as or better than the technology it is replacing. Performance tuning is usually a never-ending effort (or at least not ending until the product life ends). ZFS is no exception, and it is exceptionally young compared to the other production file systems such as UFS and Veritas Storage Foundation (the VXVM volume manager and VXFS file system). The only performance question more controversial than "Which is faster?" is "How do you prove which is faster?" The debate in general is continuous and unsolvable. There are certainly claims that ZFS is very fast, and faster than other file systems for certain operations. There are also counter claims that ZFS is slower at other operations. The StorageMojo blog has been following the debate and is a good site to watch. One posting [4] is especially interesting, showing ZFS compared with hardware RAID.

In my opinion, ZFS is a fundamentally fast volume manager/file system. It gets many aspects of storage very right. However, it cannot in software make up for one feature of hardware RAID: NVRAM cache. Nonvolatile cache allows writes to memory to take the temporary place of writes to disk. Because memory is much faster than disk, NVRAM is a great performance win. So, for example, using a Sun server containing local disk as a NAS device will have worse random write performance than a good NAS appliance that contains NVRAM. One solution to this performance delta is to use hardware RAID arrays that include NVRAM to provide individual LUNs to a system, and then use ZFS to create RAID sets and manage those LUNs as if they were individual disks. The NVRAM provides a performance boost, while all of the ZFS features remain available. Of course, in cases where random write performance is not critical (say, media servers and backup disk pools) NVRAM is not needed and ZFS is fine using local disks.

Aside from these performance challenges, ZFS is doing well at many sites. It is mostly being used in development, testing, and utility environments but is making its way into production. As more improvements are made to the feature set and more field experience drives acceptance, ZFS use should greatly increase.

The Future Feature List and the Future of ZFS

In spite of the massive list of ZFS features, there are still features that are desirable but not yet included in ZFS.

Probably the most important and useful would be the use of ZFS as the root file system, which would enable all of the above features for system administration. Imagine creating an instant snapshot of "/" and installing a patch in "/" and rolling the system back to that snapshot if the patch did not have the desired effect. Or imagine creating a snapshot every minute of the day to allow easy detection of changed files and restoration to the file's previous state. Once ZFS can be used as a root file system, zones will also be able to use ZFS for their root file systems. (Actually they already can have a ZFS root, but such a system cannot be upgraded to the next release of Solaris, as the upgrade code does not understand ZFS.) Fortunately, bootable ZFS has been added to OpenSolaris and should make its way into the commercial Solaris release in the future. It can be used currently via the various noncommercial Solaris distributions [5].

Native CIFS support is in OpenSolaris as well, so expect CIFS exporting as a future feature—no Samba (or other dancing) required.

Encryption is complicated to implement for a file system, mostly because of the key management. There is currently a ZFS encryption project underway for OpenSolaris [6], and alpha test code has already been released.

Removing disks (aside from hot spares) from a pool is an obvious need. Also missing is the ability to expand the size of a pool by adding individual disks. Currently, a set of disks can be added, for example as a RAIDZ set concatenated to a RAIDZ pool, and ZFS will cleverly stripe data across the two RAIDZ sets to maximize performance. However, adding a single disk to a ZFS pool simply has the disk concatenated to that pool, leaving for example a RAIDZ-plus-a-concatenated-disk pool rather than the much more desirable RAIDZ-expanded-to-include-the-new-disk pool.

The current quota system is a per-filesystem rather than a per-user one, which has pros and cons. There do not seem to be any plans to implement per-user quotas as well.

Scrubbing is currently done as a low-priority I/O task, but even lower-overhead user-definable scrubbing rates (in which a pool is gradually scrubbed over a period of time) are already planned for Solaris.

The ZFS intent log (ZIL), the place where ZFS stores changes that are to be applied to a ZFS pool, currently resides within the disks of that pool. OpenSolaris already includes the ability to put that log somewhere else, helping to improve write and especially random write performance. A natural next step would be to use a device dedicated to the ZIL. This could be an NVRAM device (and at least one company makes a PCI-based NVRAM card for Solaris) or a flash-based device that has been optimized for writes.

Another performance improvement could come from Brendan Gregg (of DTraceToolkit fame). He has added support in OpenSolaris for a level-2 adaptive replacement cache (L2ARC). This allows buffers to be evicted out of DRAM into a storage medium that fits between DRAM and the disk in terms of capacity and performance. The L2ARC and flash-based solid state drives (SSDs) seem to be a natural fit, and this is certainly an area to watch over the next 12 to 18 months.

ZFS integration with Mac OS X has been underway for quite a while, and read/write ZFS is available for testing [7].

For performance, many database sites use direct I/O, which bypasses the buffer cache and file locking, essentially telling the operating system and file system to get out of the way of database I/O. This feature does not exist within ZFS, and database performance on ZFS is currently a work in progress. For the latest information on ZFS performance see the ZFS Best Practices Guide [8].

Certainly the future is wide open for innovation around and integration of ZFS. As one example, have a look at the Service Management Facility (SMF) services being written by Tim Foster to automate snapshots and backups of ZFS file systems [9].

Next Time

Hopefully this ZFS status check has cleared up some questions and given guidance as to whether ZFS is now or will be in the future the right file system for your systems. The revolution seems to be well on its way and the Bastille is starting to fall. ZFS rising in its place seems inevitable and desirable.

In the next PATS column I'll discuss something that is basic, important, but frequently overlooked or done ad hoc: system problem analysis. What steps are the right ones to analyze a system that is having a problem, be it reliability or performance? My hard-learned cookbook may be a useful addition to your own techniques.

REFERENCES

[1] Recommended articles about ZFS: http://opensolaris.org/os/community/zfs/intro/; http://www.samag.com/documents/s=9950/sam0602j/0602j.htm; http://www.samag.com/documents/s=9979/sam0603h/0603h.htm; http://www.sun.com/bigadmin/features/articles/zfs_overview.jsp; http://www.sun.com/bigadmin/features/articles/zfs_part1.scalable.jsp.

[2] Bill Moore's ZFS blog: http://blogs.sun.com/bill/category/ZFS.

[3] ZFS Forum: http://www.opensolaris.org/jive/forum.jspa?forumID=80.

[4] StorageMojo blog entry comparing hardware RAID to ZFS performance: http://storagemojo.com/?p=441.

[5] OpenSolaris downloads: http://opensolaris.org/os/downloads/.

[6] Alpha ZFS encryption support: http://www.opensolaris.org/os/project/zfs-crypto/.

[7] ZFS for Mac OS X: http://trac.macosforge.org/projects/zfs/wiki/.

[8] Latest on ZFS performance: http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide.

[9] Automating snapshots: http://blogs.sun.com/timf/entry/zfs_automatic_for_the_people.