A Conversation with Jeff Bonwick and Bill Moore

Photography by Steve Skoll

The future OF FILE SYSTEMS

This month *ACM Queue* speaks with two Sun engineers who are bringing file systems into the 21st century. Jeff Bonwick, CTO for storage at Sun, led development of the ZFS file system, which is now part of Solaris. Bonwick and his co-lead, Sun Distinguished Engineer Bill Moore, developed ZFS to address many of the problems they saw with current file systems, such as data integrity, scalability, and administration. In our discussion this month, Bonwick and Moore elaborate on these points and what makes ZFS such a big leap forward.

Also in the conversation is Pawel Jakub Dawidek, a FreeBSD developer who successfully ported ZFS to FreeBSD. Ports to other operating systems, such as Mac OS X, Linux, and NetBSD are already under way, and his experience could pave the way for even wider adoption of ZFS.

Leading the discussion is David Brown, who works in Sun's Solaris engineering group and is a valued member of *Queue*'s editorial advisory board.

DAVID BROWN To start things off, can you discuss what your design goals were in creating ZFS?

BILL MOORE We had several design goals, which we'll break down by category. The first one that we focused on quite heavily is data integrity. If you look at the trend of storage devices over the past decade, you'll see that while disk capacities have been doubling every 12 to 18 months, one thing that's remaining relatively constant is the bit-error rate on the disk drives, which is about one uncorrectable error every 10 to 20 terabytes. The other interesting thing to note is that at least in a server environment, the number of disk drives per deployment is increasing, so the amount of data people have is actually growing at a super-exponential rate. That means with the bit-error rate being relatively constant,

you have essentially an ever-decreasing amount of time until you notice some form of uncorrectable data error. That's not really cool because before, say, about 20 terabytes or so, you would see either a silent or a noisy data error.

JEFF BONWICK In retrospect, it isn't surprising either because the error rates we're observing are in fact in line with the error rates the drive manufacturers advertise. So it's not like the drives are performing out of spec or that people have got a bad batch of hardware. This is just the nature of the beast at this point in time.

BM So, one of the design principles we set for ZFS was: never, ever trust the underlying hardware. As soon as an application generates data, we generate a checksum for the data while we're still in the same fault domain where the application generated the data, running on the same CPU and the same memory subsystem. Then we store the data and the checksum separately on disk so that a single failure cannot take them both out.

When we read the data back, we validate it against that checksum and see if it's indeed what we think we wrote out before. If it's not, we employ all sorts of recovery mechanisms. Because of that, we can, on very cheap hardware, provide more reliable storage than you could get with the most reliable external storage. It doesn't matter how perfect your storage is, if the data gets corrupted in flight—and we've actually seen many customer cases where this happens—then nothing you can do can recover from that. With ZFS, on the other hand, we can actually authenticate that we got the right answer back and, if not, enact a bunch of recovery scenarios. That's data integrity.

Another design goal we had was to simplify storage management. When you're thinking about petabytes of data and hundreds, maybe even thousands of disk drives, you're talking about something that no human would ever willingly take care of.

By comparison, when you have a server and you want to upgrade its memory, the process is pretty straightforward. You power down the server, plug in some DIMMs, power it back on, and you're done. You don't run dimmconfig, you don't edit `/etc/dimmtab`, and you don't create virtual DIMMs that you mount on applications. The memory is simply a pooled resource that's managed by the operating system on behalf of the application. If Firefox wants another megabyte of memory, it asks for it, and if it's available, it gets it. When it's done, it frees it, and back it goes into the pool for other applications to use. It's a very simple, very natural way of thinking about storage.

With ZFS, we asked this question: why can't your on-disk storage be the same way? That's exactly what we do in ZFS. We have a pooled storage model. The disks are like DIMMs, and the file systems are like applications. You add devices into the storage pool, and now the file system is no longer tied to the concept of a physical disk. It grabs data from the pool as it needs to store your files, and as you remove or delete your files, it releases that storage back to the pool for other file systems to use. Again, it's a very natural, very simple way to administer large quantities of data.

Finally, in designing ZFS, we also wanted to consider performance. A lot of people, until they get bitten by data corruption that causes them severe pain, are of the mindset, "I don't care, I just want it to go fast. I've never lost any data. My laptop is just fine after all these years." So unless you make it fast, people will fundamentally be uninterested—except for those who have experienced data corruption firsthand.

JB All of whom become rabid converts, as it turns out. There's nothing like having data corruption that is detected and healed for you to change your perspective on the importance of that problem and the value of having it solved.

DB Can you identify other things that have changed over the years, and that are going to be disruptive, that we haven't yet realized?

JB One thing that has changed, as Bill already mentioned, is that the error rates have remained constant, yet the amount of data and the I/O bandwidths have gone up tremendously. Back when we added large file support to Solaris 2.6, creating a one-terabyte file was a big deal. It took a week and an awful lot of disks to create this file.

Now for comparison, take a look at, say, Greenplum's database software, which is based on Solaris and ZFS. Greenplum has created a data-warehousing appliance consisting of a rack of 10 Thumpers (SunFire x4500s). They can scan data at a rate of one terabyte per minute. That's a whole different deal. Now if you're getting an uncorrectable error occurring once every 10 to 20 terabytes, that's once every 10 to 20 minutes—which is pretty bad, actually.

DB You're saying that it's not just scale, it's also the rate at which we're accessing that stuff, so the failure rates are really very visible now.

BM Yes, that's right, and as the incident rate goes up, people are going to become more and more aware that this is a problem, after all. When it happens only once a year or once every several months, it's easy to chalk that up to any number of problems—"Ah, stupid Windows, it already needs to be reinstalled every six months." Now people are thinking, "Well, maybe there's a reason. Maybe Windows isn't as bad as everyone says it is."

Or pick your favorite operating system, or your favorite laptop. Bad things happen, and while I'm not saying software isn't buggy, a lot of this stuff that we tend to chalk up to buggy software, if you dig into it, often has an underlying cause.

DB I'm recalling Jim Gray a little bit and maybe channeling some of those ideas that I've heard him discuss over the years, such as the teraserver that he put together. Originally he had been shipping disks around between people, and then he discovered it was an inconvenient way of connecting, and ultimately he shipped around whole PCs and was using Ethernet and NFS (network file system) as the logical, easy point of connectivity. Did you think at all about what's an appropriate representational level? I know ZFS is a file system, but have you given any thought to those kinds of issues?

BM Yes, as a matter of fact, I was going to mention earlier that ZFS is composed of several layers, architecturally, but the core of the whole thing is a transactional object store. The bulk of ZFS, the bulk of the code, is providing a transactional store of objects.

You can have up to 2^{64} objects, each 2^{64} bytes in size, and you can perform arbitrary atomic transactions on those objects. Moreover, a storage pool can have up to 2^{64} sets of these objects, each of which is a logically independent file system. Given this foundation, a lot of the heavy lifting of writing a Posix file system is already done for you. You still have to interface with the operating system and the virtual memory subsystem, but all the stuff you don't have to worry about is kind of nice: you don't have to worry about on-disk format; you don't have to worry about consistently modify-

ing things; and you don't have to worry about FSCK (file system check).

JB We can thank the skeptics in the early days for that. In the very beginning, I said I wanted to make this file system a FSCK-less file system, though it seemed like an inaccessible goal, looking at the trends and how much storage people have. Pretty much everybody said it's not possible, and that's just not a good thing to say to me. My response was, "OK, by God, this is going to be a FSCK-less file system." We shipped in '05, and here we are in '07, and we haven't needed to FSCK anything yet.

BM Having your data offline, while you try to find the needle in the haystack, is not really where most people want to live.

DB Well, especially when the scale of the storage is many terabytes or petabytes versus circa 1972 when the file system on Unix was five megabytes.

You talked about essentially building a layer of abstraction on the physical storage so that you have a more reliable transactional storage model down low. Historically, we've had storage welded into the operating system in terms of the specific file systems that a given operating system offers. This makes it not as amenable to moving that storage around, if you want, whether it's hot-pluggability or mobility. Did you look at those things as well?

BM If you were to go back five years and ask people, "Would you mind having all your data over the network somewhere?" they would say, "Ha, ha, ha, that's funny. Way too low performance. I'll never get my data fast enough." Now you look around and gigabit is ubiquitous and 10-gigabit is starting to emerge. Right now with gigabit you get more than 100 megabytes a second over a $5 interface that's on every device you can buy—you can hardly buy anything slower anymore.

The interesting thing to note about that is the network is now faster than any local disk you can buy. One of the things we see in the HPC (high-performance computing) industry is people moving away from local storage and using high-speed, low-latency interconnects—which Ethernet will eventually catch up with—and using that as the primary means of accessing their data.

My prediction is that in the next five to 10 years, for the most part, your data is going to be somewhere else. You'll still have local storage when you're not connected to any network, but in an office environment or a corporate environment or a data center, I just don't think you're going to have a whole lot of local storage.

JB Yes, because if you've got a relatively small client compared with a relatively large server providing the data, you have to consider the cost of disk access on your local system versus the latency of going over the wire to get something that in all likelihood is in memory on the server side—and that actually is faster now that we're up to 10-gigabit Ethernet.

BM I think this will continue to be truer as we go forward, but only in recent history have the trend lines crossed to such a point that having your storage remote is acceptable and perhaps can even exceed the performance of a local storage solution.

DB I wonder if there are some emerging issues to talk about in this regard. There is ongoing discourse about NAS (network-attached storage), the pNFS (parallel NFS) type of approach versus the SAN (storage area network).

JB Whether you talk about storage as SAN or NAS, you're describing it fundamentally as a relatively dumb thing that you plug in, and then some smart thing comes along to access it. The big change that we're going to see is more intelligence in the storage itself. Rather than simply saying, "Give me the following region of bytes out of some file," you're going to come at it with a question, and then it's going to do some work for you and come up with an answer. If you think about it, this is exactly how you interact with Google. You don't go to Google and have it feed two petabytes of data over your DSL line, and then run `grep` locally. That would be kind of bad.

Instead, you send a small question over the wire to Google, which runs the compute right near the data, where it's all low latency, and then sends you back a very small answer. We see storage in general moving in that direction because you have more and more of it and you want to be able to do unstructured queries.

You need to have that kind of technology. It's too expensive to do brute-force searches without any kind of support from underlying storage systems, just because of the change in capacity. In fact, there was a prediction made about 10 years ago that by roughly today, all data would be in databases, and file systems were going away. I think it's really all about human behavior. If you want to sell to people who are organized in the way they store things, the only people you'll be selling to will be in the database market. I'd rather sell to people who are disorganized in the way they store things because there are more of them and they have a lot more stuff. People are pack rats and they don't plan.

DB I would like to ask Pawel about some of the problems and challenges he faced while porting ZFS to FreeBSD. What made it difficult? What were the things perhaps you expected to be difficult that were less so?

PAWEL JAKUB DAWIDEK At first I just wanted to see how much work it would take to port ZFS to FreeBSD. I started by making it compile on FreeBSD, and once I did that, I was quite sure it would take at least six months to have the first prototype working. The funny thing was that after another week or so, ZFS was running on my test machine. It was truly surprising that the code was so portable; it was self-contained and I had initial read-write support after 10 days of work. The parts that need to be ported are actually only ZVOL, ZPL, and VDEV; all the rest is mostly self-contained and doesn't need a lot of work. Of course, after those 10 days there was still a lot of work to do and I encountered some problems. I found that VFS was the hardest part in ZFS's ZPL layer, but I managed to make it work.

There were some parts I needed to port from OpenSolaris itself, such as the GFS (generic file system) framework, which in ZFS is responsible for handling the `.zfs/` directory and snapshot mountpoint. There were a few really tough problems, but the project was so exciting that it somehow kept me working. I had this feeling that what I'm doing is really cool and that I should continue. So, basically, it went much, much better and faster than I ever expected.

The FreeBSD port of ZFS is almost feature-complete. We even have support for FreeBSD's jails working the same way as ZFS support for Solaris zones. We, of course, support snapshots, ZVOLs, NFS exports, etc.

The biggest problem, which was recently fixed, was the very high ZFS memory consumption. Users were often running out of KVA (kernel virtual address space) on 32-bit systems. The memory consumption is still high, but the problem we had was related to a vnode leak. FreeBSD 7.0 will be the first release that contains ZFS.

DB Is that going to be 64 bits?

PJD We want to release ZFS for every architecture we support, but we don't recommend ZFS for 32-bit architectures.

JB We debated that in the early days because obviously it has been a bug for a long time now. Back when virtual memory was first created, the idea was that you had far more virtual addressing than you had physical memory. As physical memory capacity crossed the four-gigabyte (32-bit) threshold, however, the contents of the bag got bigger than the bag. That's an awkward situation to be in, to have an eight-gigabyte system and only two or four gigabytes that are actually addressable at any one time.

I had actually hoped that this problem would go away a little sooner than it has—and Bill is laughing because he gave me a hard time about this—but as we were developing ZFS, I basically said, "Look, by the time we actually ship this thing, this problem will have taken care of itself because everything will be 64-bit." I wasn't entirely off base.

BM You were right within a decade.

JB Maybe even within half a decade, but it's one of those things where you look at it and say, "Gee, the code is so much simpler if I don't have to deal with that whole set of problems," keeping only a subset of the data mapped.

DB I think the assumption is pretty valid in terms of what has happened with hardware commoditization and cost. Hardware has become free and you can have large tracts of physical memory and disk and everything, but, unfortunately, people still have these old applications, relying on these old operating systems.

JB Sometimes it's new ones, too. It's not your next desktop; it's your next laptop, or PDA, or cellphone, and all of a sudden, you have this new class of device and you're back to square one. You have more limited addressing, a smaller amount of memory, and less compute power, and you want to be able to fit in those devices.

DB The "ZFS for my cellphone" kind of problem.

BM Actually, we've released ZFS not as a be-all, end-all product; we've tried to design and maintain it in such a way that it's really not just an endpoint but rather a starting point. It's a framework, with the transactional object store and the pooled storage, on top of which you can build far more interesting things.

DB What kinds of interesting things do you have in mind?

BM There are a lot of other things we've been planning to do with it, and haven't quite found the time to do. One thing we have found the time for is basically to take one of these objects and export it as a raw block device, as a virtual disk, if you will.

With this virtual disk you can do anything you would do with the real disk. You can put UFS (Unix file system) on it, or you can export it as an iSCSI device. You can run your favorite database on it or you can cat(1) it, if that's what you're into.

Imagine if you took, say, a database such as Postgres or MySQL and put the SQL engine on top of ZFS's transactional object store. Then you would have something that has all the file system and volume manager stuff integrated into it, so there's no waste doing the same job five different times on your way down from your database app before it gets to disk. It's all handled once, yet you still have the same programming interface and the SQL search engine that you know and love and are used to.

JB If you have a database sitting on top of a transactional file system, the database is sitting up there being very careful about ordering its writes to the log and being clear about saying when it wants to know this stuff is flushed out. Then beneath it, you've got this transactional file system creating transaction groups, putting in a bunch of changes, and then committing those out to disk atomically.

Why not have a blending of the layers where basically the whole back end to the database—the part that isn't parsing the SQL—participates directly in the transaction groups that ZFS provides, so that consistency is no longer a problem the database has to solve? It can be handled entirely by the storage software below.

BM That is especially nice when you have a transactional object store and you have the capability of rolling back. If bad things, such as a power fail, should happen, you don't ever have to do an offline FSCK, as you would in a traditional file system.

DB What are the provocative problems in storage that are still outstanding, and does ZFS help? What's next? What's still left? What are the things that you see down the pike that might be the big issues that we'll be dealing with?

JB There are not just issues, but opportunities, too. I'll

give you an example. We were looking at the spec sheets for one of the newest Seagate drives recently, and they had an awful lot of error-correction support in there to deal with the fact that the media is not perfect.
BM They're pushing the limits of the physics on these devices so hard that there's a statistical error rate.
JB Right, so we looked at the data rates coming out of the drive. The delivered bandwidth from the outer tracks was about 80 megabytes per second, but the raw data rate—the rate that is actually coming off the platter—was closer to 100. This tells you that some 20 percent of the bits on that disk are actually error corrections.
BM Error correcting, tracking, bad sector remapping.
JB Exactly, so one of the questions you ask yourself is, "Well, if I'm going to start moving my data-integrity stuff up into the file system anyway—because I can actually get end-to-end data integrity that way, which is always stronger—then why not get some additional performance out of the disk drive? Why not give me an option with this disk drive?" I'll remap all the bad sectors, because we don't even have to remap them. It suffices to allocate it elsewhere and basically deliberately leak the block that is defective. It wouldn't take a whole lot of file-system code to do that.

Then you can say, "Put the drive in this mode," and you've got a drive with 20 percent more capacity and 20 percent higher bandwidth because you're running ZFS on top of it. That would be pretty cool.
DB That's a really exciting idea. Have you had those discussions with the drive vendors about whether they would offer that mode?
BM Not quite, because they're most interested in moving up the margin chain, if you will, and providing more unreliable devices that they sell at a lower cost; it isn't really something they care to entertain all that thoroughly. I think we will see this again with flash, however, because flash has a much more interesting failure mode. With flash, as you wear out a given region of the device, you wind up getting statistically higher and higher bit-error rates in that region. In ZFS we have a notion of built-in compression. Say the application gives us 128 kilobytes of data. We compress it down to 37 kilobytes. Then we write a single 37-kilobyte block of data to disk, and to the application it looks like it's reading and writing 128 kilobytes, while we're compressing it to whatever size that works.

Conversely, you could go another way, which is potentially to expand the data with some sort of error-correcting code such that when you read the data back—or before you write the data—you would say, "All right, with the media I'm going to write to, I expect this kind of error rate, and I want this kind of recoverability. Let me expand it using my own error-correcting code, write it out, and then when I read it back, undo that and correct any errors you wind up with." The idea being that now that it's under ZFS control, it could be software controlled. As you wear out a region of a flash device, you say, "All right, that region is getting worn out. Maybe I'll start expanding the data in that region of the device a little more," or, "This area is not worn out, so maybe I won't expand those as much and get the capacity."

There are some very interesting ideas that can come from putting everything up at the file-system level.
DB I think it's also a very provocative point. When we thought about nonvolatile storage through most of our history in the field, it's been disk drives. Then all of a sudden, we've got flash, which has totally different properties than the disk drive. When you started out with ZFS, were you thinking at all about the application of hybrid storage?
BM Not explicitly. We were thinking more about keeping everything as compartmentalized as possible, so that as time moves on and things we couldn't have anticipated come up, the amount of code we have to change to implement some new radical technology should be minimal.
JB We did have an overall design principle of not designing the file system assuming that you're writing fundamentally to cylindrical storage, that other things will come into existence. At the time it wasn't clear that it was going to be flash; that wasn't even on the radar screen. We were thinking it would be more like MRAM or Ovonic unified memory or holographic memory. It's hard to say which one of these things is ultimately going to take over, but I don't think that on the deck of the *Enterprise* we'll have rotating rust.
BM Every year inside Sun, the Solaris kernel engineers have a session at the beginning of the year called predicteria where we make one-, three-, and six-year predictions for ourselves and the industry. For the past 10 years, Jeff has been betting that five years out we'll have nonvolatile memory and there will be no more DRAM in computers. So far he has been losing that bet every time it comes up, but one of these years he'll probably be right. I just don't know which year that'll be. Q

LOVE IT, HATE IT? LET US KNOW
feedback@acmqueue.com or www.acmqueue.com/forums