Linux FAST Summit '16 Summary

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

Following FAST '16, 26 people met for the 2016 USENIX Research in Linux File and Storage Technologies Summit (Linux FAST '16) to discuss file systems, storage, and the Linux kernel. I've learned that the reason behind these discussions is to help people get their changes into the kernel, or at least to help people understand what the process is like. Ted Ts'o (Google) has pointed out at past Linux FAST workshops that there are already over 120 file systems in Linux, and getting new ones added is not going to be easy.

We began the workshop by going around the room and sharing our names, affiliations, and our reason for attending this year. Often, people attend because they just presented a file system, or code that extends an existing one, at the FAST workshop and want to understand how to get their code added to the Linux kernel. Others attend because they are working on new devices or need support for doing things in file systems that haven't been done before. We had all these interests this year. But let's start with the perennial issue: getting your code accepted into the Linux kernel.

Advice on how to do this was sprinkled throughout the workshop, and rather than mimic that distribution, I thought I would try to condense it into something more coherent.

Also, before going any further, I want to mention there was also a BSD File System workshop on the same day as FAST '16 tutorials. I couldn't attend because of the tutorials, but Kirk McKusick shared a link to the agenda, which also includes some notes [1] about what was covered. I did drop by and counted 31 people in attendance, seated lecture style. When I visited later, the meeting had broken up into working groups.

Going Upstream

Getting your code added to the Linux kernel, maintained by Linus, means having your code accepted *upstream*. All distros start with the upstream code and then add what distinguishes their distro from others. Also, getting new code into the kernel is definitely trying to swim upstream—it's not easy.

Ted Ts'o, the first US contributor to the Linux kernel, and someone who has attended every Linux FAST I've been to, always starts by suggesting you join, not the kernel mailing list, but a storage-specific mailing list appropriate to the type of code you've developed. There are lists (http://vger.kernel.org/vger-lists.html) for the block devices, SCSI, block caches, btrfs, and even Ceph, and all of them much more focused than the generic kernel mailing list. There is also a device mapper mailing list on a separate server (dm-devel@redhat.com).

This time, Ted Ts'o also suggested posting early code to the appropriate list. Ted's rationale for doing this is to quickly learn whether your approach is workable or not and whether there are problems before you have spent a lot of time working on a solution.

There was also some discussion about which kernel version to use. Erez Zadok (Stony Brook University) pointed out that it sometimes takes years for a graduate student to complete a

project, but Ric Wheeler (Red Hat) explained that there is one stable kernel per year (approximately), and working with a stable kernel is best.

Another big issue with getting a new file system accepted upstream is having someone to support the code in the future. You can't just toss your code over the wall—someone must continue to fix bugs and adapt the code as other kernel code changes. That someone should have industry support: in other words, work for a company that wants the code maintained. While you might wonder how common that is, Linux FAST always has a number of people who work for companies doing this. Linux FAST '16 had kernel developers working for Facebook, Google, Intel, HP Enterprise, Huawei, Micron, Red Hat, and Seagate, and four of those companies had multiple coders present. Industry-supported kernel hackers outnumbered students and faculty at this year's Linux FAST.

Ric Wheeler noted that there is another Linux file system and storage workshop that is more appropriate for industry users, rather than academics and developers, called Vault [2]. Unlike Linux FAST, where the goal is discussion, Vault has keynotes and talks.

I was sitting next to Jian Xu, a PhD student from the University of California, San Diego, who had presented a paper about a new file system for NVRAM (NOVA [3]). While NOVA is much faster than F2FS, the currently used flash file system for Linux-based devices, Ted pointed out that ext2 is actually faster on flash than F2FS for key-value stores but that F2FS does better in SQLite benchmarks. I suggested that Xu try to find industry backing—some company with a device that would benefit from using NOVA over F2FS. Ted had also suggested finding some "economic actor" that would support his project.

Shingled Magnetic Recording

SMR was a big topic in the past and continued to be this year. Ted Ts'o announced that he had been working with someone at CMU on host-aware SMR drive management, leading to some discussions throughout the afternoon. SMR disk drives get added capacity by eliminating the space between most tracks on a platter. Adrian Palmer (Seagate) pointed out that a single zone, or writeable area, on a typical SMR drive is 256 MB, whereas in conventional drives, the write unit is a sector, or 4 kB. In ext4, the largest block size is 128 MB, half the zone size in SMR.

Current SMR drives are device managed, which means that the SMR drives you can buy today hide the fact that they are SMR: they behave like drives with small sectors by handling all writes as sequential writes to a zone. That implies that SMR drives perform block mapping behind the scenes, and must also perform garbage collection, dealing with the holes created in zones when files are deleted or truncated. These activities are hidden from users (and the operating system), except when they cause unexpected latency. I overheard someone say that, when using an SMR drive, they could finish a benchmark in 30 seconds or 12 minutes, depending on the internal state of the drive. Revealing the internal state of drives was discussed to some extent during Linux FAST and was the topic of Eric Brewer's (Google) keynote at FAST '16 [4].

Ted Ts'o and people at Carnegie Mellon University have been working with Seagate on host-aware SMR drives, which are still self-managed but accept input from the host operating system to optimize performance. Peter Desnoyers (Northeastern University) and an associate have been working with Western Digital on the same problem but are using WD drives. Shaun Tancheff is a software engineer, consulting for Seagate, working on the problem from the manufacturer's side. Shaun asked for flags that can be included with requests to host-aware and host-managed SMR drives. Jens Axboe (Facebook) said that it is possible to add modifier flags to SCSI requests. Andreas Dilger (Intel) mentioned that he has been using some bits in the stream ID, but Jens Axboe said that he was not opposed to adding flags to the file struct for doing this.

There is another type of SMR drive, host-managed. The people at Linux FAST who I've suggested were working on host-aware drives may actually have been working on host-managed drives. They probably can't confirm that, however, because of the conditions they work under (NDAs). Host-managed drives take control of the SMR drive, the exact opposite of a device-managed drive. Host-managed drives must always write at the write-pointer, the furthest point in an SMR zone that has recently been written. Having the file system or deeper kernel layers manage an SMR drive means more than having to be aware of the 256 MB zones: the OS must also handle block mapping, copying blocks to better locations, as well as handling garbage collection. In some ways, SMR requires software very like the Flash Translation Layer (FTL) found in SSDs.

Adrian Palmer brought up the issue of out-of-order writes being a problem when working with SMR drives. Drive manufacturers have been making the assumption that the write-queue will be ordered (logical block addresses in either monotonically increasing or decreasing order). In practice, they had seen non-ordered writes. John Grove (Micron) also shared interest in having a mode where block I/O ordering is guaranteed through the block stack. Jens Axboe took the concerns seriously and suggested that people propose solutions. Jens also pointed out that in a multi-queue environment, ordering would practically require that all order-dependent I/Os go through one queue.

Trim

The `trim` command was created as a method of telling SSDs that certain logical blocks were no longer in use—the file system had

deleted the files containing those blocks, or truncated a file, also releasing blocks. Trim means an SSD can change the mappings of blocks to unused, and in theory this could help SSD performance by helping to reduce garbage collection overhead.

Initially, the trim command could not be queued: any commands in a queue for a drive would have to complete before the trim command could be issued. Later versions of the standard (ATA 3.1) allowed trim commands to queue.

Ted Ts'o pointed out that there are a number of SSDs that have been blacklisted by kernel developers because of data corruption issues when the trim command was used in Linux. See [5] for a list of blacklisted drives.

Trim also affects file systems and drivers for SMR drives, as SMR drives also need to perform garbage collection, dealing with freed space, and trim offers a method of communicating to a drive which logical blocks have designated as unused.

Tim Feldman (Seagate) opened the discussion of trim by mentioning that Seagate works with T10 and T13 standards bodies, which affect both stream IDs and trim for both flash and disk drives. Tim also suggested that some internal states of drives, which are actually intelligent devices, could be communicated back to the kernel: for example, the failure of a single head or other health characteristics. Ric Wheeler said that it would be useful to know when a drive has a non-volatile (NV) cache enabled, and Tim answered that this is well-defined in standards, but in practice, results may not be correct. Andreas Dilger said the standards consider this optional, and Tim agreed that NV cache state should be exposed.

Trim for a drive-managed SMR drive could change the write pointers, but Shaun pointed out that the problem is how to share this information with block device drivers. Hannes Reinecke (SUSE) had posted some code for supporting host-managed SMR drives [6], and his post was mentioned in the context of trim for SMR.

Jens Axboe mentioned that he is working on patches that support sharing information about timing/delays, write-stream IDs for flash devices, to reduce write amplification and some latency improvement. A lot of this work has been on the standards side so they can support it in the kernel. At this point, they can push a million-plus I/Os through the kernel.

Non-Volatile Memory

The Intel Micron 3D XPoint NVRAM was on lots of people's minds. About 1000 times faster than flash, and very likely arranged in cache-line-sized blocks (64–128 bytes) instead of kilobyte- or megabyte-addressable blocks, 3D XPoint will first appear on the memory or PCIe busses. And unlike flash, which could conveniently be treated as if it were a disk device, 3D XPoint needs to be treated more like a persistent form of DRAM. While not as fast as DRAM, NVRAM-like 3D XPoint will be much denser than DRAM, allegedly allowing a server to have as much as 6 TB of fast persistence storage. For HPC, this means that burst buffers (see Bent et al.'s HPC storage article in this issue) would go away, to be replaced with CPU-board storage for checkpointing.

Suparna Bhattacharya (HP Enterprise) asked whether 3D XPoint would appear as a storage device or be more like memory. Dan Williams (Intel) replied that today it appears that 3D XPoint will first appear as memory. When you read from 3D XPoint, lines get loaded into the appropriate CPU cache, and when you flush, lines should be flushed back. The current way of mapping a file into memory using mmap() will likely be extended to work with 3D XPoint and similar devices. Dan said that while some people want more control over cache behavior, he doesn't believe that they should be able to do this: the CPU is in the best position to make decisions about the cache. But fsyncing an mmapped file should result in the portions of the file in cache being copied to non-volatile storage, as happens with fsyncing data back to a disk. Dan says that the decisions on how to handle this have not completed, and perhaps fsyncing the device should force a cache flush.

Dan also introduced DAX/DMA into persistent memory, the biggest ticket item for persistent memory. DAX was developed for NVRAM, like 3D XPoint, but looks to have other applications as well. While mmap() memory maps files, DAX provides a pointer right into memory, and will be useful not just for NVRAM, but also in file systems like Jens Axboe's and ext4 (but not btrfs), where being able to overwrite a section of a file is useful. With DAX, you write, then commit, and once you commit the process blocks until the cache has been successfully flushed. DAX sounds like it will solve some of the problems people have with reworking mmap() to work with NVRAM.

BetrFS

Several people from Stony Brook University, including some of the authors of the Best Paper Award-winning "Optimizing Every Operation in a Write-Optimized File System" [8] were present. Rob Johnson (Stony Brook University) said that the primary reason for staying for Linux FAST was to learn more about getting their file system into the kernel. Rob said that the core of their optimizations (B-epsilon trees [9]) was part of a commercial product, and it was likely that someone from their crew would be hired to work on that product. That would mean that someone would be paid to maintain any changes to the kernel to support BetrFS.

Ceph

There were also several people present from Ceph, a distributed file storage product. While Ceph is a user-level overlay, currently used for block and object store, there appeared to be things that the Ceph folks would like to see in the kernel, such as a having

a key-value store there. Greg Farnum (Red Hat) seemed more interested in having access to unwritten file extents in user space. `fallocate()` won't expose unwritten blocks, because that's a security issue, but in the case of Ceph, being able to have more control over where Ceph writes its data and metadata would help them improve performance [10]. The key-value store is less interesting, as a transactional store would be more useful. The BetrFS crew also expressed some interest in transactional storage, leading to objections from Ted Ts'o.

Ted had two concerns: first, that an application would crash during a transaction, leaving the transaction orphaned, and second, that an application might be greedy and spool up so much data into one transaction that the transaction would dominate the log (and work that could currently be done). Rob Johnson said they would be happy to have limits on the log, and timeouts could handle the crashing during a transaction issue. Greg Farnum wrote that Ceph doesn't really *need* a POSIX file system but wants a transactional key-value store that runs in kernel space. Listening to this discussion, I thought such changes seem currently unlikely. But big changes have occurred, such as the discontinuation of ext3 in recent kernels and some distros now making btrfs the default file system.

References

[1] FreeBSD NewStorage Technologies Summit 2016: https://wiki.freebsd.org/201602StorageSummit/NewStorage Technologies#Agenda.

[2] Vault: http://www.linuxfoundation.org/news-media/announcements/2014/08/linux-foundation-launches-new-conference-vault-address-growing.

[3] Jian Xu and Steven Swanson, "NOVA: A Log-Structured File System for Hybrid Volatile/Non-Volatile Main Memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16),* 2016: https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu.

[4] Eric Brewer, "Spinning Disks and Their Cloudy Future," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16),* slides and audio: https://www.usenix.org/conference/fast16/technical-sessions/presentation/brewer.

[5] `Trim` Shortcomings: https://en.wikipedia.org/wiki/Trim_(computing)#SCSI.

[6] ZBC host-managed device support, SCSI mailing list: https://lwn.net/Articles/653187/.

[7] DAX: https://lwn.net/Articles/618064/.

[8] Jun Yuan, Yang Zhan, William Jannen, Prashant Pandey, Amogh Akshintala, Kanchan Chandnani, Pooja Deo, Zardosht Kasheff, Leif Walsh, Michael Bender, Martin Farach-Colton, Rob Johnson, Bradley C. Kuszmaul, and Donald E. Porter, "Optimizing Every Operation in a Write-Optimized File System, in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16),* 2016: https://www.usenix.org/conference/fast16/technical-sessions/presentation/yuan.

[9] Michael A. Bender, Martin Farach-Colton, William Jannen, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter, Jun Yuan, and Yang Zhan, "An Introduction to Bε-trees and Write-Optimization." *;login:*, vol. 40, no. 5, October 2015: https://www.usenix.org/publications/login/oct15/bender.

[10] See Ts'o's comment on `fallocate()` after Linux Fast: http://marc.info/?l=linux-api&m=145704481128395&w=2.