LSF '07: 2007 Linux Storage & Filesystem Workshop

San Jose, CA
February 12–13, 2007
Summarized by Brandon Philips (brandon@ifup.org)

Fifty members of the Linux storage and filesystem communities met in San Jose, California, to give status updates, present new ideas, and discuss issues during the two-day Linux Storage & Filesystem Workshop. The workshop was chaired by Ric Wheeler and sponsored by EMC, NetApp, Panasas, Seagate, and Oracle.

JOINT SESSION

Ric Wheeler opened the workshop by explaining the basic contract that storage systems make with the user to guarantee that the complete set of data will be stored, bytes are correct and in order, and raw capacity is utilized as completely as possible. It is so simple that it seems that there should be no open issues, right?

Today, these basic demands are met most of the time, but Ric posed a number of questions. How do we validate that no files have been lost? How do we verify that bytes are correctly stored? How can we utilize disks efficiently for small files? How do errors get communicated between the layers?

Through the course of the next two days some of these questions were discussed, others were raised, and a few ideas were proposed. Continue reading for the details.

■ *Ext4 Status Update*

Mingming Cao gave a status update on ext4, the recent fork of the ext3 file system. The primary goal of the fork was the move to 48-bit block numbers; this change allows the file system to support up to 1024 petabytes of storage. This feature was originally designed to be merged into ext3 but was seen as too disruptive [1]. The patch is also built on top of the patch set that replaces the indirect block map and with extents [2] in ext4. Support for greater than 32K directory entries will also be merged into ext4.

On top of these changes a number of ext3 options will be enabled by default in ext4; these include directory indexing to improve file access for large directories, resize inode, which reserves space in the block group descriptor for online growing, and 256-byte inodes. Users of ext3 can use these features today by using mkfs.ext3 -I 256 -O resize_inode dir_index /dev/device.

A number of RFCs are also being considered for inclusion into ext4. This includes a patch that will add nanosecond timestamps [3] and the creation of persistent file allocations [4], which will be similar to posix_fallocate but won't waste time writing zeros to the disk.

Currently, ext4 stores a limited number of extended attributes in-inode and has space for one additional block of extended attribute data, but this may not be enough to satisfy xattr-hungry applications. For example, Samba needs additional space to support Vista's heavy use of ACLs, and eCryptFS can store arbitrarily large keys in extended attributes. This led everyone to the conclusion that someone needs to collect data on how xattrs are being used, to help developers decide how to best implement xattrs. Until larger extended attributes are supported, application developers need to pay attention to the limits that exist on current file systems (e.g., one block on ext3 and 64K on XFS).

Online shrinking and growing was briefly discussed and it was suggested that online defragmentation, which is a planned feature, will be the first step toward online shrinking. A bigger issue, however, is storage management. Ted T'so suggested that the Linux filesystem community can learn from ZFS how to create easy-to-manage storage systems. Christoph Hellwig sees the disk management issue as being a user-space problem that can be solved with kernel hooks and sees ZFS as a layering violation. Either way, it is clear that disk management should be improved.

■ *The fsck Problem*

Zach Brown and Valerie Henson were slated to speak on the topic of filesystem repair, but there was a slight delay as Val's laptop was booting. To pass the time she introduced us to the latest fashion: laptop rhinestones. They would make a great discussion piece if you are waiting on a fsck and, if Val's fsck estimates for 2013 come true, having a strategy to pass the time will become very important.

With her system booted Val presented an estimate of 2013 fsck times. She first measured a fsck of her 37-GB home directory with 21 GB in use, which took 7.5 minutes and read 1.3 GB of filesystem data. Next, she used projections of disk technology from Seagate to estimate the time to fsck a 2013 home directory, which will be 16 times larger. Although 2013 disks will have a fivefold bandwidth increase, seek times will only improve by 20%, to 10 ms, leading to a fsck time of 80 minutes! The primary reason for long fscks is seek latency, since fsck spends most of its time seeking over the disk, discovering and fetching dynamic filesystem data such as directory entries, indirect blocks, and extents.

Reducing seeks and avoiding the seek latency punishment are key to reducing fsck times. Val suggested one solution: Keeping a bitmap on disk that tracks the blocks that contain filesystem metadata; this would allow for reading all data in a single arm sweep. This optimization, in the best case, would make a single sequential sweep over the disk and on the 2013 disk reading all filesystem data would only take around 134 seconds, which is a big improvement. A full explanation of the findings and possible solutions can be found in the paper Repair-Driven File System Design [5]. Also, Val announced that she is working full time on a file system called chunkfs [6] that will make speed and ease of repair a primary design goal.

Zach Brown presented a blktrace of e2fsck. The basic outcome of the trace is that the disk can stream data at 26 Mbps and fsck is achieving 12 Mbps. This situation could be improved to some degree without on-disk layout changes if the developers had a vectorized I/O call. Zach explained that in many cases you know the block locations that you need, but with the current API you can only read one at a time.

A vectorized read would take a number of buffers and a list of blocks to read as arguments. Then the application could submit all of the reads at once. Such a system call could save a significant amount of time, since the I/O scheduler can reorder requests to minimize seeks and merge requests that are nearby. Also, reads to blocks that are located on different disks could be parallelized. Although a vectorized read could speed up the fsck, eventually filesystem layout changes will be needed to make fsck really fast.

■ *Libata: Bringing the ATA Community Together*

Jeff Garzik gave an update on the progress of libata, the in-kernel library to support ATA hosts and devices. First, he presented the ATAPI/SATA features that libata now supports: PATA+C/H/S, NCQ, FUA, SCSI SAT, and CompactFlash. The growing support for parallel ATA (PATA) drives in libata will eventually deprecate the IDE driver, and Fe-

dora developers are helping to accelerate testing and adoption of the libata PATA code by disabling the IDE driver in Fedora 7 test 1.

Native Command Queuing (NCQ) is a new command protocol introduced in the SATA II extensions and now supported under libata. With NCQ the host can have multiple outstanding requests on the drive at once. The drive can reorder and reschedule these requests to improve disk performance. A useful feature of NCQ drives is the force unit access (FUA) bit, which will ensure that data in write commands with this bit set will be written to disk before returning success. This has the potential of enabling the kernel to have both synchronous and nonsynchronous commands in flight. There was a recent discussion [7] about both NCQ FUA and SATA FUA in libata.

Jeff briefly discussed libata's support for SCSI ATA translation (SAT). SAT lets an ATA device appear to be a SCSI device to the system. The motivation for this translation is the reuse of error handling and support for distro installers, which already know how to handle SCSI devices.

There are also a number of items slated as future work for libata. Many drivers need better suspend/resume support, and the driver API is due for a sane initialization model using an allocate/register/unallocate/free system and use of "Greg blessed" kobjects. Currently, libata is written under the SCSI layer and debate continues on how to restructure libata to minimize or eliminate its SCSI dependence. Error handling has been substantially improved by Tejun Heo and his changes are now in mainline. If you have had issues with SATA or libata error handling, try an updated kernel to see whether those issues have been resolved. Tejun and others continue to add features and to tune the libata stack.

■ *Communication Breakdown: I/O and File Systems*

During the morning a number of conversations sprang up about communication between I/O and file systems. In the case of errors, file systems should be getting information on nonretryable errors and passing that data up to user space. Particularly bad can be situations where retries are happening over and over when the I/O layer knows that an entire range of blocks is missing.

A "pipe" abstraction was discussed to communicate data on byte ranges that are currently in error, under performance strain (because of a RAID5 disk failure), or temporarily unplugged. If a file system was aware of ranges that are currently handling a recoverable error, have unrecoverable errors, or are temporarily slow, it might be able to handle the situations more gracefully.

File systems currently do not receive unplug events, and handling unplug situations can be tricky. For example, if a fibre channel disk is pulled for a moment and plugged back in, it may be down for only 30 seconds, but how should the file system handle the situation? Currently, ext3 remounts the entire file system as read only. XFS has a configurable timeout for fibre channel disks that must be reached before it sends an EIO error. And what should be done with USB drives that are unplugged? Should the file system save state and hope the device gets plugged back in? How long should it wait, and should it still work if it is plugged into a different hub? All of these questions were raised but there are no clear answers.

FS TRACK

■ *Security Attributes*

Michael Halcrow, eCryptFS developer, presented an idea to leverage SELinux and make file encryption/decryption based on application execution. For example, a policy could be defined so that the data would be unencrypted when OpenOffice is using the file but encrypted when the user copies the file to a USB key. After presenting the mechanism and mark-up language for this idea, Michael opened the floor to the audience. The general feeling was that SELinux is often disabled by users and that per-mount-point encryption may be a more useful and easier-to-understand user interface.

■ *Why Linux Sucks for Stacking*

Josef Sipek, Unionfs [8] maintainer, went over some of the issues involved with stacking file systems under Linux. A stacking file system, such as Unionfs, provides an alternative view of a lower file system. For example, Unionfs takes a number of mounted directories, which could be NFS, ext3, etc., as arguments at mount time and merges their name space.

The big unsolved issue with stacking file systems is handling modifications to the lower file systems in the stack. Several people suggested that leaving the lower file system available to the user is just broken and that by default the lower layers should only be mounted internally.

The new fs/stack.c file was discussed, too. This file currently contains a simple inode copy routine that is used by Unionfs and eCryptfs, but in the future more stackable filesystem routines should be pushed to this file.

Future work for Unionfs includes getting it working under lockdep and additional experimentation with an on-disk format. The on-disk format for Unionfs is currently under development and will store white-out files and persistent Unionfs inode data.

■ *B-trees for a Shadowed FS*

Many file systems use b-trees to represent files and directories. These structures keep data sorted, are balanced, and allow for insertion and deletion in logarithmic time. However, there are difficulties in using them with shadowing. Ohad Rodeh presented his approach to using b-trees and shadowing in an object storage device, but the methods are general and useful for any application.

Shadowing may also be called copy-on-write (COW). The basic idea is that when a write is made the block is read into memory, modified, and written to a new location on disk. Then the tree is recursively updated, starting at the child and using COW, until the root node is atomically updated. In this way the data is never in an inconsistent state; if the system crashes before the root node is updated then the write is lost but the previous contents remain intact.

Replicating the details of his presentation would be a wasted effort as his paper, "B-trees, Shadowing and Clone" [9], is well written and easy to read. Enjoy!

■ *eXplode the Code*

Storage systems have a simple and important contract to keep: Given user data, they must save that data to disk without loss or corruption even in the face of system crashes. Can Sar gave an overview of eXplode [10], a systematic approach to finding bugs in storage systems, to help root out the bugs that can break this contract.

eXplode systematically explores all possible choices that can be made at each choice point in the code to make low-probability events, or corner cases, just as probable as the main running path. And it does this exploration on a real running system with minimal modifications.

This system has the advantage of being conceptually simple and very effective. Bugs were found in every major Linux file system, including a fsync bug that can cause data corruption on ext2. This bug can be produced by doing the following: Create a new file, B, which recycles an indirect block from a recently truncated file, A, then call fsync on file B and crash the system before file A's truncate gets to disk. There is now inconsistent data on disk and when e2fsck tries to fix the inconsistency it corrupts file B's data. A discussion of the bug has been started on the linux-fsdevel [11] mailing list.

FS TRACK

■ *NFS*

The second day of the file systems track started with a discussion of an NFS race. The race appears when a client opens up a file between two writes that occur during the same second. The client that just opened the file is unaware of the second write and keeps an out-of-date version of the file in cache. To fix the problem a change attribute was suggested. This number would be consistent across reboots, would be unitless, and would increment on every write.

In general everyone agreed that a change attribute is the right solution; however, Val Henson pointed out that implementing this on legacy file systems will be expensive and will require on-disk format changes.

Discussion then turned to NSFv4 access control lists (ACLs). Trond Myklebust said they are becoming standard and Linux should support them. Andreas Gruenbacher is working on patches to add NFSv4 support to Linux but currently only ext3 is supported; more information can be found on the Native NFSv4 ACLs on the Linux [12] page. A possibly difficult issue will be mapping current POSIX ACLs to NFSv4 ACLs, but a draft document, "Mapping Between NFSv4 and Posix Draft ACLs" [13], lays out a mapping scheme.

■ *GFS Updates*

Steven Whitehouse gave an overview of the recent changes in the Global File System 2 (GFS2), a cluster file system where a number of peers share one large file system. The important changes include a new journal layout that can support mmap, splice, and other system calls on journaled files, page cache level locking, readpages() and partial writepages() support, and ext3 standard ioctls lsattr and chattr.

The readdir() function was discussed at some length, particularly the ways in which it is broken. A directory insert on GFS2 may cause a reorder of the extendible hash structure GFS2 uses for directories. In order to support readdir, every hash chain must be sorted. The audience generally agreed that readdir is difficult to implement and Ted Ts'o suggested that someone should try to go through committee to get telldir/seekdir/readdir fixed or eliminated.

■ *OCFS2*

A brief OCFS2 status report was given by Mark Fasheh. Like GFS2, OCFS2 is a cluster file system, designed to share a file system across nodes in a cluster. The current development focus is on adding features, as the basic filesystem features are working well.

After the status update the audience asked a few questions. The most requested OCFS2 feature is forced unmount and several people suggested that this should be a future virtual filesystem (vfs) feature. Mark also said that users really enjoy the easy setup of OCFS2 and the ability to use it as a local file system. A performance hot button for OCFS2 is the large inodes that occupy an entire block.

In the future Mark would like to mix extent and extended attribute data in-inode to utilize all of the available space. However, as the audience pointed out, this optimization can lead to some complex code. In the future Mark would also like to move to GFS's distribute lock manager.

■ *DualFS: A New Journaling File System for Linux*

DualFS is a file system by Juan Piernas that separates data and metadata into separate file systems. The on-disk format for the data disk is similar to ext2 without metadata blocks. The metadata file system is a log file system, a design that allows for very fast writes, since they are always made at the head of the log, which reduces expensive

seeks. A few performance numbers were presented: under a number of micro- and macro-benchmarks, DualFS performs better than other Linux journaling file systems. In its current form, DualFS uses separate partitions for data and metadata; this forces the user to answer a difficult question: How much metadata do I expect to have?

More information, including performance comparisons, can be found on the DualFS LKML announcement page [14] and the project homepage [15]. The currently available code is a patch on top of 2.4.19 and can be found on SourceForge [16].

■ *pNFS Object Storage Driver*

Benny Halevy gave an overview of pNFS (parallel NFS), which is part of the IETF NFSv4.1 draft [17] and tries to solve the single-server performance bottleneck of NFS storage systems. pNFS is a mechanism for an NFS client to talk directly to a disk device without sending requests through the NFS server, fanning the storage system out to the number of SAN devices. There are many proprietary systems that do a similar thing, including EMC's High Road, IBM's TotalStorage SAN, SGI's CXFS, and Sun's QFS. Having an open protocol would be a good thing.

However, Jeff Garzik was skeptical of including pNFS in the NFSv4.1 draft particularly because to support pNFS the kernel will need to provide implementations of all three access protocols: file storage, object storage, and block storage. This will add significant complexity to the Linux NFSv4 implementation.

Benny explained that the pNFS implementation in Linux is modular to support multiple layout-type specific drivers, which are optional. Each layout driver dynamically registers itself using its layout type and the NFS client calls it across a well-defined API. Support for specific layout types is optional. In the absence of a layout driver for some specific layout type, the NFS client falls back to doing I/O through the server.

After this overview Benny turned to the topic of OSDs: object-based storage devices. These devices provide a more abstract view of the disk than the classic "array of blocks" abstraction seen in today's disks. Instead of blocks, objects are the basic unit of an OSD, and each object contains both metadata and data. The disk manages the allocation of the bytes on disk and presents the object data as a contiguous array to the system. Having this abstraction in hardware would make filesystem implementation much simpler. To support OSDs in Linux Benny and others are working to get bi-directional SCSI command support into the kernel and support for variable-length command descriptor blocks (CDBs).

■ *Hybrid Disks*

Hybrid disks with an NVCache (flash memory) will be in consumers' hands soon. Timothy Bisson gave an overview of this new technology. The NVCache will have 128–256 MB of nonvolatile flash memory that the disk can manage as a cache (unpinned) or the operating system can manage by pinning specified blocks to the nonvolatile memory. This technology can reduce power consumption or increase disk performance.

To reduce power consumption, the block layer can enable the NVCache Power Mode, which tells the disk to redirect writes to the NVCache, thereby reducing disk spin-up operations. In this mode the 10-minute write-back threshold of Linux laptop mode can be removed. Another strategy is to pin all filesystem metadata in the NVCache, but spin-ups will still occur on nonmetadata reads. An open question is how this pinning should be managed when two or more file systems are using the same disk.

Performance can be increased by using the NVCache as a cache for writes, resulting in a long seek. In this mode the block layer would pin the target blocks, ensuring a write to the cache instead of incurring the expensive seek. Also, a file system can use the NVCache to store its journal and boot files for additional performance and reduced system start-up time.

If Linux developers decide to manage the NVCache there are many open questions. Which layer should manage the NVCache, the file system or block layer? And what type of API should be created to leverage the cache? Another big question is how much punishment these caches can take. According to Timothy it takes about a year (using a desktop workload) to fry the cache if you are using it as a write cache.

■ *Scaling Linux to Petabytes*

Sage Weil presented Ceph, a network file system that is designed to scale to petabytes of storage. Ceph is based on a network of object-based storage devices, and complete copies of each object are distributed across multiple nodes, using an algorithm called CRUSH. This distribution makes it possible for nodes to be added and removed from the system dynamically. More information on the design and implementation can be found on the Ceph homepage [18].

CONCLUSION

The workshop concluded with the general consensus that bringing together SATA, SCSI, and filesystem people was a good idea and that the status updates and conversations were useful. However, the workshop was a bit too large for code discussion. More targeted workshops will need to be held to work out the details of some of the issues discussed at LSF '07. Topics for future workshops include virtual memory and filesystem issues and extensions that are needed to the VFS.

REFERENCES

[1] http://lwn.net/Articles/187336/.

[2] http://lwn.net/Articles/187321/.

[3] http://article.gmane.org/gmane.comp.file-systems.ext4/986.

[4] http://article.gmane.org/gmane.comp.file-systems.ext4/899.

[5] http://infohost.nmt.edu/~val/review/repair.pdf.

[6] http://www.usenix.org/events/hotdep06/tech/prelim_papers/henson/henson.pdf.

[7] http://article.gmane.org/gmane.linux.ide/15942/.

[8] http://unionfs.filesystems.org.

[9] http://www.cs.huji.ac.il/~orodeh/papers/ibm-techreport/H-0245.pdf.

[10] www.stanford.edu/~engler/explode-osdi06.pdf.

[11] http://marc.theaimsgroup.com/?l=linux-fsdevel&m=117148291716485&w=2.

[12] http://www.suse.de/~agruen/nfs4acl/.

[13] http://www.citi.umich.edu/projects/nfsv4/rfc/draft-ietf-nfsv4-acl-mapping-03.txt.

[14] http://lwn.net/Articles/221841/.

[15] http://ditec.um.es/~piernas/dualfs/.

[16] http://sourceforge.net/project/showfiles.php?group_id=187143&package_id=218377.

[17] http://www.nfsv4-editor.org/drafts/drafts.html.

[18] http://ceph.sourceforge.net/.