2008 Linux Storage & Filesystem Workshop (LSF '08)

San Jose, CA
February 25–26, 2008

SUMMARY, LIGHTNING TALKS, AND WRAP-UP

Summarized by James Bottomley

James Bottomley opened the Lightning Talks with the presentation of some slides from Doug Gilbert about standards changes. Bottomley's main worry was that the new UAS (USB Attached SCSI) would end up having the compliance level of current USB with all the added complexity of SCSI. The INCITS decision to try to close off access to all SCSI standards was mentioned and discussed, with the general comment being that this was being done against the wishes of at least the T10 and T13 members. Ted Ts'o observed that INCITS is trying to make money selling standards documents and perhaps it was time for a group such as the Free Standards group to offer the SCSI Technical Committees a home.

Val Henson told everyone how she'd spent her summer vacation trying to speed up fsck by parallelizing the I/O, an approach which, unfortunately, didn't work. The main problems were that threaded async isn't better and that the amount of read ahead is small. A questioner from the floor asked if what we really want is to saturate the system. Val answered that only about a quarter of the buffer cache is used but that we'd like to hint to the operating system about our usage patterns to avoid unnecessary I/Os. Chris Mason commented that both BTRFS and EXT3 FS could use this.

Nick Bellinger asked about the tradeoff between putting things in the kernel and putting them in user space. His principal concern was the current target driver implementation in user space, which might lower the I/O per second in iSCSI. Martin Petersen asserted that we didn't really have enough data about this yet. There followed a discussion in which the KVM virtual I/O drivers were given as a counter example (the initial storage driver being a user space IDE emulation), which wound up with everyone concluding that KVM wasn't a good example. The final discussion was around NFS, with the majority of people suggesting that it went in-kernel not for performance reasons but for data access and concurrency reasons. The question of where to draw the line between kernel and user space in any implementation is clearly still a hot topic.

STACKING

Erez Zadok, State University of New York, Stony Brook

Erez Zadok discussed the problems of stacking one file system on top of other and possible solutions to those problems.

Page cache consumption: Each layer maintains its own copies of objects, and it is observed that some stackable file systems don't change the data between layers, resulting in duplicate pages with the same data.

Stack page consumption: Another problem is that every filesystem layer adds to the system stack. A possible short-term solution of having a larger kernel stack was suggested; having a linked list of operations and VFS iterating through it (such as Windows I/O manager) was suggested as a long-term solution.

Persistent inode numbers: Stackable file systems don't have persistent inode numbers. They need unique persistent inode numbers if they are to be exported; moreover, some user space tools demand it. A lower filesystem's inode number could be used, but this disallows crossing into other lower filesystems. Trond Myklebust suggested creating a mount point like NFS to combat this problem of crossing filesystem boundaries. Storing mapping between path names and inode numbers was also suggested, but it makes hard links difficult to deal with.

VFS Stacking Issues For Union Mounts, UnionFS, etc

Jan Blunck

Jan Blunck discussed whiteouts and ways to remove them. The implementation of whiteouts is straightforward: Add a new filetype and let the file system choose whether it wants to support it.

Jan's proposal was to convert readdir to be an inode operation and use it with a specialized filldir helper function to remove whiteouts from a logical empty directory and use the normal unlink of the directory otherwise. Al Viro proposed that Jan should let the file system itself choose how to remove logical empty directories. File systems know how they lay out the directory and implement whiteouts, so they could possibly have optimizations for removing them as well.

Also, there is a problem with readdir, seekdir, etc., all of which have been designed without stacking in mind. The question was raised whether we should design a new stacking-aware interface to be able to open files on specific layers and how we want to do union readdir. It was proposed that it would eventually be necessary to do duplicate removal and whiteout suppression in user space. This is because allocation of huge chunks in memory for caching readdir results is not a great idea.

FILESYSTEM AND NFS COMMUNICATIONS

Trond Myklebust, NetApp; Dave Chinner, SGI

Starting with problematic filesystem features, Trond Myklebust discussed the different models of Access Control Lists (ACLs) available in Linux (POSIX, NTFS, NFSv4) and mismatches between them. The mapping between POSIX and NFSv4 models is not lossless. NFSv4.1 might fix the correct rendering of POSIX ACLs, which Andreas Grunebacher is currently working on.

Moving further, Trond discussed extended attributes (xattrs) for NFS. Project "Labeled NFS" is designed to address the security issue and not the general xattrs. Questions were raised whether it could be used as general xattrs. Although NFSv4 named attributes could be used as xattrs, the model doesn't seem to fit well.

One use case put forth for xattrs was object-based metadata, where small 512- to 1,000-byte size xattrs could be used as tags.

Cache consistency issues still exist with NFS, with clients being unable to detect file changes on server. Change time and modification time provide means for update notification. Ext4 uses a change attribute flag but it is not clear whether other file systems are going to follow suit. Whether a persistent flag in the superblock can be used for update notification was discussed.

ADDRESS SPACE OPERATIONS

Steven Whitehouse, RedHat; Christoph Hellwig, SGI; Mark Fasheh, Oracle

Starting with an update on GFS2 (Global Filesystem 2), Steven Whitehouse briefed us on the latest developments, including smaller code base, shrunken in-memory data structures, and faster block map. GFS2 is now using page_mkwrite instead of filemap_fault. Also, writepage and writepages have been rewritten to avoid jdata deadlock.

Moving on to the address space, they found that some operations are almost duplicated and lots of redundancy can be done away with if the locking is handled properly. For example, readpage and readpages are very similar codes with the exception of lock ordering in the distributed file systems.

The next topic concerned extending the writes to multiple pages. Currently, locking and transactions impose a lot of overhead owing to per-page write operations. Also lock ordering with respect to source pages needs to be followed. Nick Piggin already has patches out doing this and could pave the way for a further line of thinking to solve this problem. Although there is a lot of overhead in reserving blocks in a file system, it is more useful to do reservation once than for each page.

In a general discussion about address space operations and consolidating ops to reduce redundancy, Christoph Hellwig pointed out the problems with writepages operation in XFS; help from virtual memory folks is required to improve it. He also said that improving the readpage operation is not easy

and starting with writepage could be a better way instead. Chris Mason mentioned that the readpage in BTRFS is small code but suboptimal.

PNFS

Tom Talpey, Ricardo Labiaga, and Trond Myklebust, NetApp

pNFS is about creating a parallel file system using NFS. It uses version 4.1 of NFS. Tom Talpey confirmed the wide and growing interest of users of pNFS, with applications such as MPI and MPI-IO being of natural interest. Also, there is a growing interest in NFS over Remote Direct Memory Access (NFS RDMA), which is going to come out in 2.6.25.

Currently there is no native pNFS-aware file system. A server exports whatever VFS has, and there is no built-in code yet. Typically, users use cluster backends with pNFS exports, which is difficult, having to plug in GPFS, for example, and then pNFS. Tom put forth the need for an easily deployable parallel solution.

Tom introduced simple pNFS (spNFS), a pNFS server developed and contributed by NetApp. As Tom explains, the code is small, simple, and entirely in user space. spNFS is linearly scalable once the metadata server is out of the way. The major advantage of spNFS is that it is filesystem agnostic and works with any backend file system. Also, spNFS doesn't require any modification to a NFSv3 server and only a slight modification to NFSv4. Major disadvantages of spNFS are that it is very simple with a manual configuration and it is difficult to restripe and modify the layouts. Client writethrough to the metadata server is possible, but painful. Also, the hooks in the metadata server are not elegant.

A question was raised whether the server should be in the kernel, and whether there is nothing in the kernel that it depends on. Tom mentioned that the layout cache is in memory, but said there is no architectural reason to not have it in the user space. Further goals of spNFS would be to increase the filesystem agnosticism and use arbitrary layout management for arbitrary backend file systems.

NEXT-GEN FILESYSTEM TOPICS, BTRFS

Zach Brown, Oracle

Zach started off by envisioning the next-generation general file system and the design goals of BTRFS. BTRFS has two trees, a giant metadata tree and an allocation tree. With the basic btrees, writeable snapshots are easy to have, allowing filesystem repairs to be localized instead of being proportional to the size of the disk.

Zach explained that BTRFS maintains an internal map between logical and device blocks. BTRFS also does block-level replication and Zach questions whether everything should be replicated. Erez asked whether deduplication is possible or planned. Chris said that it is possible.

Moving on to the integrity of the file system, Zach explained that BTRFS maintains per-block UUID, checksum, transaction number, and block id. BTRFS has checksums for every 4k of the file. Since inode numbers are unique in a subvolume of BTRFS and there is no global inode number space, they correspond to the key in the tree.

Zach discussed how BTRFS differs from ZFS. BTRFS has simple allocators as compared to ZFS. In BTRFS, there is only one metadata tree instead of one per directory. Dave Chinner said that ZFS has tree locking and parallel access features, which BTRFS currently does not have. Ted thought it would be advantageous to have concurrent access when the tree is longer. Chris said that the BTRFS disk format needs to be fixed before we can worry about CPU usage.

Zach pointed out that BTRFS can internally refer to multiple devices. Val Henson recalled that this was a pain in ZFS. Matt Mackall suggested that redefining the interface would be a nice thing to do. Christoph Hellwig suggested looking at XFS, which supports multiple data volumes. Matt pointed that one interesting thing about direct multi-pathing (DM) is that it is not possible to "cat" a device, and it is difficult to duplicate in a file system. Christoph said that the problem of doing this in LVM is bad.

Chris Mason explained that the transaction numbers in BTRFS are currently copy-on-write (COW). Transaction numbers go up and a restricted walk can be performed by using transaction numbers. With this, it is possible to find data newer than a given transaction number.

FILEBENCH

Spencer Shepler, Eric Kustarz, and Andrew Wilson, Sun Microsystems

Spencer described Filebench, a filesystem performance testing tool that generates artificial workloads resembling real-life applications. Since the current benchmarking tools cover only 10% of the important applications and are mainly micro benchmarks, the current approach for Filebench is to use expensive labor intensive benchmarks covering full application suites, such as Oracle using TPC-C. Actual applications can be benchmarked by installing and running them, but this involves a great deal of administrative overhead. SPECsfs is available for NFS, but it is tied to the workload and tests only the server, which is suitable for NFSv3 but not for NFSv4.

Filebench uses a model-based methodology in which it captures application-level behavior, understands it, snapshots it, and then replays it with desired scaling curves (e.g., throughput latency versus working set size). Ric asked whether tests are run long enough to see how filesystem aging affects the performance, since, because of fragmentation, older file systems behave differently than newer ones. Spencer said that Filebench tries to get close to this length, but practically it is impossible to achieve. It was suggested that a longer trace might help with aging a file system.

Filebench uses flow states to replicate different application behaviors. Flows are defined to represent I/O workload patterns. Threads are assigned to different processors, each using these flows. Threads with flows synchronize at points where the flow blocks until the completion of other flows. Ric Wheeler informed us that SNIA has been collecting I/O workload traces in academia which could be used to imitate real-world workloads. Spencer said that Filebench struggles with gathering data and understanding the application behavior. The difficult part is taking the application knowledge and transforming it into an equivalent workload. Considerable application knowledge is required for this. Dave Chinner pointed to a research paper presented at FAST '07 that dealt with generating and replaying the traces. Though mostly the work was done as a part of the OpenSolaris project, working versions of Filebench exist on Linux, MacOS, and FreeBSD. Current work in progress involves having composite flows allowing metaflow operations.

SCALABILITY IN LINUX STORAGE SUBSYSTEM

Dave Chinner, SGI

Dave Chinner tried to put forth the biggest challenges the Linux storage subsystem has in the coming 3 to 5 years.

Dave declared direct I/O to be a solved problem: things haven't changed for a long time now. This indicates that block- and driver-layer scalability looks good, but numerous micro-optimizations need to be done. Dave suggested having an infrastructure in place to expose the underlying geometry and storage topology dynamically. Having knowledge of the underlying geometry is helpful for many things, such as allocating large data blocks, getting full stripe writes, avoiding redundant data, and having a self-healing file system. Also it is helpful for determining optimal I/O size. Knowledge of disk hotspots would allow one to determine the best locations for placing redundant data and enable the file system to redistribute data.

Dave pointed out that the pdflush daemon writing the pages to the disk is single-threaded within a file system and does not scale well with too many writebacks. Currently, pdflush is only optimized for file systems that do not do I/O in write_inode operation. Problems with pdfush are not evident because people use ext2/3/4.

According to Dave, other things that could affect the storage subsystem in the near future include making DM/MD useful on common hardware RAID configurations and readying to the challenge of 50,000 IOPS per disk. Grant Grundler suggested that things can be learned from the network stack

architecture and IOPS can be handled similarly to the way network packets are handled, although it will still need to be way more scalable with many more interfaces. For this, both hardware and software need to evolve (e.g., HBAs can be made more intelligent to deal with multiple disks).

IPV6 SUPPORT FOR NFS

Chuck Lever, Oracle

Chuck gave timelines for both kernel- and user-space components required for NFS IPv6. Pre-2.6.23 kernels have some RPC server support for client-side rpcbind versions 3 and 4. Version 2.6.23 has the string-based NFS mount option parsing needed for IPv6 as well as for NFS over Remote Direct Memory Access (NFS RDMA) and cacheFS. Version 2.6.24 adds IPv6 support in the in-kernel RPC client, and 2.6.25 will add IPv6 infrastructure in the NFS client. It is expected that 2.6.26 will have NLM and NSM support and other remaining patches will support IPv6 in the in-kernel NFS server.

On the user-space side, many of the components required for NFS IPv6 are already in place. Library libtirpc provides IPv6-enabled RPC facilities in user space. The rpcbind daemon, which is now in the Fedora kernel, replaces portmapper, and rpc.statd and rpc.mountd now support IPv6. On the client-side command-line tools, IPv6 NFSv4 support can be added easily to mount.nfs, but NFSv2 and v3 would require version and transport discovery for NFS and NLM. exportfs on the server side would need to support specifying IPv6 addresses in the export rules.

Among the things that are missing are RPC pipefs changes, IPv6 support for NFSv4 callbacks and referrals, and significant test capabilities. Chuck said it is expected that basic IPv6 support in all NFS components will become available for distribution in 2008.

NFS RDMA

Tom Talpey, NetApp

Tom talked about the current state of NFS over Remote Direct Memory Access (NFS RDMA) transport. NFS RDMA provides five times the speed of NFS. RPC RDMA is now in 2.6.24 and has been working since. Tom mentioned similar changes in the server, but these are harder to generalize. Listening to all the clients all the time is difficult to achieve without a fundamental change, and Tji's code needs an architectural overhaul.

It is likely that 2.6.25 will have an end-to-end RDMA client– server. Currently, it works in the test environment, operating on any InfiniBand hardware supported by the kernel and there is a nice family of adapters to choose from.

Surprisingly, Tom said, cache writes did not use 100 Mbps largely because of pdflush and dirty ratio. The bigger the memory footprint, the longer it took for NFS to push writes. Tom pointed out that pdflush is a real problem, and they are struggling to get cached writes faster.

ISSUES WITH DATA-ORDERED AND BUFFER HEADS

Mark Fasheh, Oracle

Data-ordered mode uses buffer heads for block sizes less than the page size. Buffer heads are attached to pages and are written out at commit time. Mark suspects possible metadata or data lifetime issues. Also, using block size in address operations takes many extra lines of code. Plus, the overhead of using buffer heads for logical to physical mapping of data is harmful to extents.

To move toward a proposed solution, Mark pointed out that write_begin and write_end allow page lock reordering. Also, page_mkwrite allows allocating before writepage is called. Mark suspects that journaled data could be a problem for ext3. As a solution to the buffer-headless ordered-data mode, it is suggested that the file system should handle accounting of data ordering. Logical to physical mapping can be maintained by using an internal extent map. Journal_dirty_data can be replaced, and the file system can handle the truncate. The default behavior of JBD would remain same. Jan Kara suggested that every file system can handle its own data and not have the complexity of journaled data code, and dirty data can be flushed before truncate is started, which is simple but not performant.

Andreas Dilger suggested that JBD2 be dissolved and its features be merged with JBD instead. Chris pointed out that getting rid of buffer heads might make sense only for ext3 and ext4. Ted wondered whether using JBD is a way to get rid of buffer heads, and Jan explained that it holds true only for data. Val Henson pointed out that Ingo Molnar's patches make relative atime (access time) with batching much smarter; atime provides updates in memory and inodes are not marked dirty. Val pointed out a problem: If the inodes are not written to the disk, the atime could go backward. The Windows operating system does a timeout and writes out on disk after the timeout, said Erez Zadok. Ted Ts'o suggested that tricks such as writing out the inodes if they are adjacent to where the current inodes are written could be explored. Dave Chinner added that XFS has relative atime and not atime.

CEPH DISTRIBUTED FILE SYSTEM

Sage Weil, University of California, Santa Cruz

Ceph is made up of user-space daemons for configuring, handling name spaces, and communicating with OSDs for data and metadata. A cluster of monitors provides filesystem configuration. OSD clusters are storage clusters with 4 to 10,000 nodes. The cluster size is dynamic, but there needs to be at least three monitors for high availability. The file open operation, for example, goes to the metadata server (mds) and file I/O goes to OSDs, which can talk to each other. Objects are replicated on multiple nodes. The mds embeds inodes inside the directory and has a long journal to aggregate changes and limit directory metadata writes. The mds allows dynamic migration and does load balancing. Objects are stored and replicated across a dynamic cluster and can communicate with each other for various things such as failure detection. A given object identifier locates the OSD and makes dynamic reorganization very easy. OSDs have the capability to migrate in the background.

Sage said that the kernel client for Ceph is basically functional, but it still lacks the parts of I/O path and write-sharing. The actual inode size in the implementation is 64 bits. The metadata behavior is similar to that of NFS but is moving toward being stateful. Sage indicated that many additional features, such as locking and direct I/O, would be added soon. In answering Mark Fasheh's question, Sage explained that mmap is planned but currently at a low priority. Sage also said that the share-writeable feature is a best-effort one.

Sage explained that near-out-of-memory cases are something to be dealt with gracefully. The communication model complications add to the problem of memory reservation. Also, dual write acknowledgments are required from both OSDs and during the commit on the disk. Compound atomic transactions and asynchronous commit notifications are object storage requirements. Currently, ebofs is used and performs well but requires more code to maintain. As an alternative, Sage suggested adding a layer over existing file systems, but fsync degenerates behavior in most file systems. Along with a complete kernel client, Sage listed usability tools, distributed quota architecture, and fine-grained snapshots among future tasks.

LEASES ON CLUSTERED FILE SYSTEM

Felix Blyakher, SGI

Felix Blyakher discussed extending the lease functionality to the cluster file systems. He discussed a prototype implementation and brought up issues that exist. The function setlease allows local caching and the lease is broken on conflicting operations. Leases can be requested using the fcntl() system call. NFS uses leases for delegations and Samba also uses them for local caching.

Explaining how setlease evolved over time, Felix pointed out that previously individual file systems were unaware of setlease. Currently, they do return the error code EINVAL but misses out on possible optimizations. Felix suggested that for clusterwide setlease, file systems evaluate the lease over the cluster before granting the lease. Felix explained a prototype implementation of setlease over a cluster. The server knows which files are opened and in what mode. The client asking for a lease does an RPC call to the server, which either cancels the request or grants it depending upon whether the file is already open in a conflicting mode.

It is expensive to notify all nodes clusterwide when a lease is broken. The server has to notify the lease owner about the conflicting file open, and it has to wait until the client responds with the cached copy of the file. It was suggested that the server can use timeouts while waiting for clients to respond to the lease break. Andreas Dilger pointed out the same problem with lusterfs when the server is too busy to drop the current lease. Instead, the server should be able to extend the lease.