## 2008 Linux Storage & Filesystem Workshop (LSF '08)

# San Jose, CA
# February 25–26, 2008

## Storage Track

*Summarized by Grant Grundler (grundler@google.com)*



Thanks are also owed to James Bottomley, Martin Petersen, Chris Mason, and the speakers for help in collecting material for this summary.

Several themes came up over the two days:

*Theme 1: Solid State Drives*
SSDs (Solid State Disks) are coming. There was a good presentation by Dongjun Shin (Samsung) on SSD internal operation, including some discussion on which parameters were needed for optimal operation (theme #2). The I/O stack needs both micro-optimizations (performance within driver layers) and architectural changes (e.g., you have to parameterize the key attributes so that file systems can utilize SSDs optimally). Intel presented SCSI RAM and ATA_RAM drivers to help developers tune the SCSI, ATA, and block I/O subsystems for these orders-of-magnitude-faster (random read) devices.

Hybrid drives were a hot topic at LSF '07 but were only briefly discussed in the introduction this year. The conclusion was that separate cache management was a nonstarter for hybrid drives (which was the model employed by first-generation hybrid drives). If they're to be useful at all, the drive itself has to manage the cache. The hybrid drives are expected to provide methods allowing the OS to manage the cache as well. It was suggested that perhaps Linux can just be given the hybrid drive flash as a second device to the filesystems.

*Theme 2: Device Parameterization*
The device parameters discussion is just beginning on how to parameterize device characteristics for the block I/O schedulers and file systems. For instance, SSDs want all writes to be in units of the erase block size if possible, and device mapping layers would like better control over alignment and placement. The key object here is how to provide enough parameters to be useful but not so many that "users" (e.g., the file system) get it wrong. The general consensus was that having more than two or three parameters would cause more problems than it solved.

*Theme 3: I/O Priorities*
I/O priorities and/or bandwidth sharing have lots of folks interested in I/O schedulers. There was consideration about splitting the I/O scheduler into two parts: an upper half to deal with different needs of feeding the Q (limit block I/O resource consumption) and a lower half to rate limit what gets pushed to the storage driver.

*Theme 4: Network Storage*
Two technologies were previewed for addition to the Linux kernel: pNFS (Parallel NFS) and FCoE (Fiber Channel over Ethernet). Neither is ready for kernel.org inclusion but some constructive guidance was given on what directions specific implementations needed to take.

The issues facing iSCSI were also presented and discussed. User- versus kernel-space drivers were hot topics within those Networked Block Storage forums.

### INTRODUCTION AND OPENING STATEMENTS: RECAP OF LAST YEAR

*Chris Mason and James Bottomley (a.k.a. jejb)*

This session was primarily a scorecard of how many topics discussed last year are fixed or implemented this year. The bright spots were the new filesystem (BTRFS, pronounced "butter FS," which incorporates B-trees for directories and an extent-based filesystem with $2^{64}$ maximum file size) and emerging support for OSD (Object-base Storage Device) in the form of bidirectional command integration (done) and long CDB commands (pending); it was also mentioned that Seagate is looking at producing OSD drives.

Error handling was getting better (e.g., no more breaking a 128k write up into individual sectors for the retry) but there's still a lot of work to be done and we have some new tools to help test error handling. The 4k sector size, which was a big issue last year, has receded in importance because manufacturers are hiding the problem in firmware.

## SSD

*Dongjun Shin, Samsung Electronics*

Solid State Disk (SSD) storage was a recurring theme in the workshop. Dongjun gave an excellent introduction and details of how SSDs are organized internally (sort of a two-dimensional matrix). The intent was to give FS folks an understanding of how data allocation and read/write requests should be optimally structured. "Stripes" and "channels" are the two dimensions to increase the level of parallelization and thus increase the throughput of the drive. The exact configurations are vendor specific. The tradeoff is to reduce stripe size to allow multithreaded apps to have multiple I/Os pending without incurring the "lock up a channel during erase operation" penalty for all pending I/Os. Hard disk drives (HDDs) prefer large sequential I/Os, whereas SSDs prefer many smaller random I/Os.

Dongjun presented postmark (mail server benchmark) performance numbers for various file systems. An obvious performance leader seemed to be nilfs for most cases, and it was never the worst. Successive slides gave more details on some of the FSs tested. Some notable issues were that flush barriers kill XFS performance and that BTRFS performance was better with 4k blocks than with 16k blocks.

Dongjun shared what specific standards bodies were available or being added to enable better performance of SSDs: a trim command to truncate and mark space as unused and an SSD identify to report page and erase block sizes (for FS and Vol Mgr).

His last slide neatly summarized the issues, covering the entire storage stack from FS, through the block layer, to the I/O controller.

jejb asked which parameter was the most important one. The answer wasn't entirely clear but it seemed to be the erase block size followed by the stripe size. This was compared to HD RAID controller design, which would like to export similar information.

Flush barriers are the only block I/O barriers defined today and we need to revisit that. One issue is the flush barriers kill performance on the SSDs since the flash translation layer could no longer coalesce I/Os and had to write data out in blocks smaller than the erase block size. Ideally, the file system would just issue writes using erase block sizes. We (the Linux kernel community) need to define more appropriate barriers that work better for SSDs and still allow file systems to indicate the required ordering/completions of I/Os.

## ERROR HANDLING

*Ric Wheeler, EMC*

Ric Wheeler introduced the perennial error-handling topic with the comment that bad sector handling had markedly improved over the "total disaster" it was in 2007. He moved on to silent data corruption and noted that the situation here was improving with data checksumming now being built into filesystems (most notably BTRFS and XFS) and emerging support for T10 DIF. The "forced unmount" topic provoked a lengthy discussion, with James Bottomley claiming that, at least from a block point of view, everything should just work (surprise ejection of USB storage was cited as the example). Ric countered that NFS still doesn't work and others pointed out that even if block I/O works, the filesystem might still not release the inodes. Ted Ts'o closed the debate by drawing attention to a yet to be presented paper at FAST '08 showing over 1,300 cases where errors were dropped or lost in the block and filesystem layers.

Error injection was the last topic. Everybody agreed that if errors are forced into the system, it's possible to consistently check how errors are handled. The session wrapped up with Mark Lord demonstrating new hdparm features that induce an uncorrectable sector failure on a SATA disk with the WRITE_LONG and WRITE_UNC_EXT commands. This forces the on-disk CRCs to mismatch, thus allowing at least medium errors to be injected from the base of the stack.

## POWER MANAGEMENT

*Kristen Carlson Accardi, Intel*

Arjan van de Ven wrote PowerTOP and it's been useful in tracking down processes that cause CPU power consumption but not I/O. Although kjournald and pdflush are shown as the apps responsible, obviously they are just surrogates for finishing async I/O. For example, postfix uses sockets, which triggers inode updates. Suggestions for preventing this include using lazy update of nonfile inodes and virtual inodes.

With ALPM (Aggressive Link Power Management, http://www.lesswatts.org/tips/disks.php), up to 1.5 watts per disk can be saved on desktop systems. Unlike disk drives, no hardware issues have been seen with repeated powering up or down of the physical link so this is safer to implement. The problem is that AHCI can only tell which state the drive is currently in (link down or not) and not how long or *when* the link state changes (making it hard to gather metrics or determine performance impact). Performance was of interest since trading off power means some latency will be associated with coming back up to a full-power state. The transition (mostly from Async Negotiation (AN) when restoring power to the Phys) from SLUMBER to ACTIVE state costs about ~10 ms. Normal benchmarks show no performance hit as the drive is always busy. We need to define a bursty power benchmark that is more typical of many environments.

Kristen presented three more ideas on where Linux could help save power. The first was to batch-average group I/O; 5–30 seconds is normal to flush data, so instead wait up to 10 minutes before flushing these. The second suggestion was a question: Can the block layer provide hints to the low-level driver? For example, "Soon we are going to see I/O; wake up." The third suggestion was making smarter timers to limit CPU power-up events—that is, coordinate the timers so they can wake up at the same time, do necessary work, then let the CPU go to a low-power state for a longer period of time. But we need a new interface that specifies how much variability each timer can tolerate.

Ric Wheeler (EMC) opened up the discussion on powering down disks since the savings there are typically 6–15 watts per disk. But powering up disks requires coordination across the data center. Otherwise network traffic could cause lots of drives to spin up at the same time. He also suggested using SSD to cache and wondered how that might work.

Eric Reidel (Seagate) mentioned EPA requirements: Should we idle CPU versus the hard drive? One would be trading off power consumption for data access. He said that Seagate can design for higher down/up lifecycles. Currently, it's not a high count only because Seagate is not getting data from OEMs on how high that count needs to be. It was noted that one version of Ubuntu was killing drives after a few months by spinning them down or up too often.

The final question involved whether ALPM commands work over the SATA port multiplier. Kristen thought it should but had not tested it. She also suggested using a "Watts Up" meter to measure actual power savings.

## CFQ AND CONTAINERS

*Fernando Luis Vazquez Cao*

Cao touched on three related topics: block I/O (BIO) resources and cgroups, which define arbitrary groupings of processes; I/O group scheduling; and I/O bandwidth allocation (ioband drivers, which manage I/O bandwidth available to those groups). The proposals were not accepted as is but the user-facing issues were agreed upon. The use case would be Xen, KVM, or (I assumed) VMware.

Currently, the I/O priority is determined by the process that initiated the I/O. But the I/O priority applies to *all* devices that process is using. This changed in the month preceding the conference and the speakers acknowledged that. A more complex scheme was proposed that supports hierarchical assignment of resource control (e.g., CPU, memory, I/O priorities). A graphical example was given on slide 9 (of 18).

Proposed was page_cgroup to track write bandwidth (but is not needed or used for read bandwidth tracking). One must track when the page is dirtied—forking across dirty pages is a trouble area. The page would get assigned to a cgroup when the BIO is allocated. One advantage of the get_context() approach is that it does *not* depend on the current process and thus would also work for kernel threads.

Slide #12 proposed three ideas on group-aware scheduling/bandwidth control. This was a critical slide for this presentation and much of the content that follows will refer to those three ideas. This revisited the "stackable requests" discussion and it was clear that Request-DM (Device Mapper) multi-path needs the same infrastructure.

Idea #1 proposed a layer between the I/O scheduler and the I/O driver. This requires some changes to elevator.c and additional infrastructure changes. Jens Axboe pointed out that one can't control the incoming queue from below the block I/O scheduler. The scheduler needs to be informed when the device is being throttled from below in order to prevent the I/O scheduler queue from getting excessively long and consuming excessive memory resources. Jens suggested they start with #1 since it implements fairness.

Idea #2 was generally not accepted. This made the last section of the talk (Slides 15 to 17) moot. For idea #3 (group scheduler above LVM make_request), adding a hook so cgroup can limit I/O handed to a particular scheduler was proposed and this idea got some traction. Jens thought #3 would require less infrastructure than #1. Effectively, #3 would lead to a variable sized Q-depth. And #3 would limit BIO resource allocation.

Fernando/Hiroaki's I/O bandwidth control implementation was using Token Buckets. This implements proportional sharing of available bandwidth. They also wanted a max bandwidth limit for each cgroup. Ric Wheeler observed that latency of seekiness reduces the usable bandwidth and makes workload/bandwidth prediction impossible.

Discussion about what needs to be scheduler-independent ensued. Most folks are attracted to this idea because code-sharing is good and one wouldn't have to touch any of the schedulers directly. If integrated into the I/O scheduler, I/O bandwidth control would add some overhead for all users, even those who don't want it. Jens was encouraging as well.

But the idea has some downsides. After LSF, Naveen Gupta (Google) talked with Jens Axboe and thought he convinced Jens that this may not be such a good idea. One can implement such a bandwidth-limiting policy in many ways and different schedulers will interact

differently with each one. And it seems that the scheduler issuing its own I/O (e.g., anticipatory reads) will just break the B/W limiting and make the wrong tradeoffs. One also can't merge cgroup I/Os together, and the I/O scheduler would have to know that.

"Tracking I/O" was about how to track write bandwidth, not for read. The Slide 13 diagram showed the proposed data structures. It would associate ownership (a.k.a. I/O priority) of dirtied pages with the cgroup or original process. A dirty page would get assigned to a cgroup when allocating the BIO. One must track dirtied pages for this to work, but forking with dirtied pages is going to be ugly. James Bottomley preferred the io_context approach.

### NCQ EMULATION

*Gwendal Grignou, Google*

Gwendal started by explaining what Native Command Queuing (NCQ) was, his test environment (fio), and which workloads were expected to benefit. In general, the idea is to let the device determine (and decide) the optimal ordering of I/Os since it knows current head position on the track and the seek times to any I/Os it has in its queue. Obviously, the more choices the device has, the better choices it can make and thus the better the overall throughput the device will achieve. Results he presented bear this out, in particular for small (<32k), random read workloads (e.g., for a classic database).

But the problem is that since the device is deciding the order, it can chose to ignore some I/Os for quite a while too. And thus latency-sensitive applications will suffer occasionally with I/Os taking more than 1–2 seconds to complete. He implemented and showed the results of a queue plugging that starved the drive of new I/O requests until the oldest request was no longer over a given threshold. Other methods to achieve the same effect were discussed but each had its drawbacks (including this one).

He also showed how by pushing more I/O to the drive, we also impact the behavior of block schedulers to coalesce I/O and anticipate which I/Os to issue next. Briefly discussed (and rejected) was replicating the drive topology knowledge in the I/O scheduler. The German *CT Magazine* was stated to have written a program to accurately characterize drive models. This was rejected since it introduced a substantial maintenance issue and also required knowing RPM and tracking the current head position. Jens Axboe also noted that I/O priorities and B/W allocation policies become harder to enforce.

And although NCQ was effective on a best-case benchmark, it was debated how effective it would be in real life (perhaps <5%). The actual performance gains depend too much on workload and often the best performance improvements come from merging I/O requests (which also reduces number of rotations of the platters for a given I/O sequence).

### MAKING THE IO SCHEDULER AWARE OF THE UNDERLYING STORAGE TOPOLOGY

*Aaron Carroll and Joshua Root, University of New South Wales*

Disclosure: Grant Grundler arranged the grant from Google to fund this work. HP is also funding a portion of this work.

Aaron and Joshua have created an infrastructure to measure the performance of any particular block trace and were interested in seeing how I/O schedulers behave under particular workloads.

The performance slides are graphs of how the various schedulers perform as one increases the number of processes generating the workload. They tested the following schedulers: AS (Anticipatory Scheduler), CFQ (Completely Fair Queueing), Deadline, FIFO, and NOOP. Note that with the idle-window parameter turned off, the (IIRC) CFQ scheduler looks like the NOOP scheduler.

They tested a few different configs: RAID 0 sequential, async; single-disk random and sequential; and 10-disk RAID 0 random and sequential. Of the various parameters—queue depth, underlying storage device type, and RAID topology—they wanted to establish which parameters were relevant and find the right way to determine those parameters (e.g., by user input, with runtime microbenchmark measurements, by asking lower layers). Queue depth is generally not as important nor is it very helpful for any sort of anticipation. For device type, it would be obvious to ask the underlying device driver but we need a suitable level of abstraction; later slides discussed that. For RAID topology, the key info was "stripe boundaries" (which others refer to as "alignment") and width. He suggested that one could do per-spindle scheduling, but that wasn't warmly embraced.

Ric Wheeler said that he can see differences in performance depending on the seek profile if most I/Os are to one disk at a time and if Array is doing read ahead. Random reads for RAID 3/5/6 depend on worst case (i.e., the slowest drive).

Jens mentioned that disk type could be exported easily by plugging (stopping Q to build a bigger I/O) or through an anticipatory maneuver (starting new I/O, after the previous one has completed but before the application has requested the data/metadata).

We discussed how to split fairness/bandwidth sharing/priorities (or whatever you want to call it) so that a component above the SW RAID md driver would manage incoming requests. A lower half of the scheduler would do a time slice (fairness per device instead of per process). One needs to define an API for this.

It was also noted that CFQ can unfairly penalize bursty I/O measurements. One suggestion was to use Token Bucket to mitigate bursty traffic.

Discussion resumed on which parameters to export.

Partitioned LUNs are a simple case of stupidity ("Don't do that on devices that allow varying the size of the LUN") that exporting the preferred alignment and size could avoid. The problem is that the RAID stripe becomes misaligned if the start of the LUN doesn't happen to align with the underlying RAID stripe. XFS does attempt to align with the RAID stripe and gets messed up on partitioned LUNs.

Aaron and Joshua introduced two new schedulers that might be useful in the future: FIFO (true fifo, without merging) and V(R) SSTF. There was no discussion on these.

There was also a brief discussion about the CFQ bug they (Aaron and Joshua) found. This was related to the case where CFQ scheduler tries to measure Q-depth.

### DMA REPRESENTATIONS: SG_TABLE VS. SG_RING IOMMUS AND LLD'S RESTRICTIONS

*Fujita Tomonori*

(LLD stands for Low Level Driver, e.g., a NIC or an HBA device driver.)

Fujita did an excellent job of summarizing the current mess that is used inside the Linux kernel to represent DMA capabilities of devices. As Fujita dove straight into the technical material with no introduction, I'll attempt to explain what an IOMMU is and the Kernel DMA API.

Historically, I/O devices that are not capable of generating physical addresses for all of system RAM (e.g., ISA, EISA, or, more recently, 32-bit PCI) have always existed. The solution without an IOMMU is a "bounce buffer" in which you DMA to a low address the device can reach and then memcpy to the target location. I/O Memory Management Units (IOMMUs) can virtualize (a.k.a. remap) host physical address space for a device and thus allow these legacy devices to directly DMA to any memory address. The bounce buffer is no longer necessary and we save the CPU cost of the memcpy. IOMMUs can also provide isolation and containment of I/O devices (preventing any given device from spewing crap over random memory—think Virtual Machines), merge scatter-gather lists into fewer I/O bus addresses (more efficient block I/O transfers), and provide DMA cache coherency for virtually indexed/tagged CPUs (e.g., PA-RISC).

The PCI DMA Mapping interface was introduced into the Linux 2.4 kernel by Dave Miller primarily to support IOMMUs. James Bottomley updated this to support noncache coherent DMA and become bus-agnostic by authoring the Documentation/DMA-API.txt in Linux 2.6 kernels. (Just to be clear, neither author did this alone—see the documents for shared credits.)

The current DMA API also does not require the IOMMU drivers to respect the max segment length (i.e., IOMMU support is coalescing DMA into bigger chunks than the device can handle). The DMA alignment (i.e., boundaries a DMA cannot cross) has similar issues (e.g., some PCI devices can't DMA across a 4-GB address boundary. Currently, the drivers that have either length or alignment limitations have code to split the DMA into smaller chunks again. The max_seg_boundary_mask in the request queue is not visible to IOMMU since only struct device * is passed to IOMMU code.

Slide 7 proposes adding a new struct device_dma_parameters to put all the DMA-related parameters into one place and make them visible to IOMMUs. This idea was rejected as overkill. The solution preferred by jejb and others was just to add the missing fields to struct device. Slide 10 summarized where all the various parameters currently live; adding another struct to deal with it was just adding more pointers that we didn't really need. Most devices do DMA, and adding the fields directly to struct device was the cleanest approach.

jejb also confessed that he added u64 *dma_mask to avoid ripping the DMA_mask out of pci_dev and thus annoying Dave Miller (who was the primary author of the PCI DMA API). He agreed that this needs to happen.

The next issue discussed was IOMMU performance and I/O TLB flushing. The IOMMU driver (and HW) performance is critical to good system performance. New x86 platforms support virtualization of I/O and thus it's not just a high-end RISC computer problem.

Issues discussed related to IOMMU mapping and unmapping included the following:

1. How does one best manage IOMMU address space? Through common code? Some IOMMU drivers use bitmap (most RISC); Intel uses a "Red Black" tree. He tried converting POWER to use Red/Black tree and lost 20% performance with netperf.

jejb and ggg agree that the address allocation policy needs to be managed by the IOMMU or architecture-specific code since I/O TLB replacement policy dictates the optimal method for allocating IOMMU address space.

2. When should we flush I/O TLB? One would like to avoid flushing the I/O TLB since (a) it's expensive (as measured in CPU cycles) and (b) it disturbs outstanding DMA (forces reloading I/O TLB). However, if we flush the entries when the driver claims the DMA is done, we can prevent DMA going to a virtual DMA address that might have been freed and/or reallocated to someone else (i.e., flushing I/O TLB is required to prevent devices from corrupting RAM). This is not a common problem, but having an IOMMU to enforce DMA access to RAM is the only way to conclusively determine this (assuming you don't have expensive bus analyzers).

The bottom line is that there is a tradeoff between performance and safety (a.k.a. robustness).

3. Should we just map everything once? (Assume that IOMMU can span all of RAM.) IIRC and some RISC archs did this in the 2.2 kernel. The performance advantage is that you don't need to map, unmap, and flush I/O TLB for individual pages but the tradeoff is isolation (since any device can DMA anywhere), which can be useful in some cases (e.g., embedded devices such as an NFS server).

The last DMA-mapping-related issue was SG chaining versus SG rings. This is a settled issue (SG chaining is preferred) and Rusty wasn't present to argue the other side. The scsi_sglist() macro was changed after 2.6.24 (slide 20 of 27). SCSI data accessors do allow insertion between chains, with the last entry having a flag to indicate that the next chain is valid. Slide 24 (of 27) shows how to walk the scsi_sglist().

It was mentioned that the (pass-through) sg (SCSI generic) driver can build very a large sg_list—bigger than the midlayer can handle. Boaz Harrosh pointed out the sg_list from the BIO layer can be inserted or extended also.

### ISCSI TRANSPORT CLASS SIMPLIFICATION

*Mike Christie and Nicholas Bellinger*

The main thrust here is that common libs are needed to share common objects between transport classes. In particular, Mike called out the issues that the lsscsi maintainer has faced across different kernel versions where /sys has evolved. James Bottomley conceded that there were issues with the original implementation. Mike also mentioned problems with parsing /sys under iSCSI devices. The goal is to provide a common starting point for user space visible names.

Mike proposed a scsi_transport_template that contained new scsi_port and scsi i_t_nexus data structures. iSCSI also needs an abstraction between SCSI ports—an I_T_nexus. Other users of I_T nexus were also discussed.

James Bottomley pointed out that libsas already has an I_T_nexus abstraction. It provides a host/port/phy/rphy/target/lun heirarchy for /sys. However, the exported paths need to be more flexible. Mike floated the idea of a new library to encapsulate the SCSI naming conventions so that tools like lsscsi wouldn't have to struggle.

Development for iSCSI focuses on Linux-iSCSI.org. iSCSI exposed issues with error recovery. The slides neatly summarize most of the points Nicholas wanted to make.

"Advanced Features" discussed starting with Multiple Connections per Session (MC/S), which was stated to be faster than Ethernet bonding using Gigabit Ethernet. iSER (RFC-5045 and related RFC-5040/-5044) standards will help iSCSI to support 10GigE link speeds and direct data placement (e.g., iSCSI over Infiniband). ISNS (RFC-4171) would rework fabric discovery and is extensible for other storage fabrics.

The project status of the various pieces related to iSCSI start on slide 8 (of 19): SCSI Target (current design), SCST (older Target design), LIO-SE (Linux iSCSI.Org Storage Engine) and LIO-Target, and IET.

Slide 14 showed the relationships among front ends (e.g., iSER/iWARP), the common storage engine (SE), and the data structures used in the kernel to track storage (e.g., struct page and struct scsi_device).

Slide 15 made some good arguments for sharing SCSI CDB (command block) emulation code in the kernel, but implementation wasn't acceptable (yet).

Slide 16 (of 19) started the contentious discussion over user versus kernel space implementations. A relevant quote from Linus: is that the only split that has worked pretty well is connection initiation/setup in user space, with actual data transfers in kernel space.

The lively but inconclusive debate left me thinking that most of the code will forced to live in user space until evidence is presented otherwise. iSCSI, FC, and SAS would be better in kernel because concurrency control fundamentally resides in the kernel. And LIO-SE assumes most drivers belong and are implemented in kernel space because transport APIs force middle code into kernel. KVM performance suffers because of movement among virtual kernels (for I/O IIRC).

## REQUEST-BASED MULTI-PATHING

*Kiyoshi Ueda and Jun'ichi Nomura, NEC*

The key point was proposed multi-path support below the I/O schedule; this seems to be the favored design.

Problems are expected with request completion and cleaning up the block layer. An RFC for a request stacking framework was posted to linux-scsi and linux-ide mailing lists. See the last slide (37) for URLs to postings.

The big advantage of request-based DM multi-path is that, since BIOs are already merged, the multi-path driver can do load balancing since it knows exactly how many I/Os are going to each available path.

Three issues were raised (see Slide 7):

Issue 1: How do you avoid deadlock during completion?

Issue 2: How do you keep requests in mergeable state?

Issue 3: How do you hook completion for the stacking driver?

Issue 1: blk_end_request() will deadlock because the queue lock is held through the completion process. jejb suggested moving completions to tasklet (soft IRQ) since SCSI at one point had the same issue. There was also some discussion about migrating drivers to use blk_end_request instead of __blk_end_request().

Issue 2: Busy stack drivers won't know when the lower driver is busy, and once a request is removed from the scheduler queue, it's no longer mergeable. Slides 14–21 have very good graphic representations of the problem. jejb suggested prep and unprep functions to indicate whether requests are mergeable or not; "prepared" means it's unmergeable and "unprep" would push it back to a mergeable state. Someone volunteered that Jens Axboe had objections, but Jens had left the room a bit earlier.

One basic difference between BIO (existing code) and proposed Request DM is that device locking (queue lock) will be required for both submission and completion of the Request DM handler I/Os and is not required by BIO.

Issue 3: The problem is that req->end_io() is called too late and is called with a queue lock held. Solutions were offered and discussed in the remaining slides (29–36):

Regarding issue 1, one should only allow use of nonlocking drivers (i.e., drivers that do not lock in the completion path): All SCSI drivers, cciss, and i2o already meet this criterion; Block Layer is using locking completion; a DASD driver change is needed. There was a discussion about how much work it was to convert other drivers.

If there is no submission during completion, then pass submission to a work queue (basically, a work-around solution).

It was suggested that the free and completion functions be separated, then a work queue be used to harvest free requests.

jejb said that they also served other purposes but agreed they could be separated. SCSI already does nonlocking drivers.

The issue 2 busy_state is not a queue condition exclusively. Other resources may result in a busy response (e.g., a mapping resource or other prep work might fail).

Possible solutions for issue 3: Stacking Hook: (A) add end_io (was the RFC already posted?); (B) move the end_io calling place; (C) use end_io as is. For (A), one can't use end_io for the stacking driver. end_io is called to destroy the request. The call to end_io has to move until no one needs to reference the request. There was discussion around how to make req->blk_end_io() work: allow it to point at md_end_io(), end_io(), or free(). (B) had one easy question from Boaz: How many callers are there of end_io()? With (A), there are two: sr and sd. But there are very few end_io functions, so one just has to make sure it's called exactly once. (C) wasn't considered a solution. Partial completion couldn't be supported by stacking drivers.

## FS AND VOLUME MANAGERS

*Dave Chinner, SGI*

Dave covered several major areas: a proposal he called "BIO hints" (which Val Hansen called "BIO commands"); DM (Device Mapper) multi-path; chunk sizes; and I/O barriers.

BIO hints is an attempt to let the FS give the low-level block hints about how the storage is being used. The definition of "hint" was something that the storage device could_(but was not required to) implement for correct operation. Suggestions he offered were "release" (indicate space is no longer in use), COW (take a snapshot, with a preallocated set of ranges), and Don't COW (identify a range of blocks we never snapshot, e.g., journal).

The function mkfs could provide the "space is free" hints and would be good for RAID devices, transparent security (zero released data blocks), and SSDs, which could put unused blocks in its garbage collection. Some filesystem-aware storage devices already implement some of this (e.g., the FAT file system on USB sticks).

DM multi-path has a basic trust issue. Most folks don't trust it because the necessary investment wasn't made to make it trustworthy. This is a chicken-and-egg problem. Ric Wheeler said that EMC does certify DM configs.

Other complaints were poor performance, the lack of proper partitioning, the poor user interface for management tools, and the total lack of support for existing devices. (This is just an unverified list.)

Power of 2 chunk sizes don't work with HW RAID, which uses 4+1 or 8+1 disks.

Barriers today are only for cache flushing, to both to force data to media and to enforce ordering of requests. Ordered commands (which SCSI offers at a perf cost) might be helpful. But despite how painful flush barriers are, disks with WCE implemented perform better despite the flush barriers.

jejb suggested implementing commit on transaction.

### OSD-BASED PNFS

*Benny Halevy and Boaz Harrosh, Panasas*

Benny first described the role of the layout driver for OSD-based pNFS. Layouts are a catalog of devices, describing the byte range and attributes of that device. The main advantage of the layout driver is that one can dynamically determine the object storage policy. One suggestion was to store small files on RAID1 and large files on RAID5. Striping across devices is also possible. By caching the layouts (object storage server descriptions), one can defer cataloging all the OSD servers at boot time and implement on-demand access to those servers.

Current device implementations include iSCSI, iSER, and FC. SCSI over USB and FCoE are also possible. Functional testing has been done and performance was described as being able to "saturate a GigE link." Future work will include OSD 2.0 protocol development, and it's already clear there will be changes to the OSD protocol.

Requirements of the Linux kernel to support OSD pNFS were discussed. Bidirectional SCSI CDB support is in 2.6.25-rcX kernels. There are no objections to patches for variable length CDBs, which might go into 2.6.26. Recent patches to implement "Long Sense Buffers" were rejected; a better implementation is required.

The discussion ended on DM and ULD (Upper Level Driver; e.g., sd, tape, cd/dvd). DM contains the desired striping functionality, but it also takes ownership of the device. Distributed error handling is not possible unless the DM would pass errors back up to high layers. Each ULD is expected to register an OSD type. But the real question is whether we want to represent objects as block devices (segue to the next talk) and how to represent those in some name space.

### BLOCK-BASED PNFS

*Andy Adamson, University of Michigan; Jason Glasgow, EMC*

Afterward, PNFS was summarized to me as "clustered FS folks . . . trying to pull coherency into NFS." The underlying issue is that every clustered files system (e.g., Lustre) requires coherency of metadata across nodes of the cluster. NFS historically has bottlenecked on the NFS server since it was the only entity managing the metadata coherency.

The first part of this talk explained the Volume Topologies and how pNFS block devices are identified (fsid). Each fsid can represent arbitrarily complex volume topologies, which under DM get flattened to a set of DM targets. But they didn't want to lose access to the hierarchy of the underlying storage paths in order to do failover.

The proposal for "Failover to NFS" survived Benny's explanation of how a dirty page would be written out via block path and if that failed, then via NFS code path. The main steps for the first path would be write, commit, and logout commit and, for the failover path, write to MDS and commit. This provoked sharp criticism from Christoph Hellwig (and others): This is stupid. It adds complexity without significant benefit. The client has two paths that are completely different, and the corner cases will kill us.

The complexity he referred to was the unwinding of work after starting an I/O request down the block I/O code path and then restarting the I/O request down a completely different code path. A lively debate ensued around changes needed to Block I/O and VFS layers. Christoph was not the only person to object and this idea right now looks like a nonstarter.

The remaining issue covered block size: 4k is working but is not interoperable with other implementations.

### FS AND STORAGE LAYER SCALABILITY PROBLEMS

*Dave Chinner, SGI*

Dave offered random thoughts on 3- to 5-year challenges. The first comment was "Direct I/O is a solved problem and we are only working on micro-optimizations."

He resurrected and somewhat summarized previous discussion on exposing the geometry and status of devices. He wanted to see independent failure domains being made known to the FS and device mapper so that those could automate recovery. Load feedback could be used to avoid hot spots on media I/O paths. Similarly, failure domains and dynamic online growing could make use of loss-redundancy metrics to automate redistribution of data to match application or user intent.

Buffered I/O writeback (e.g., using pdflush) raised another batch of issues. It's very inefficient within a file system because the mix of metadata and data in the I/O stream causes both syncing and ordering problems. pdflush is also not NUMA aware and should use CPUsets (not Containers) to make pdflush NUMA aware. James Bottomley noted the I/O completion is on the wrong node as well (where the IRQ is handled). Finally, different FSs will use more or less CPU capacity and functionality such as checksumming data and aging FS might saturate a single CPU. He gave an example where the raw HW can do 8 GB/s, but only sees 1.5 GB/s throughput with the CPU 90% utilized. And like the FS, knowing the preferred alignment of the underlying storage would improve performance dramatically in those cases too. pdflush was summarized as "a bunch of heuristics" and no one objected to adding more.

Dave also revisited the topic of error handling with the assertion that given enough disks, errors are common,. He endorsed the use of the existing error injection tools, especially scsi_debug driver.

His last rant was on the IOPS (I/O per second) challenge SSDs present. He questioned that Linux drivers and HBAs are ready for 50k IOPS from a single spindle. Raw IOPS are limited by poor HBA design with excessive per transaction CPU overhead. HBA designers need to look at NICs. Using MSI-X direct interrupts intelligently would help but both SW and HW design to evolve.

I'd like to point folks to mmio_test (see gnumonks.org) so they can measure this for themselves. Disclaimer: I'm one of the contributors to mmio_test (along with Robert Olsson, Andi Kleen, and Harald Welte). Any MMIO (Memory Mapped I/O) reads in the driver performance path will prohibit the level of performance Dave Chinner described. And most SATA/SAS drivers have some MMIO reads. I estimate most commercial HBA designs are ~3 years behind NICs—e.g., RNIC (RDMA NIC)—which is not surprising given the cost models. There are some exceptions now (e.g., Marvell's 8-port 664x SAS/SATA chip) and probably others I'm not aware of.

Jörn Engel added that tasklets were added about 2 years ago which now do the equivalent of NAPI ("New API" for NIC drivers). NAPI was added about 5 or 6 years ago to prevent incoming NIC traffic from live-locking a system. All the CPU cycles could be consumed exclusively handling interrupts. This interrupt mitigation worked pretty well even if HW didn't support interrupt coalescing.

### T10 DIF

*Martin Petersen, Oracle*

Martin pointed to the FAST '08 paper "An Analysis of Data Corruption in the Storage Stack" by Bairavasundaram et al. (See the related article in this issue of *;login:*.)

His first point was that data can get corrupted in nearly every stage between host memory and the final storage media. The typical data-at-rest corruption (a.k.a. "grown media defects") is just one form of corruption. Remaining data corruption types are grouped as "while data is in flight" and applications need to implement the first level of protection here. He also characterized Oracle's HARD as the most extreme implementation and compared others to "bong hits from outer space." Given the volume of data being generated, there was agreement that the trivial CRCs would not be sufficient.

Although some vendors are pushing file systems with "logical block cryptographically strong checksumming" and similar techniques as bullet proof, they only detect the problems at read time. This could be months later, when the original data is long gone. The goal of the TDIF (T10 Data Integrity Feature) standard was to prevent bad data from being written to disk in the first place.

HW RAID controllers routinely reformat FC and SCSI drives to use 520-byte sectors to store additional data integrity/recovery bits on the drive. The goal of TDIF was to have end-to-end data integrity checks by standardizing and transmitting those extra 8 bytes from the application all the way down to the media. This could be validated at every stop on its way to media and provide end-to-end integrity checking of the data.

Slide 3 described what TDIF implements in the additional 8 bytes it adds to the normal 512-byte sector. Slide 4 explains which parts of the data path each of the competing standards covers and successive slides summarize more of the respective standards. Slide 16 neatly summarized the "Application/OS Challenges" of how a robust end-to-end protection could be achieved.

He pointed out which changes are needed in the SCSI; one of those (variable length CDBs) is already in the kernel. James Bottomley observed he could no longer get SCSI specs to implement new features like this one owing to recent changes in distribution. He also pointed out the FS developers could use some of the tag CRC bits to implement a reverse-lookup function they were interested in. The best comment, which closed the discussion, came from Boaz Harrosh: Integrity checks are great! They catch bugs during development!

### FCOE

*Robert Love and Christopher Leech*

Robert and Christopher took turns giving a description of the project, providing an update on current project status, and leading a discussion of issues they needed help with.

FCoE is succinctly described as "an encapsulation protocol to carry Fibre Channel frames over Ethernet" and standardized in T11. The main goal of this is to integrate existing FC SAN into a 10-GigE network and continue to use the existing FC SAN management tools. The discovery protocol is still under development. James Bottomley observed that VLAN would allow the FC protocol to pretend there is no other traffic on the Ethernet network since the on-wire protocol supports 802.1Q tags.

Slide 4 (of 20) nicely maps the FC-0 and FC-1 layers to IEEE 802.3 PHY and MAC layers, respectively. The key here is that the FC standard was originally intended to support both TCP/IP and FC protocol traffic over FC transport and the layering already existed in the FC standard to enable a simple addressing scheme (Fabric Id maps to MAC address). And following Martin Petersen's talk on TDIF, the presenters pointed out that the FC protocol implements its own end-to-end CRC.

Open-FCoE.org seems to be making good progress on several areas but it's not ready for production use yet. The current code has a functional initiator and an SW gateway. Tools for wireshark to decode the FCoE protocol are also upstream and working. But a rearchitecture is underway and new functional development on the current tree has temporarily been suspended. They are considering libraryizing the FCP support (e.g., libfc) instead of putting everything into scsi_transport_fc.

Current problems discussed included the complexity of the code, frustration with the (excessive) number of abstractions, and wanting to take advantage of current NIC offload capabilities. Current rework is taking direction from James Smart, making better use of existing Linux SCSI/FC code and then determining how much code could be shared with existing FC HBA drivers.

Discussion covered making use of a proposed "IT_Nexus" support. Robert and Christopher agreed that IT_Nexus would be useful for FCoE as well, since they had the same issues as others managing the connection state. James Bottomley also pointed out their current implementation didn't properly handle error states and got a commitment back that Robert would revisit that code. Use of sysfs versus ioctl to send command came around to "Why not use SG Passthru?" Also, the SCST was recommended as the target mode but that was debated since Target can be out of tree until it's stable. Current target support is STGT and implemented in user space. This resurrected an earlier discussion where iSCSI was told to "put everything in user space."

### LINUX STORAGE STACK PERFORMANCE

*Kristen Carlson Accardi and Mathew Wilcox, Intel*

Kristen and Mathew "willy" Wilcox provided forward-looking performance tools to address expected performance issues with the Linux storage stack when used with SSDs (see http://iou.parisc-linux.org/lsf2008/IO-latency-Kristen-Carlson-Accardi.pdf). This follows the "provide data and the problem will get solved" philosophy. Storage stacks are tuned for seek avoidance (waste of time for SSDs) and SSDs are still fairly expensive and uncommon. The underlying assumption is lack of SSDs in the hands of developers means the data won't get generated and no one will accept optimizations that help SSDs.

Kristen's first slides, providing a summary of SSD cost/performance numbers (e.g., Zeus and Mtron), showed that a single device is now capable of 50,000+ IOPS (I/O per second). Current rotational media can only do 150–250 IOPS per device on a random read workload (3000–4000 if it's only talking to the disk cache) and largely depend on I/O request merging (larger I/O sizes) to get better throughput. Ric Wheeler pointed out EMC's disk array can actually do much more but it requires racks of disks. Her point was that this level of performance will be in many laptops soon and it would be great if Linux could support that level of performance.

SCSI RAM and ATA_RAM drivers are the new kernel components that would allow developers to emulate SSD performance and focus on measuring performance of the protocol layers. Willy recently submitted those drivers to their respective mailing lists (http://marc.info/?l=linux-scsi&m=120331663227540&w=2 and http://www.ussg.iu.edu/hypermail/linux/kernel/0802.2/3695.html).

Concurrently, he developed iolat, which would provide a reasonable workload _and_ performance measurements (CPU utilization) conveniently rolled into one tool. Personally, I'm not happy about yet another benchmark and would rather just see fio (or some other simple benchmark) and a performance monitoring tool rolled into one script.

Criticisms of the iolat benchmark results were pointed out. The first was that the readprofile() data presented had the wrong symbols in its lookups. The second was that readprofile() uses timer ticks to sample and thus will never measure code called in an interrupt handler or held under a lock acquired with spinlock_irqsave(). In this case, only the latter issue is a problem since the RAM driver doesn't generate any interrupts like a normal I/O device would. I suggested using the IOAT (DMA engine for offloading user/kernel space copies) to emulate a real I/O device; this also churns the CPU cache and consumes memory bandwidth without burning CPU cycles.

Despite criticisms of iolat, the results (slide 12 of 22) comparing ATA_RAM, SCSI_RAM, RD (RAM disk), and normal disks are enlightening. Small, direct-random-read performance of SCSI_RAM and ATA_RAM is approximately 1/4 that of the RD driver. This is all due to latency in the protocol layer code path.

The outcome of this is that Willy started rewriting iolat to use oprofile instead. We need accurate profile data to determine where the CPU is spending time. Kristen also suggested libata stop using the SCSI midlayer and thus avoid the SCSI-to-ATA translation layer (which is costing another 10% in performance), but we need to be careful with HBA drivers that support both SAS and SATA.

## SYSFS REPRESENTATIONS

*Hannes Reinecke and Kay Sievers, SuSE*

*Summarized by James Bottomley (James.Bottomley@HansenPartnership.com)*

Hannes Reinecke and Kay Sievers led a discussion on sysfs in SCSI. They first observed that SCSI represents pure SCSI objects as devices with upper layer drivers (except sg: SCSI Generic) being SCSI bus drivers. However, everything else, driven by the transport classes, gets stored as class devices. Kay and Greg want to eliminate class devices from the tree, and the SCSI transport classes are the biggest obstacle to this. The next topic was object lifetime. Hannes pointed to the nasty race SCSI has so far been unable to solve where a nearly dead device gets readded to the system and can currently not be activated (because a dying device is in the way). Hannes proposed the resurrection patch set (bringing dead devices back to life). James Bottomley declared that he didn't like this and a heated discussion ensued during which it was agreed that perhaps simply removing the dying device from visibility and allowing multiple devices representing the same SCSI target into the device list (but only allowing one to be visible) might be the best way to manage this situation and the references that depend on the dying device.

Noncontroversial topics were reordering target creation at scan time to try to stem the tide of spurious events they generate and moving SCSI attributes to default attributes so that they would all get created at the correct time and solve a race today where the upward propagation of the device creation uevent races with the attribute creation and may result in the root device not being found if udev wins the race.

The session wound up with James demanding that Greg and Kay show exactly what the sysfs people have in store for SCSI, with the topics of multiple binding (necessary to allow sg to bind to an already bound driver and also to allow the transport classes to attach through the driver infrastructure) and elimination of the SCSI_device class in favor of the same information provided by /sys/bus/scsi. James was OK with this in principle but pointed out that we have a lot of tools in existence today that depend on things like the /sys/class/scsi_device so compatibility links would have to be provided.

## LIGHTNING TALKS

*Summarized by James Bottomley*

James Bottomley opened the Lightning Talks with the presentation of some slides from Doug Gilbert about standards changes. James'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.

## File System Track

*Summarized by Amit Gud (amitgud@vmware.com)*

Many thanks to Rik Farrow, the Program Chair, the Program Committee members, and all the speakers for helping collect the material required for this summary. Special thanks to Grant Grundler, my co-scribe, for all the help in collecting and hosting the material.

### 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.

Past attempts to solve this problems have been centered on passing the page to the lower layer, by either explicitly copying to the lower page and then releasing the lower page after the lower op is done or by reassigning the host mapping of the page to the lower inode. The first approach does reduce average memory pressure but not under stress, whereas the second approach is racy. Another past attempt includes implementing a Virtual Memory Area (VMA) operation fault, in which vma->vm_file would be assigned to the lower file before calling the lower vma->fault operation. Although this technique doesn't require address_space operation, it still is racy and needs the pointers to be fixed up when the calls across layers return.

One possible technique to address these problems is to tweak the page cache to be able to identify a page by context irrespective of which filesystem layer it belongs to. These content-addressable pages could save lot of physical memory. Ted Ts'o mentioned a VMware paper in OSDI about this area. In another proposed solution, a point was raised about whether different parts of the filesystem layer carry out different operations, but the consensus was to be away from inode operations.

*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.

There was also a suggestion of having a helper function to allow growing of the stack. XFS was deemed as another use case of having larger kernel stack.

*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.

*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 posed a question and a possible solution. How should we do whiteout removal when removing a logical empty directory? 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. The problem boils down to what we want the interface to look like. Al suggested that we should focus on new systems and forget about maintaining compatibility with statically linked binaries against old glibc versions. Therefore it would be enough if we teach the glibc about whiteouts and use getdents as before. Seeking on directories wouldn't be possible with union mount in the future.

That leaves the task open to push the whiteout handling to glibc and get it accepted by the upstream community.

### 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.

Dave_Chinner said that file systems can optimize the operations coming from NFS servers. Currently they are unable to do so. Dave Chinner suggests that an open flag, O_NFSD, could be added to allow the identification of the caller. Filesystem write calls by NFSD could be optimized by picking the right values for the minimum and maximum write sizes.

ENOSPC (Error: No Space) on a server is not handled gracefully yet, leading to data loss. The problem exists when many clients write to the server and the server runs out of space. Overcommitting because of client-side caching seems to be the problem. pNFS handles this by having an RPC-type operation, whereas NFSv4 also has equivalent operation, called getlayout(), although it is optional. Dave suggested using reservation pools on the NFS server, which could be a guaranteed disk space per client marked by the file system as used.

FIEMAP, suggested by Andreas Dilger, provides a means of efficiently discovering offline blocks.

Premount notifiers are required in order to purge the cache when unexporting the shares. Since the export table keeps references of dentries and vfsmnt, unmounting is not possible because of open file references.

### 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. Also, there are similarities between implementation of launder_page and writepage, except for the release of the page lock in writepage, that write out buffers prior to direct I/O (DIO). Steven suggests that the lock ordering problems with writepage can be avoided with writepages.

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.

To an inquiry about the splice system call, Christoph said that it is better to share more infrastructure for common functions for file systems than to duplicate code; file systems can then implement their own splice functionality.

### 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.

Tom discussed the idea of having an spNFS pseudo file system with a layout driver hook allowing parsing layout requests by probing lower file systems or using some other technique. Adding layout operations to the filesystem export API or to the VFS switch would mean changes to all or all eligible file systems. Another option of having a layout library API did not seem convincing given the added complexity. Questions were raised as to whether layout detection would be further complicated by the copy-on-write (COW) file systems. Tom pointed out that if the layout changes, clients need to be informed, and there needs to be a way for file systems to do it like pNFS, which issues a local callback. Also, it was suggested that the layout detection need not be in the form of a layer, but laid on the side like GPFS. It was pointed out that there exists an ioctl() call to get block mapping information; it works on block basis and needs to be called for every 4k.

Questions were raised about whether the layout management should be in user space or in kernel space. Matt Mackall suggested that for block-based layouts, it needs to be in the kernel for consistency issues, but for file-based layouts it can be moved to user space as consistency can be taken care of. Also it was suggested that if the support for any file system is desired, it is better to be in the kernel.

### 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 disk.

Answering a question regarding whether the worst case, scanning an entire file system, still exists, Zach explained that back pointers are the key to avoid entire filesystem scans. Anything that has a link has a back link. Ted Ts'o suggests that this can be an area of research of locally repairing the localized damage or repairing around it. In response to Erez Zadok's question about snapshot granularity and speed, Zach replied that the entire metadata tree is the granularity of the snapshot, but the actual I/O required is very small. Taking a snapshot at the file or directory level is difficult because of reference counting. Andreas Dilger suggested that directory-level snapshots could be created by taking a directory and creating a volume out of it, but Matt thought too much administrator overhead would be involved in this and hence and it would not be practical. Also, hard links are a problem, Chris Mason pointed out.

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. Ted questioned how BTRFS avoids duplicate inode numbers. Chris explained that it works like the extent allocation: You look through the tree and find a key not there.

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.

In a discussion about how fsck can be improved for BTRFS, someone asked whether a snapshot can be taken and fsck'd. Zach Brown explained that part of what BTRFS needs to fsck is the bookkeeping structures themselves. Ric Wheeler stated that 20% of errors are due to bad software and 80% are due to hardware, the majority of which are localized; therefore with reverse mapping it could be possible to detect bad sectors and repair the file system online.

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.

Chris mentioned, in responding to Ted Ts'o's question, that BTRFS does not retain any information about deleted files.

Zach mentioned that checksums are performed on blocks of size 4k. Ric finds this useful for validating the objects stored by the user. Christoph suggested using extended attributes for storing the checksums.

Jörn Engel suggested spinning down all the devices except the one being used for writing and using exec in place at the device level.

### 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. Erez Zadok suggested that dd-ing an old file system is an inexpensive way of creating an aged 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. Ted T'so asked whether the underlying log device could be rate limited; Spencer explained that traces don't have timestamps and flows could be rate limited to time.

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. Erez mentioned similar research at CMU and SUNY Stony Brook; the major problem with SNIA gathering traces is that they are unable to decide on the standards of collecting the traces and lack of manpower. Ted suggested that this could be done as a part of the Google Summer of Code.

Answering Jan Kara's question, Spencer explained that Filebench currently deals with local file systems but dealing with multiple nodes is planned in the future.

Filebench works with filesets, which is a fractal tree of files having depth and size. Filebench has a predefined set of typical workloads.

In detailing Filebench updates and features, Spencer explained that Filebench has been around for the past 3 years, but the past 18 months have mainly been spent in cleaning up the code and solving some problems. 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 is an issue since Filebench today is CPU and memory intensive. Work in progress also involves NFS and CIFS plugins, in which a dynamic library providing flow operations is under consideration. Erez stressed that the combination of client and server matters: performance numbers could vary drastically if either of them is changed.

### 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 (e.g., to minimize cacheline bouncing and to make sure that the optimizations doesn't cause regressions).

Dave suggested having an infrastructure in place to expose the underlying geometry and storage topology dynamically. Currently, there is no kernel infrastructure, although a few ioctls and dmsetup tools in user space could be used.

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.

Dave said that pdflush becomes CPU intensive as the file system is aged. Parallelizing the pdflush daemon could be a good option. There can be more than one pdflush thread per backing device, but this is tricky since only one thing can operate on the device.

It was suggested that pdflush talk to the backing device with the unit of the stripe; if the stripe is of 2 Mb, it can writeback 2-Mb-sized pages. It is desired that pdflush writes the stripe in full and not just part of it. He reminded us that pdflush does not work because of memory pressure; its primary goal is to get the dirty pages out as efficiently as possible. Dave's suggestions entail adding more heuristics from the file system and block layers to pdflush.

A disconnect in the error paths can cause file systems to hang with a cause not known to the file system. Dave stated that it is desirable that the errors (from, e.g., devices disappearing and path failovers) are passed on to the file system from the layers below it.

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 Lever discussed the current state of NFS IPv6 support and the future timeline. IPv6 has been around for a long time now and it is not reasonable to expect a significant transition period. The U.S. federal government requires that all software released this fiscal year and next have IPv6 support and the Department of Defense and NIST requirements on this have yet to be given out.

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.

Someone asked whether pNFS supports IPv6 and the answer was affirmative, since pNFS is addressing-mechanism agnostic.

Chuck called for NFS IPv6 testers and developers to help with the implementation.

### 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.

Tom mentioned a bug in RPC RDMA that occurs when doing large metadata operations and occurs only on PPC. 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. Tom recalled an interesting case with DDR adapters. He said that the client performance is above anything NFS has ever achieved with data copy avoidance, which normally takes 100% CPU, but the RDMA number is down to under 20% with the same bandwidth.

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.

Ted asked whether testing RDMA could be any easier. Tom pointed out a Broadcom device that currently is not supported on Linux, but is on Windows. Broadcom hasn't yet revealed the hardware details.

### ISSUES WITH DATA-ORDERED AND BUFFER HEADS

*Mark Fasheh, Oracle*

Mark addressed problems with buffer heads in data-ordered mode and solutions for buffer-head-less ordered-data mode.

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. He 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. The code especially gets complicated for OCFS2 with blocksize, clustersize, and pagesize trifecta.

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. Chris Mason questioned whether, if data is put in an ordered list, overwriting the middle of the file causes the bufferheads to be ordered. Mark said it does, and OCFS2 used to do this until last year when he realized that existing data is never rewritten. Although this is slower for users, Jan added, it is true only for overwriting of a file.

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. Lease can either be read or write lease. Currently, the leases are local, and the goal is to make the lease work over a cluster.

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.

The discussion then turned to grace periods. These can be granted by the server to the client when the lease needs to be reclaimed. When the grace period is over, regular locking is allowed to continue. Felix pointed out that there also exist issues with leases on local file systems for hard links.