

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.

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 has 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 was a hot topic in Networked Block Storage forums.

■ *Introduction and Opening Statements: Recap of Last Year*

Chris Mason and James Bottomley

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 2⁶⁴ 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, 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

LSF '08: 2008 Linux Storage & Filesystem Workshop

*San Jose, CA
February 25–26, 2008*

STORAGE TRACK

*Summarized by Grant Grundler (grundler@google.com)
Copyright 2008 Google, Inc. (Creative Commons Attribution License, <http://code.google.com/policies.html> or <http://creativecommons.org/licenses/by/2.5/>)*

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

last year, has receded in importance because manufacturers are hiding the problem in firmware.

■ **SSD**

Dongjun Shin, Samsung Electronics

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 FSes tested. Some notable issues were that flush barriers kill XFS performance and that BTRFS performance was better with 4k blocks than with 16k blocks.

Flush barriers are the only block I/O barriers defined today, and the flush barriers killed 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.

■ **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, noting that the situation here was improving with data checksumming now being built into file systems (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 file system might still not release the inodes. Ted Ts’o closed the debate by drawing attention to the paper by Gunawi et al. 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. 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.

Ric Wheeler (EMC) opened 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.

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.

■ **Block IO Resources and cgroups**

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 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 speaker acknowledged that. A more complex scheme was proposed that supports hierarchical assignment of resource control (e.g., CPU, memory, I/O priorities). Proposed was `page_cgroup` to track write bandwidth. 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.

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. 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.

■ **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 choose 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 affect the behavior of block schedulers to coalesce I/O and anticipate which I/Os to issue next. And although NCQ was effective on a best-case benchmark, it was debated how effective it would be in real life (perhaps <5%).

■ **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.

They tested a few different configs: RAID 0 sequential, async; single-disk random and sequential; and IO-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. For RAID topology, the key info was “stripe boundaries.”

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

■ **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 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 memcopy 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 memcopy. 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. 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.

The next issue discussed was IOMMU performance and I/O TLB flushing. The IOMMU driver (and HW) performance are 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 included the following:

1. How does one best manage IOMMU address space? Through common code? Some IOMMU drivers use bit-map (most RISC); Intel uses a "Red Black" tree. Fujita tried converting POWER to use Red/Black tree and lost 20% performance with netperf. Bottomley and Grundler 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. The bottom line is that there is a tradeoff between performance and safety (a.k.a. robustness).

3. Should we just map everything once? 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 (SCSI Generic) chaining versus SG rings.

■ **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 iSCSI 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` hierarchy 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. The lively but inconclusive debate left me thinking that most of the code will be 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.

■ Request-based Multi-pathing

Kiyoshi Ueda and Jun'ichi Nomura, NEC

The key point was proposed multi-path support below the I/O scheduler; 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 (Device Mapper) 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. The first issue was that `blk_end_request()` will deadlock because the queue lock is held through the completion process. Bottomley 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()`. The second issue involved busy stack drivers that 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. Bottomley suggested prep and unprep functions to indicate whether requests are mergeable or not. 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. The third issue was 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.

■ 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 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. 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.

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 com-

plaints were poor performance, the lack of proper partitioning, the poor user interface for management tools, and the total lack of support for existing devices.

Barriers today are only for cache flushing, both to force data to media and to enforce ordering of requests. Bottomley 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 namespace.

■ 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 filesystem (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: this 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 that the I/O completion is on the wrong node as well (where the IRQ is handled). Finally, different FSes 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. 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). Jörn Engel added that about 2 years ago tasklets were added 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.

He pointed out which changes are needed in the SCSI; one of those (variable-length CDBs) is already in the kernel. James Bottomley observed that he could no longer get SCSI specs to implement new features like this one, owing to recent changes in distribution. He also pointed out that 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.

Open-FCoE.org seems to be making good progress on several areas but it’s not ready for production use yet. 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 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 that their current implementation didn’t properly handle error states; he got a commitment back that Robert would revisit that code.

■ 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://www.usenix.org/events/lsf08/tech/Carlson_Accardi_powermgmt.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 that 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 that EMC’s disk array can actually do much more, but it requires racks of disks. Kristen’s 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.

■ 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 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 re-added 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. 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 Bottomley demanding that Greg and Kay show exactly what the sysfs people have in store for SCSI.

For the complete summaries, see <http://www.usenix.org/events/lsf08/lsf08sums.pdf>.