Improving Block-level Efficiency with \textit{scsi-mq}

Blake Caldwell
Oak Ridge National Laboratory
Oak Ridge, TN
Email: blakec@ornl.gov

\textbf{Abstract}—Current generation solid-state storage devices are exposing new bottlenecks in the SCSI and block layers of the Linux kernel, where IO throughput is limited by lock contention, inefficient interrupt handling, and poor memory locality. To address these limitations, the Linux kernel block layer underwent a major rewrite with the blk-mq project to move from a single request queue to a multi-queue model. The Linux SCSI subsystem rework to make use of this new model, known as \textit{scsi-mq}, has been merged into the Linux kernel and work is underway for \textit{dm-multipath} support in the upcoming Linux 4.0 kernel. These pieces were necessary to make use of the multi-queue block layer in a Lustre parallel filesystem with high availability requirements. We undertook adding support of the 3.18 kernel to Lustre with \textit{scsi-mq} and \textit{dm-multipath} patches to evaluate the potential of these efficiency improvements. In this paper we evaluate the block-level performance of \textit{scsi-mq} with backing storage hardware representative of a HPC-targeted Lustre filesystem. Our findings show that SCSI write request latency is reduced by as much as 13.6\%. Additionally, when profiling the CPU usage of our prototype Lustre filesystem, we found that CPU idle time increased by a factor of 7 with Linux 3.18 and \textit{blk-mq} as compared to a standard 2.6.32 Linux kernel. Our findings demonstrate increased efficiency of the multi-queue block layer even with disk-based caching storage arrays used in existing parallel filesystems.

I. INTRODUCTION

The Lustre [1] parallel filesystem achieves high aggregate IO throughput for parallel applications by splitting IO requests across many backing storage devices, known as Object Storage Targets (OST’s). Once file layout has been set and metadata operations have been handled by the Metadata Server (MDS), clients may send IO requests for data blocks directly to the OST. An OST handling IO requests from a large number of clients often becomes IO throughput bound due to the block storage device that backs the OST. For assuring data integrity and reliability, these block devices are typically RAID volumes with hardware-caching capabilities that improve upon the latency qualities of the underlying magnetic hard disks.

Even though the filesystem IO latency can be amortized by splitting the IO requests among many OST’s, the request latency of the slowest OST will still be the determinant of overall completion time.

Hardware caches in RAID storage arrays improve both the latency and throughput within the bounds of the available cache size such that IOP rates can rival, and may even exceed the capabilities of SSD devices. The storage array used in our evaluation is capable of more than 300,000 IOP/s per RAID device when performing sequential writes. While the handling capacity of the Linux block subsystem is limited to somewhere between 400,000 [2] and 800,000 IOP/s [3], a penalty is paid as IOP/s near the upper bound of that range in terms of increased lock contention, cache invalidations, and interrupt servicing. However, when a multi-queue block layer is employed, as in [2], Linux itself can scale to over 10 million IOP/s.

Previous performance evaluations of \textit{blk-mq} have been done using the \textit{null-blk} device driver [2], [3], which simply acknowledges the IO operation, without writing to the data to any storage device [3]. We build upon these evaluations, but take a different angle of evaluating caching storage arrays such as those providing OST’s for production Lustre filesystems. To demonstrate the applicability of block-level throughput and latency improvements, we have built a Lustre filesystem on the 3.18 Linux kernel patched with \textit{scsi-mq} support.

The remainder of this paper is structured as follows. In Section II we describe the Linux block subsystem and block-layer enhancements. Section III describes our prototype filesystem and experimental setup. In Section IV we present the results from our block-level throughput and latency tests and kernel code profiling with \textit{scsi-mq}. Then Section V discusses our findings on block-layer improvements in the context of a Lustre filesystem. We present related works in Section VI and we conclude with Section VII which summarizes our findings and areas for future work.

II. BACKGROUND

A. Legacy block layer

Even prior to the \textit{blk-mq} update, the Linux block layer was adequate for magnetic hard disks with slow seek latencies, where IOP rates weren’t constrained by lock contention or interrupt handling. Rather than optimizing the block layer for
minimum latency, it was often even advantageous to leave the queue in a “plugged” state where requests sat in the queue waiting to be coalesced with other requests into a single sequential IO. With rotational drives, the delay introduced by request merging and reordering is far outweighed by the reduction in seek latencies by making IO sequential.

However, with SSDs capable of servicing millions of IO operations per second [4], the tradeoffs made by the existing Linux block layer needed to be re-evaluated. The designers of blk-mq detailed three major scalability concerns in the legacy Linux block layer in [3]. Firstly, insertions, removals and IO scheduling operations involving the request queue require obtaining a global lock, where cores competing for the same lock will block waiting for the operation to complete. Secondly, beyond just one hardware interrupt per IO completion, the exact core handling the IO request may still need to be signaled by an inter-processor interrupt (IPI). Thirdly, cache coherency must be maintained. Multiple cores must contend for the per block device request queue lock. Each transfer of ownership of the lock between cores will cause cache line invalidations and cache coherency traffic across the memory bus. If the lock remains in memory on within a different NUMA locality region, then acquiring the lock will involve a remote memory access.

Because of these effects, Bjørling et al. find that the Linux block layer is limited to 800,000 IOP/s, and cannot benefit from the use of additional cores [3].

B. Multi-queue enhancements

To address the three concerns outlined in [3] and restated above, the block layer was restructured in the Linux 3.13 kernel with two levels of queues. The blk-mq design employs per-core software submission queues, where an IO request is enqueued on the same core as the application submitting the IO. The queue obeys simple FIFO semantics, removing the need for locking of the queue for each insertion or removal. Additionally, accessing the queue structure is done entirely within the memory locality region of the core submitting the request. A second level of queues has the responsibility of dispatching IO’s to the underlying storage device driver. These queues are usually equal in number to the degree of parallelism supported by the device. For example, SSD drives can exploit the parallelism of independent NAND flash chips. Some PCIe SSD devices have up to 32 channels per controller [5] while SATA-attached SSD’s have up to 10 channels [6]. For legacy block device drivers such as those for rotational hard disks, the number of hardware dispatch queues defaults to one.

While the legacy block layer takes advantage of time an IO request remains in the submission queue to sequentially reorder requests, the blk-mq design simply interleaves IO requests from the per-core software submission queues onto the hardware dispatch queues. With solid state devices, capable of higher random IO performance, the benefit from sequential ordering is lessened and the FIFO queues avoid locking overheads that become more pronounced at high IOP/s rates. This trade-off may be troublesome for rotational media, but as noted in [3], there is room for IO scheduling to be done between the two queue levels. To date, this scheduling has not been implemented.

Another enhancement added by blk-mq, is support for tagged completions. The block layer associates a tag for each IO, where the tag corresponds to the position in the hardware dispatch queue. The tag is then preserved in the device driver code and can be used again by the block layer to identify the IO upon completion. This increases the speed at which the IO completion can be identified and then passed on for handling by the core that was responsible for its submission.

The multi-queue block layer design is shown on the left in Figure 1 where ost_io threads on each core submit IO’s through the Linux kernel page cache to a dedicated submission queue. These submission queues map to an equal number of dispatch queues supported by the SRP initiator device driver, with multiple RDMA channels between the OSS and the SRP target implemented in the storage array. This design contrasts to the right half of Figure 1, with a single queue per block device, where ost_io threads on different cores are competing for the same lock.

The performance reported with scsi-mq has ranged from 1 million IOP/s [7] using an array of 16 SAS SSD’s to 12 million using the null-blk device driver which just acknowledges the request and discards the data [3].

C. SCSI subsystem work

To make use of the multi-queue block layer with SCSI devices, the SCSI IO submission path had to be updated. The scsi-mq work was functional in the 3.17 kernel and lower level device drivers are being updated to support multiple hardware dispatch queues. Of interest to this study, is the SRP initiator driver ib-srp which was updated in the Linux 3.19 kernel (and in our patched 3.18 kernel) to support scsi-mq and multiple RDMA channels [2].
Other work is underway on the `dm-multipath` driver to support `scsi-mq`. This feature is highly desirable for Lustre deployments and the `dm-multipath` maintainers are targeting inclusion of an updated multi-queue compatible driver in the upcoming Linux 4.0 kernel release.

### D. Lustre IO patterns

A common measure of the performance of parallel filesystems in HPC is the aggregate IO handling capacity, and to reach near 1 TB/s, parallelism is used at multiple levels. Lustre stripes objects across multiple block storage devices and each block device is backed by a RAID volume that stripes its data across multiple hard disks. Large IO requests can benefit from this parallelism, roughly by a factor equal to the stripe count. However each operation incurs a certain amount of overhead associated with the metadata tracking, ordering, and consistency checking, where small IO requests are disproportionately affected by individual request latency. Lustre metadata operations, such as a `stat` typically involve IO requests 4 KB in size and bulk IO operations to the OST’s are ideally done in 1 MB segments.

It is not immediately apparent how the large 1 MB IO requests to Lustre OST’s from would benefit from either increased IOP/s capacity or decreased request latency. In fact prior studies of multi-queue block-layer performance have used 512-byte sized IO’s to the reach high the IOP/s rates where `blk-mq` shines. With 1 MB IO’s, an IOP/s rate of just 1024 is required to reach 1GB/s of throughput. The streaming throughput capacity of the RAID volume will become the bottleneck before the legacy Linux block layer in this case. A reduction in request latency is unlikely to be noticeable with 1 MB object storage requests because a data transfer rate of 1 GB/s means that latency will be roughly 1 ms, compared to the few µs difference in IO handling latency.

However, for 4KB IO operations, IOP/s rates begin to approach what the Linux block layer can handle. The vendor of the storage hardware used in this study claim that the array we used is capable of 840,000 IOP/s out of the cache. While we were only able to attain 500,000 IOP/s at this block size, such a rate is still approaching the limitations of the Linux block layer. Additionally, request latencies of these small IO’s becomes important for the interactivity of the filesystem to the end-user.

### III. Evaluation

Our prototype filesystem for evaluating the performance impact of `scsi-mq` was built using hardware similar to known production filesystems. The testbed was built around a DataDirect Networks SFA 10K-X storage appliance housing 600 2TB SATA disks. The disks are organized in 60 RAID-6 pools of 10 disks each. The pools are each presented as a block device via SCSI RDMA Protocol (SRP) over InfiniBand to four Lustre object storage servers (OSS). Typically four OSS are needed to meet the bandwidth capacity of the storage array, but only one OSS was used for this evaluation. We tested cases of multiple block devices, or Logical Unit Numbers (LUNs), attached to a single host, but not the case of using multiple OSS.

Each OSS has two Intel E5-2650 Ivy Bridge processors, 64GB RAM, and 2 QDR InfiniBand links for the SRP transport to the storage array. Using `dm-multipath`, and the dual InfiniBand links between the OSS and the storage array, highly available block devices could be used, which would survive the failure of one of the two storage controllers in the storage array. Since the `dm-multipath` drivers add an abstraction layer above the block-device, which could negatively affect the performance, we set out to quantify the performance penalty, and then use the raw block device to for the remainder of the block-level tests. The patches to enable `dm-multipath` to work with `blk-mq` were added to the 3.18 kernel used in this evaluation. These patches are proposed for inclusion in the Linux 4.0 kernel.

Since we are studying the performance capabilities of the Linux block device layer, we turned on write-back caching on the storage array. We acknowledge that the sustained IOP rates of each rotational drive-backed RAID volume would be significantly lower and thus become the bottleneck in tests that exhausted the array’s cache capacity of 8 GB mirrored between the controllers. The storage array has an additional cache setting called ReACT, which disables the write-back caching for aligned full-stripe writes. We left this feature enabled, as is recommended by the vendor for use with production filesystems, but it was not impactful to the tests run in this evaluation using 4 KB IO’s since the array’s configured stripe width is 1MB.

In keeping with the goal of evaluating `scsi-mq` performance in an environment representative of current Lustre filesystem deployments, different software stacks were deployed on the OSS in using a diskless provisioning technique. The complete configuration and OS install are captured in system image that differs only where necessary between the software stacks evaluated. Three images were constructed: one with Red Hat Linux 6.6, and kernel version 2-6.32-431, and the other two with Red Hat Linux 7.0 with kernel 3.18. The only difference between the two Red Hat 7.0 images is the setting of the kernel configuration parameter `CONFIG_SCSI_MQ_DEFAULT`. With this setting enabled, in addition to per-core software queues, the SRP lower-level SCSI device driver made use of multiple hardware dispatch queues and equal to the number of SRP channels configured. Disabling this parameter meant that a single hardware dispatch queue would be used, but with multiple per-core software queues. These scenarios with the 3.18 kernel are referred to as `mq+mc` and `mq` respectively in Section IV.

The SRP transport protocol protocol involves drivers on both the OSS (SRP initiator) and storage array (SRP target). The SRP target driver has supported multi-channel operation previously, but recent patches to the SRP initiator driver `ib_srp` from Bart Van Assche added options to configure the number of channels and set the completion request affinity. In this study, we tried two different channel configurations. First, with 16 channels, one per core, and then two channels, with one per
processor socket. We did not observe a significant difference in performance when using 16 compared to two channels, so for the results presented below, we used 16 channels for comparison to other studies that used one channel per core [9].

At the time of completion of this study, support for the 3.18 kernel was not yet part of the Lustre code. Since it was a primary objective of this study to determine what performance benefits scsi-mq could bring to Lustre, we made the necessary changes to the Lustre code to be compatible with the patched 3.18 kernel which included scsi-mq support. One of the changes required was rebasing patches from the upstream Lustre client driver to the master branch of the Intel Lustre tree. Many production Lustre filesystems use a special filesystem format called ldiskfs for the object and metadata target devices. Unlike other backing filesystems such as zfs, which are less-commonly used with Lustre, especially for metadata targets, ldiskfs is a series of patches on top of the ext4 filesystem. These patches were rebased for the ext4 code in the Linux 3.18 kernel for this study. We have made all of our patches to Lustre for the 3.18 kernel publicly available on GitHub [11].

As with other works involving storage performance on NUMA architectures [4], [12], we had to pay careful attention to processor and interrupt affinity in setting up the experiments. The InfiniBand card connecting the OSS to the storage array has affinity to NUMA node 1, so we directed all interrupts from the Infiniband device to the cores on the single processor socket which makes up NUMA node 1. Also numaclt was used to bind the workload generation processes to the cores on node 1.

The workload in each test was generated with fio [13] using the libaio IO engine. Parameters for each of the tests were a runtime of 60 seconds, iodepth of 127, and IO’s batched in units of 62. The average of tree trials for each test is presented in the figures in Section IV. For the SRP initiator parameters, we set max_cmd_per_lun to 62, and queue_size to 127. We ran all block-level tests with a workload of 4 KB sequential writes, representative of a specific use case in a Lustre filesystem, metadata IO. Writes were chosen instead of reads for an initial in-depth study, to avoid the need to account for the effects of a read-ahead prefetch behavior. An evaluation of read performance is a subject we aim to include in future work. We only studied sequential performance with the rotational disk-backed storage array, because performance with random IO’s involving large seek times drops to a level such that the bottleneck was never the Linux block layer. While this would be a useful measure or performance on a filesystem backed by flash storage media, capable of high random IOP/s rates, it is not applicable to our study with rotational media.

IV. RESULTS

As a first point of comparison, Figure 2 shows the throughput effects by increasing the number of fio threads submitting IO’s to a single block device. The Linux 3.18 stack with multiple per-core submission queues has an edge over the 2.6.32-431 kernel with a single submission queue from 1 to 4 threads per LUN. However, the values converge at 8 threads per LUN. We expect that lock contention is greatest in the 8 thread per LUN case, which should have benefited the 3.18 mq stack, but it does not appear to be the bottleneck since the legacy block layer is able to equal IOP/s at just over 500,000. As is the case throughout our evaluations, the 3.18 kernel with multiple SRP channels fails to show any benefit from increased parallelism. We did not have the opportunity as part of this study to determine what is preventing the mq+mc stack from scaling, but this is an area we intend to study in future work.

An unexpected finding in our tests was that the performance of multiple threads per LUN exceeded that of writing to multiple LUNs. In Figure 3 the throughput of both the legacy block layer and 3.18 mq tops out near 300,000 IOP/s. This hints that a bottleneck exists somewhere outside the factors addressed by blk-mq. The performance of multiple threads writing to a single block device in Figure 2 exceeds the 300,000 IOP/s mark. We would need to perform additional analysis to determine whether that is a result of cache optimizations, such as request reordering either the block level or within the storage array hardware.

As discussed in Section II-D reduced request latencies for 4 KB IO’s could have a positive impact on filesystem interactivity. We list the average latency of 100 timings of 4 KB requests with the pre-scsi-mq kernel and a Linux 4.0 release candidate with the dm-multipath patches for scsi-mq in Table I. A significant 13.6% improvement in 4 KB write latency is seen with the scsi-mq kernel. The request latency with dm-multipath remained the same between the two software stacks, however.
TABLE I

| Kernel     | dm-multipath | raw block |
|------------|---------------|-----------|
| 2.6.32-431 | 130.50        | 119.92    |
| 4.0rc1     | 130.56        | 103.58    |

In addition to latency and throughput performance metrics, we looked for signs of increased efficiency at the filesystem-level. Using the `perf` tool, kernel stack traces were collected at 100 Hz, and compiled into FlameGraphs [14] shown in Figure 4 with `scsi-mq` and the legacy block layer. A striking difference is seen in the relative amount of cycles spent idle vs. busy in Lustre `ll_ost_io` threads. With the 2.6 Linux kernel, 7.72% of cycles are spent idle, while 55.06% of sampled cycles are idle cycles with the 3.18 kernel. The increase in idle cycles can be attributed in large part to decreased time spent in `ll_ost_io` threads while handling IO requests and their completion.

V. Discussion

The workloads evaluated in this study are focused on the use case of Lustre metadata operations, which have an IO size of 4 KB. These small IO requests have higher sensitivity to changes in latency, and thus the benefit from a 13.6% reduction in block-level request latency could benefit a real filesystem-level workload. One such benefiting workload would be a recursive directory listing that gathers the total size of a file's contents underneath a directory. There is a data dependency between reading the contents of a directory entry, and subsequent reads of each inode part of the directory entry. These data dependencies force a synchronous order of metadata operations where the completion latency of each one directly impacts the overall delay a user would experience. Filesystem scans that walk the whole directory structure also encounter a data dependency in metadata operations, and could see a benefit from reduced 4 KB request latency.

Metadata throughput of a single block device in Lustre has not been measured at a rate near the IOP capacity of the single request queue block layer. An OpenSFS-funded project [15] measured up to 200,000 IOP/s during concurrent `stat` operations by 16 clients, but improvements in subsequent Lustre versions could have pushed the IOP/s rate close to or above the handling capacity of a single block device without `scsi-mq`. An example of an event that would trigger high IOP/s rates to the block device would be cache write-back events on the MDT. A large stream of cache write-backs would occur at less interesting times such as when unmounting a filesystem, but perhaps there a more comprehensive study of metadata target block IO might reveal other times. Another time the MDT might become IOP-limited is during filesystem recovery when the journal is being played back and committed to the filesystem.

A different scenario where the multi-queue block layer design could be beneficial is when multiple IO’s are being issued to the same block device, but one is more latency sensitive that the rest. In fact, a benefit mentioned in [3] of `blk-mq` was that 2-level queuing allows for IO scheduling, and potentially a quality of service mechanism to intercede as IO’s from the per-cpu software queues are inserted into the hardware dispatch queues. Reserving queue resources for smaller or more-latency sensitive IO operations could lead to predictable latency variation during times of high load. Furthermore, the ability of `scsi-mq` to preserve IO request tags used in the block layer into the SCSI device driver level, and on completion of the IO adds an accounting mechanism for the timing of individual IO requests through completion.

VI. Related Work

The `blk-mq` design was detailed by Bjørling et al. in [3], and then merged in Linux 3.13 kernel. In order to add support of this new IO path to SCSI subsystem, the `scsi-mq` work was merged in the 3.17 kernel [7]. Each of the works [3, 7], contributed performance evaluations, with the former utilizing the `null-blk` device driver for measuring pure software overheads and the latter using an SSD-back RAID array. The results in [7] have applicability to some real Lustre filesystem deployments, but many of the highest performing Lustre filesystems make use of a RAID array and the SRP transport which Bart Van Assche added multi-queue support to in Linux 3.19 [2]. Van Assche also measured the performance of `scsi-mq` before the 3.17 merge and published results using the `null-blk` driver in [9], [16], and detailed SRP multi-queue status in [2], [17].

While previous evaluations examined the software efficiency of `blk-mq`, our work examines the impact on performance with a caching storage array and representative parallel filesystem IO patterns. This is, to our knowledge, the first evaluation of parallel filesystem making use of the multi-queue Linux block layer.

VII. Conclusion

In this paper we have detailed our experimental evaluation of a prototype Lustre filesystem using the redesigned multi-queue block layer in a patched Linux 3.18 kernel. We have made our Lustre code modifications for the Linux 3.18 kernel with `scsi-mq` publicly available on GitHub [11]. Our block-level tests revealed a 13.6% improvement in write request latency over the standard block layer in the Linux 2.6.32 kernel and that idle CPU cycles increased by a factor of 7.
Within the context of a Lustre filesystem, we explored benefits that these improvements could bring to overall filesystem performance. While these results are promising, this study has also highlighted areas in need of further investigation.

As a subject of study in future work, we intend to understand the reasons behind the poor scaling performance of multi-channel support in the SRP driver. We found performance improvements with the use of per-core software submission queues, but utilizing multiple hardware queues actually degraded performance. This is contrary to what we expected given the performance benefits of a multi-channel SRP driver in [9]. Additional channels should expose additional parallelism in the storage array hardware and avoid contention for shared queues. We focused on write performance in this paper, but an in-depth investigation of read performance is also necessary to understand the full impact of the redesigned block layer.

We have narrowed our target for filesystem improvements from scsi-mq to metadata operations where IOP rates near the capacity of the existing Linux block layer, and requests are sensitive to latency improvements. The caching storage arrays with rotational media used in this study were capable of IOP rates near the cited limits of the single request queue block layer, but we were unable to verify the observed IO throughput was limited by inefficiencies in the software stack. Performing evaluations on solid-state media capable of higher IOP/s rates and high random throughput could highlight these limitations in future work.

ACKNOWLEDGMENT

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] “Lustre File System,” [http://opensfs.org/lustre/](http://opensfs.org/lustre/)
[2] B. Van Asche, “Increasing SCSI LLD driver performance by using the SCSI multiqueue approach,” [http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf](http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf), Boston, MA, Mar. 2015.
[3] M. Bjartling, J. Axboe, D. Nellans, and P. Bonnet, “Linux block IO: Introducing multi-queue SSD access on multi-core systems,” in *Proceedings of the 6th International Systems and Storage Conference*, ser. SYSTOR ’13. New York, NY, USA: ACM, 2013, pp. 22:1–22:10.
[4] D. Zheng, R. Burns, and A. Szalay, “Toward millions of file system IOPS on low-cost, commodity hardware,” in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2013 International Conference for, Nov 2013, pp. 1–12.
[5] M. Cornwell, “Anatomy of a solid-state drive,” *Queue*, vol. 10, no. 10, pp. 30:30–30:36, Oct. 2012.
[6] C. Intel, “Intel X25-M and X18-M Mainstream SATA Solid-State Drives,” [http://download.intel.com/newsroom/kits/ssd/pdfs/X25-M_34nm_ProductBrief.pdf](http://download.intel.com/newsroom/kits/ssd/pdfs/X25-M_34nm_ProductBrief.pdf)
[7] C. Hellwig, “High performance storage with blk-mq,” [http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf](http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf), Dusseldorf, Germany, Oct. 2014.
[8] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, S. Gupta, D. Tiwari, S. S. Vazhkudai, J. H. Rogers, D. Dillow, G. M. Shipman, and A. S. Bland, “Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 217–228.
[9] B. Van Asche, “Re: scsi-mq V2,” [https://lkml.org/lkml/2014/6/27/300](https://lkml.org/lkml/2014/6/27/300).
[10] Data Direct Networks, “SFA 10K-X Datasheet,” [http://www.ddn.com/download/resource_library/brochures/block_storage/SFA10K-X_Datasheet.pdf](http://www.ddn.com/download/resource_library/brochures/block_storage/SFA10K-X_Datasheet.pdf)
[11] “Patch for Lustre source for Linux 3.18 on GitHub,” [https://github.com/bacaldwell/lustre](https://github.com/bacaldwell/lustre).
[12] L. Shelton, “High performance I/O with NUMA systems in Linux,” [http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf](http://events.linuxfoundation.org/sites/events/files/slides/scsi.pdf), Philadelphia, PA, May 2013.
[13] “Flexible I/O Tester,” [http://github.com/axboe/fio](http://github.com/axboe/fio).
[14] B. Gregg, “Flame Graphs,” [http://www.brendangregg.com/flamegraphs.html](http://www.brendangregg.com/flamegraphs.html).
[15] R. Henwood, “Final Report for the SMP Node Affinity Sub-project of the Single Metadata Server Performance Improvements of the SFS-DEV-001 contract,” [http://wiki.opensfs.org/images/2/2d/MDS_SMPNodeAffinity_Demonstration_FinalReport.pdf](http://wiki.opensfs.org/images/2/2d/MDS_SMPNodeAffinity_Demonstration_FinalReport.pdf).
[16] B. Van Asche, “Re: scsi-mq,” [https://lkml.org/lkml/2014/6/17/291](https://lkml.org/lkml/2014/6/17/291).
[17] B. V. Asche, “SRP and the scsi-mq project,” [http://downloads.openfabrics.org/Architecture/Media/Monterey_2014/Monday/13_IBDG%20SRP.pptx](http://downloads.openfabrics.org/Architecture/Media/Monterey_2014/Monday/13_IBDG%20SRP.pptx), Monterey, CA, Mar. 2014.