New Lustre Features to Improve Lustre Metadata and Small-File Performance

John Fragalla, Bill Loewe, PhD and Torben Kling Petersen, PhD
Cray Inc
Seattle, WA, USA
[jfragalla, bloewe, tpetersen]@cray.com

Abstract—**As HPC I/O evolves beyond the challenges of large I/O performance, several new Lustre features - Distributed Namespace (DNE) 2, Progressive File Layout (PFL), and Data on Metadata (DoM), when combined with flash-based Lustre targets, can provide metadata and small-file performance improvements to applications transparently. To improve the execution of single-directory operations and increase the scalability of overall metadata operations, Striped or Remote Directories with DNE2 can be configured with multiple MDTs. For small-file I/O, Progressive File Layout with flash and disks can be used to optimize small- and large-I/O by seamlessly storing files on flash-based MDTs or specific flash OSTs. This paper will share performance results that demonstrate the scalability benefits of metadata performance using striped or remote directories as well as small-file performance with DoM and Flash OSTs with PFL, both with and without Cray's block I/O accelerator (NXD) configured.**

Keywords-component; Lustre, metadata, flash acceleration, benchmarking, performance.

I. INTRODUCTION

As new community versions of Lustre are released, planned enhancements and features come available. With the latest major releases of Lustre (2.10, 2.11 and 2.12) came significant improvements to several technologies enabling enhanced performance and storage efficiency. Specifically, technologies such as Data on MDT (DoM) [1], Progressive File Layouts (PFL) [2] as well as previously introduced flash acceleration technologies from Cray, such as NXD [3, 4], alleviate some of the historical challenges posed by working with Lustre, namely IOPS and small I/O dependent workflows. In addition, improvements in Lustre metadata management, specifically, the ability to manage a larger number of metadata in a single directory, requires better understanding and fine tuning. Several of these technologies and techniques can be used in strategic combinations to make later versions of Lustre more capable.

As flash-based storage solutions in the form of accelerators, storage tiers or stand-alone storage solutions become more common, the interaction between Lustre enhancements and IOPS-capable storage components is increasingly important to understand. Only a few sites worldwide use all flash-based storage, while most utilize flash as an integrated component (a.k.a "Burst Buffer [5]). Optimizing an expensive, flash-based storage solution today requires extensive planning and fine tuning.

As demands on HPC storage increase and will continue to grow in the ExaScale era, overprovisioning with additional hardware is no longer an acceptable method to improve system performance. Using hardware more efficiently, adding intelligent software features to enable data placements and staging, as well as providing tools for data movement will achieve significantly better improvement gains at lower cost than adding hardware. The new Lustre features described above, coupled with the interaction between traditional HDD-based storage and newer solid-state components, can support true ExaScale systems in the data-centric world of high-performance computing.

II. AIMS OF THIS STUDY

The focus of this study is to objectively evaluate several recent performance enhancements in Lustre and compare their use on current productions systems, as well as examine their effects on more complex workloads. While NXD has been the topic of several previous studies [6, 7], the combination of new Lustre features paired with several different flash based acceleration technologies is also a significant part of this study.

III. SYSTEMS AND METHODOLOGY

The following hardware and software system configurations were used for the study.

A. Storage System and Clients

- 2x metadata units (ClusterStor L300 AMMU), each with two MDS nodes and two flash-based MDTs, each in a 10-device RAID 10 configuration using 20 Seagate (ST3200FM0033) SSDs and 20 Samsung (MZILS3T8HMLH/007) SSDs in separate enclosures for a total of 4 all flash MDTs.
- 1x ClusterStor L300F with two OSS nodes each with a single RAID-10 OST using 10x Seagate (XS3200LE10003) SSDs
- 2x ClusterStor L300N each with 2 OSS nodes and 2 OSTs based on 6 TB Seagate Makara drives (GridRAID 41[8+2-2]) and 2 SSDs for NXD
- Up to 64 client nodes (FDR connectivity)
- EDR InfiniBand non-blocking fabric

B. Software

- Lustre 2.11.0 client and server software
- CentOS Linux release 7.5 (client and server OS)
- Spectre/Meltdown - enabled kernels on clients, disabled on servers
    - o Client: 3.10.0-862.el7.x86_64
    - o Server: 3.10.0-693.21.1.x3.1.9.x86_64

- ClusterStor Neo 3.1 B23
- NXD 3.3.0
- IOR version 2.10.3 [8]
- MDTEST version 1.9.3[9]

C. *Test Environment*

To clearly demonstrate the Lustre features analyzed in this study, a number of different I/O methodologies are used.

1) *Effects of DoM on Random 4KB Writes I/O*

Using various DoM sizes between 64 KB and 4 MB, and writing 1,000 to 250,000 files, using different file sizes ranging from 32 KB to 8 MB to keep a constant aggregate dataset size. As the file size increases, the number of files written would decrease accordingly. DoM was configured with DNE2 to transparently use all four of the flash MDTs. Two sets of benchmarks were run, one with NXD turned off and one with NXD turned on using a bypass (breakpoint) of 32 KiB. All NXD data was flushed between each run to ensure an empty cache. The files were written with IOR, using random I/O with a 4KB block size, to examine write IOPs. The benchmark setup used a PFL layout where the first extent of the files was written to MDT with DoM and the remaining extent of the files were written to the HDD tier with and without NXD enabled, to analyze the results with random 4KB I/O.

2) *Effects of PFL on Random 4KB Writes I/O Using an All-Flash Tier*

Using a tiered namespace with 2 flash based OSTs and 4 HDD based OSTs separated into 2 OST pools, and using PFL to place the initial writes on the flash pool and the subsequent data on the HDD pool. Both the size of the initial PFL writes (64 KB – 4 MB) as well as the file sizes were varied (32 KB – 8 MB), and by adjusting the file count from 1,000 to 250,000, a constant aggregate dataset size was maintained. As the file size increases, the number of files written would decrease accordingly. As in experimental setup 1, two sets of benchmarks were run, one with NXD turned off and one with NXD on the HDD tier turned on using a bypass of 32 KiB. All NXD data caches were flushed between each run to ensure an empty cache. The files were written with IOR using random I/O with a 4KB block size, to examine write IOPs. The benchmark setup was similar to a PFL layout with DoM, but instead of using MDT to write the first extent layout, this setup uses flash OSTs with the HDD tier and also has a benchmark setup with a PFL layout where the first extent of the files were written to the flash OST tier and remaining extent of the file written to the HDD tier with and without NXD enabled to analyze the results with random 4KB I/O. The paper will analyze some differences between a PFL layout with DoM compared to a PFL layout with a flash OST tier for the first extent of the file, while writing the remaining extent of the file to the HDD tier.

3) *The "Noisy Neighbor" Problem*

A set of benchmarks examining the effects of two conflicting workflows on the file system were run. PFL was used with the all-flash tier in an attempt to isolate a smaller I/O workflow.

The "foreground" I/O process used IOR to create large, multi-GB sequential writes to the storage using 64 MB transfers and a fixed time of 180 seconds for writing and 60 seconds for reading. With this large sequential workload running across 3 iterations, the "noisy neighbor" I/O was created using MDTEST writing 1MB or 4MB files. NXD was negligible for this test due to the larger file size being generated by MDTEST for this workload, and we wanted to generate a "noisy neighbor" workload indicative of real application workloads. The test used flash targets, SSD-based OSTs on the L300F, with PFL to absorb the noisy neighbor workload. The paper will also compare results of absorbing the "noisy neighbor" workload on the MDT flash targets instead of using L300F flash OSTs to achieve similar results.

4) *DoM and Sharded Directories*

Using a sharded directory over all 4 MDTs, MDtest was used to create 4 million empty files (1M per MDT). The resulting file operations of creates, stats, reads, and unlinks were compared using both unique and shared directories as well as with or without DoM.

IV. Results

The different benchmark models used in this paper were designed to highlight specific features and their possible benefits to complex I/O workflows. While the data used for the specific I/O patterns were created by standard benchmarking tools, the specific parameters were chosen to simulate workflows that the authors have seen in production systems.

1) *Effects of DoM on Random 4KB Writes I/O*

When writing files of different sizes to the storage solutions, the results only focused on the PFL layout where the file size is written to both the flash MDTs with DoM and to the HDD tier, to examine the effect of NXD disabled and enabled has on this workload (Figure 1). As we focused specifically on write IOPS (rather than throughput), a significant benefit of DoM plus NXD enabled on the HDD tier can be seen when comparing the measured write IOPS of each file with compared with write IOPS achieved with DoM and the HDD pools with NXD disabled. Due to this specific workload, using a random 4KB block size to write various file size, the combination of flash MDTs to store the first extent of the file with the remaining extent of the file written to the HDD tier with NXD enabled has a positive impact on increasing write IOPs due to the fact NXD is enabled, which fits nicely within the bypass size of 32KB block size. However, it is important to note that if the random I/O benchmark setup chose a block size larger than the NXD bypass size, e.g. 40KB instead of 4KB, than NXD will have zero impact on improving IOPs.

2) *Effects of PFL on Random 4KB Writes I/O Using an All-Flash Tier*

In this set of experiments, the DoM flash component was replaced with all flash OSTs storage pool optimized for IOPS rather than throughput. As it was not the intent to compare a pure SSD-based OST pool with an HDD-based one, we used the same basic PFL layout to allow the first extent of the file

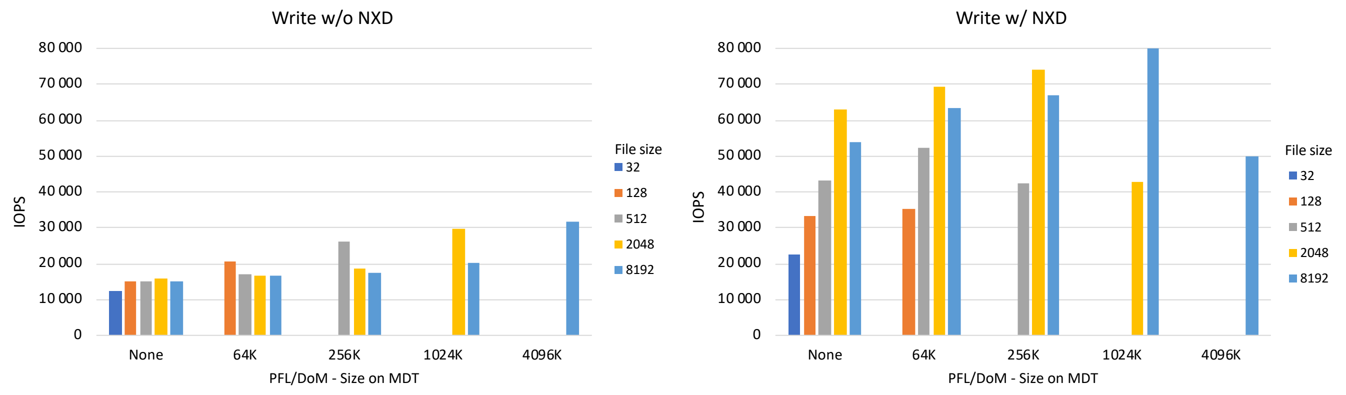

Figure 1 - Effects of DoM using various PFL settings with and without NXD. Data not hitting the HDD tier are removed for clarity.

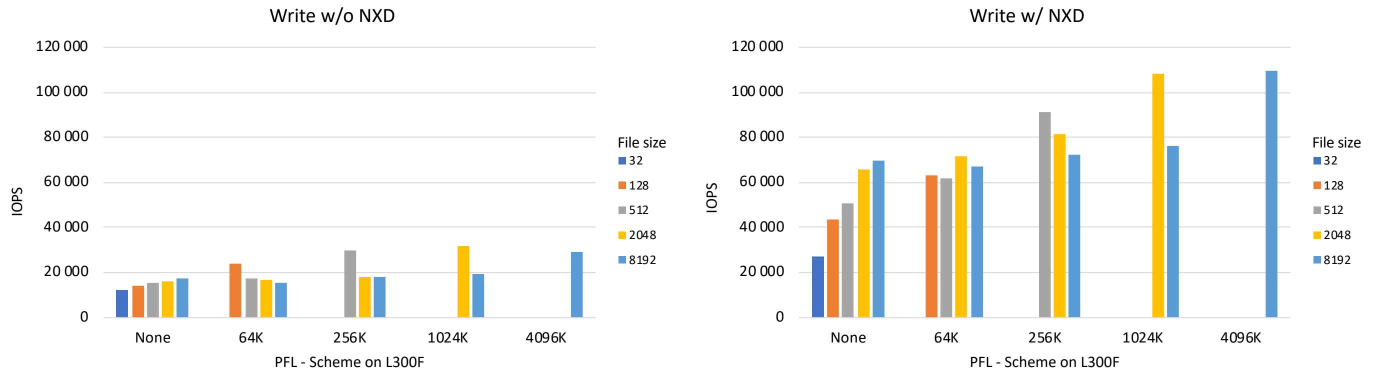

Figure 2 - Effects of PFL with a flash-based tier on IOPS using various PFL settings with and without NXD. Data not hitting the HDD tier are removed for clarity.

to use the flash tier as the primary storage target and the remaining extent of the file be written to the HDD tier, with and without NXD enabled. The results in Figure 2 only focused on a PFL layout in which the file is written to both the flash and HDD tiers, to analyze the effects of NXD of improving 4KB IOPs. Again, as in study 1, NXD enabled clearly increased write IOPs, working together on the same data set split on the flash tier, as it did using DoM with flash MDTs with the HDD tier, with regards to the ability to handle large IOPS. As previously stated, NXD with flash OSTs working on the same data set when the file size exceeds the PFL size on flash and splits between flash OSTs and HDD OSTs with NXD enabled, the results show an additional increase in write IOPs as compared to just flash OSTs alone

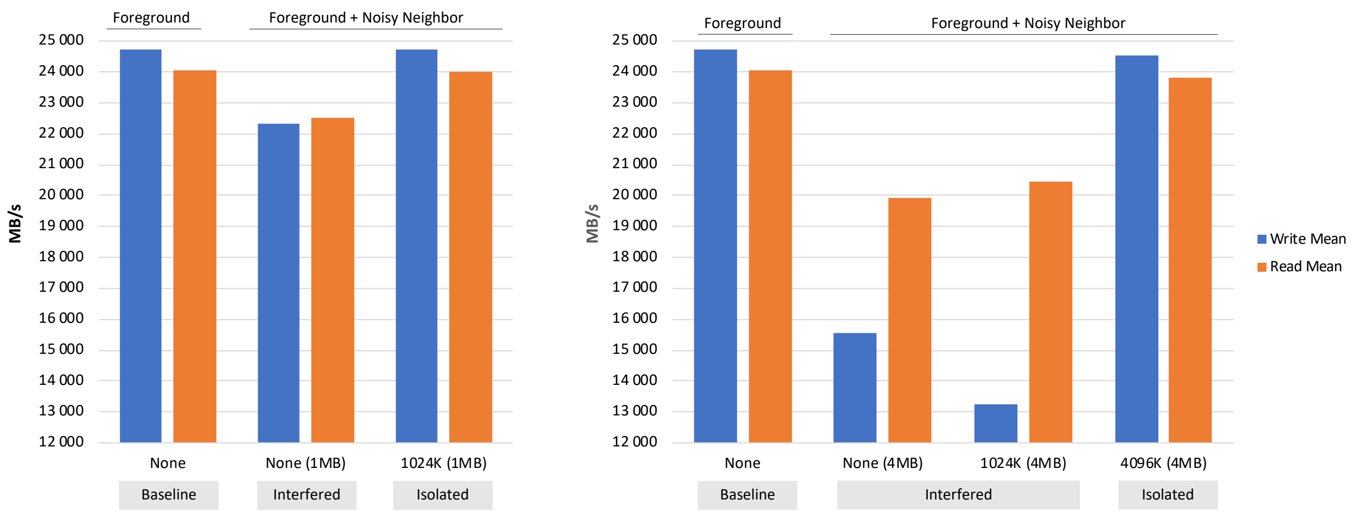

Figure 3 - Effects of PFL using a flash-based tier on "Noisy Neighbor"

DNE Striping	Files/MDT	File Create/s	File Stat/s	File Read/s	File Unlink/s
0 KB Files - Unique Directory					
Sharded Directory - 4x MDTs	1 048 576	167 611	753 885	602 834	346 796
Sharded Directory - 4x MDTs (64K DoM)	1 048 576	352 809	1 053 564	787 548	373 597
0 KB Files - Shared Directory					
Sharded Directory - 4x MDTs	1 048 576	148 974	428 402	605 334	187 857
Sharded Directory4x MDTs (64K DoM)	1 048 576	174 572	332 047	823 025	189 968

Table 1 - Effects of DoM on sharded (DNE phase 2b) directories using zero length files.

for this benchmark setup, due to the random 4KB block size that was used to write out the data sets, which fits within the NXD bypass setting of 32KB block size.

3) *The "Noisy Neighbor" Problem*

The baseline results depicted in Figure 3 show the basic throughput of a 2 SSU ClusterStor L300N system showing that under optimal settings, each SSU can deliver more than 12 GB/s read or write performance from Lustre 2.11 to a sufficient number of clients. Adding the "noisy neighbor" workload to compete with the sequential streaming workload has a significant effect on overall throughput with a 20 – 30% impact. Adding a PFL slice on the all flash-based L300F alleviates the impact, but to reach baseline levels, the PFL size needs to be able to accommodate the entire small I/O file size. The results illustrate that, with the 4MB file size competing workload on a PFL layout of only writing first 1MB of the first extent size of the file to flash OSTs, performance degrades further than just having the two workloads compete for resources on the HDDs alone. This indicates that the I/O profile needs to be well understood to tune the PFL layout and optimize performance.

4) *DoM and Sharded Directories*

Comparing metadata performance (Table 1) using DNE version 2b and a shared directory spanning 4 MDTs with or without DoM showed significant variability where DoM significantly improved file creates (110%), stat (40%) and reads (30%) using unique directories but had no effects on unlinks.

On shared directories, the impact was much less pronounced with file creates being slightly enhanced and reads were up some 30%. Stats was significantly impacted whereas the other components were essentially unaffected. Even though the performance gain is slightly enhanced, single shared directory capabilities are improved with the ability of leveraging multiple MDTs which increases functionality over just single MDT.

V. Discussions and Conclusions

The results generated in the different experimental setups are an extract of a much larger set of benchmark studies. However, the main findings do highlight the intended aims of this study while still generating some surprises.

Based on the results from tests 1-3, NXD has a profound effect on the write IOPS performance of a Lustre system working together with flash-based targets, MDTs or OSTs. But let's examine the results obtained without use of NXD for a moment. DoM has been heralded as the solution for writing large numbers of small files to a Lustre file system as the data would require significantly fewer RPCs to secure the data on disk, whereas earlier versions of Lustre required the metadata to be created, the OST assigned and the data committed to said OST despite it being only a few bytes in size. However, the current implementation of DoM (Lustre 2.11) does not reflect the anticipated small I/O performance on writes due to the additional open-close to commit the data to the DoM partition still needs to perform from the client to the MDS, compared to flash OSTs.

Comparing the PFL of using DoM for the first extent compared to Flash OSTs, looking at Figure 2 and comparing the same results from Figure 1, with or without NXD enabled in the HDD tier, flash OST demonstrates higher write IOPs versus MDT with DoM using 4KB block size. Reviewing Lustre architecture with less RPC overhead and less latency, writing to DoM should have at least shown comparable results to flash OSTs, but the results indicate otherwise. It is possible that the random I/O benchmark setup is better suited to flash OSTs rather than a MDT with DoM or there might be a write limitation with DoM when using Lustre DNE2 at the same time.

However, even with this additional open-close commit, DoM increased the write performance compared to writing small files to the HDD OSTs between 20 – 40% depending on file size. But this performance is not a given as the specific PFL settings to achieve a DoM enhanced I/O need to be adapted to the I/O pattern of any system. In addition, all the data reported here is for write performance. Comparing read performance using DoM, the effect is ~ 100% better over HDD (data not shown) as each read requires a single RPC and the data is read from SSDs and read performance for DoM outperforms reads on flash OSTs by up to 3x. Even if the results show flash OSTs are favoring write performance over flash MDTs, reads on DoM are favorable over flash OSTs due to the less RPC overhead and less latency reading data already written to the MDTs.

Comparing the effect of DoM with an OST pool based on all flash arrays is also interesting as the concept is basically the same: small files and the first blocks of larger files are committed to a flash-based tier but in this case, the flash component is significantly larger and used exclusively for data I/O. The overall results of experiment 1 and 2 show a very similar profile with PFL on the all flash arrays, providing high IOPS results as one would expect. However, the fact is that PFL is consistent in behavior and delivers significant performance enhancements when tuned correctly. And therein lies a challenge in understanding the I/O profile. Before using PFL on a production system, one needs to look closely at the pattern of user I/O on a given HPC system. While it would be helpful to provide guidance on specific settings that improve general purpose I/O, the problem is that no two production systems are identical, nor do they use the same set of software packages, have the same types of data or the same number of concurrent users. Therefore, such guidelines would not improve performance of a given system; more likely they would have a suboptimal impact and the investment in expensive SSD or NVMe based components would not be justified.

Lustre is designed for large streaming I/O and it is commonly known that mixed I/O workloads significantly reduce the performance of a storage solution, especially if one of the workloads is a small, random I/O pattern. While we have shown before that NXD can solve some of this problem, specifically random small block I/O, [7] it does not allow for dynamic allocation nor works for random I/O with varying data block sizes. Using PFL and an addressable flash tier, MDT or Flash OSTs, allows for better dynamics and can essentially restore the streaming I/O on a Lustre file system with complex I/O patterns. While this currently requires careful analysis of the I/O patterns, it could conceivably be enhanced by real time analysis of I/O with associated adjustment of the PFL scheme on a given directory of an OST pool.

One additional area where Lustre is improving is metadata performance. However, metadata performance is not a single measurement, but rather consists of many different components, each with different characteristics and limitations. For example, file create is known to be a single threaded process in the metadata server and scales, more or less, linearly with CPU clock frequency (data not shown), regardless of the devices used for the MDT storage. Metadata stats, on the other hand, are exclusively read dependent and benefits greatly from SSD based MDTs. Using multiple metadata targets and sharded directories did indicate a positive effect of DoM when writing a million files into unique directories but had no effect on performance when using a single shared directory. DoM did produce a significant benefit on file reads regardless of the type of directory structure used. However, with Lustre 2.11, the overall assessment of DoM on metadata performance was not entirely clear and there is a lot of variability. The benefits of multiple MDTs with sharded directories is seen when doing single directory operations, in which multiple MDTs can participate to improve operations in a single directory, where historically Lustre can only use a single MDT for single directory operations.

To summarize, the recent enhancements seen in Lustre 2.10 and above do indeed show improvements in performance, specifically IOPS dependent I/O. However, with the current set of tests, these improvements depend heavily on understanding the application I/O profile and tuning Lustre for the specific profile to accelerate mixed I/O. What is clear, however, is that SSDs are here to stay and will in time take most , if not all, I/O intensive (whether that being throughput or IOPS) from current enterprise HDD based systems although the latter will be around for many years to come to satisfy the ever increasing need of capacity. With increased use of flash, and high capacity SSDs and HDDs, the number of files in a single namespace is also growing, and with new Lustre Features, such as DNE2, multiple MDT targets are becoming a requirements to manage larger amount of files and directories, and single directory operations are becoming critical to support more entries than historically what was allowed.

Bridging the paradigm shift requires careful planning and insights into one's workflow patterns as well as tools to manage automatic tiering, data staging/de-staging as well as more advanced data movement and migration.

Future Work

The results reported in this paper are based on pure benchmarking tests and should be seen as indicative regarding real-world production codes and I/O workflows. As Lustre 2.11 and newer versions gain prominence in the user community, similar studies using in-house developed software or commercial codes should be done to further validate or modify the findings in this study.

Acknowledgment

As usual, no paper such as this could be achieved solely by the principal authors, and many people contributed to gather and analyze the test data and determine the results. The authors would like to thank (in no particular order) the following collaborators for their hard work and support:

Rich Jenke, Kris Woosley, Amanda Ma, Zack Rankin, Tim Watson, Wolfgang Szoecs, Rob Roy, Peter Bojanic, Patrick Farrell, Nathan Rutman, Mark Wiertalla, Larry Jones, Nate Schumann, Charlie Carroll, Cory Spitz and the Cray Lustre Development Team

References

[1] OpenSFS. "Data on MDT," http://wiki.lustre.org/Data_on_MDT_High_Level_Design.

[2] OpenSFS, "Progressive File Layouts," 2018.

[3] T. Kling Petersen, "Flash Acceleration of HPC Storage – NXD," https://www.cray.com/resources/flash-acceleration-hpc-storage-ndx-performance-comparison?leadsource=email&srcdes=NXDpaperpromo&campaign=7010b000000ayIh&elqTrackId=4d0df9ee28ee45d4b66c17d8e89201ac&elq=3378cd58c24740a0ab0ca2e7f8134c09&elqaid=1131&elqat=1&elqCampaignId=511, 2018.

[4] T. Kling Petersen, and P. Balakrishnan, "Flash Acceleration of HPC Storage - Nytro Intelligent I/O Manager," http://seagate.com/?lipi=urn%3Ali%3Apage%3Ad_flagship3_profile_view_base%3BEf0NR4%2FvRXGtPrsZfZH9fg%3D%3D, 2017.

[5] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in 28th IEEE MSST conference, 2012.

[6] T. Kling Petersen, and J. Bent, “Hybrid flash arrays for HPC storage systems: An alternative to burst buffers,” in 2017 IEEE High Performance Extreme Computing Conference (HPEC), 2017.

[7] T. Kling Petersen, and B. Loewe, "The Role of SSD Block Caches in a World of Networked Burst Buffers," 2018.

[8] IOR. http://sourceforge.net/projects/ior-sio/.

[9] MDtest. http://sourceforge.net/projects/mdtest/.