Evaluation of A Flash Storage Filesystem on the Cray XE-6

Jay Srinivasan
Computational Systems Group
NERSC, Lawrence Berkeley National Laboratory
Berkeley, USA
Email: jsrinivasan@lbl.gov

Richard Shane Canon
Technology Integration Group
NERSC, Lawrence Berkeley National Laboratory
Berkeley, USA
Email: scanon@lbl.gov

***Abstract*—Flash storage and other solid-state storage technologies are increasingly being considered as a way to address the growing gap between computation and I/O. Flash storage has a number of benefits such as good random read performance and lower power consumption. However, it has a number of challenges too, such as high cost and high-overhead for write operations. There are a number of ways Flash can be integrated into HPC systems. This paper will discuss some of the approaches and show early results for a Flash file system mounted on a Cray XE-6 using high-performance PCI-e based cards. We also discuss some of the gaps and challenges in integrating flash into HPC systems and potential mitigations as well as new solid state storage technologies and their likely role in the future.**

***Keywords*-Flash storage; Cray XE-6; HPC systems**

I. Introduction

The effective use of large HPC systems depends greatly on the effectiveness of the I/O subsystem. While the performance of HPC systems has increased by several orders of magnitude over the last few years due to increases in the number of compute cores, system scale, and the use of accelerators, the I/O systems have largely failed to keep pace, resulting in a gap between computing and I/O, which is made only more prominent by the advent of data centric computing models that require greater amounts of I/O. Additionally, application I/O on HPC systems does not always conform to a steady, streaming pattern and may involve bursts of reading or writing which tax the ability of the storage subsystem to meet these requirements. Increasing the performance of the I/O subsystem cannot be simply done by using traditional methods of scaling up spinning disk storage, and so the use of Flash and other solid-state storage technologies are being considered.

The present costs of solid-state storage technologies makes it impractical as a drop-in replacement for traditional spinning disk storage and thus, any use of high-performance storage will require software solutions to effectively utilize them in conjunction with traditional lower-performant storage. The integration of a I/O path utilizing Flash-based storage is still evolving, although in general, one may consider the addition of the Flash-based storage to be one more tier in a storage hierarchy of the I/O subsystem of an HPC system.

In this paper, we discuss some of these approaches and present one example of a proof-of-concept deployment of a Lustre filesystem built on high-performance PCI-e-based Flash storage cards on a Cray XE-6. The system is architected not as a hierarchy with higher performing storage in the path to lower performing storage, but instead presents Flash-based and Disk-based filesystems to the compute nodes and explicitly migrates data between the two. We start by providing an introduction to Flash technology in Section II. In Section III, we provide a potential architecture using Flash. In Section IV, we explore the performance of the Flash-based storage and compare it to a concurrently available traditional ”scratch” storage system. We then discuss the use of Flash and provide a cost analysis in Section V. We close with some final thoughts and ideas for Future work.

II. Flash Storage Introduction

A. Flash Technology

While it is beyond the scope of this paper to provide an exhaustive background on Flash storage, it is useful to summarize some of its important characteristics and their implications [1]. Flash is a short-hand name for NAND Flash storage which is currently the most ubiquitous form of Solid State Storage (SSS). NAND has certain characteristics that make it advantageous over disk storage such as fast random read access and low-power consumption (especially for data at rest). However, compared to volatile DRAM, it suffers from high overhead for writes, since storage cells must first be erased before storing new data. This erase cycle can take several hundred microseconds to perform for a single block. Flash also suffers from ”wear” which causes the storage cells to fail after a certain number of erase-write cycles. The susceptibility to wear varies between technologies. Traditionally enterprise devices have used Single Level Cell (SLC) chips since they can typically provide over 1000x more write cycles versus multi-level cell (MLC) chips. However, increasingly vendors are using MLC chips, which are cheaper, and relying on more sophisticated logic to hide some of these differences. Furthermore, as manufacturers move to newer fabrication technologies with smaller feature sizes (i.e. from 32 nm to 19 nm) to increase capacity, the ability to reliably store data is decreasing. As a result, device manufacturers must utilize increasingly sophisticated logic to provide the same reliability as previous generations of devices. New solid-state storage

technologies such as Phase Change Memory (PCM) [2] or Memristor [3] could potential address many of the deficiencies of NAND Flash (both reliability and write penalties). However, these technologies have yet to reach large scale production.

Solid State Storage can be connected in a variety of methods to a system. Solid State Disks (SSDs) are perhaps the most common since they are often found in laptops and are increasingly used in high-end server systems. SSDs come in different grades (consumer versus enterprise) that typically vary in interface (SATA versus SAS), sophistication of the controller logic, and Flash technology (SLC versus MLC). When using SSDs, the controller the drives are attached are critical since they can become a performance bottleneck. The controllers also typically cannot account for failure characteristics within the SSD. For example, the controller cannot easily shift data from failing cells on one SSD to another SSD. Instead the controller must simply view the SSD as healthy or unhealthy.

Direct attached PCI-e devices are another class of Flash storage. These devices are typically more expensive compared to SSD solutions, but often offer higher IO rates, more sophisticated logic that is closely integrated with the Flash components, and higher bandwidth. In many high-end PCI-e Flash devices, the Flash Translation Layer (FTL) is programmed into a field-programmable gate array (FPGA). This enables the algorithm and control logic to modified over time. These algorithms are responsible for activities such as "garbage collection", balancing data across cells, and detecting failing cells. The ability to change this logic is important since, in many cases, the best methods are still being determined.

The PCI-e cards often employ techniques to hide the high impact of erase-write cycles by maintaining a pool of erased blocks that can store in-coming writes. However, once this pool is exhausted, the device will slow down new writes while it aggressively starts to prune the storage. This behavior can be clearly seen in sustained write tests. Eventually, the devices typically reach a steady-state where new blocks are pruned and consumed at a consistent rate. A plot of this behavior is seen in Fig 1.

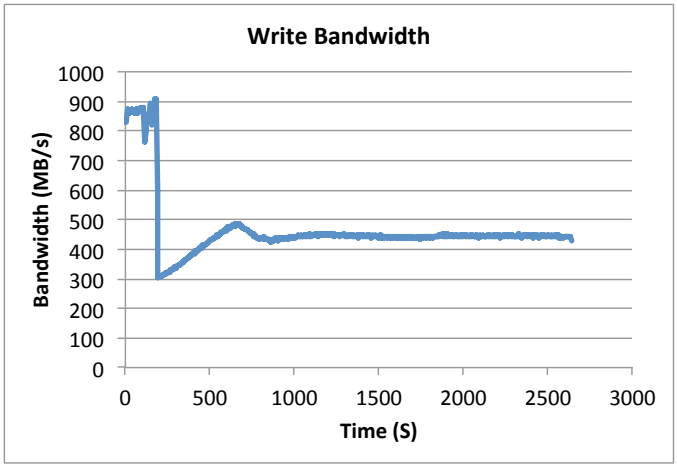

Fig. 1. A plot of the write bandwidth of a full device over time. Initially pre-erased blocks are used to provide a high burst rate. Eventually these blocks are exhausted and blocks must be erased as they are consumed.

B. Integration of Flash into HPC Systems

There are a number of ways that Flash can be integrated into HPC system. One model is to place Flash locally on each compute node. The Flash could be used either to extend the memory hierarchy or as a local block storage device. The later is the most prevalent today, although there is ongoing research exploring how to use Flash in the memory hierarchy. The are several ways the local block based Flash storage could be used. It could be treated as a swap device (again to extend the available memory for applications), treated as a local buffer cache, or aggregated into a distributed file system. Research projects are exploring how to utilize Flash as local buffer cache [4]. One approach would be to utilize checkpointing libraries like SCR [5] which already support using local storage to store checkpointing files. Aggregated local Flash storage can be accomplished with distributed file systems like Ceph [6] or GPFS using its recently developed "shared nothing" model [7]. The biggest strength to integrating Flash into the compute node design is that the bandwidth and the capacity scale linearly with the system and it can reduce the pressure on the global interconnect. This is one of the reasons that frameworks like Hadoop have championed this approach [8]. Extending the memory hierarchy also allows Flash to address potential challenges with providing sufficient memory for applications. However, these approaches depend on the systems having support for Node-local Flash storage which is infeasible with most of todays blade-form factor based HPC systems.

Another approach is to place the Flash in the storage hierarchy and rely on the storage server or an intelligent controller to manage its use. There are storage products on the market that provide this capability such as NetApp's SANtricity SSD Cache [9]. In addition, some local file systems such as ZFS support using solid-state storage to accelerate logging and ingest. This approach is interesting because it is the most transparent to the application. If the controller is virtualizing the Flash storage, even the file system layer may not need to be altered to take advantage of the Flash storage. However, the application and, potentially, the file system lose some control over how the Flash is utilized which may limit its effectiveness.

A final approach is to use the Flash as a discreet storage pool or file system and rely on the application or a service to migrate the data between the Flash storage and disk storage. For example, an application would write checkpointing files to the Flash storage, then trigger a background migration process while it continues to perform calculations. At the beginning of the next checkpointing cycle, it would need to insure that the previous checkpoint was successfully migrated. Alternatively, it could perform the migrations on some longer cycle to conserve disk bandwidth. A modified version of this approach would be to utilize some out of band migration service that would be triggered through an API or semaphore

files. This could be tightly integrated into the file system or could be implemented as middle-ware. Some of the advantages to this approach is it can provide a high-level of flexibility on how the Flash storage is used. It also lends itself to simple implementations for exploration.

III. Flash Storage Architecture

The evaluation was carried out using a small Cray XE-6 test development system (TDS). This system already has a small external disk-based Lustre file system consisting of a single Object Storage Server (OSS) serving four Object Storage Targets (OSTs). The server is a single Dell R710 with two LSI 8600 storage arrays. The server is attached to a Quad Data Rate InfiniBand network via a 4x host adapter. Two Lustre Network (LNET) Routers bridge the XE-6 Gemini network and the InfiniBand network.

The Flash storage was configured as a separate Lustre file system. A total of four Virident tachIOn cards were configured on two IBM x3650 M2 Westmere-based Object Storage Servers. Each Virident card has around 400 GB of SLC-class NAND storage and are capable of delivering around 1.1 GB/s of read and write bandwidth and 160k Read IO operations per second.

Figure 2 shows a schematic of the architecture of the evaluation system.

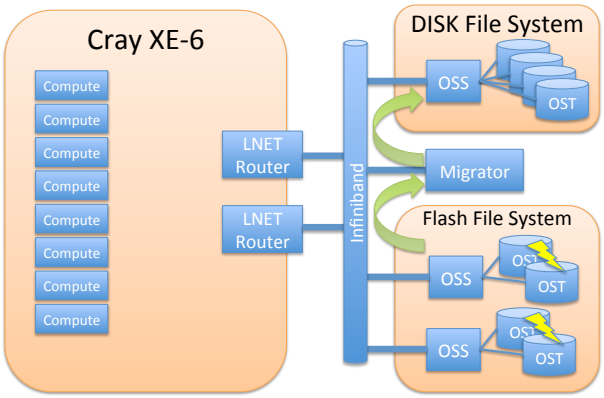

Fig. 2. Architecture Diagram for Evaluation Configuration.

IV. Performance of Flash vs. Traditional Storage

We ran two kinds of tests on the traditional and Flash based storage: a standard I/O benchmark (IOR) [10] using POSIX I/O to gauge the baseline performance of the filesystems and a benchmark designed to mimic a workload composed of both I/O and compute. This benchmark (called `flashio`), consists of a simple matrix-matrix multiplication done several times in a loop (the amount of computation is adjustable). Additionally, the benchmark performs I/O (designed to mimic a checkpoint operation) from each task to a designated filesystem. For the case when the I/O is performed to the Flash storage, we have a "migrator" process running that moves the files out of Flash storage into standard storage in the background (see Figure 2).

The migrator is a small program that runs continuously in the background and checks to see if a specifically named "semaphore" file is written, indicating that the checkpoint is complete. If the file exists, the migrator moves the corresponding "checkpoint" file from the Flash storage into the disk-based filesystem. In its current form, the migrator process is a per-user migration tool and is currently only single threaded and moves files one by one. There are several obvious optimizations we can perform to improve the performance of the migrator depending on the I/O requirements of the users of the Flash storage – for example, by multithreading the program, addition of the ability to specify locations to move at job run-time and make the migrator a more generic program not requiring user intervention.

The migrator process can run on any node that has access to both the Flash and the disk-based filesystems. In this case, we ran it on an "external" login node of the Cray XE-6 that has access to the disk-based Lustre filesystem directly via the InfiniBand network. For the case when I/O is performed to traditional disk storage, no additional moves of files are performed. Table I shows the results of IOR runs to the disk-based "scratch" and the Flash-storage filesystems on the Cray XE-6. As can be seen, the disk subsystem performs at about 300-400 MB/sec for writes, while the Flash storage has a peak of almost 4GB/sec for writes. (IOR results were obtained using file-per-process access, sequential offsets for ordering and with a transfer size of 4MiB and block size of 1GiB). Figure 3 shows a plot of these results.

TABLE I
IOR runs on Flash and Disk storage

Nodes	Tasks/Node	Aggregate Filesize (GiB)	Flash Write B/W (MB/s)	Disk Write B/W (MB/s)
2	1	2	1132.44	406.46
2	2	4	1786.57	316.02
4	2	8	2066.55	326.52
12	2	24	3491.05	397.47
12	4	48	3520.62	401.62
12	5	60	3649.77	406.10
12	12	144	3781.97	377.39
12	24	288	3534.52	368.19

Table II shows the results of `flashio` runs at various concurrencies with I/O performed to the Flash storage and disk-based "scratch" storage.

The `flashio` runs to both the disk-based storage and Flash storage are adjusted to perform the same amount of computation. In the case of I/O to disk, once the files are written, no further I/O is performed. In the case of Flash storage, once the files are written to the Flash filesystem, the migrator process (which runs continuously in the background) moves the files to a disk-based store. The computational loop continues while this migration is in progress. It is therefore possible that for small amounts of computation that the run may complete before the migration is complete. The results

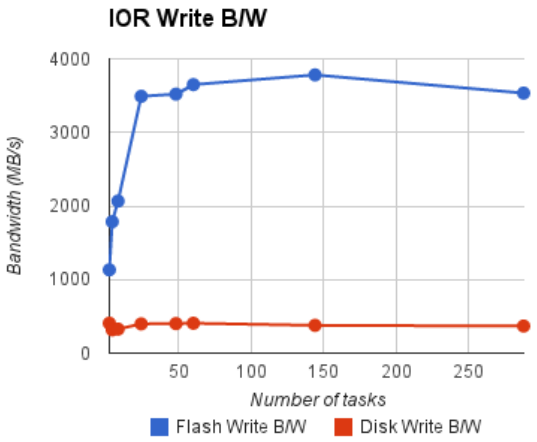

Fig. 3. IOR runs on Flash and Disk storage.

TABLE II
FLASHIO RUNS ON FLASH AND DISK STORAGE

Tasks	Disk Filesystem Write B/W (MB/s)	Disk Filesystem Run time (secs)	Flash Filesystem Write B/W (MB/s)	Flash Filesystem Run time (secs)
2	408.96	1436	1087.10	1435
4	479.72	1453	1760.29	1447
8	454.63	1478	1815.86	1452
24	418.40	1504	3572.85	1447
48	488.32	1567	3445.97	1467
60	437.68	1626	4289.22	1479
144	526.15	1841	4980.90	1524
288	476.56	2221	5588.29	1561

reported in table II, however, are from runs where the amount of computation exceeds the time for I/O (to both disk-based and Flash-based storage) and the compute cycles finish well after the migration process is complete, and thus the longer times for the disk-based runs are solely due to the slower I/O performed to disk. Figures 4 and 5 show the time for runs of varying concurrency to both Flash and disk-based storage.

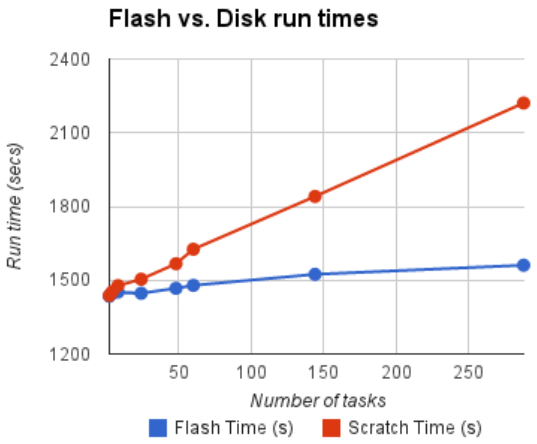

Fig. 4. Flash and Disk storage benchmark I/O times

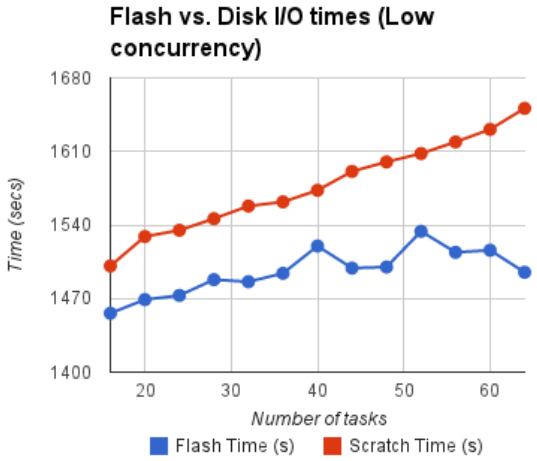

Fig. 5. Flash and Disk storage benchmark I/O times (detail of lower concurrencies)

V. DISCUSSION

A. Cost Analysis Using Hybrid Flash Architecture

The fact that the the Flash file system is significantly more powerful in terms of bandwidth relatively to the disk space system is useful to explore a potential design point for future systems. Using Flash storage to provide bandwidth, while depending on disk to provide capacity is one proposed approach for addressing the ever growing bandwidth gap. In the evaluation configuration, the Flash system is around 10x faster than the disk based solution which is perhaps a bit extreme. However, this could be a realistic model for future systems as disk performance improvements continues to lag. For example, let's assume that an application needs to checkpoint 1,200 TB ever hour. If the file system provided 1 TB/s of bandwidth, it would require 20 minutes to checkpoint. Consequently, 33% of the run-time would be used to perform checkpointing operations. However, if a hybrid approach was employed which used Flash storage providing 2.25 TB/s, the applications would require around 15% of the time to perform checkpointing. Furthermore, the disk storage bandwidth could potentially be reduced by approximately 60% since the entire checkpoint period (1 hour) could be used to migrate the data between the Flash storage and the disk storage.

A cost analysis helps illustrate the potential value of a hybrid approach. While Flash capacity is still much more costly than disk storage, on a bandwidth basis it can be very competitive. Table III provides a current estimate of the cost for bandwidth and capacity of Enterprise-class Flash and disk storage. Obviously pricing can vary widely for different vendors, models, and configuration, so these estimates are mainly intended to provide an illustration of how Flash could be a cost effective option. Using the example above and the cost estimates in Table III, the hybrid and disk-only approaches would be roughly equivalent in cost (see Table IV). However, the hybrid approach would provide around 28% more compute time since the checkpoints can complete more quickly. The

main drawback in this scenario is there is 62% less disk capacity since it was in essence traded off for bandwidth provided by the Flash storage. Assuming disk capacities growth continues to outpace disk bandwidth improvements, this potential trade-off may become more acceptable over time. This analysis neglects some of the additional performance advantages and power savings of Flash. Since Flash provides superior random read performance to disk solutions, data can be more efficiently reorganized before it is transferred to disk. In addition, Flash storage requires significantly less energy to deliver the equivalent bandwidth compared to disk-based storage.

TABLE III
APPROXIMATE COST FOR ENTERPRISE CLASS FLASH STORAGE AND DISK STORAGE ON A BANDWIDTH AND CAPACITY BASIS.

Storage	Bandwidth Cost ($ per GB/s)	Capacity Cost ($ per TB)
Flash (Enterprise grade)	$6,000	$6,000
Disk (Enterprise grade)	$22,000	$400

TABLE IV
A COST COMPARISON BETWEEN A HYBRID BASED SOLUTION THAT EMPLOYS BOTH FLASH AND DISK AND A DISK ONLY SOLUTION.

	Hybrid	Disk Only
Flash Storage Bandwidth (TB/s)	2.25	-
Disk Storage Bandwidth (TB/s)	0.39	1.00
Flash Storage Capacity (PB)	2.25	-
Disk Storage Capacity (PB)	20.9	53.3
Example Application		
Checkpoint Volume (TB)	1200	1200
Checkpoint Iteration (seconds)	3600	3600
Time for Checkpoint (s)	533	1200
Time remaining for Compute (s)	3067	2400
Percentage of Time in I/O	15%	33%
Improvement in Compute Time	28%	-
Cost		
Flash Cost ($ Thousands)	$13,500	-
Disk Cost ($ Thousands)	$8,609	$22,000
Total Cost ($ Thousands)	$22,109	$22,000

B. Trends in Solid State Storage Technology

Today the solid-state storage market is dominated by NAND Flash. However, a number of technical hurdles stand in the way of continuing improvements in NAND storage capacity and performance. As the feature sizes decrease, the reliability of the storage decreases. Consequently, more aggressive measures are required to maintain reliability and some predict that NAND Flash will fail to maintain the rate of improvement seen in the past [11]. For these reasons, new technologies are expected to supplant NAND in the not too distant future. Memristor storage which has been championed by HP and Hynix allows the resistance of an element to be changed in a persistent way which can then be read out to obtain the stored value. One advantage of this technology is that it can use conventional manufacturing techniques. However, challenges still remain in perfecting the switching mechanisms. Perhaps even closer to market is Phase-Change Memory (PCM) which stores data by changing the material in the chip between a crystalline and amorphous state. Both Memristor and PCM are expected to address some of the most serious deficiencies of NAND storage. For example, the expensive erase-cycle required in NAND Flash is avoided in the new technologies. This should allow for better write performance. More importantly, the wear issues of NAND are also avoided. This means the new technologies should provide longer life-times and require less sophisticated logic to manage the chips. It is possible that these new technologies could replace both NAND Flash and current DRAM technologies, thus potentially providing a persistent general purpose RAM.

While these new technologies are promising, current challenges in transitioning to manufacturing and other technical hurdles remain. Consequently, NAND Flash is expected to remain the market leader for solid state storage for several years to come. Several manufacturers of NAND storage have recently announced products using fabrication processes in the sub 20nm scale [12] and future products are already anticipated at the 15nm feature size. In addition, 3D stacking is being considered as a way to extend the roadmap for NAND [13]. The most serious hurdles are expected as manufacturing is scaled down to the 10nm scale and below. Consequently, NAND Flash will likely remain a contender for the next several years.

VI. CONCLUSION

Flash-based storage and other solid state storage technologies are promising tools to address the growing I/O challenges encountered in large scale HPC systems. There are a number of ways to integrate these technologies into the storage hierarchy. In this evaluation, we have explored one such method – the use of a discrete pool of very high-performance storage and asynchronous staging of files to and from this storage into larger pools of lower-performing storage. This approach was chosen because it was straight forward to implement and evaluate. One feature of this implementation is that it requires the user to select what I/O to redirect to the Flash file system. We believe that it is important that the user have some explicit control over how the Flash is utilized and how the movement of data between Flash and long-term disk storage is managed. Methods that integrate Flash more seamlessly into the storage hierarchy may have advantages in ease of use, but may lead to inefficient use of the scarce capacity provided by the Flash storage. This challenge is similar to what is encountered in many hierarchical designs. As in memory hierarchies, the use of the Flash-storage as a cache requires consideration of coherency issues between various levels of the cache. For storage caches, as the absolute timescale for ensuring coherence is high, it is an additional reason to ensure the users are not kept oblivious of the I/O path, and are able to exercise some control over it. Another potential problem with the approach we have demonstrated is that the Flash storage is treated as a shared resource and, consequently, is susceptible to contention from other users. This is no different than what

is typically encountered in disk-based file systems today. A more ideal solution would allow users to either provision bandwidth and capacity through a quality of service resource or have private pools of Flash storage allocated to applications. Another potential solution is to have Flash integrated into the node design (i.e. Node Local storage) which opens up other possible uses of the Flash. Regardless of the specific implementation, having APIs or other mechanisms for users to control the use of Flash and the migration of data would be beneficial. We have also provided a simple cost analysis which demonstrates that even today's Flash storage is a cost effective way to increase bandwidth if storage capacity can be sacrificed. We expect industry trends may help reinforce this calculus in the future, especially, if new solid state storage technologies transition to market. We plan to pursue exploring the use of Flash-based technologies for I/O acceleration by expanding the scope of the current study. Additional, Flash storage cards can be added to the Flash file system which should increase the performance to over 20 GB/s. Scaling up the test by integrating the Flash filesystem into a larger Cray XE or XC based system is being looked at, as is improving the performance of the migrator process. Additionally, we are looking at ways of making I/O to the Flash filesystem more seamless and predictable to the users.

ACKNOWLEDGMENT

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

The authors would like to thank the staff of the Computational Systems Group and the on-site Cray personnel at NERSC for their assistance.

REFERENCES

[1] N. M. Master, M. Andrews, J. Hick, S. Canon, and N. J. Wright, "Performance analysis of commodity and enterprise class flash devices," in *Petascale Data Storage Workshop (PDSW), 2010 5th*. IEEE, 2010, pp. 1–5.

[2] H. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[4] K. Iskra, "The NOLOSS project," Febuary 2011, http://science.energy.gov/.

[5] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*. IEEE, 2010, pp. 1–11.

[6] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.

[7] K. Gupta, R. Jain, I. Koltsidas, H. Pucha, P. Sarkar, M. Seaman, and D. Subhraveti, "GPFS-SNC: An enterprise storage framework for virtual-machine clouds," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 2–1, 2011.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[9] "NetApp SANtricity solution brief," http://www.netapp.com/us/media/ds-3395.pdf.

[10] "IOR: A parallel filesystem i/o benchmark," https://github.com/chaos/ior.

[11] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of nand flash memory," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, pp. 2–2.

[12] Y. Li, S. Lee, K. Oowada, H. Nguyen, Q. Nguyen, N. Mokhlesi, C. Hsu, J. Li, V. Ramachandra, T. Kamei, M. Higashitani, T. Pham, M. Honma, Y. Watanabe, K. Ino, B. Le, B. Woo, K. Htoo, T.-Y. Tseng, L. Pham, F. Tsai, K. ho Kim, Y.-C. Chen, M. She, J. Yuh, A. Chu, C. Chen, R. Puri, H.-S. Lin, Y.-F. Chen, W. Mak, J. Huynh, J. Chan, M. Watanabe, D. Yang, G. Shah, P. Souriraj, D. Tadepalli, S. Tenugu, R. Gao, V. Popuri, B. Azarbayjani, R. Madpur, J. Lan, E. Yero, F. Pan, P. Hong, J. Y. Kang, F. Moogat, Y. Fong, R. Cernea, S. Huynh, C. Trinh, M. Mofidi, R. Shrivastava, and K. Quader, "128gb 3b/cell nand flash memory in 19nm technology with 18mb/s write rate and 400mb/s toggle mode," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, 2012, pp. 436–437.

[13] "SK Hynix readying 3d stacked memory," http://www.i-micronews.com/news/SK-Hynix-readying-3D-stacked-memory-commercialization-closer,10000.html.