

DASH-IO: an Empirical Study of Flash-based IO for HPC

Jiahua He Jeffrey Bennett Allan Snavely
San Diego Supercomputer Center, University of California, San Diego
jiahua@gmail.com, jab@sdsc.edu, allans@sdsc.edu

ABSTRACT

HPC applications are becoming more and more data-intensive as a function of ever-growing simulation sizes and burgeoning data-acquisition. Unfortunately, the storage hierarchy of the existing HPC architecture has a 5-order-of-magnitude latency gap between main memory and spinning disks and cannot respond to the new data challenge well. Flash-based SSDs (Solid State Disks) are promising to fill the gap with their 2-order-of-magnitude lower latency. However, since all the existing hardware and software were designed without flash in mind, the question is how to integrate the new technology into existing architectures. DASH is a new Teragrid resource aggressively leveraging flash technology (and also distributed shared memory technology) to fill the latency gap. To explore the potentials and issues of integrating flash into today's HPC systems, we swept a large parameter space by fast and reliable measurements to investigate varying design options. We here provide some lessons we learned and also suggestions for future architecture design. Our results show that performance can be improved by 9x with appropriate existing technologies and probably further improved by future ones.

Categories and Subject Descriptors

B.4 [INPUT/OUTPUT AND DATA COMMUNICATIONS]: Performance Analysis and Design Aids

1. INTRODUCTION

HPC applications are becoming more and more data-intensive. We have participated in and analyzed some user interviews [25, 24, 21, 23] and found two important data-intensive applications categories: data mining and predictive science [26, 19, 14]. Both of these two kinds of applications share the same characteristics: they are dominated by small random data access (especially read) patterns. We focus on improving random (especially read) performance in this paper.

To achieve satisfying performance for these kinds of applica-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

TeraGrid '10, August 2-5, 2010, Pittsburgh, PA, USA.

Copyright 2010 ACM 978-1-60558-818-6/10/08 ...\$10.00.

tions, short latency is the key. Unfortunately, the current architecture was not designed for these kinds of latency-critical applications. Figure 1 shows the average latencies of the existing memory storage hierarchy. As may be noticed, there is a 5-order-of-magnitude latency gap between memory and spinning disks.

To fill the latency gap, one can make use of remote DRAM memory across the network interface, which is 3-order-of-magnitude faster than spinning disks. However, it requires special hardware [8] or software [10, 1, 16] to support. Flash-based SSD [13, 17, 15, 20, 22, 18] is another relatively new choice, which is 2-order-of-magnitude faster than spinning disks and promises potentially to take the place of them. We adopted both these technologies to build a data-intensive supercomputer called DASH, which is a new Teragrid resource. In this paper, we will focus on the flash drives.

How best to integrate flash technology into the existing HPC architecture is still an open problem. Since most the existing hardware and software were designed for slow spinning disks, there will be a lot of surrounding components like operating system and RAID (hardware or software), which might not

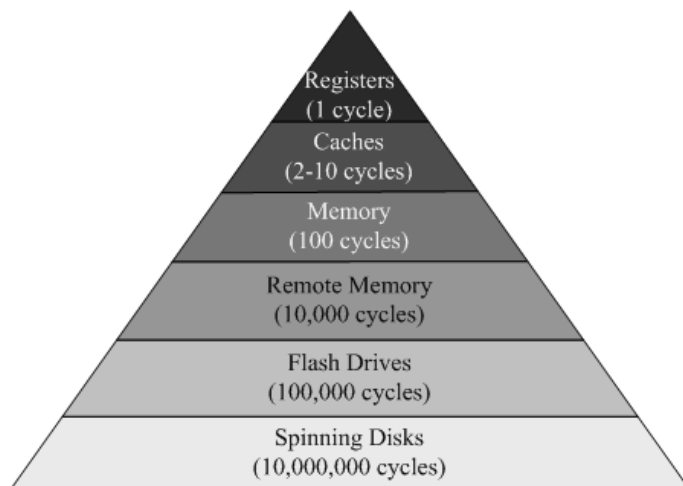


Figure 1: The existing memory storage hierarchy. There is a 5-order-of-magnitude latency gap between memory and spinning disks. Our solution is to adopt distributed shared memory and flash drives to fill the gap.

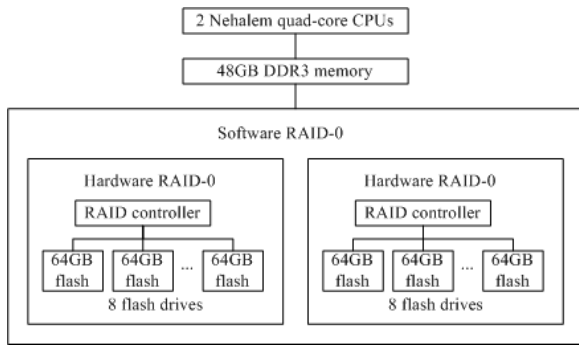


Figure 2: The original design of IO nodes. Each eight drives are grouped by a hardware RAID controller into a hardware RAID-0. Another software RAID-0 is set up on top of the two hardware RAID-0s. The best random read IOPS achieved is about 88K, which is about only 15% of the theoretical upper bound.

be able to catch up with the high-speed flash drives.

To explore the potentials and issues of the flash technology, we swept a large parameter space by fast and reliable measurements to investigate different design options. We would like to share some lessons we learned and suggestions for future architecture design.

2. DASH SYSTEM ARCHITECTURE

DASH is composed of four so-called supernodes connected by InfiniBand [5]. Each supernode is a 16-way cluster with 16 compute nodes and 1 IO node. All the nodes are equipped with two Intel[®] Nehalem quad-core 2.4GHz CPUs and 48GB DDR3 memory. Each IO node has in addition 16 Intel[®] X25-E [6] 64GB flash-based SSDs (Solid State Disks), with the total capacity of 1TB, serving the affiliated supernode. The whole supernode (including the compute nodes and the I/O node) is virtualized into a single system image by the vSMP system from ScaleMP[®] inc. [10]. From users’ perspective, a supernode has 128 cores, 768GB main memory, and 1TB flash drives. We are running a single Linux image above the vSMP system. In fact, DASH is just a prototype system for the even larger Teragrid system called Gordon coming next year. Gordon will have more (32) and larger (32-way) supernodes.

DASH is our prototype solution to the memory-disk latency gap problem. Two innovations are adopted to fill the gap: flash drives and distributed shared memory. With flash drives, we can accelerate IO by about 2 orders of magnitude in term of latency. Access to remote DRAM memory provided by the vSMP system can improve the performance by another order of magnitude albeit to a smaller(768GB) pool of storage. In the following sections, we will focus on the flash technology and try to explore the design space.

3. FLASH-BASED IO DESIGN SPACE EXPLORATION

Figure 2 shows our original design for IO nodes using hardware RAID plus software RAID. To connect the 16 flash drives, we adopted two Intel[®] RS2BL080 PCIe 2.0 RAID

controllers with 8 up-to-6Gb/s SAS/SATA ports each. Every eight drives are configured as a hardware RAID-0. Another software RAID-0 was set up above the two hardware ones. There are several RAID levels like 0, 1, 5, 6. Since our flash drives will be used as a working area instead of a permanent storage system, we don’t guarantee redundancy or reliability. To pursue high performance without any parity computation overhead, we only investigate RAID-0 in this paper. Without any tuning, the out-of-box random read performance is about 46K IOPS with 4KB requests. After exhaustive tuning, we achieved about 88K 4KB IOPS, which was about 2x improvement. However, since each X25-E drive can perform about 35K 4KB IOPS, the upper bound performance of 16 drives should be about 600K and we only obtained about 15% of the upper bound, which was quite disappointing. After some investigations, we realized that the bottleneck could be the embedded 800MHz IO processor of the RAID controller, which was designed for spinning disks and might not be able to work with the much faster flash drives. One obvious solution to the issue is to use faster RAID controllers. However, RS2BL080 was already the best RAID controller we could find at the time the machine was built (fall 2009). It seems the existing hardware RAID controllers are not ready for flash drives yet. In fact, we are not the only one encountering the issue. Previous work [13] observed the same problem. Another workaround is to avoid hardware RAID and adopt software RAID instead. By this approach, we can leverage our powerful Nehalem processors of the host. To verify our hypothesis that this might improve matters, we attached 6 flash drives to the on-board HBA (Host Bus Adapter) of the motherboard and repeated the test with software RAID. With only 6 drives, we could easily achieve about 150K IOPS, which is about 2x of hardware RAID performance.

With our hypothesis confirmed, we switched controllers for simple HBAs (LSI[®] 9211-4i) with 4 up-to-6Gb/s SAS/SATA ports each. Besides the software/hardware RAID issue, we believed there are more design dimensions, such as stripe size, stripe width (number of drives), file system, IO schedulers, queue depth, write caching, and read ahead, that are critical to the performance of the flash storage system. To explore the complete parameter space might cause exponential explosion in design space and seems infeasible. In fact, we can reason about the appropriate range for each dimension and limit the space to explore. In the following subsections, we will discuss how we chose the parameter space, set up the experiments, and what the results imply.

3.1 Experiment Configurations

Figure 3 shows the IO node configuration after switching to the LSI[®] HBAs. Instead of the hierarchical structure with hardware RAID plus software RAID, this time we simply group up all 16 drives with a single software RAID-0. There are quite a few IO benchmarks around and we chose one of the most accurate and stable ones: XDD [11]. To measure the pure performance of the flash drives without interference from OS buffer cache, we performed direct IO across all our tests.

Table 1 shows all the parameter dimensions we measured and their tested values. With direct IO, write caching and read ahead become irrelevant. Since we are targeting appli-

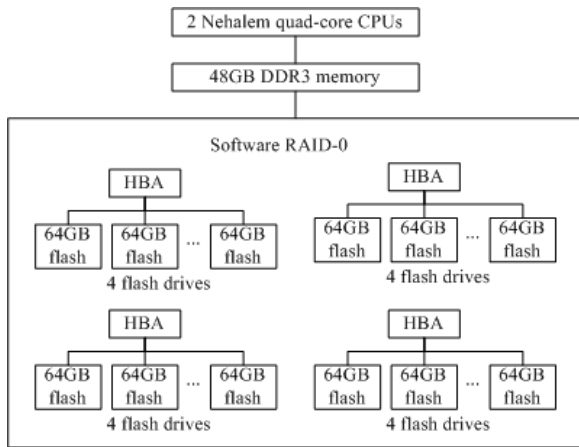


Figure 3: The IO node design after switching to simple HBAs. All 16 drives are set up as a single software RAID-0. The random read IOPS was improved by about 4x comparing with the original design up to about 350K.

cations with large amount of small random accesses, write caching and read ahead are not helping.

Stripe size is the chunk size in which a RAID spreads out data into different drives. The range from 16KB to 256KB should cover the reasonable sizes. Stripe width is the number of drives in a RAID. We are trying to test the numbers from 1 to 16 to investigate the performance scalability.

File systems may add some overhead to the system. We would like to compare the situation with and without file system to see how big the overhead is. XFS [12] is a file system designed for large compute and storage system with excellent performance and scalability. Read ahead is the mechanism the operating system tries to prefetch data the user will request. As referred to above, read ahead should not be effective with direct IO, which is confirmed by our results (not shown in this paper). Linux has four IO scheduler choices: No-op, CFQ, Deadline and Anticipatory. We explored to see how these scheduling algorithms perform with flash drives.

Request size means how big the requests generated by the IO benchmark (XDD in our case) are. 4MB should be large enough to guarantee the sequential access behavior and 4KB is a common size used in random tests and is found in many applications. Seek positions are the start addresses of the IO requests; these can be sequential or random. One thing worth mentioning is that random seeking does not necessarily mean random accesses. Random-seeking with 4MB requests still generates sequential accesses i.e. the pattern is occasional repositioning of start address followed by a long sequential access. Queue depth is the number of outstanding IO requests that can be outstanding at a given time. Small numbers like 1, 4, 16 should be enough for sequential tests. For random tests, we will try large numbers like 32, 128, 512 to see what the performance impact is. The final dimension is read or write. Here we only consider 100% read and 100% write. We tested 14 combinations of the above

four parameters as follows. In the following sections, we will refer to them as test types and use the numbers from 1 to 14 to present them in some of our figures.

- 4MB sequential tests with sequential seeking, queue depth 1, read and write;
- 4MB sequential tests with random seeking, queue depth 1, 4, and 16, read and write;
- 4KB random tests with random seeking, queue depth 32, 128, and 512, read and write.

For each test, we repeated it five times. There were in total 16,800 runs (it can be seen why some search-space limiting is needed). By limiting each run to 10 seconds, we managed to finish the entire data gathering experiment in about four days - analysis took a lot longer.

For each run, the benchmark will report both bandwidth and IOPS (Input/Output Per Second). Though people usually talk about bandwidth of sequential tests and IOPS of random tests, they are in fact the same thing and can be translated to each other as long as you know the request size. Since our sequential tests and random tests are using different request sizes, we would like to present the bandwidth numbers for a uniform view when we talk about both sequential and random tests in the following sections.

3.2 Data Pre-processing

As referred to above, we repeated each test five times. After collecting the data, we pre-processed the data and searched for outliers with Chauvenet's criterion [3]. According to the criterion, with five measured data, one can be classified as an outlier if it is 1.65 standard deviations away from the mean value. With the criterion, we found 1155 outliers out of the 16,800 measured data (about 7 percent).

Figure 4 and Figure 5 show the average measured values over the 5 passes for read and write tests respectively. Here we can observe some interesting phenomena. The first value of each test tends to be off the trend. It seems the performances of the flash drives are quite sensitive to the pre-conditions, but they can adapt to the conditions very fast right after the first run. The write performance is more sensitive than the read one, which is almost constant.

With the above observations, we dropped all the data of the first pass and repeated the Chauvenet test. This time we found no outliers at all. Figure 6 and Figure 7 show the coefficients of variation across all the tests before and after the change. The data became much more stable after removing the outliers.

3.3 Stripe Size

The stripe size of a RAID is critical for performance. Small sizes are easier for applications to make use of the parallelism across the composing drives and increases the resultant bandwidth. However, too small sizes may cause significant overhead of striping and IO processing.

Figure 8 shows the average bandwidth with different stripe sizes. The X axis represents the test types referred above.

Table 1: Parameter Dimensions and Their Values

Parameters	Descriptions	Values
Stripe size	The chunk size of RAID	16KB, 64KB, 256KB
Stripe width	Number of drives	1, 2, 4, 8, 16
File system	With or without XFS	Raw, XFS
Read ahead	Linux prefetching	Off, On
IO scheduler	Linux IO scheduling algorithms	No-op, CFQ, Deadline and Anticipatory
Request size	Size of IO requests	4MB for sequential tests; 4KB for random tests
Seek position	Start addresses of IO requests	sequential-seeking, random-seeking
Queue depth	Number of outstanding IO requests	1, 4, 16 for sequential tests; 32, 128, 512 for random tests
IO operation	Read or write	100% read, 100% write

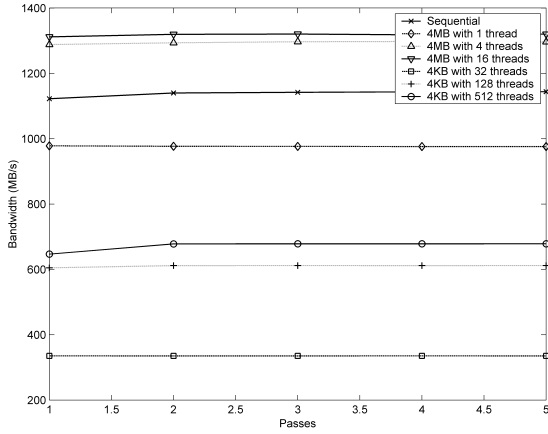


Figure 4: Average bandwidth over five passes of read tests. The first pass tends to off the trend.

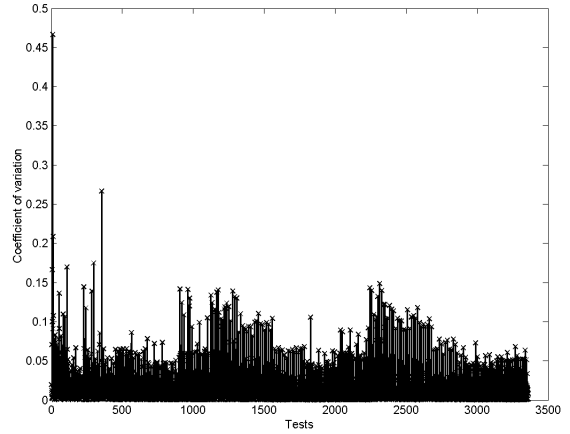


Figure 6: Coefficients of variation before outliers removal. With Chauvenet’s criterion [3], we found 1155 outliers out the 16,800 measured data.

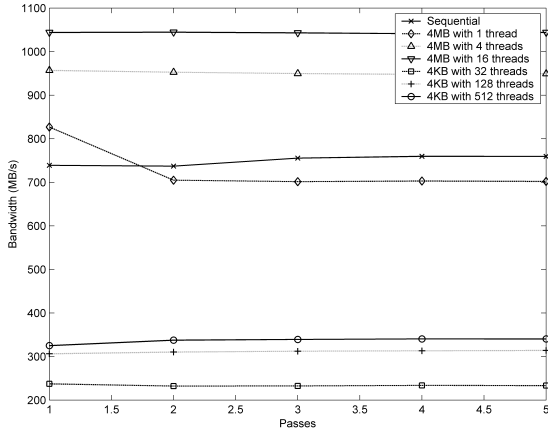


Figure 5: Average bandwidth over five passes of write tests. The first pass tends to off the trend. The write tests are more sensitive to pre-conditions but can adapt quickly right after the first run.

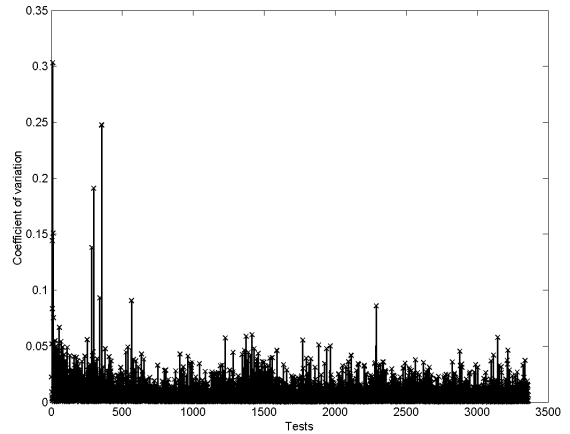


Figure 7: Coefficients of variation after outliers removal. After removing the first pass of each test, all the outliers are removed.

We can see that 16KB is too small to achieve reasonable performance. The performance of 64KB and 256KB are almost the same. 256KB is a little bit better.

Since our random test size (4KB) is smaller than all the stripe sizes, a single request would not be striped across more than one drive. However, our queue depths are all larger than the drive number (16) and the requests should

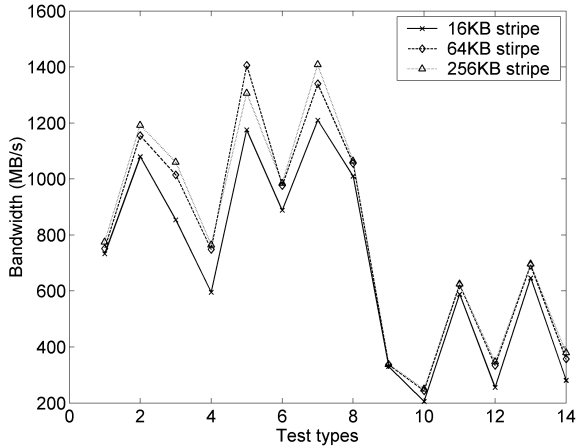


Figure 8: Average bandwidth with different stripe sizes. Deciding stripe size is a trade-off between parallelism and striping overhead.

spread out evenly onto different drives. In this situation, smaller stripe sizes cannot take advantage of parallelism. Instead, they suffer from the striping overhead. For sequential tests with the request size of 4MB, all the stripe sizes tested can benefit from IO parallelism. Within the range, smaller stripe sizes will suffer from the striping overhead again. As a conclusion, the optimal stripe size depends on the application characteristics. The sweet spot may be the largest stripe size to make the requests span over all the drives. Of course the queue depth is also a factor. We will have more discussion of this later in the queue depth section.

3.4 Stripe Widths and Performance Scalability

The stripe width (the number of composing drives) of a RAID is another important factor for performance. Ideally, we would like to see the bandwidth or throughput (like IOPS) scale up with the number of drives. We are especially interested in the IOPS performance of the random tests for a couple of reasons. First, our target applications are dominated by small random accesses. Also, since the IOPS number of a single flash drive is already pretty high compared to spinning disks, scaling the performance up can be challenging for the surrounding components such as RAID. In this section, we will see how the random IOPS scale up from 1 drive to 16 drives.

Figure 9 shows the random read IOPS scaling over the number of drives. The good news is that comparing with the 88K IOPS result of our original design with hardware RAID, we managed to improve the performance by about 4x, leveraging the horsepower of our Nehalem host processors. How about the scalability? Here things become more tricky. The performance scales almost linearly up to 8 drives, which is great. However, it almost stops scaling after 8 drives. With 16 drives, the performance can only scale up a little bit to about 350K IOPS from 250K with 8 drives.

Our further investigation showed that the issue origins from

the high IOPS of flash drives, which introduces a flood of hardware interrupts. The existing Linux kernel and HBA drivers do not work well enough to balance the workload to all the CPU cores. To be more specific, there is an existing issue [9] in the current kernel to configure interrupt binding for devices using message signaled interrupts (MSI) [7] but without MSI per-vector masking capability, which is the case for our HBAs. To work around the issue, we disabled the MSI and fell back to the legacy pin-based interrupt. Figure 10 shows the same results with the new configuration. With MSI disabled, now the random read performance can scale almost linearly up to 16 drives with the queue depth of 512. The peak performance is about 460K IOPS, which is about 80% of the theoretical upper bound (600K IOPS). However, the legacy interrupt may have some disadvantages such as limited number of interrupts and higher overheads. As you might have noticed, the performances with queue depth of 32 drop about 2x. That is because we don't have enough outstanding requests to overlap and hide the long overheads in these tests.

The curves with the same queue depth are represented with the same color in both the figures. It is easy to see that the queue depth is very important. More threads usually means better performance.

3.5 File Systems

File system is an important factor at the operating system level. Since most of the existing file systems were designed before flash drives came into batch production, they might not be able to work with flash drives well. We are interested in figuring out what the overhead of existing file systems will be. Here we chose XFS [12] as an example. XFS was designed for large compute and storage system from day one and is famous for its excellent performance and scalability. It will be a good fit for our large-scale data-intensive super-computer if its overhead on flash drives is small enough.

Figure 11 compares the bandwidth with and without XFS. The sequential performances (the first eight categories in the figure) are almost the same. That is because the overhead of the file system is amortized because of the large request size. However, the overhead becomes dominant in the random tests and XFS performs quite poor. One interesting phenomenon is that the random write performances (the categories 10, 12, and 14 in the figure) drops dramatically, whose reason is still under investigation. It will be interesting to also compare the performance with other file systems such as ext4 [4] or Brtfs [2]. At this point, we would like to save it as our future work.

3.6 IO Schedulers

For traditional spinning disks, people developed all kinds of IO scheduling algorithms mainly to serve three purposes: adjacent request merging, request reordering (elevator scheduling), and request prioritizing. Most of these optimizations are redundant for flash drives. For example, since there is no head moving and the seeking cost is trivial for flash drives, elevator scheduling is not necessary. We don't need request prioritizing either for our target applications. Without any benefits, these optimizations might even introduce unnecessary overhead for flash drives. There are four IO schedulers in Linux: No-op, CFQ (Completely Fair Queuing), Dead-

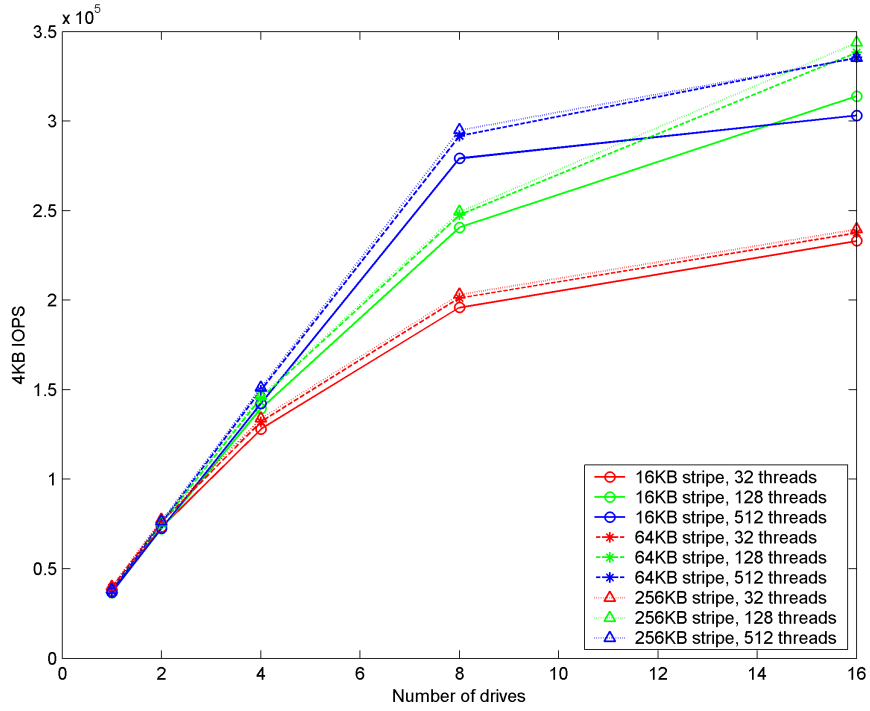


Figure 9: Random read IOPS scaling over drive amount. It scales almost linearly up to 8 drives but not after that.

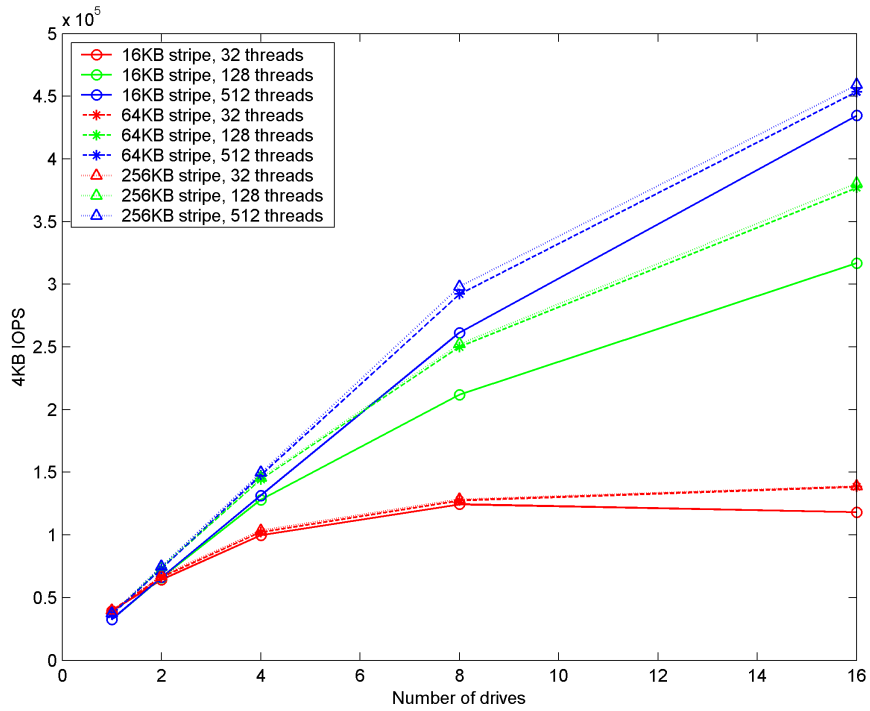


Figure 10: Random read IOPS scaling over drive amount without MSI-X. With MSI disabled, it scales almost linearly up to 16 drives.

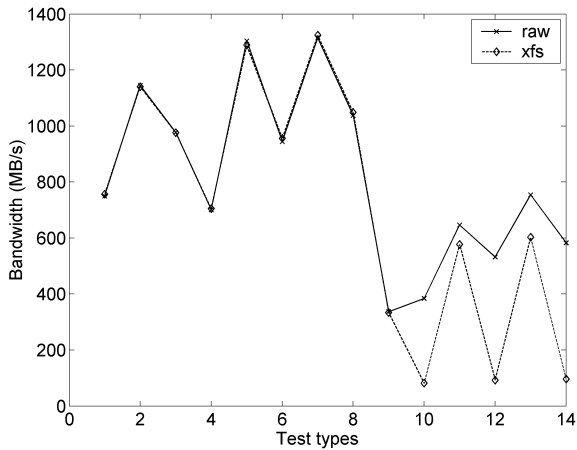


Figure 11: Average bandwidth with and without file system. The sequential performances with and without XFS are almost the same while the XFS's random (especially write) performances are worse.

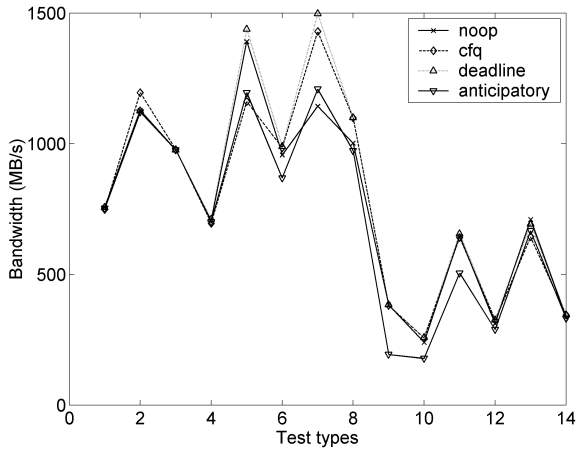


Figure 12: Average bandwidth with different IO schedulers. Simple algorithms like No-op and Deadline work best. Most advanced optimizations designed for spinning disks, such as elevator scheduling, are not necessary, even harmful for flash drives.

line, and Anticipatory. No-op is the simplest one with only request merging function. We would like to see if the other advanced algorithms can bring any benefits for flash drives.

Figure 12 shows the average bandwidth with different IO schedulers. Deadline is the best for sequential (random-seeking with 4MB requests) tests; No-op is the best for random (with 4KB requests) tests; CFQ varies a lot; while Anticipatory is systematically bad. We can see that advanced optimizations designed for spinning disks are not necessary for flash drives. Simple algorithms like No-op and Deadline work best.

3.7 Queue Depths

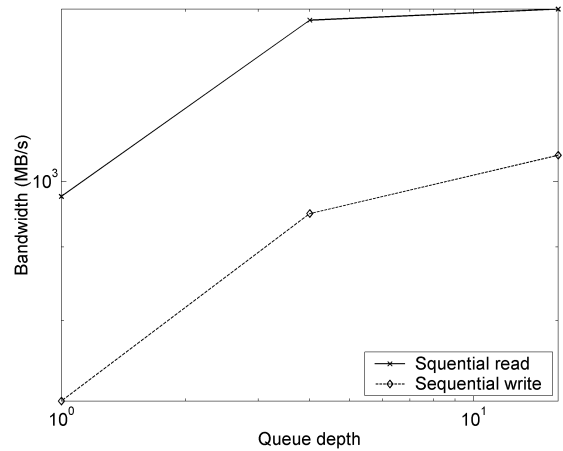


Figure 13: Average bandwidth of sequential tests with different queue depths 1, 4 and 16. Although the request size (4MB) can span across all the drives, higher queue depth can still improve bandwidth because of the internal parallelism of each drive.

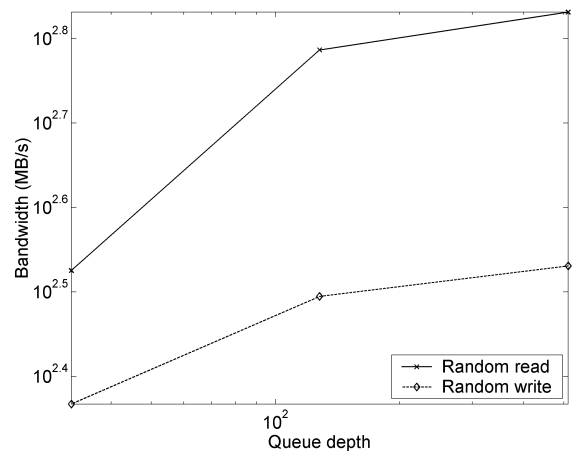


Figure 14: Average bandwidth of random tests with different queue depths 32, 128 and 512. Performance increases until the queue depth 128 only because 512 exceeds the aggregated parallelism of the 16 drives.

A traditional spinning disk is a serial device, which can only access one data block at a given time. Flash drives are totally different. Even a single drive may contain tens of data buses and packages internally, which can be accessed in parallel. To explore the full potential of a flash drive, a user has to utilize asynchronous IO or multi-thread to keep the drive busy.

Figure 13 and Figure 14 show the average bandwidth with different queue depths for sequential and random tests respectively. Both figures are logarithmic. All our tests (sequential or random, read or write) shared a similar rising trend with increasing queue depth. Even though the request size (4MB) of the sequential tests is large enough to

span all the composing drives, higher queue depth can still improve bandwidth. The extra performance improvement comes from the internal parallelism of each drive. For random tests, we observed a similar trend until the queue depth reaches 128. This makes sense because 512 is a big number for 16 drives with tens of internal packages each. In another word, queue depth of 512 exceeds the aggregated parallelism of the 16 drives.

4. CONCLUSIONS

Data-intensive HPC applications are becoming more and more common. Existing memory storage hierarchy has a 5-order-of-magnitude gap between memory and spinning disks, and is not able to respond to the challenge well. We adopted distributed shared memory and high-speed flash drives to fill the gap, and built the prototype system called DASH, which is a Teragrid resource. In this paper, we focused on the flash technologies and tried to explore the complete design space of a flash storage system. We swept a large multi-dimensional parameter space to figure out how the other system components, such as operating system and RAID (hardware and software), interfere with flash drives. With these exhaustive searching, we managed to improve the performance from our original design by about 9x. However, we found that some existing technologies like RAID don't fit the new technology very well. In the future, we will keep investigating the origins of these limitation and try to remove them and release the potential of the flash technology. We are also interested in other emerging storage technologies such as PCM (Phase Change Memory) and STTM (Spin-Torque Transfer Memory). Our goal is to re-shape the memory storage hierarchy to fit the high performance data trend ahead.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge Gordon team members including Michael Norman, Shawn Strande, Arun Jagatheesan, Eva Hocks, Larry Diegel, Mahidhar Tatini, Thomas Hutton, Steven Swanson, Lonnie Heidtke and Wayne Pfeiffer.

This work was sponsored in part by the National Science Foundation under NSF OCI #0951583 entitled "I/O Modeling EAGER", by NSF OCI #0910847 entitled "Gordon: A Data Intensive Supercomputer".

6. REFERENCES

- [1] 3leaf system inc. <http://www.3leafsystems.com/>.
- [2] Btrfs project. https://btrfs.wiki.kernel.org/index.php/Main_Page.
- [3] Chauvenet's criterion page at wikipedia. http://en.wikipedia.org/wiki/Chauvenet%27s_criterion.
- [4] EXT4 page at Wikipedia. <http://en.wikipedia.org/wiki/Ext4>.
- [5] InfiniBand page at Wikipedia. <http://en.wikipedia.org/wiki/InfiniBand>.
- [6] Intel x25-e datasheet and technical documents. <http://download.intel.com/design/flash/nand/extreme/extreme-sata-ssd-datasheet.pdf> and <http://www.intel.com/design/flash/nand/extreme/technicaldocuments.htm>.
- [7] Message Signaled Interrupts page at Wikipedia. http://en.wikipedia.org/wiki/Message_Signaled_Interrupts.
- [8] Overview of recent supercomputers 2009. http://www.nwo.nl/nwohome.nsf/pages/NWOA_F7X8HXB_FEng.
- [9] RedHat bugzilla: Setting IRQ affinity does not work with MSI devices. https://bugzilla.redhat.com/show_bug.cgi?id=432451.
- [10] Scalemp inc. <http://www.scalemp.com/>.
- [11] XDD benchmark, version 6.5. <http://www.ioperformance.com/>, retrieved in September 2009.
- [12] XFS project. <http://oss.sgi.com/projects/xfs/>.
- [13] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and P. R. Design tradeoffs for ssd performance. pages 57–70. USENIX Annual Technical Conference, 2008.
- [14] D. Bader, editor. *Petascale Computing: Algorithms and Applications*. Chapman & Hall/CRC Press.
- [15] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ASPLOS '09*, pages 217–228, New York, NY, USA, 2009. ACM.
- [16] M. Chapman and G. Heiser. vnuma: A virtual shared-memory multiprocessor. USENIX Annual Technical Conference, 2009.
- [17] F. Chen, D. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. pages 181–192. SIGMETRICS/Performance, 2009.
- [18] N. M. et al. Bit error rate in nand flash memories. pages 9–19. IEEE International Reliability Physics Symposium, 2008.
- [19] A. Funk, V. Basili, L. Hochstein, and J. Kepner. Analysis of parallel software development using the relative development time productivity metric. *CT Watch*, 2:4A, 2006.
- [20] L. Grupp, A. Caulfield, J. Coburn, E. Yaakobi, S. Swanson, and P. Siegel. Characterizing flash memory: Anomalies, observations, and applications. MICRO, 2009.
- [21] P. Kogge, editor. *ExaScale Computing Study: Technology Challenges in Achieving Exascale System*. <http://www.sdsc.edu/~allans>.
- [22] S. Park and K. Shen. A performance evaluation of scientific i/o workloads on flash-based ssds. Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS'09), 2009.
- [23] A. Snaveley, X. Gao, C. Lee, N. Wolter, J. Labarta, J. Gimenez, and J. P. Performance modeling of hpc applications. ParCo, 2003.
- [24] A. Snaveley, G. Jacobs, and D. A. Bader, editors. *Workshop Report: Petascale Computing in the Biological Sciences*. <http://www.sdsc.edu/~allans>.
- [25] A. Snaveley, R. Pennington, and R. Loft, editors. *Workshop Report: Petascale Computing in the Geosciences*. <http://www.sdsc.edu/~allans>.
- [26] A. Szalay and J. Gray. Science in an exponential world. *Nature*, 440:413–414, 2006.