Jitter-Free Co-Processing on a Prototype Exascale Storage Stack

John Bent
john.bent@emc.com

Sorin Faibish
sfaibish@emc.com

Jim Ahrens
ahrens@lanl.gov

Gary Grider
ggrider@lanl.gov

John Patchett
patchett@lanl.gov

Percy Tzelnic
tzelnic@emc.com

Jon Woodring
woodring@lanl.gov

Abstract

In the petascale era, the storage stack used by the extreme scale high performance computing community is fairly homogeneous across sites. On the compute edge of the stack, file system clients or IO forwarding services direct IO over an interconnect network to a relatively small set of IO nodes. These nodes forward the requests over a secondary storage network to a spindle-based parallel file system. Unfortunately, this architecture will become unviable in the exascale era.

As the density growth of disks continues to outpace increases in their rotational speeds, disks are becoming increasingly cost-effective for capacity but decreasingly so for bandwidth. Fortunately, new storage media such as solid state devices are filling this gap; although not cost-effective for capacity, they are so for performance. This suggests that the storage stack at exascale will incorporate solid state storage between the compute nodes and the parallel file systems. There are three natural places into which to position this new storage layer: within the compute nodes, the IO nodes, or the parallel file system. In this paper, we argue that the IO nodes are the appropriate location for HPC workloads and show results from a prototype system that we have built accordingly. Running a pipeline of computational simulation and visualization, we show that our prototype system reduces total time to completion by up to 30%.

1 Introduction

The current storage stack used in high performance computing (HPC) is recently imperiled by trends in disk technology and harsh economic realities. The storage requirements in HPC are bursty and predictable. Although there is a small amount of other traffic, the vast majority of IO is comprised of frequent checkpoints and infrequent restarts. Although sites and workloads vary, there are several general rules of thumb. One, checkpoint and restart workloads are highly concurrent due to the tight coupling of the running jobs. Two, checkpoint frequency is on the order of a small number of hours. Three, HPC sites have a minimum required bandwidth in order to store a checkpoint quickly enough to ensure sufficient forward progress and a minimum required capacity to store a sufficient number of checkpoints for a variety of reasons such as time-series analysis and back-tracking due to computational steering.

Until recently, the number of disks required for capacity has been larger than the number required for bandwidth. In other words, buying the number of disks required for capacity has provided excess bandwidth essentially for free. However, disk capacity is increasing much faster than disk performance. New technologies such as shingled disks [4] are only exacerbating this trend. The result is that the number of disks required for capacity has now become fewer than the number required for bandwidth. Unfortunately, purchasing disks for bandwidth is cost-prohibitive [9]. Solid-state drives (SSDs) however are cost-effective for bandwidth but cost-prohibitive for capacity. Clearly, a hybrid hierarchy in which SSDs are used for checkpoint bursts and disks are used for checkpoint storage is the answer.

However, it is not immediately obvious whether to place SSDs inside the compute nodes, inside the storage system, or somewhere in the middle. Although each placement may be appropriate for some workloads, we believe that placement in the middle is best for HPC. Placement on the compute nodes is problematic due to computational jitter [17] as any perturbation caused by asynchronously copying data from the SSDs to the disks can ripple devastatingly through the tightly-coupled computation [11]. Placement inside the storage system is also problematic as it would require a more expensive storage network matched to the higher bandwidth of the SSDs instead of being matched to the lower bandwidth of the disks.

Therefore, in this paper we examine a prototype exascale storage stack built with SSDs placed in spe-

978-1-4673-1747-4/12/$31.00 ©2012 IEEE

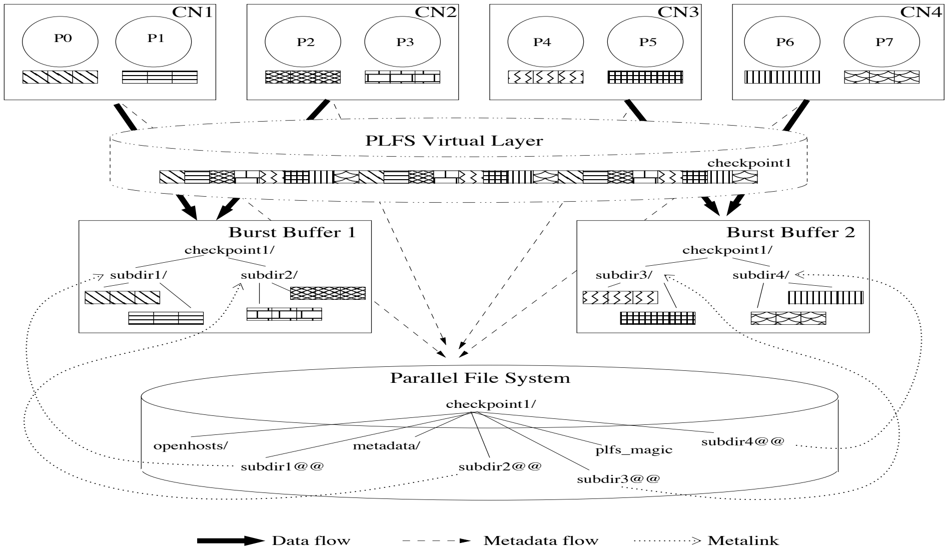


Figure 1: **A Burst Buffer Storage Architecture.** *Eight processes spread across four compute nodes intersperse writes into a shared file. By using the PLFS middleware layer, the illusion of a single file is preserved in a manner completely transparent to the application and the user. Physically however, PLFS transforms the IO to leverage both the global visibility of the parallel file system as well as the faster performance of the SSDs in the burst buffers.*

cialized *burst buffers* stitched transparently into the storage stack using a modified version of the PLFS middleware [6]. Evaluations with a workload of simulation and visualization show a speedup in total time to completion of up to thirty percent. The remainder of the paper is as follows: we describe our architecture in Section 2, our results in Section 3, related work in Section 4, and our conclusions in Section 5.

2 Design and Implementation

Our design consists of both a new hardware architecture and a modified software stack. The hardware architecture is relatively straight-forward. In order to transparently weave the SSDs into the storage stack, we extend the notion of IO nodes [3, 10, 20] to add buffering to quickly absorb the bursty checkpoint workloads; we call these augmented IO nodes *burst buffers*. As is typical of IO nodes, we place the burst buffers within the computational fabric and also attach them to the secondary storage network.

The modified software stack is a bit more complex. We extend the PLFS middleware layer [6] to incorporate the burst buffers. PLFS is a middleware virtual file system which transparently interposes on application IO. This interposition can dramatically improve write bandwidth by rearranging the workload into one requiring less locking within the parallel file system (PFS). This is primarily achieved by decoupling writes to a shared logical file into writes to multiple physical files; each physical file is then written by only a single process. PLFS also maintains metadata sufficient to reconstruct the logical file for reads.

PLFS stores file data and its metadata into *containers*. Containers are implemented as physical directories on the underlying file system which PLFS uses as its store. *Subdirs* within the container distribute the large number of files within the container into multiple directories to avoid performance problems with too many files in a single directory [15]. Since version 2.0 [5], PLFS has used a new abstraction called *metalinks* which allow the subdirs to physically exist in *shadow* containers. This allows sites with multiple metadata servers to configure PLFS such that its load is distributed across them.

By default, PLFS randomly places subdirs; we modified it to allow each compute node to specify a location for its subdirs. In this way, the relatively small amount of PLFS metadata is stored on the PFS but the larger amount of file data is stored in the burst buffers as shown in Figure 1. We then augmented the PLFS API by adding two functions to allow the user to control the management of the burst buffers: one to start an asynchronous copy of the file data from the shadow subdirs into into the container on the PFS and a second to remove the shadow subdirs and their contents.

An important feature of PLFS is that it is transparent to users and applications and runs with unmodified programs. Although we preserved this as much as possible, existing interfaces such as POSIX and MPI-

8 Compute Nodes	2 Burst Buffers	Storage	2 File System Blades
• 40 GB RAM • Dual socket, quad core, 2.8 GHz Intel Nehalem • Mellanox QDR Infiniband port	• 128 GB RAM • Quad socket, 10 core, 3.1 GHz Intel Westmere • 512 core nVidia Tesla 2090 GPGPU • 4 Mellanox QDR Infiniband ports • 16 Samsung 200 GB SSDs	• EMC VNX 7500 • 24 GB RAM • 10 Near-Line 2TB 7200 RPM SATA drives • 4 8gb fiber channel ports	• Lustre 1.8.0 • 16 GB RAM • Dual socket, quad core, 2.8 GHz Intel Nehalem • Mellanox QDR Infiniband port • 2 8gb fiber channel ports

Table 1: **Evaluation System Specifications.** *The SSDs on the burst buffers were grouped into four RAID-0 arrays, mounted using ext4, and exported via NFS. In all cases, the operating system was 64-bit CentOS 6.0. Both file system blades ran a Lustre Object Storage Server; one also ran a metadata server which used a local Samsung 200 GB SSD. The fiber channels were directly connected; the other ports used a Mellanox QDR Infiniband switch.*

IO do not support burst buffer management: thus we were forced to modify our simulation to initiate the asyncronous copy and our visualization to remove the shadows. Aside from these modifications, the burst buffers were transparent to the user and the application. The files were always accessed via the same path regardless of the data's physical location; the only difference perceivable is a faster bandwidth when the data is moving to or from the burst buffers.

3 Evaluation

To evaluate our design, we built a small prototype burst buffer system consisting of eight compute nodes, two burst buffers, and a PFS. On the compute nodes, we ran the PLFS client through which the simulation could checkpoint and restart. The client was also on the burst buffers in order for the visualization to read the checkpoint data. Lustre running on an EMC VNX 7500 served as our PFS. Hardware specifications are listed in Table 1 and a depiction is shown in Figure 2.

The workload consisted of a simulation running for eight timesteps on the compute nodes. After each timestep, the simulation checkpointed using the PLFS driver in the MPI-IO library. Each checkpoint file was then processed by a visualization program which accessed the file via POSIX through a FUSE mount of PLFS.

The simulation was a fluid dynamics model coupled with a wind turbine model [19] used to study effects of terrain and turbine placement on other downstream wind turbines. The software used for the visualization was ParaView [1], an open-source, large-scale, parallel visualization and analysis tool. The visualization analyzed the forces on the wind turbine, the air flow, and the vortex cores created by the movement of the wind turbine.

We ran the same workload using two configurations. One using the compute nodes and the PFS but without the burst buffers; the second added the burst buffers. For the remainder of the paper, we'll refer to the first configuration as *NoBB* and the second as *WithBB*. Using NoBB the visualization was *post-processed*: it did not run until the eight computational timesteps and checkpoints were complete. By adding the burst buffers however, we were able to *co-process* the visualizations by running them on the burst buffers and pipelining them with the simulation. Additionally, the checkpoint latency between computational timesteps was reduced with WithBB since the checkpoint bandwidth was increased approximately 400%. Remember that the checkpoints were still ultimately saved on the PFS; WithBB just allowed that slower data movement to be asynchronous and pipelined with the computation.

The results are shown in Figure 3. Notice that the simulation, checkpoint, and visualization phases are serialized using NoBB whereas they are pipelined when using WithBB. These graphs show that WithBB provides four distinct benefits. First, the simulation phase finishes about ten percent more quickly. Second, the complete workload finishes about thirty percent more quickly. Third, because the visualization is co-processed instead of post-processed, computational steering is possible much earlier (after about four minutes instead of thirty) as is shown by the elapsed time when the first visualization completes. Finally, since WithBB allows the visualization and asynchronous copying to the PFS to be done on the burst buffers, there is no jitter introduced and therefore the compute phase is not slowed as it might be if the SSDs were placed inside the compute nodes. This can be observed from the duration of each individual simulation phases (*i.e.* the widths for the simulation bars are the same).

We acknowledge that it is not completely fair to compare NoBB to WithBB since WithBB uses additional hardware. However, we believe our results are still valid for several reasons. First, the changing economics of disks and SSDs seem to dictate the need for SSDs in the exascale era. Our work is an important early exploration in this area. Additionally, the thirty percent speedup that we observe in our work-

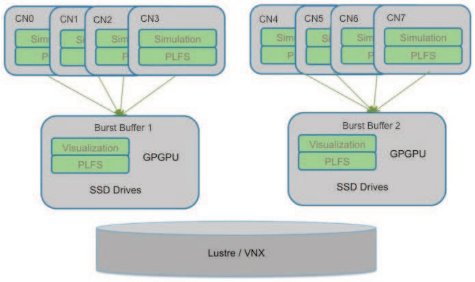


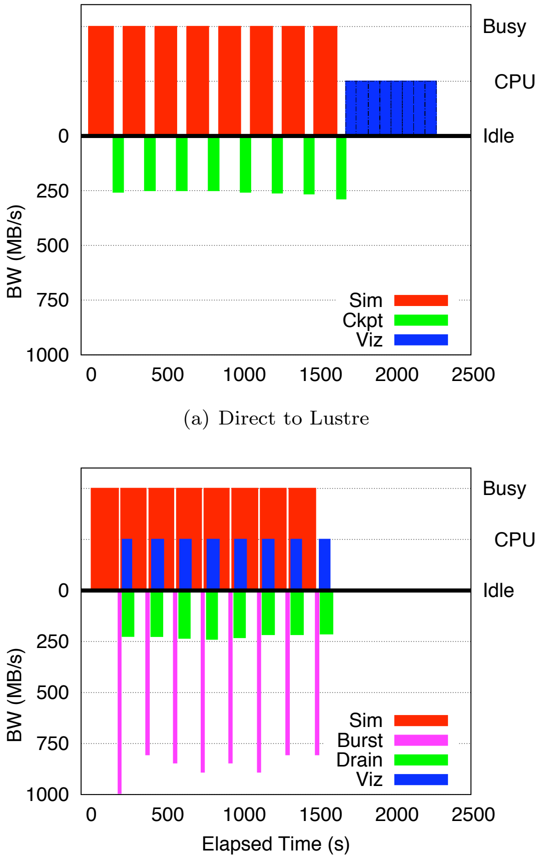
Figure 2: **Our Prototype Burst Buffer System.** *This figure shows the prototype exascale storage system we built using eight compute nodes in which the wind-turbine simulation saved checkpoints through PLFS into two burst buffers. Our visualization program then ran on the burst buffers and read the wind-turbine data through PLFS as the data was simultaneously copied onto the parallel file system. After it was no longer needed, the copy on the burst buffers was removed in order to reclaim space.*

load possibly under-represents the potential of burst buffers because we evaluated only one small workload that may not be completely reflective of anticipated exascale workloads.

One reason is that the simulation we used checkpointed only five percent of the total RAM in the compute cluster (16 out of 320 GB). Exascale applications may checkpoint a much larger ratio; therefore reductions in checkpoint time will reduce total time to completion more than we have shown here. A second reason is that our simulation was configured to checkpoint at a fixed interval of computation. Figure 3 shows that the simulation checkpointed the same number of times even though it ran for a shorter time. In production settings, applications tend to adjust themselves to checkpoint at a fixed interval of wallclock time. Therefore, a reduction in checkpoint latency will increase computational efficiency even more than we have shown here.

4 Related Work

Performing visualization and analysis simultaneously with a running simulation is done using one of two different techniques, *in situ* and *co-processing*. *In situ* performs visualization and analysis in the same process space as the simulation code, usually through linked library analysis code. This allows the visualization and analysis to have direct access to simulation memory and to perform computations without data movement. Several large-scale *in situ* analysis libraries have recently emerged [8, 16, 23] as have several that are application-specific [2, 21, 24]. Although *in situ* avoids a data copy, it can not be pipelined with the simulation as we have done. Both *in situ* and co-processing will be used in exascale.



(a) Direct to Lustre

(b) Using Burst Buffers

Figure 3: **Workflow.** *Both graphs show the elapsed time running the same workload of eight timesteps of the wind-turbine simulation, eight checkpoints, and eight visualizations. The upper graph shows the total time to completion using the default system without the burst buffers and post-processed visualization; the lower with the burst buffers and the co-processing they enable. In both graphs, the height above the x-axis merely indicates whether processing is occuring (the differing heights for the simulation and the visualization are merely to differentiate between the two); the height below the x-axis indicates bandwidth (lower is better).*

Other examples of co-processing analyses have been performed with scientific data staging technologies, including ADIOS [12] and GLEAN [22]. Our contribution is an approach which leverages file system semantics so that unmodified applications can use our burst buffer system using standard POSIX and MPI-IO interfaces.

As opposed to placing storage in the middle as we have studied here, other work [7, 14] studied similar workloads in which they place storage on the compute nodes. Although this was not measured in their work or ours, we fear this placement can slow down the foreground computation due to jitter [11, 17].

Another project to use SSDs between the compute nodes and the storage system [18] also uses SSDs as a temporary store before migrating data to the PFS. Unlike our work however, they use SSDs from a subset of the compute nodes that have SSDs attached. Although this places the SSDs on the compute nodes, they can choose to use them as burst buffers in order to avoid jitter. Additionally, our work leverages the IO nodes in existing HPC architectures which allows a reduction in the size of the storage network.

Finally, IOFSL [3] aggregates the IO from multiple compute nodes onto a smaller number of IO nodes. Similar to our work, SCR [13] has also shown large checkpoint improvements by bursting checkpoints to the memory of neighboring nodes and then asynchronously migrating them to the PFS. We believe that both of these, or similar technologies, are likely to be included in the exascale storage stack. These projects are complementary to ours and we are currently working with both to integrate all three.

5 Conclusion

Economic trends in storage technologies and the storage requirements for exascale computing indicate that SSDs will be incorporated into the HPC storage stack. In this paper, we have provided an important initial exploration of this large design space. Using commodity hardware, modified HPC file system middleware, we evaluated our design with a real HPC workload consisting of simulation and analysis. We demonstrated that placing SSDs in between the compute nodes and the storage array allow jitter-free coprocessing of the visualization and reduce total time to completion by up to thirty percent.

We are currently exploring ways to leverage the burst buffers to enable Map-Reduce style analytics as well as incorporating pre-staging of data sets into the scheduler. Additionally, we are developing an analytical model to help determine the ratio between burst buffers and compute nodes for large systems. Finally, we are continuing to test our design and software on increasingly large system to study its scalability.

References

[1] J. Ahrens, B. Geveci, and C. Law. Paraview: An end user tool for large data visualization. *The Visualization Handbook*, pages 717–731, 2005.

[2] J. Ahrens, K. Heitmann, M. Petersen, J. Woodring, S. Williams, P. Fasel, C. Ahrens, C. Hsu, and B. Geveci. Verifying scientific simulations via comparative and quantitative visualization. *IEEE Computer Graphics and Applications*, 30(6):16–28, 2010.

[3] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, and L. Ward. Scalable i/o forwarding framework for high-performance computing systems. In *IEEE International Conference on Cluster Computing*, Cluster 2009, New Orleans, LA, Sept. 2009.

[4] A. Amer, D. D. E. Long, E. L. Miller, J.-F. Pris, and T. Schwarz. Design issues for a shingled write disk system. In *26th IEEE Symposium on Massive Storage Systems and Technologies*, MSST 2010, May 2010.

[5] J. Bent et al. Plfs 2.0. sourceforge.net/projects/plfs/files/.

[6] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 21:1–21:12, New York, NY, USA, 2009. ACM.

[7] D. Camp, H. Childs, A. Chourasia, C. Garth, and K. I. Joy. Evaluating the Benefits of An Extended Memory Hierarchy for Parallel Streamline Algorithms. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization*, LDAV. IEEE Press, 2011.

[8] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96. IEEE, Oct. 2011.

[9] G. Grider. Speed matching and what economics will allow. HEC-FSIO 2010, Aug. 2010.

[10] G. Grider, H. Chen, J. Nunez, S. Poole, R. Wacha, P. Fields, R. Martinez, P. Martinez, S. Khalsa, A. Matthews, and G. A. Gibson. Pascal - a new parallel and scalable server io networking infrastructure for supporting global storage/file systems in large-size linux clusters. In *IPCCC'06*, pages –1–1, 2006.

[11] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing variability in the io performance of petascale storage systems. In *SC '10: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2010. ACM.

[12] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, CLADE '08, page 1524, New York, NY, USA, 2008. ACM.

[13] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[14] X. Ouyang, S. Marcarelli, and D. K. Panda. Enhancing checkpoint performance with staging io and ssd. In *Proceedings of the 2010 International Workshop on Storage Network Architecture and Parallel I/Os*, SNAPI '10, pages 13–20, Washington, DC, USA, 2010. IEEE Computer Society.

[15] S. V. Patil, G. A. Gibson, S. Lang, and M. Polte. GIGA+: Scalable Directories for Shared File Systems. In *Petascale Data Storage Workshop at SC07*, Reno, Nevada, Nov. 2007.

[16] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H. Shen, T. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 105–112. IEEE, Oct. 2011.

[17] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 55–, New York, NY, USA, 2003. ACM.

[18] R. Prabhakar, S. S. Vazhkudai, Y. Kim, A. R. Butt, M. Li, and M. Kandemir. Provisioning a multi-tiered data staging area for extreme-scale machines. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 1–12, Washington, DC, USA, 2011. IEEE Computer Society.

[19] E. K. Rodman R. Linn. Determining effects of turbine blades on fluid motion, 05 2011.

[20] G. Shipman, D. Dillow, S. Oral, and F. Wang. The spider center wide file system: From concept to reality. Cray User Group Conference, May 2009.

[21] A. Tikhonova, C. Correa, and K. Ma. Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1551–1559, 2010.

[22] V. Vishwanath, M. Hereld, and M. E. Papka. Toward simulation-time data analysis and I/O acceleration on leadership-class systems. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–14. IEEE, Oct. 2011.

[23] B. Whitlock, J. Favre, and J. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 100–109, 2011.

[24] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In situ sampling of a Large Scale particle simulation for interactive visualization and analysis. *Computer Graphics Forum*, 30(3):1151–1160, June 2011.