FusionFS: a distributed file system for large scale data-intensive computing

Dongfang Zhao*, Chen Shou*, Zhao Zhang†
Iman Sadooghi*, Xiaobing Zhou*, Tonglin Li*, Ioan Raicu*‡
*Department of Computer Science, Illinois Institute of Technology
†Department of Computer Science, University of Chicago
‡Mathematics and Computer Science Division, Argonne National Laboratory

I. Introduction

Today's science is generating datasets that are increasing exponentially in both complexity and volume, making their analysis, archival, and sharing one of the grand challenges of the 21st century. Exascale computing, i.e. 10^{18} FLOPS, is predicted to emerge by 2019 with current trends. Millions of nodes and billions of threads of execution, producing similarly large concurrent data accesses, are expected with the exascale.

Current state-of-the-art yet decades long storage architecture of high-performance computing (HPC) systems would unlikely provide the support for the expected level of concurrent data access. The main critique comes from the topological allocation of compute and storage resources that are interconnected as two cliques. Even though the network between compute and storage has high bandwidth and is sufficient for compute intensive petascale applications, it would not be adequate for data-intensive petascale computing or the emerging exascale computing (regardless if it is compute or data intensive).

We introduce FusionFS, a distributed filesystem particularly crafted for extreme scale HPC systems. FusionFS leverages FUSE [1] to work in user space and provides a POSIX interface, so that neither the OS kernel nor applications need any changes. Non-Volatile Memory(NVM) has proven to offer large gains for high-performance I/O-intensive applications [3], and FusionFS complies with Gordon [4] architecture by taking local NVM as local storage coexisting with processors. FusionFS has a completely distributed metadata management based on an implementation of distributed hash table (i.e. ZHT [7]) to achieve a scalable metadata throughput. FusionFS also delivers a scalable high I/O throughput based on maximizing the data locality in typical read/write data access patterns.

II. Design and Implementation

Figure 1 illustrates the allocation of different node types in a typical supercomputer setup, i.e. IBM BlueGene/P. The traditional parallel filesystem (e.g. GPFS) is mounted on the storage nodes. The fact that compute nodes need to access the remotely connected storage nodes was not an issue for compute-intensive applications. However this architecture would seriously jeopardize large scale data-intensive applications. Burst Buffer [8] alleviates the issue in the sense of elevating data from storage nodes to I/O nodes as a persistent cache. This architecture clearly has at least two advantages: (1) the network latency is improved by reducing the hops from 2 to 1, conceptually; (2) the data concurrency is increased from O(100) to O(1K). Nevertheless, Burst Buffer is still a "remote" storage from the perspective of compute nodes. We propose that each compute node should actively participate into both the computation and the data I/O, which is illustrated as the green layer. This would fully exploit the high speed bandwidth (e.g. 3D-torus) between compute nodes, and make data locality explicit for computation.

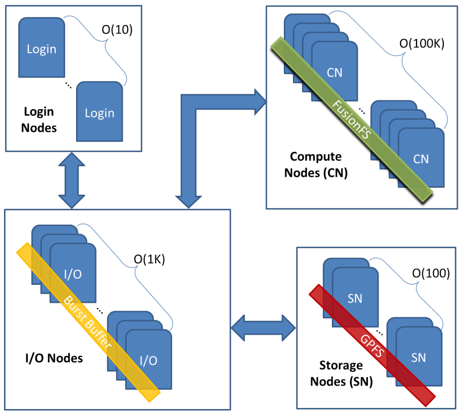

Fig. 1. Storage spectrum of IBM BlueGene/P

FusionFS is implemented with C/C++ and Shell scripts, excluding two third-party libraries: the Google Protocol Buffers [2] and UDT [5]. The software stack of FusionFS is shown in Figure 2. Three services (metadata, data transfer, and provenance) are on top of the stack, that are supported by FusionFS Core and FusionFS Utilities interacting with the kernel FUSE module.

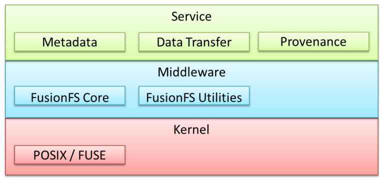

Fig. 2. FusionFS software stack

III. Evaluation

All experiments in this section were conducted on an IBM BlueGene/P supercomputer Intrepid [6] at Argonne National Laboratory, unless otherwise mentioned.

For metadata performance, as shown in Figure 3, when each node creates 10K files in its private directory at 1K-node scale, FusionFS has nearly two orders of magnitude higher performance over GPFS. The gap between GPFS and FusionFS metadata access cost will continue to grow as 8 nodes seem to be enough to saturate the metadata servers of GPFS.

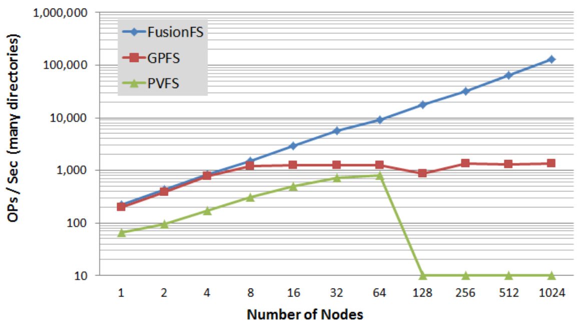

Fig. 3. File create (many directories)

For throughput, Figure 4 shows that FusionFS is only slightly better than GPFS and PVFS on a single node. FusionFS shows linear scalability from 1 to 1K nodes. PVFS shows its limitation on scalability: the throughput starts to drop down at 1K nodes. Up to 64 nodes, GPFS is flattened because every 64 compute nodes share one I/O node on Intrepid, forming a PSET. This is the reason why GPFS does not seem to scale up to 64 nodes, but does beyond that.

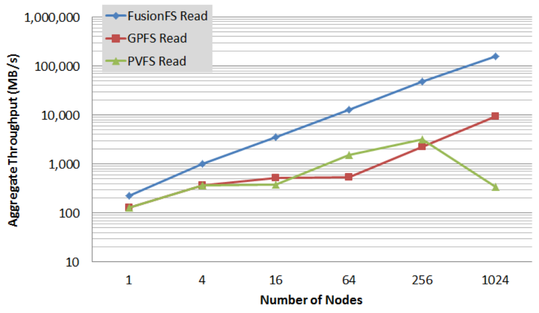

Fig. 4. Read Throughput (block size 128KB)

We show how real applications perform on FusionFS. BLAST (the Basic Local Alignment Search Tool) searches one or more nucleotide or protein sequences against a sequence database and calculates similarities. It has been implemented with different parallelized frameworks, e.g. ParallelBLAST [9]. We carried out a weak scaling experiment with ParallelBLAST with 4GB database on every 64-nodes (i.e. PSET), and increased the database size proportionally to the number of nodes.

As shown in Figure 5, there is a huge (more than 1 order of magnitude) performance gap between GPFS and FusionFS at all scales (except for the trivial 1-node case). FusionFS has up to 32X speedup (i.e. at 512 nodes), and an average of 23X improvement between 64-nodes and 1024-nodes. At 1 node scale, we believe the GPFS kernel module to be more effective in accessing an idle parallel filesystem.

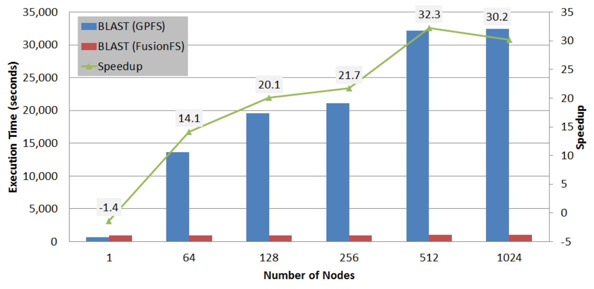

Fig. 5. BLAST execution time

IV. Conclusion and Future Work

This work proposes to break the accepted practice of segregating storage resource from computational resources, and to leverage the abundance of processing power, bisection bandwidth, and local I/O commonly found in today's and future high-end computing systems. We believe the radical storage architecture changes proposed by FusionFS will make future exascale computing more tractable.

References

[1] FUSE Project. http://fuse.sourceforge.net/.
[2] Google protocol buffers. http://code.google.com/p/protobuf/.
[3] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson. Understanding the impact of emerging non-volatile memories on high-performance, io-intensive computing. 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.
[4] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS XIV, New York, NY, USA, 2009. ACM.
[5] Y. Gu and R. L. Grossman. Supporting Configurable Congestion Control in Data Transport Services. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 31–41, 2005.
[6] Intrepid. https://www.alcf.anl.gov/resource-guides/intrepid-file-systems.
[7] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu. ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. IEEE IPDPS 2013, Boston, MA, 2013.
[8] 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 *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–11, 2012.
[9] D. R. Mathog. Parallel BLAST on split databases. *Bioinformatics*, 19(4):1865 – 1866, 2003.