Supporting Large Scale Data-Intensive Computing with the FusionFS Distributed File System

Dongfang Zhao and Ioan Raicu
Department of Computer Science
Illinois Institute of Technology
Technical Report, August 2013

Abstract

State-of-the-art yet decades-old architecture of HPC storage systems has segregated compute and storage resources, bringing unprecedented inefficiencies and bottlenecks at petascale levels and beyond. This paper presents FusionFS, a new distributed file system designed from the ground up for high scalability (16K nodes) while achieving significantly higher I/O performance (2.5TB/sec). FusionFS achieves these levels of scalability and performance through complete decentralization, and the co-location of storage and compute resources. It supports POSIX-like interfaces important for ease of adoption and backwards compatibility with legacy applications. It is made reliable through data replication, and it supports both strong and weak consistency semantics. Furthermore, it supports scalable data provenance capture and querying, a much needed feature in large scale scientific computing systems towards achieving reproducible and verifiable experiments.

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.

As shown in our previous study [1], 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, as shown in Figure 1. 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). Future storage systems need to be re-architected to co-locate storage and compute resources in order to be able to better support the extreme level of concurrency expected with future computing systems. These future storage systems should leverage the higher bi-section bandwidth of modern torus interconnects, and the abundance of computational resources of entire system (generally 2 to 3 orders of magnitude larger than the resources found in a dedicated segregated distributed storage system).

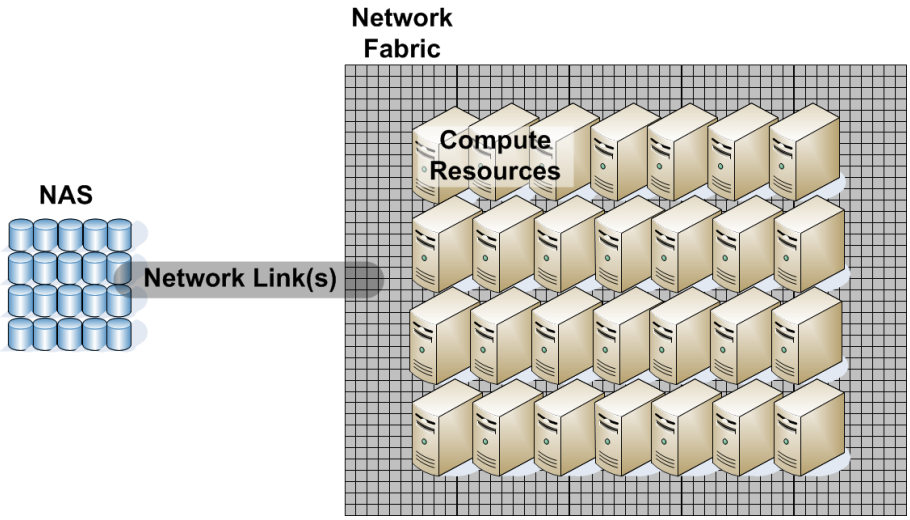
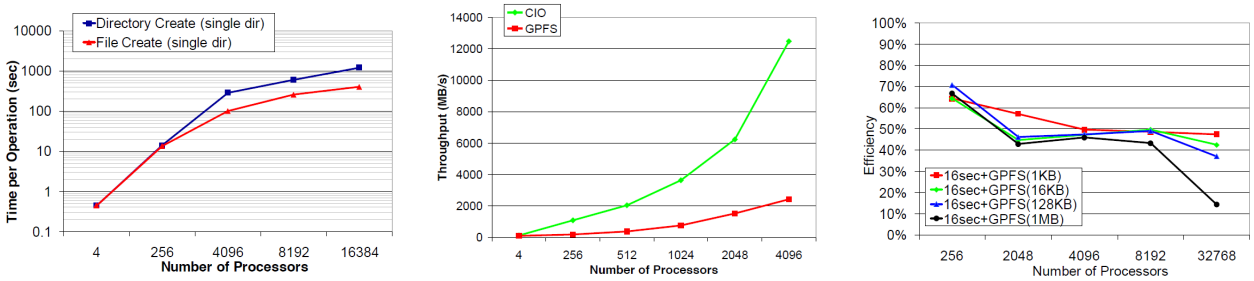


Fig. 1. Modern HPC system achitecture

The following examples (Figure 2) we experienced from IBM BlueGene/P at Argonne National Laboratory (ANL) would give a more concrete idea on why storage systems in future exascale systems will be the Achilles heel.



(a) File/dir create; 16K-cores [2] (b) Read throughput [3] (c) Write efficiency [3]

Fig. 2. GPFS performance on IBM BlueGene/P

Metadata operations on parallel file systems can be inefficient at large scale. Early experiments on the BlueGene/P system at 16K-core scales (see Figure 2(a)) shows the various costs (wall-clock time measured at remote processor) for file/directory create on GPFS. Ideal performance would be to have a flat line. If care is not taken to avoid lock contention, performance degrades rapidly, with operations (e.g. create directory) that took milliseconds on a single core, taking over 1000 seconds at 16K-core scales. [2, 3]

Read/Write: Reading performance of common datasets (e.g. application binaries, read-only databases) is challenging. The experiment in Figure 2(b) shows the distribution of data from the GPFS file system to the compute nodes with two approaches: 1) pushing out the data over a spanning tree (CIO), and 2) pulling the data from each compute node independently (GPFS). At the relatively modest scale of 4k-cores, the CIO approach outperforms GPFS by a factor of five, due to better utilization of the bi-section bandwidth of the torus network. Writing from many compute nodes directly to parallel file systems is also challenging; Figure 2(c) shows the poor efficiency achieved (15%-70%) with 16 second tasks producing 1KB to 1MB output. [2, 3]

While the parallel filesystem on network attached storage (NAS) for HPC has been researched and incrementally improved over the past decades, a more interesting and challenging problem is the design and implementation of a distributed filesystem particularly crafted for HPC systems by leveraging local persistent storage on hundreds of thousands of compute nodes, and scalable to millions of nodes for the emerging exascale era. None of the existing distributed filesystems meet all the HPC requirements: for example, some do not have a POSIX interface(e.g. HDFS [4]), some have centralized metadata management (e.g. GFS [5], HDFS [4]), Some couple data and metadata management making locality hard to achieve (e.g. Ceph [6]), while others assume segregation of compute and storage resources (e.g. [7], [8], [9]). Moreover, none of these filesystems are optimized for solid-state drive (SSD) which should be a perfect fit for concurrent data access in HPC: (1) SSD delivers a much higher bandwidth and lower latency than the traditional spinning hard disk drive (HDD); (2) SSD can be accessed concurrently, which is a especially preferable feature with recent significant improvement on multi-core and many-core technology. The benefit of SSD for a single node file system has been extensively studied recently, for example, SCMFS [10] is a prototype built with storage class memory, DFS [11] is developed on virtual flash storage, and the HyCache [12] user-level filesystem aimed at delivering SSD-like performance at the cost of traditional mechanical hard drives.

We introduce FusionFS, a distributed filesystem particularly crafted for extreme scale HPC systems. FusionFS leverages FUSE [13] 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 [14], and FusionFS complies with Gordon [15] 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 [16, 17]) 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. Both synchronous or asynchronous data replications are supported to maintain the system reliability. FusionFS also addresses the issue with the traditional provenance collection method [18], which would potentially become a performance bottleneck especially for file systems meant for extreme-scales.

This paper's contributions lie in the design and implementation of a distributed filesystem optimized for high-end computing systems, delivering:

- ***A POSIX interface for backwards compatibility***
- ***Scalable metadata throughput and I/O bandwidth***
- ***High reliability with both strong and weak consistency semantics***
- ***Distributed data provenance collection and query***

II. Related Work

There have been many shared and parallel file systems proposed since the 1980s, such as the Network File System (NFS)[19], Andrew File System (AFS)[20], General Purpose File System (GPFS)[7], Parallel Virtual File System (PVFS)[8], Lustre[9], Panases[21], Microsoft's Distributed File System (DFS)[22], GlusterFS[23], XtreemFS[24], OneFS [25], and POHMELFS[26]. While the majority of these file systems expose a POSIX-like interface providing a global namespace, and many have been adopted in cluster computing, grid computing, and even supercomputing, the biggest critique of these file systems is their vision that compute resources should be completely agnostic of the data locality on the underlying storage system. All of these file systems assume that the storage nodes/servers are significantly fewer than the client/compute nodes that will access the file system, resulting in an unbalanced architecture for data-intensive workloads.

A variety of distributed file systems have been developed to address the unbalance from parallel file systems to support data-intensive computing, such as GFS [5], HDFS [4], Sector [27], CloudStore [28], Ceph [6], GFarm [29], MooseFS [30], Chirp [31], MosaStore [32], PAST [33], Circle [34], and RAMCloud [35]. However, many of these file systems are tightly coupled with execution frameworks (e.g. MapReduce [36], Hadoop [4]), which means that scientific applications not using these frameworks must be modified to use these underlying non-POSIX-compliant file systems. For those that offer a POSIX-like interface, they lack distributed metadata management. And for those few (e.g. Ceph, PAST, Circle) that also have distributed metadata management, they fail to decouple data and metadata management making maximizing data locality difficult. The majority of these systems also fail to expose the data locality information for general computational frameworks (e.g. batch schedulers) to harness the data locality through data-aware scheduling. Also, with the exception of RAMCloud, none of the filesystems were designed and optimized for non-volatile memory storage (NVM). It is worth noting that the majority of these distributed file systems were not designed specifically for high-end computing (HEC) and scientific computing workloads, and the scales that HEC are anticipating in the coming years.

Some efforts have been made to address specific components of parallel/distributed file systems. GIGA+ [37] addressed challenges from big directories where millions to billions of small files are created in a single directory. The metadata throughput of GIGA+ significantly outperforms the traditional distributed directory implementations at up to 32-node scales. However, it is not clear if this design would suffice for larger scale, e.g. 1K nodes and beyond. Burst Buffer [38] proposed an SSD layer on I/O nodes to alleviate the storage bottleneck on an IBM BlueGene/P system at, and had shown its effectiveness with simulations. DASH [39] presented a prototype supercomputer with a 1TB SSD I/O node for each super compute node. Obviously, SSD cache on I/O nodes can deliver better bandwidth and latency than remote storage nodes. However, compute nodes still need to talk to I/O nodes via network for every single data access, inevitably implying more latency and congestion to some degree. On the other hand, FusionFS writes data directly onto the on-board NVM, and moreover, is a real working system deployed on BlueGene/P at 8K nodes.

III. Design and Implementation

Figure 3 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 [38] 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 every compute node should actively participate in the metadata and data management, leveraging the abundance of computational power many-core processors will have and the many orders of magnitude higher bisection bandwidth in multi-dimensional torus networks as compared to available cost effective bandwidth into remote network persistent storage. The benefits of this new architecture lie in its enabling of some workloads to scale near-linearly with systems scales by leveraging data locality and the full network bisection bandwidth. We believe the next generation of HPC systems would be equipped with local NVM (e.g. SSD), coexisting on the compute nodes to allow the system to leverage the data locality [42].

A. POSIX Interface

In general, it is important for a filesystem to provide POSIX interface for HPC applications, since it is one of the most widely used standard. Most HPC applications assume the underlying filesystem supports POSIX interface (many for legacy reasons). For the sake of backward compatibility, POSIX should be supported if at all possible.

FUSE has been criticized for its efficiency on traditional HDD-based file systems. In native UNIX file systems (e.g. Ext4) there are only two context switches between the caller in user space and the system call in kernel spaces. However for a FUSE-based file system, context needs to be switched four times: two switches between the caller and VFS; and another two between libfuse and FUSE. A detailed comparison between FUSE-enabled and native file systems was reported in [43], which

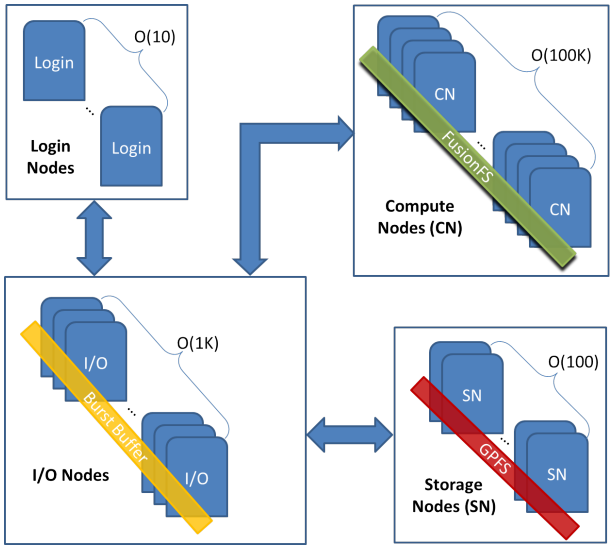


Fig. 3. Storage spectrum of IBM BlueGene/P

shows that a Java implementation of FUSE has about 60% overhead compared to the native file system mounted on commodity spinning hard drives.

Our previous work [12] has shown that FUSE overhead in the context of a HPC system, particularly with C/C++ bindings on NVMs, could be greatly compensated by the high concurrency offered by NVM. For example, Figure 4 shows that a FUSE+SSD filesystem could achieve 580 MB/sec aggregate bandwidth (at 12 concurrent processes, the same number of hardware threads of the test bed) for concurrent data accesses, which is about 85% of the bandwidth of the raw SSD device (i.e. SSD Ext4).

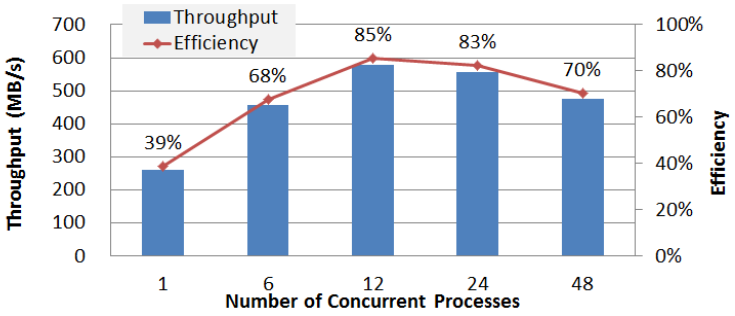


Fig. 4. FUSE Efficiency on SSD

B. Distributed Metadata Management

FusionFS metadata management relies on a distributed hash table [44], namely Zero-Hop Distributed Hash Table (ZHT) [16, 17]. ZHT is a distributed key-value store, rather than a ready-to-use metadata management system. All it manipulates are <key,value> entries. A key can be as simple as a string, and a value is a data blob. For FusionFS metadata, it is straightforward to save the file name as a key in ZHT. However, the metadata of a file usually has a strict structure, e.g. user permissions, time stamps, size, etc. Therefore we need a protocol to serialize/deserialize FusionFS metadata to/from ZHT values. The Google Protocol Buffer [45] is a small, fast, and simple implementation for serialization and deserialization, and has been adopted by FusionFS. We plan to integrate a more efficient encode/decode tool (e.g. Facebook/Apache Thrift [46]) in our future work.

FusionFS has different data structures for managing regular files and directories, although both regular files and directories do share some common fields. Conventional i-node information like permissions can be found in both file types. For a regular file, a field called *addr* records the node address of its primary copy. Replicas are stored in the k (default is 2) nearest neighbors, so the replica address does not need to be stored. For a directory, there is a field called *filelist* to record all the files under this directory. This field is particularly useful for providing an in-memory speed for directory read, e.g. "ls /dev/FusionFS".

C. Data Movement

The local persistent storage in FusionFS enables many strategies for data read/write. In the traditional HPC architecture, a compute node has to connect to an I/O node, to further connect to a storage node to access data. FusionFS, however, has multiple choices to access data: the local compute node, the remote compute node, or the remote storage node. Conceptually, FusionFS introduces two extra layers in the storage hierarchy: the local SSD and the remote SSD. In Figure 5, compute node A is able to access its local SSD A and its remote SSD B, both of which are accessible via FusionFS.

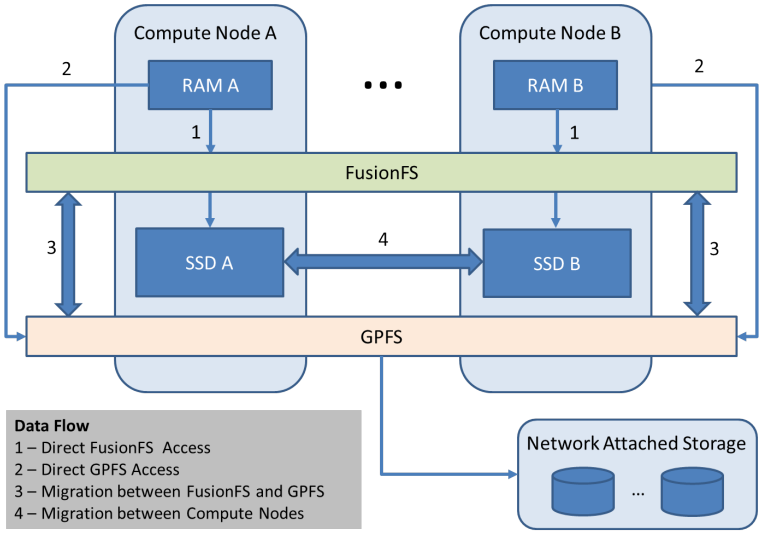


Fig. 5. Storage Hierarchy with FusionFS

If data is accessed in the local SSD, FusionFS invokes the corresponding UNIX system calls. If data needs to be accessed in a remote compute node, then we need to migrate the data across nodes. Ideally, the data transfer mechanism should be efficient, reliable and light-weighted. User Datagram Protocol (UDP) is efficient on transferring but is an unreliable protocol. Transmission Control Protocol (TCP), on the other hand, is reliable but has a relatively lower efficiency. Ideally, a hybrid UDP/TCP protocol might be best, essentially a protocol that is both reliable and efficient.

We have developed our own data transfer service Fusion Data Transfer (FDT) on top of UDP-based Data Transfer (UDT) [47], which is a reliable UDP-based application level data transport protocol for distributed data-intensive applications. UDT adds its own reliability and congestion control on top of UDP which thus offers a higher speed than TCP.

The top consideration of our I/O design is to achieve high aggregate write throughput, as reported in [48] that supercomputer I/O loads are quite dominated by writes. Therefore, FusionFS always tries to write files on the local SSD, in order to maximize the write throughput. This strategy implies linearly scalable bandwidth, because there is no interference between compute nodes (with the exception of data replication which might introduce network congestion). In theory, the aggregate write bandwidth is the summation of bandwidth of all SSDs (assuming the network bandwidth is sufficient to keep up with the replica management). Note that checkpointing workloads should work extremely well using this strategy.

Unlike data write, FusionFS cannot control whether the data to be read resides right on the local node, as that is decided by the job/task scheduler which manages the computation. In the case of "remote read", requested data are transferred from the remote compute node(s) by FDT, and then can be read locally by the process which initiated the original read request. For large files, a better strategy is to move the job/task to the data, rather than the data to the job/task. For example, in [49, 50], we proposed a "data diffusion" approach to perform cooperative caching and schedule computation close to the data, and in [51] we presented a method to predict the I/O performance given the applications' level of I/O concurrency and I/O amount.

When a particular node has a possibility to overflow its local SSD, it would move the data to another node that has enough SSD space, as indicated by arrow 4 in Figure 5. When the aggregate FusionFS usage reaches some threshold, then the data on FusionFS need to be swapped to the remote NAS. On the other hand, when FusionFS has enough space, some data on NAS can be prefetched/cached into FusionFS. This bidirectional data transfer is illustrated as arrow 3 in Figure 5. In some sense, FusionFS works as a distributed persistent cache as Mercury [52] (proposed for virtual machines), rather than the traditional transient cache e.g. Panache [53]. We have explored different caching mechanisms for persistent cache in the context of SSD and HPC systems in our previous work [12], and will integrate a similar mechanism in our future work.

D. Data Reliability

The current mechanism for FusionFS to keep data reliable is data redundancy, sharing the same philosophy of GFS [5] and HDFS [4]. FusionFS maintains k replicas of every file (i.e. the primary copy) into its k nearest neighbors. k is default to 2 in FusionFS: every primary copy has two replicas on its previous and next compute nodes in the logical ring-shaped topology, in a similar way employed in ZHT [17]. k value could be changed by end users to meet their application-dependent and/or platform-specific needs. For example, some applications checkpoint huge files, so replicating would not be feasible. In this case k could be set to 0 to disable replications.

Two semantics are supported for replica synchronization: (1) strong consistency - each write operation triggers the synchronization between the primary copy and its replicas, and will not return until the synchronization is completed; (2) weak consistency - the write operation will fork another thread to synchronize with the replicas and will return immediately when the primary copy is updated. Strong consistency is a necessity for those applications requiring a consistent global view of data (e.g. financial transaction), whereas weak consistency achieves better performance if eventual consistency is acceptable. Weak consistency is the default semantic in FusionFS to synchronize replicas.

We are currently working towards the support for information dispersal algorithms [54] as an alternative to replication, in order to improve storage efficiency and to reduce network bandwidth needed.

E. Data Provenance

Data provenance is the service describing how data was derived. Some file systems have so far proposed a central system for provenance collection [18]. This is a performance bottleneck, especially for file systems meant for extreme-scales. This section describes the development of such a distributed provenance capture and query service, that has been integrated into FusionFS as a fundamental storage service.

As an introductory example, Figure 6 shows a typical query tree (compliant to the OPM Provenance Model [55]) in filesystem provenance. This tree will be available when a user asks something like "what happened to ∼/origin_file". The query can be interpreted as the following. One node (12.0.0.1) creates a file (origin_file), which is then copied by a bunch of other nodes (12.0.0.2, ..., 12.4.0.16) and saved as a local copy on each (backup_a, ..., backup_b).

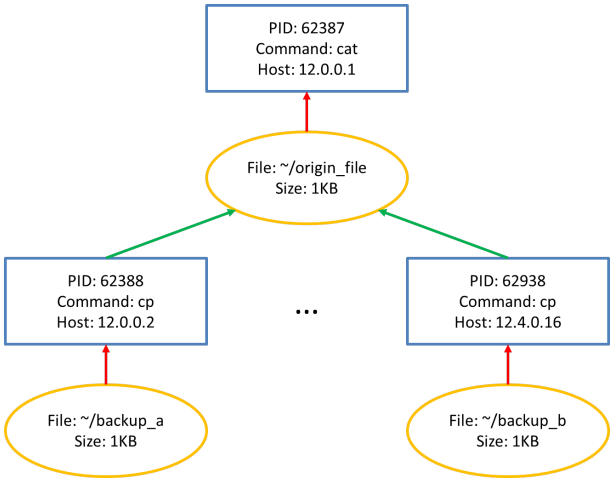


Fig. 6. An output example of provenance query

To capture the provenance information, FusionFS collects every block-level POSIX operation, and caches it in a local hash table (not the local instance of ZHT), where each key represents a vertex in the query tree. All entries of this local hash table will be saved to the underlying ZHT instance (either local or remote, depending on the hash value of the file name) only when the file is closed. In this two-stage way, we avoid the overhead introduced by many small writes to ZHT.

F. Implementation

The software stack of FusionFS is shown in Figure 7. 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.

IV. Evaluation

We have scaled FusionFS up to 16K nodes on BlueGene/P. The aggregate throughput shows an excellent scalability, as reported in Figure 8.

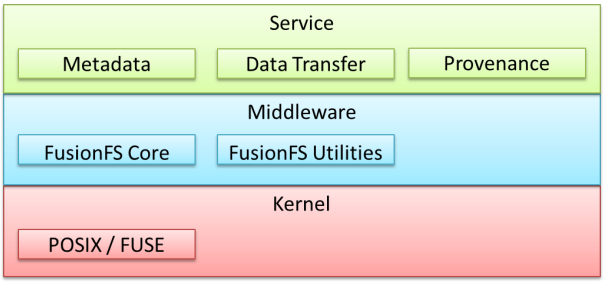


Fig. 7. FusionFS software stack

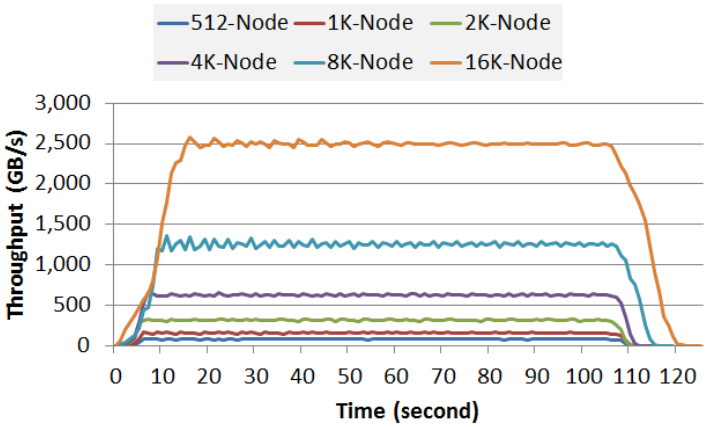


Fig. 8. FusionFS throughput

V. Use Case: Efficient Checkpointing

The combination of FusionFS' policy that writes are always local, along with asynchronous replications for data reliability, offers a great opportunity for reducing the optimal checkpointing interval (OPT) [68]:

$$OPT = \sqrt{2\delta(M+R)} - \delta,$$

where δ is the checkpointing time, M is the mean-time-to-failure (MTTF), and R is the rebooting time. IBM claims each BlueGene/P node has 1000-year MTTF. Then the MTTF of the 1K-node system is 1 year, and the rebooting time (in minutes) is negligible comparing to the MTTF. Assuming each node writes 128MB data, FusionFS OPT and GPFS OPT are shown in Figure 9. FusionFS shows a significantly smaller OPT compared to GPFS at all scales (where smaller is better). As mentioned earlier, PVFS could not scale beyond 256 nodes.

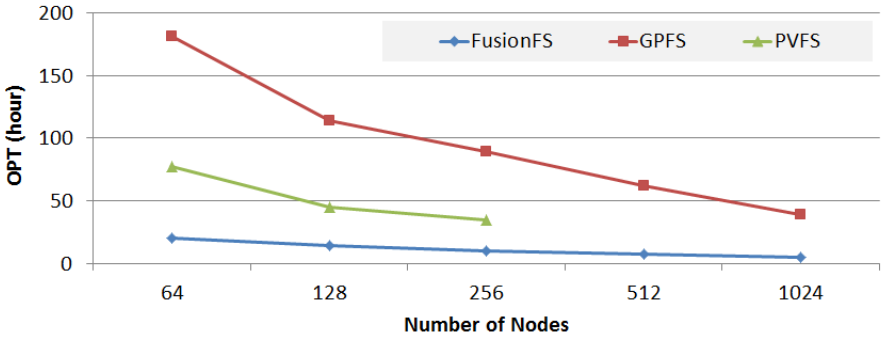


Fig. 9. Optimal Checkpointing Time comparing FusionFS, PVFS, and GPFS

VI. Conclusion and Future Work

This paper presented FusionFS, a distributed filesystem for large scale data-intensive applications. In addition to the usual features (POSIX and reliable), FusionFS also supports distributed metadata management and distributed data provenance capture and query. Our main message is that, by combining lessons learned from parallel file systems (e.g. GPFS), distributed file systems (e.g. HDFS), peer-to-peer systems (e.g. distributed hash tables), along with new advances in hardware (e.g. SSD), we can define a new storage architecture that is optimized for future high-end computing at extreme scales. 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.

We are working on a library of APIs exposed to application developers to completely bypass the FUSE layer, which would further improve the performance of those applications that do not require POSIX interfaces. We have developed a hybrid SSD/HDD caching system [12] to provide a SSD-level performance while keeping the space capacity and per-GB cost similar to HDD. This caching system will be integrated into FusionFS to make it more cost-effective. We have also explored the potential of a GPU-based erasure coding mechanism to achieve highly efficient data reliability than data replications. We also intend to study the suitability of FusionFS as a file system supporting highly efficient checkpointing operations. We also plan to expose data locality, and offer interfaces to scheduling systems (e.g. Falkon [58], Slurm [69]) and parallel programming systems (e.g. Swift [57]) to allow better scheduling that can optimize data locality. Finally, we plan to scale FusionFS to the largest systems available, to O($\sim$100K) nodes, with the ultimate scalability goals at O(1M) nodes.

Acknowledgement

This work was supported by the National Science Foundation (NSF) under grant OCI-1054974. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory (ANL), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] Dongfang Zhao, Da Zhang, Ke Wang, and Ioan Raicu. Exploring reliability of exascale systems through simulations. 21st High Performance Computing Symposia, SpringSim'13, San Diego, CA, 2013.

[2] Ioan Raicu, Zhao Zhang, Michael Wilde, Ian T. Foster, Peter H. Beckman, Kamil Iskra, and Ben Clifford. Toward loosely coupled programming on petascale systems. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, page 22, 2008.

[3] Zhao Zhang, Allan Espinosa, Kamil Iskra, Ioan Raicu, Ian T. Foster, and Michael Wilde. Design and Evaluation of a Collective I/O Model for Loosely-coupled Petascale Programming. In *IEEE Many-Task Computing on Grids and Supercomputers (MTAGS08), SC '08*, 2008.

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[6] Sage A. Weil et al. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[7] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[8] Ibrahim F. Haddad. PVFS: A Parallel Virtual File System for Linux Clusters. *Linux J.*, 2000(80es), November 2000.

[9] Philip Schwan. Lustre: Building a file system for 1,000-node clusters. In *Proceedings of the linux symposium*, page 9, 2003.

[10] Xiaojian Wu and A. L. Narasimha Reddy. SCMFS: a file system for storage class memory. In *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, SC '11, 2011.

[11] William K. Josephson, Lars A. Bongo, Kai Li, and David Flynn. DFS: A file system for virtualized flash storage. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 12–12, Berkeley, CA, USA, 2010. USENIX Association.

[12] Dongfang Zhao and Ioan Raicu. HyCache: a user-level caching middleware for distributed file systems. International Workshop on High Performance Data Intsensive Computing, IEEE IPDPS 2013, Boston, MA, 2013.

[13] FUSE Project. http://fuse.sourceforge.net.

[14] Adrian M. Caulfield et al. Understanding the impact of emerging non-volatile memories on high-performance, io-intensive computing. In *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[15] Adrian M. Caulfield, Laura M. Grupp, and Steven 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.

[16] Tonglin Li, Raman Verma, Xi Duan, Hui Jin, and Ioan Raicu. Exploring distributed hash tables in HighEnd computing. *SIGMETRICS Perform. Eval. Rev.*, 39(3):128–130, December 2011.

[17] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. IEEE IPDPS 2013, Boston, MA, 2013.

[18] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[19] Mike Eisler, Ricardo Labiaga, and Hal Stern. Managing NFS and NIS, 2nd ed. *O'Reilly & Associates, Inc.*, 2001.

[20] John H. Howard. On overview of the andrew file system. In *USENIX Winter*, pages 23–26, 1988.

[21] David Nagle, Denis Serenyi, and Abbie Matthews. The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 53–, Washington, DC, USA, 2004. IEEE Computer Society.

[22] Microsoft Inc. http://www.microsoft.com/windowsserversystem/dfs/default.mspx.

[23] GlusterFS. http://www.gluster.com/.

[24] Felix Hupfeld et al. The XtreemFS architecture – a case for object-based file systems in Grids. *Concurr. Comput. : Pract. Exper.*, 20(17):2049–2060, December 2008.

[25] OneFS. http://www.emc.com/storage/isilon/isilon-onefs-operating-system.htm.

[26] POHMELFS: Parallel Optimized Host Message Exchange Layered File System. http://www.ioremap.net/projects/pohmelfs/.

[27] Yunhong Gu, Robert L. Grossman, Alexander S. Szalay, and Ani Thakar. Distributing the sloan digital sky survey using udt and sector. In *e-Science*, page 56, 2006.

[28] CloudStore. http://kosmosfs.sourceforge.net/.

[29] Xiaohui Wei, Wilfred W. Li, Osamu Tatebe, Gaochao Xu, Liang Hu, and Jiubin Ju. Implementing data aware scheduling in gfarm(r) using lsf(tm) scheduler plugin mechanism. In *GCA*, pages 3–10, 2005.

[30] MooseFS. http://www.moosefs.org/.

[31] Douglas Thain, Christopher Moretti, and Jeffrey Hemmes. Chirp: A practical global file system for cluster and grid computing. *Journal of Grid Computing*.

[32] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. The case for a versatile storage system. *SIGOPS Oper. Syst. Rev.*, 44(1):10–14, March 2010.

[33] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *In HotOS VIII*, pages 75–80, 2001.

[34] Circle. http://savannah.nongnu.org/projects/circle/.

[35] John Ousterhoutet al. The case for ramclouds: scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, January 2010.

[36] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI, 2004.

[37] Swapnil Patil and Garth Gibson. Scale and concurrency of GIGA+: file system directories with millions of files. FAST'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[38] Ning 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.

[39] Jiahua He, Arun Jagatheesan, Sandeep Gupta, Jeffrey Bennett, and Allan Snavely. DASH: a recipe for a flash-based data intensive supercomputer. In *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, SC '10, 2010.

[40] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):7:1–7:25, November 2008.

[41] Intel Product Manual. Intel X25-E SATA Solid State Drive. http://download.intel.com/design/flash/nand/extreme/319984.pdf, 2009.

[42] Ioan Raicu, Ian T. Foster, and Pete Beckman. Making a case for distributed file systems at exascale. In *Proceedings of the third international workshop on Large-scale system and application performance*, LSAP '11, pages 11–18, New York, NY, USA, 2011. ACM.

[43] Aditya Rajgarhia and Ashish Gehani. Performance and extension of user space file systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, Sierre, Switzerland, March 2010.

[44] Ion Stoica et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003.

[45] Google Protocol Buffers. http://code.google.com/p/protobuf/.

[46] Apache Thrift. http://thrift.apache.org/.

[47] Yunhong Gu and Robert 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.

[48] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. Characterizing output bottlenecks in a supercomputer. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, SC '12, 2012.

[49] Ioan Raicu, Yong Zhao, Ian T. Foster, and Alex Szalay. Accelerating large-scale data exploration through data diffusion. In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 9–18, New York, NY, USA, 2008. ACM.

[50] Ioan Raicu et al. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 207–216, New York, NY, USA, 2009. ACM.

[51] Zhao Zhang, Daniel S. Katz, Michael Wilde, Justin M. Wozniak, and Ian Foster. MTC Envelope: Defining the capability of large scale computers in the context of parallel scripting applications. In *Proceedings of the 22nd ACM international symposium on High performance distributed computing*, HPDC '13, New York, NY, USA, 2013. ACM.

[52] Steve Byan, James Lentini, Anshul Madan, and Luis Pabon. Mercury: Host-side flash caching for the data center. In *MSST*, pages 1–12, 2012.

[53] Marc Eshel, Roger Haskin, Dean Hildebrand, Manoj Naik, Frank Schmuck, and Renu Tewari. Panache: a parallel file system cache for global file access. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 12–12, Berkeley, CA, USA, 2010. USENIX Association.

[54] Dongfang Zhao, Corentin Debains, Pedro Alvarez-Tabio, Kent Burlington, and Ioan Raicu. IStore: Towards high efficiency, performance, and reliability in distributed data storage with information dispersal algorithms. under review at IEEE Cluster'13, 2013.

[55] The OPM Provenance Model. http://openprovenance.org/.

[56] Intrepid. https://www.alcf.anl.gov/resource-guides/intrepid-file-systems.

[57] Yong Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *Services, 2007 IEEE Congress on*, pages 199 –206, july 2007.

[58] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, and Mike Wilde. Falkon: a Fast and Light-weight tasK executiON framework. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 43:1–43:12, New York, NY, USA, 2007. ACM.

[59] Amazon S3. http://aws.amazon.com/s3/.

[60] Intrepid File System. https://www.alcf.anl.gov/user-guides/intrepid-challenger-eureka-file-systems.

[61] Ashish Gehani and Dawood Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. ACM/USENIX Middleware, pages 101–120, 2012.

[62] Chen Shou, Dongfang Zhao, Tanu Malik, and Ioan Raicu. Towards a provenance-aware distributed filesystem. In *5th USENIX Workshop on TaPP, NSDI 2013*, Lombard, IL, 2013.

[63] Heshan Lin, Pavan Balaji, Ruth Poole, Carlos Sosa, Xiaosong Ma, and Wu-chun Feng. Massively parallel genomic sequence search on the blue gene/p architecture. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 33:1–33:11, Piscataway, NJ, USA, 2008. IEEE Press.

[64] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. CloudBLAST: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, eScience '08, pages 222–229, Washington, DC, USA, 2008. IEEE Computer Society.

[65] David R. Mathog. Parallel BLAST on split databases. *Bioinformatics*, 19(4):1865 – 1866, 2003.

[66] Zhao Zhang, Daniel S. Katz, Justin M. Wozniak, Allan Espinosa, and Ian T. Foster. Design and analysis of data management in scalable parallel scripting. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, SC '12, page 85, 2012.

[67] Zhao Zhang, Daniel S. Katz, Matei Ripeanu, Michael Wilde, and Ian T. Foster. AME: an anyscale many-task computing engine. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 137–146, New York, NY, USA, 2011. ACM.

[68] John Daly. A model for predicting the optimum checkpoint interval for restart dumps. In *Proceedings of the 2003 international conference on Computational science*, ICCS'03, pages 3–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[69] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.