Distributed File Systems for Exascale Computing

Dongfang Zhao* and Ioan Raicu*†
*Department of Computer Science, Illinois Institute of Technology
†Mathematics and Computer Science Division, Argonne National Laboratory
dzhao8@hawk.iit.edu, iraicu@cs.iit.edu

I. Introduction

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. This degree of computing capability is similar to that of a human brain and will enable the unraveling of significant scientific mysteries and present new challenges and opportunities.

Unfortunately, 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 works well for current compute intensive petascale applications, it would not be adequate for data-intensive petascale computing or exascale computing. 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.

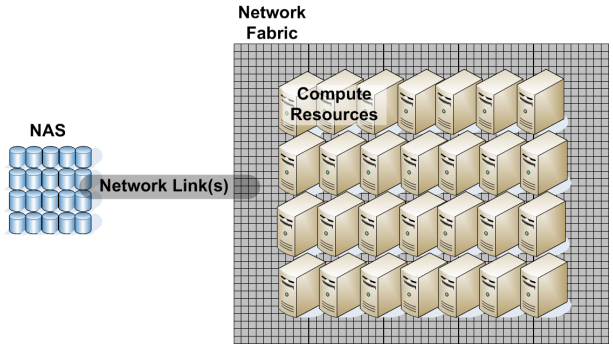

Fig. 1. The architecture of modern HPC systems: compute and storage resources are segregated and interconnected by high-bandwidth network.

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. FusionFS has a completely distributed metadata management based on an implementation of distributed hash table (i.e. ZHT [4]) to achieve a scalable metadata throughput.

II. Design and Implementation

In our previous work [5], we propose that 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, as shown in Figure 2. NVM would be an excellent candidate for the local persistent storage, that can be observed from two aspects: (1) NVM delivers a much higher bandwidth and lower latency than the traditional spinning hard disk drive (HDD); (2) NVM can be accessed concurrently, which is a especially preferable feature with recent significant improvement on multi-core and many-core technology. The local high throughput and low latency persistent storage would alleviate the traffic congestion on the network between compute nodes and the NAS. The distributed filesystem mounted on the local persistent storage could (or should) coexist with the parallel filesystem on NAS.

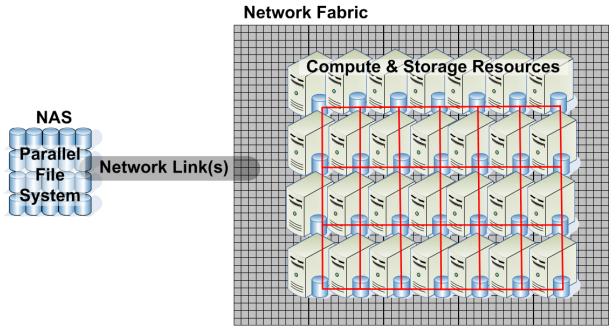

Fig. 2. The architecture of HPC systems with local persistent storage: distributed filesystem deployed on the local persistent storage coexists with remote parallel filesystem on NAS.

The FusionFS metadata management relies on a distributed hash table, namely Zero-Hop Distributed Hash Table (ZHT) [4]. 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 1) 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". This list also plays the key role to maintain the tree-like hierarchy of FusionFS.

The high-level modules of FusionFS implementation is shown in Figure 3. Each compute node behaves the same role in FusionFS and nothing is centralized at all. These compute nodes are normally interconnected by some high performance network (e.g. 3-D torus in IBM BlueGene/P). The high bandwidth of the node-to-node communication is crucial to the success of FusionFS.

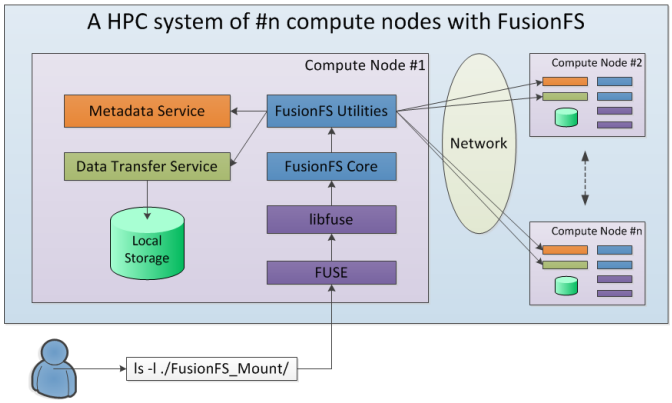

Fig. 3. Overview of FusionFS architecture

Components of each compute node are briefly explained in the following:

1) **FUSE:** FUSE is the Linux kernel module that monitors any I/O requests made to FusionFS.
2) **libfuse:** libfuse is a user-level library that interprets the incoming POSIX-compliant file requests into FusionFS implementation.
3) **FusionFS Core:** This module implements all the FUSE interfaces to manipulate POSIX file operations.
4) **FusionFS Utilities:** This module provides miscellaneous utilities supporting local FusionFS Core module and local services, as well as communication to remote compute nodes.
5) **Metadata Service:** It is a daemon service dedicated for metadata manipulations.
6) **Data Transfer Service:** It is a daemon service that handles data transfer.
7) **Local Storage:** We assume there is a high performance persistent storage (e.g. SSD) attached to each compute node.

FusionFS is implemented with C/C++ and Shell scripts, consisting of about 20K lines of code in total, excluding 2 third-party libraries: the Google Protocol Buffers [2] and UDT [3]. Protocol Buffers is used to serialize C/C++ structures into strings and deserialize strings back to structures. UDT provides the underlying infrastructure and APIs to transfer data in FusionFS.

III. Evaluation

We deployed FusionFS on the IBM BlueGene/P supercomputer (i.e. Intrepid) at Argonne National Laboratory. Each node has one 850MHz quad-core processor and 2GB RAM. Figure 4 shows that when each node creates 10K files at 1024-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.

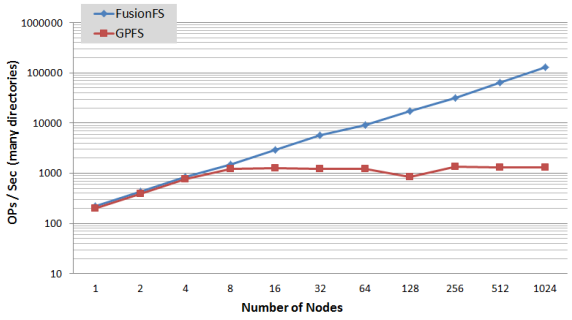

Fig. 4. Comparison of metadata performance between FusionFS and GPFS on IBM BlueGene/P (many directories)

IV. Conclusion

We believe the radical storage architecture changes proposed by FusionFS will make exascale computing more tractable. Our main message is that by combining lessons learned from parallel file systems and distributed file systems, 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.

References

[1] FUSE Project. http://fuse.sourceforge.net/.
[2] Google protocol buffers. http://code.google.com/p/protobuf/.
[3] 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.
[4] 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. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13.
[5] I. Raicu, I. T. Foster, and P. 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.