Scalable Distributed Directory Implementation on Orange File System

Shuangyang Yang, Walter B. Ligon III
Parallel Architecture Research Laboratory
Clemson University, Clemson, SC 29634, USA
{shuangy, walt}@clemson.edu

Elaine C. Quarles
Clemson Computing and Information Technology
Clemson University, Clemson, SC 29634, USA
elaine@clemson.edu

Abstract

In Orange File System, large data files are striped across multiple servers to provide highly concurrent access, however, contents of large directories are only stored in a single server, which is becoming a bottleneck in handling a large number of requests accessing the same directory concurrently. In this paper, a scalable distributed directory for Orange File System is implemented and evaluated in a large-scale system. The throughput performance is measured by a modified version of UCAR metarates benchmark. The result shows great scalability in concurrently creating and removing large numbers of files under one directory by multiple clients. On a 64-servers setup and 128 clients accessing the same directory concurrently, the scalable distributed directory can achieve more than 8,000 file creations per second and over 11,000 file removals per second on average.

1. Introduction

The high-performance computing (HPC) community is walking into an new era where top clusters and systems are able to achieve a performance of petaFLOPs (10^{15}) scale easily. According to the TOP500 supercomputer list [5] of November, 2010, the top 7 systems all break the petaFLOPs barrier in their performances. And they are all composed of tens of thousands of cores and nodes running in parallel. The rapid growth in computing power and high demand for parallelism imposes significant challenges for the storage system, which is expected to handle the input/output (I/O) requests from parallel applications with good performance and great scalability.

Many parallel file systems are developed to answer that challenge[15, 1, 2, 11, 7]. File data are distributed across multiple processing nodes in parallel file systems, each with its own storage resources. In that case, concurrent I/O requests can be spread across several servers to process rather than focusing I/O on a single server [6]. Recently there's growing concern with making directories scalable on parallel file systems. It is a typical scenario for applications which do data mining and real-time application monitoring to create numerous small files under the same directory every second. As a result, it is important to build scalable directory services for parallel file systems to support efficient concurrent access to even larger directories in the future.

Parallel Virtual File System (PVFS) [7] is a production-quality parallel file system designed for use on high end computing (HEC) systems that provides very high performance access to disk storage for parallel applications. PVFS is being used at a number of sites, such as Argonne National Laboratory, NASA Goddard Space Flight Center, and Oak Ridge National Laboratory. It is also widely used for research purposes [17]. Orange File System (OrangeFS) is a branch of PVFS driven by input from a large base of user feedback [2]. Although it distributes large files across multiple servers, a directory object and its data object are still stored on a single server. This design of a directory is inadequate to scale large directories when multiple clients are accessing the same directory concurrently.

A scalable directory service [16] designed for PVFS was demonstrated based on GIGA+ [10]. A prototype was built on a parallel file system simulator, where it achieved high throughput and scalability while minimizing bottlenecks and synchronization overheads. It is necessary to incorporate the scalable directory service into a production-quality high-performance parallel file system to get benefits. In this paper, a scalable distributed directory is designed based on previous work and implemented into OrangeFS seamlessly.

The main contributions of this paper include:

- Implemented a complete scalable distributed directory service which distributes directory entries across multiple servers, and incorporated the service into Orange File System seamlessly. The number of directory entry partitions is configurable during directory creation and can be dynamically incremented with usage.
- Evaluated throughput performance in large scale under a production-level environment and compared the outcomes with that of vanilla OrangeFS. The result shows great scalability when the number of servers and clients are varied.

In the rest of the paper, Section 2 lists some related works by other researchers. Section 3 describes the design and implementation of the scalable distributed directory. Section 4 gives results of performance evaluation and Section 5 draws the conclusion and a plan of future work.

2 Background

GPFS [11] supports efficient file name lookup in large directories through *extensible hashing* [8]. Directory entries of a large directory are stored in multiple disk blocks. To map an entry to a disk block, a hash function is applied to the entry name and the n low-order bits of the hash value is used as the block number, where n depends on the size of the directory. As a directory grows, a disk block can be split in two. The logical block number of the new directory block is derived from the old block number by adding a '1' in the $n + 1^{st}$ bit position, and directory entries with a '1' in the $n + 1^{st}$ bit of their hash value are moved to the new block.

Lustre File System [1] is a massively parallel distributed file system for cluster computing. Lustre is working on distributed metadata service which seems to support one level of splitting, and some results of a pre release [13] shows over 10,000 files creates per second.

Ceph [15] is a research distributed file system which offers dynamic distributed metadata management based on current access patterns. Ceph writes a directory's content to the object storage devices using the same striping and distribution strategy as file data. Each metadata server keeps a record of the popularity of metadata within the directory and adaptively distributes metadata hierarchically with dynamic subtree partitioning strategy [14].

GIGA+ [10] is a scalable directory design for shared file systems which divides each directory into a scalable number of fixed-size partitions that are distributed across multiple servers in the cluster. A *bitmap* is used to represent a tree of partitions of the corresponding servers where '1' indicates presence and '0' for absence. At first, only the zero-th bit position in the bitmap is set to '1'. As a directory is filled, an overflowing partition with index i and depth r will move half of its hash space to a partition $i + 2^r$, and both partitions will be at depth $r + 1$. Each server manages its partitions independently. An illustration of the splitting process in GIGA+ is shown in Figure 1.

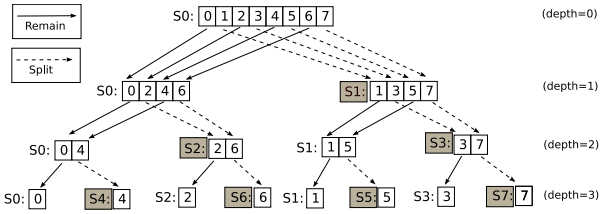


Figure 1. Illustration of a three-level splitting process in GIGA+. The hash space is divided evenly for each split.

The design of scalable distributed directory on OrangeFS employs the extensible hashing technique and the splitting strategy of GIGA+, however, it differs in other aspects in order to be incorporated into OrangeFS seamlessly.

3 Design and Implementation

3.1 OrangeFS Structure

OrangeFS organizes files and directories in the form of several storage objects. Some important objects are

- *Metadata* objects store data about files or directories. Besides usual file attributes like owner, group, permissions, etc. OrangeFS metadata object keeps file distribution information of the actual data. Metadata objects do not contain contents of files or directories.
- *Datafile* objects are blocks of actual file contents. OrangeFS stripes a file across multiple datafiles on multiple servers to facilitate parallel access.
- *Dirdata* objects contain contents of directories, or directory entries. Each directory entry is a pair of entry name and the identifier to its metadata object, either a file metadata object or a directory metadata object.

OrangeFS objects are uniquely identified by *handles*, which are unique, opaque, integer-like identifiers. This provides a concise, non path dependent mechanism for specifying what object to operate on when clients and servers communicate.

The metadata object of a file keeps a list of datafile handles identifying the actual data blocks, while the metadata object of a directory keeps only one dirdata handle representing the dirdata object at this time. The structures of these objects are visualized in Figure 2.

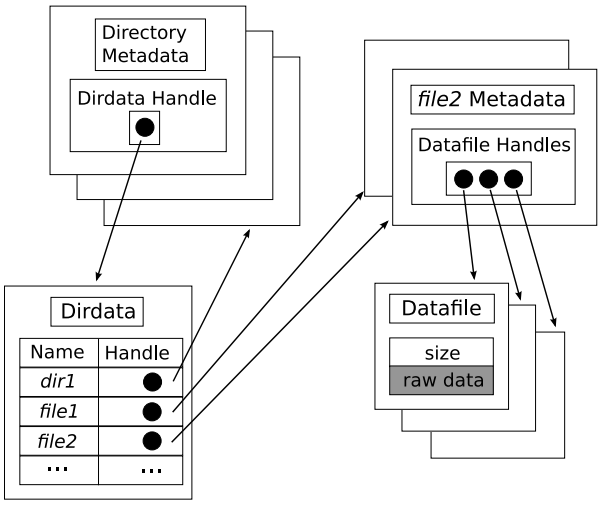


Figure 2. Diagram of structures of OrangeFS objects. A directory metadata object only has one dirdata object, which resides on the same metadata server. Usual file attributes fields are omitted in this diagram.

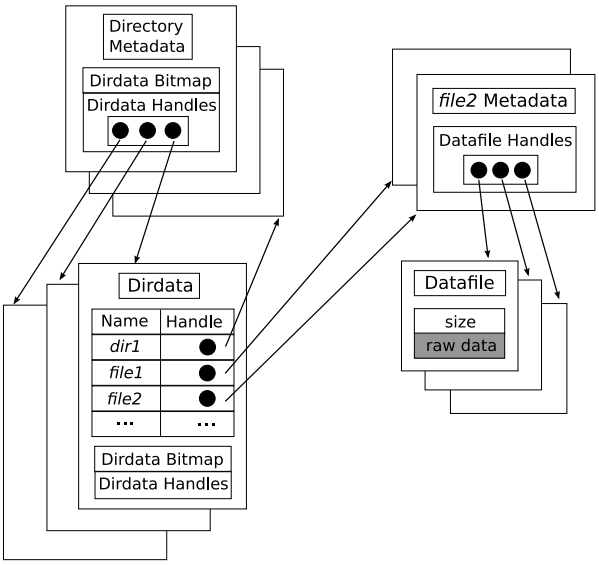


Figure 3. Diagram of object structures in scalable distributed directory OrangeFS design. A directory metadata object holds a number of dirdata objects across all metadata servers. Dirdata handles array and dirdata bitmap are added on the directory metadata object and the dirdata object.

3.2 Scalable Distributed Directory Design

3.2.1 Data Structure

An array of *dirdata handles* and a *dirdata bitmap* are added to a directory metadata object and dirdata objects to enable distribution of directory entries across multiple servers. The structures are shown in Figure 3.

When a directory is created, an array of dirdata objects is allocated with one dirdata object on each metadata server. Within each dirdata object, directory entries are indexed by Berkeley DB [9], which uses B+ tree to provide low-cost lookup, insert and delete operations plus efficient sequential access. Consequently there is no need to keep multiple dirdata objects of the same directory on one metadata server.

The dirdata bitmap keeps a one-to-one mapping between the bit position and the index of the dirdata handle array. A bit value of '1' indicates an *active* dirdata object while a bit value of '0' indicates an *inactive* dirdata object. The bitmap is initialized during directory creation to set the initial number of active dirdata objects to use. Unlike GIGA+ which always starts from one partition and increases the number of partitions gradually, the initial number of active dirdata objects is configurable. The splitting process is found to be expensive [16] and for a directory which is expected to be large, it is better to utilize all the dirdata objects to enjoy better scalability from the start.

The metada object of a directory keeps a most up-to-date copy of dirdata bitmap, which is done by accepting bitmap updates from dirdata objects. Each dirdata object also keeps a copy of dirdata bitmap and array of dirdata handles. One purpose is to verify the incoming directory entry request belongs to its hash space and another is to find the new dirdata object when issuing a split operation. Thus the dirdata bitmap on a dirdata object only needs to be locally updated and accurately reflects its own status.

3.2.2 Directory Entry Lookup

When initiating a directory request, the OrangeFS client library is responsible for fetching the scalable distributed directory attributes, including the dirdata array and dirdata bitmap, and deciding which dirdata object to contact to complete directory operations. We made this decision to simplify the design on the server side and reduce metadata server waiting time for each client request, since metadata and dirdata are probably not sitting on the same server.

Another reason is that OrangeFS uses client-side caching of object metadata to optimize the performance of file system access patterns [12]. There is an *acache* module, which stands for attribute cache, in OrangeFS system. It is used to cache metadata of storage objects including ownership, timestamp and distributed file information. By keeping a copy of distributed directory attributes on the client side, we can take advantage of the *acache* module.

The extensible hashing technique mentioned in Section 2 is used to map a directory entry to a dirdata object using the algorithm as detailed in Figure 4. A directory entry is assigned to an active dirdata handle based on its name. To achieve a random distribution, the directory entry name is encoded by a strong hash algorithm (*MD5* in our case) first. The hashed value then serves as the key to dirdata object selection. The lower R bits ($R = \lceil log_2(N) \rceil$, where N is the size of dirdata array) are taken as an initial matching index. If the dirdata bitmaps shows an inactive dirdata handle at that index, the highest bit is taken off and the dirdata bitmap is checked again with the new matching index. In the end, an active dirdata handle is picked and the corresponding operations can be processed. The dirdata object at index 0 is always active which guarantees a dirdata object is selected under any circumstances.

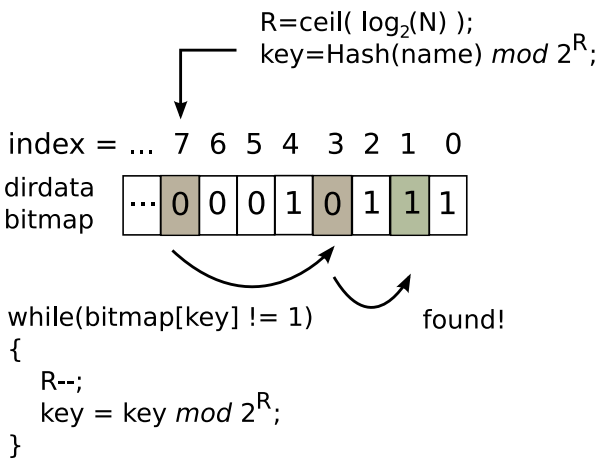


Figure 4. Illustration of the lookup process in scalable distributed directory on OrangeFS. N denotes the total number of dirdata objects. In this example, N=8 and the key starts at 7.

3.2.3 Dirdata Splitting

Dirdata splitting is supported when the number of directory entries in one dirdata object exceeds a threshold value, which is a configurable parameter with a default value. Because the maximum number of dirdata objects of a directory is fixed at the number of metadata servers, splitting is not possible, or necessary, when all of the dirdata objects are in use. In that case, the number of directory entries stored in the dirdata object is allowed to exceed the threshold. If new servers are added to the system, overloaded dirdata objects can resume splitting to dirdata objects on new servers.

The dirdata splitting process follows the splitting method of GIGA+ mentioned in Section 2. If the initial number of active dirdata objects is n, then during directory creation, the first n bits of dirdata bitmaps will be set to '1' and their depths will be calculated. Upon splitting from a dirdata object with an index of i and depth of r, the index of the new dirdata object is $i + 2^r$ if the index is still within range. Directory entries with a '1' on the $r + 1^{st}$ bit position of their hash values will be moved to the new dirdata object. Both dirdata objects will bump to a depth of $r + 1$ afterwards.

After splitting is completed between two dirdata objects, an updated bitmap is sent to the server holding the metadata object of the directory to update the bitmap there, basically setting the bit at the new dirdata position as '1'. With this mechanism, the metadata object will have the most up-to-date information of the directory at all times. If the client uses an out-of-date bitmap in its cache and finds that the directory entry is no longer on the indicated dirdata server because of splitting, it can always turn to the metadata object for the newest copy of the bitmap and start again. This simplifies the synchronization process and a client only needs a maximum of two probes to locate the correct dirdata object, comparing to possible multiple probes in GIGA+ because of the lazy client update mechanism they used [10].

3.2.4 Maintenance and Removal

Some attributes, such as the timestamp information and the number of directory entries, will be distributed with the dirdata and collected by the client if needed. Only active dirdata objects are contacted for these distributed attributes and directory entries. Other attributes like permissions are kept in the metadata object, similar to regular files.

The removal of a scalable distributed directory is conducted mainly by the server which holds the metadata object to simplify the error handling procedure. It will make sure all the dirdata objects are deleted before the removal of the metadata object of the directory. If anything goes wrong when removing dirdata objects, the dirdata array can be rebuilt to restore the functionality of the directory.

3.3 Current Status

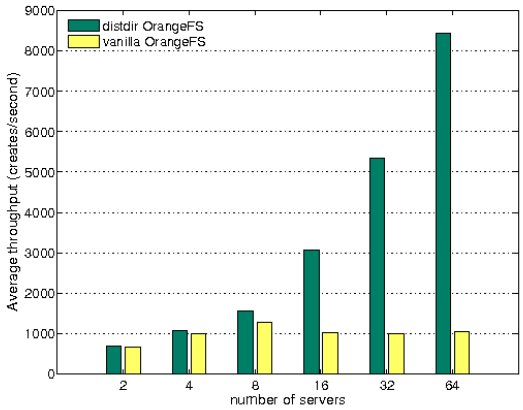
At the time when this paper is prepared, the main framework of scalable distributed directory has been implemented in OrangeFS and is available as an experimental release. Clients are able to conduct directory operations, like create and delete files, list directory, change directory names and etc. The number of active dirdata objects can be specified when creating a directory. We are still working on the function of splitting directory entries between dirdata objects.

4 Performance Evaluation

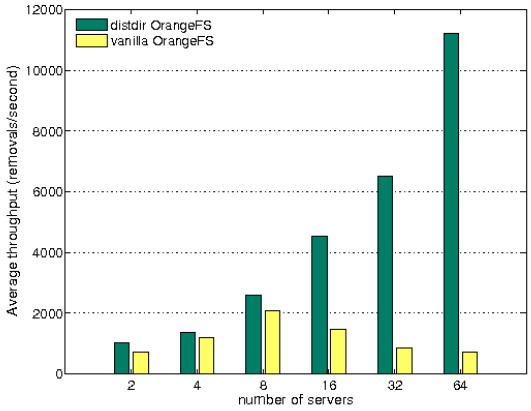
The performance of the scalable distributed directory was evaluated on the Palmetto Cluster housed by Clemson Computing and Information Technology [3]. The cluster was configured as follows at the time of our experiments. There were 1,541 nodes in total with 8 cores per node. The nodes were running Linux 2.6.18 and equipped with Intel Xeon E5345/E5410/L5420 at 2.33GHz/2.5GHz or AMD Opteron 2356 at 2.3GHz and memory sizes of 12GB/16GB. The interconnect network utilizes Myrinet 10G and high throughput storage is attached to all nodes.

The UCAR metarates benchmark [4] is an MPI application that measures throughput of file creation rates in one directory with multiple clients concurrently. We modified the benchmark program to use OrangeFS native APIs to communicate with OrangeFS servers and add a function to measure the file removal rates under a directory. The throughput results from creating and removing empty files under one directory with different numbers of OrangeFS servers is illustrated in Figure 5. The number of clients is twice the number of OrangeFS servers. The throughput of scalable distributed directory shows great scalability as the number of servers increase and can reach more than 8,000 file creations and over 11,000 file removals on average in one second with 64 servers.

Next, the number of clients is varied when running the benchmark on a 64 server setup. The results are illustrated in Figure 6. The results show that OrangeFS with scalable distributed directories can scale with the number of clients up to twice the number servers, while the vanilla OrangeFS can not scale up with the number of clients. Figure 6 also shows the results with 192 clients and 256 clients, where the average file creations per second maintains over 8,000 and the average file removals per second drops a little from 11,000 but still lingers above 8,000. This means the servers are overloaded with client requests and are reaching their full capacities. The faster drops in file removal may be due to the OrangeFS file removal implementation because the vanilla OrangeFS shows similar trend as servers are saturated.



(a) average file creation throughput with different numbers of servers



(b) average file removal throughput with different numbers of servers

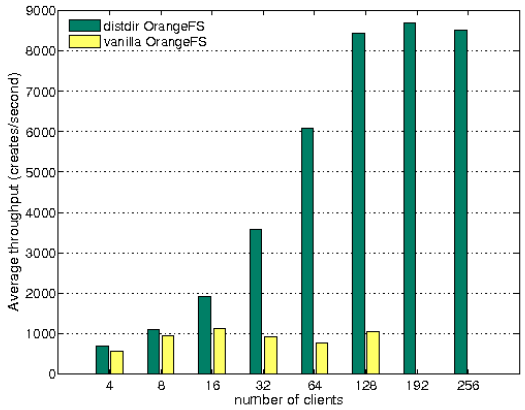
Figure 5. Scale and performance of scalable distributed directory implementation on OrangeFS with different number of OrangeFS servers. The UCAR metarates benchmark is used and the number of clients are selected to be 2x the number of servers.

5 Conclusions and Future Work

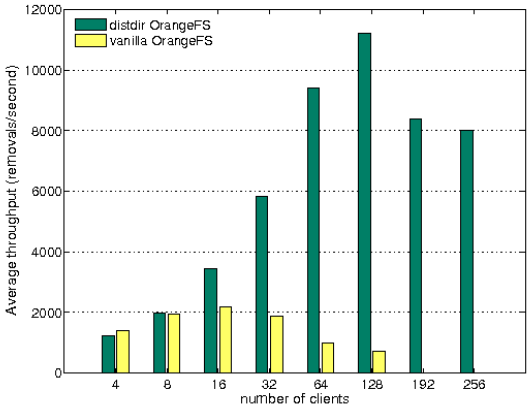
This paper describes a scalable distributed directory design and implementation on Orange File System. Similar to data files, directory entries are distributed across multiple servers. The number of partitions can be specified during directory creation and dynamically incremented when filling the directory. Files are assigned to a specific partition by hashing their names to achieve load balancing among partitions. Clients are responsible for contacting the correct server when initiating a request.

A complete scalable distributed directory implementation on OrangeFS has been carried out except the dynamic splitting function when the paper was prepared. A modified version of UCAR metarates benchmark is used to evaluate the performance of the scalable distributed directory. Comparing with vanilla OrangeFS, the scalable distributed directory shows great scalability in creating and removing large number of files by multiple clients concurrently. On a 64-servers setup and 128 clients accessing the same directory concurrently, the scalable distributed directory can achieve more than 8,000 file creations per second and over 11,000 file removals per second on average.

We are working to complete the splitting function of the



(a) average file creation throughput with different numbers of clients on 64-server system



(b) average file removal throughput with different numbers of clients on 64-server system

Figure 6. Scale and performance of 64 servers scalable distributed directory implementation on OrangeFS with different number of clients.

scalable distributed directory and optimize the current implementation, including optimizing collective communications and the readdir functionality. Afterwards, we plan to instrument the scalable distributed directory and obtain detailed measurements on performances like throughput, scalability, overheads and etc.

6 Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. CCF-0621441 and Omnibond Systems LLC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We would like to thank Clemson Computing and Information Technology for the use of the Palmetto Cluster resource. Michael Moore and Becky Ligon from Clemson Computing and Information Technology provided great assistance during the work.

7 Availability

The scalable distributed directory feature is available as an experimental release of OrangeFS at

`http://orangefs.org/download/`

References

[1] Lustre a Network Clustering FS. `http://www.lustre.org`.

[2] Orange File System. `http://orangefs.org`.

[3] The Palmetto Cluster. `http://citi.clemson.edu`.

[4] UCAR Metarates Benchmark. `http://www.cisl.ucar.edu/css/software/metarates`.

[5] TOP500 Supercomputing Sites. `http://www.top500.org`, November 2010.

[6] P. H. Carns. *Achieving Scalability in Parallel File Systems*. PhD thesis, Clemson University, Clemson, SC, USA., 2005.

[7] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th annual Linux Showcase & Conference*, Berkeley, CA, USA, 2000. USENIX Association.

[8] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible Hashing - A Fast Access Method for Dynamic Files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[9] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, Monterey, CA, USA, 1999.

[10] S. Patil and G. Gibson. GIGA+: Scalable Directories for Shared File Systems. Technical Report CMU-PDL-08-110, Carnegie Mellon University Parallel Data Lab, October 2008.

[11] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the Conference on File and Storage Technology (FAST'02)*, Berkeley, CA, USA, January 2002.

[12] B. W. Settlemyer. *A Study of Client-Based Caching for Parallel I/O*. PhD thesis, Clemson University, Clemson, SC, USA, August 2009.

[13] R. S. Studham and R. Subramaniyan. Lustre: A Future Standard for Parallel File Systems? Invited presentation at International Supercomputer Conference, June 2005.

[14] S. A. Weil, S. A. Brandt, and E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'04)*, November 2004.

[15] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th symposium on Operation systems design and implementation*, Berkeley, CA, USA, 2006.

[16] Y. Wu. A Study for Scalable Directory in Parallel File Systems. Master's thesis, Clemson University, Clemson, SC, USA, July 2009.

[17] X. Zhang, K. Davis, and S. Jiang. IOrchestrator: Improving the Performance of multi-node I/O Systems vis Inter-Server Coordination. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2010.