Survey on Bottlenecks and Best Practices of Three Implementations of Distributed File Systems

Samuel Li
Information and Computer Science Department
University of Oregon
samuelli@cs.uoregon.edu

Abstract— **File systems are crucial components of supercomputers and internet-based service providers, such as Google. Often, file systems consist of many file servers to provide capacity, performance, and reliability. They are organized as a distributed system to provide functionalities of a file system. A distributed file system can have several performance challenges, due to the distributed nature of the file system itself, and the high concurrency nature of the supercomputer applications. This survey paper studies architecture and performance issues of three widely used file systems: Google File System, GPFS, and Lustre. It provides the readers a better understanding of how distributed system works, and what to expect from a distributed file system.**

I. INTRODUCTION

File systems are a crucial component of many large-scale systems, namely the supercomputers and internet-based service providers. While the computational capacity of modern supercomputers are keep growing, the file system performance is not keep up the pace, in terms of both the storage capacity and data transfer throughput. Distributed file systems are widely used to tackle this problem. On the one hand, distributed file systems enables a large number of commodity storage devices to connect together and act like a unified storage space, which expands the storage capacity. On the other hand, distributed file systems can potentially aggregate file I/O operations from single storage devices together, providing a high data transfer throughput. This survey paper sheds some light on the architecture as well as various performance issues of distributed file systems.

This survey paper specifically looks into three popular distributed file systems: Google File System GPFS, and Lustre. Google File System [1] is designed by Google and used exclusively by Google. GPFS [2], [3] is a proprietary system owned by IBM, and it also powers many supercomputers. Lustre [4] is open-sourced and powers many of the world's fastest supercomputers. We will especially focused on three aspects of these file systems: 1) different architecture design, 2) file partitioning scheme, and 3) efforts and demonstrations to achieve higher performance using these three file systems.

II. ARCHITECTURE OF THREE FILE SYSTEMS

A. Google File System

The Google file system is the proprietary file system designed by Google [1]. It has a few impressive properties, such as the ability to reach a global scalability [5], [6], and the good performance on structured data [7]. However, publications are limited due to its proprietary nature. This section gives an overview of Google file system (GFS) based on the available publications.

The initial design choices of GFS are explained by Ghemawat et al. [1]. It started from four observations regarding the unique usages and requirements by Google:
1. “component failures are the norm rather than the exception";
2. “files are huge by traditional standards";
3. “most files are mutated by appending new data rather than overwriting existing data"; and
4. “co-designing the applications and the file system API benefits the overall system by increasing our flexibility."
The extra flexibility as described by the fourth item enables GFS to make some radical design choices,

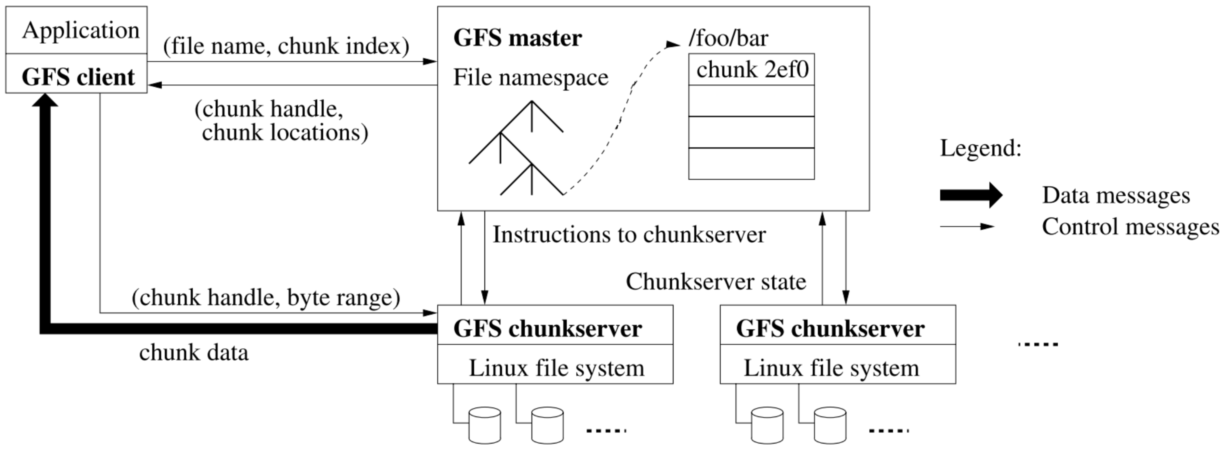

Fig. 1. GFS architecture

for example, it does not comply to any standard API. In contrast, the other two surveyed file systems both comply the standard API of POSIX. We will revisit these observations and discuss how they impact the design of GFS in the following discussion.

A.1 Master-Server Architecture: Master

GFS adopts a master-server architecture. More specifically, each GFS cluster consists of one master node and multiple chunkservers; both are commodity machines.

The roles of master include:
1. maintaining all file system metadata, including namespace, access control, mapping from files to chunks, and current locations of chunks;
2. monitoring system state by sending periodical heartbeat messages and collecting system state of individual chunkservers;
3. controlling system-wide activities, such as lease management, garbage collection, chunk migrations; and
4. answering requests from clients.

Because there is only one single master node in a GFS system, the master can easily become a system bottleneck. To prevent this from happening, the master takes a minimum involvement in the fourth task: answering requests from clients. More specifically, the master does not handle any of the actual file I/O operations; rather, it sends instructions and re-directs actual file I/O operations to the proper chunkserver nodes. These instructions include which chunkservers to look for, and what chunk handles to use. Clients then interact with file chunkservers to finish the actual file I/O operations.

A.2 Master-Server Architecture: Chunkserver

The chunkservers in GFS are machines that actually stores data files. Data is stored as chunks in GFS, so each chunkserver stores many data chunks. The chunk size is an important parameter to tune the GFS, and 64MB is decided to be a good balance between disk utilization and system performance. A chunkserver always performs tasks from either instructions from the master, or the clients. Typically, these tasks include:

1. read and write operations per requests by clients;
2. maintenance tasks such as replicating an existing data chunk;
3. lease management, including requiring a lease, extending a lease, and releasing a lease; and
4. rallying in a data flow along a chain of chunkservers.

We will discuss the last task in Section **??**.

The overview of GFS architecture is shown in Figure 1.

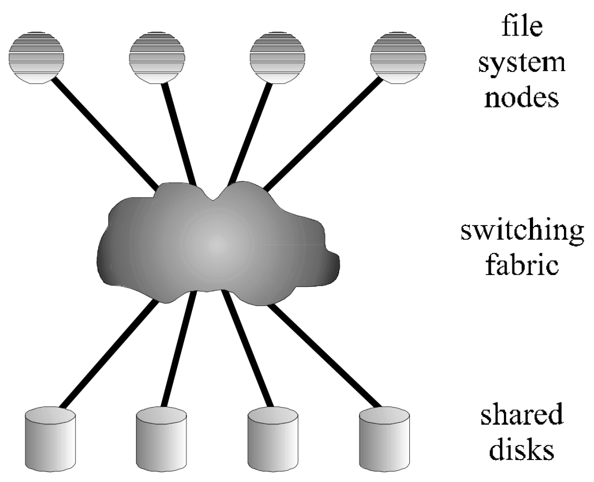

Fig. 2. GPFS architecture

B. *IBM GPFS: General Parallel File System*

The General Parallel File System (GPFS) from IBM is a popular file system that is more widely adopted than the Google File System. In fact, it powers some of the most powerful supercomputers in the world, including the third and fifth fastest supercomputers (Sequoia and Mira respectively) in the latest Top500 list [8]. Accordingly, there is more published research on GPFS than GFS. However, because of the proprietary nature of GPFS, the publications on GPFS is also relatively limited. We provide an overview of the architecture of GPFS in this section.

B.1 Decentralized Architecture

GPFS adopts a decentralized architecture. More specifically, there are cluster nodes and disks or disk subsystems. Clusters and disks are connected over a switching fabric. The architecture is decentralized in the sense that all cluster nodes have equal to the disks, and they provide equal functionalities and access to user applications [3], [2].

Data stored on GPFS is distributed into many stripes, and are essentially placed on different disks. This design enables parallel data access, which helps improving the overall I/O performance. We will discuss more about this parallel data access later in Section III-B.

The locking mechanism of GPFS is mostly distributed as well with a few tasks performed in a centralized fashion. The centralized locking routines mainly takes care of updates on metadata and configuration files. Even these routines are centralized, the central coordinator is elected from the pool of clusters, and can hardly run into a single-point failure problem. Figure 2 provides an overview of the GPFS architecture.

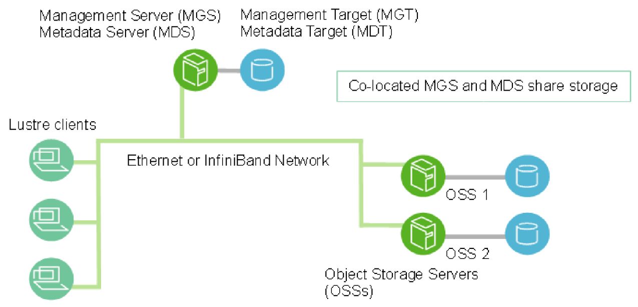

Fig. 3. Lustre architecture

C. *Lustre File System*

Lustre is an open-sourced file system. Like GPFS, Lustre has also been used by many of the world's top supercomputers [8]. It is also the most intensively researched distributed file system among the three.

C.1 Lustre File System Architecture

The Lustre file system has a similar architecture as GFS: one node serves as management server (MGS); multiple object storage servers (OSS) actually stores data; management server and object storage servers are connected via high-speed network [4], [9]. It should also be noted that later releases of Lustre do have support for multiple metadata servers to provide better reliability

On the management server side, it keeps all the metadata in object files, named metadata target (MDT). MDTs have information filenames, directories, permissions and file layout. The MGS also provides network request handles for local MDTs.

On the object storage server side, data is grouped as object storage targets (OSTs). The mapping between OSSs and OSTs can be flexible, which means one OSS can have multiple OSTs, or multiple OSSs have access to one OST, or even a more complex n to m mapping. The final choice of mapping scheme is based on the usage and the choice of hardware. For example, one OSS can host two OSTs to achieve redundancy if using commodity storage, while utilize the complex n to m mapping to achieve high performance if using enterprise-class storage arrays. Figure 3 provides an overview of the Lustre architecture.

D. Discussion on Three Architectures

The three surveyed file systems, GFS, GPFS, and Lustre represent two distinct architectures: architecture with a master node or architecture without a master node. Both architectures have advantages and disadvantages.

GFS, Lustre have a master node, and this design has a major advantage that it is easy to perform maintenance creating data replica; migrating partial data; detecting and from failures; communicating with clients; etc. The master node has this advantage because it possesses a global knowledge of the entire file system. The disadvantage is also obvious that the master node can cause a single-point failure. This disadvantage can be largely avoided by providing a replica of the master node, as what Lustre does. A less severe drawback is that the master node can become a system bottleneck. Modern supercomputers tend to equip a large amount of memory and fast storage such as solid state drives to tackle this problem.

GPFS uses a decentralized architecture without a master node. This design has an obvious advantage that it has better tolerance on failure of single nodes. This property is quite important on large scale systems. The disadvantage of this decentralized architecture is that implementation of many operations, like locks or leases, becomes complicated in many cases.

III. Distributed File Partitions

Distributed file systems provide a unified interface to applications and clients, yet they consist of a number of disks and servers. Partitioning files and put them on separate disks and servers is a common practice, because this approach provides aggregated storage capacity and bandwidth. This section surveys partitioning strategies adopted by the three file systems, and disscuss the advantages and disadvantages of them.

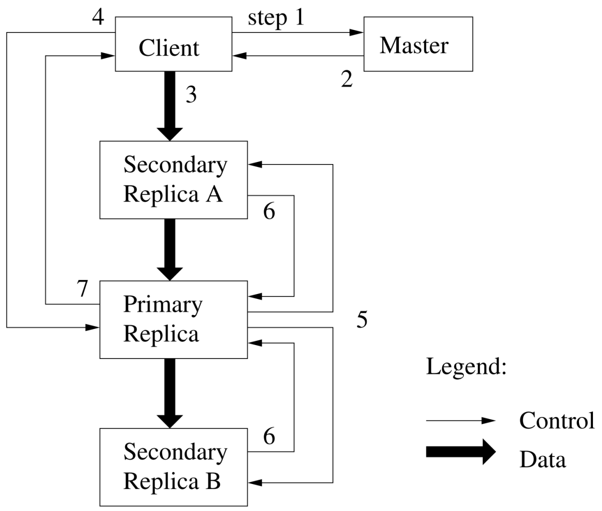

Fig. 4. GFS data flow

A. Chunks in Google File System

Google File System partition data into chunks [1]. Each chunk has a relatively big size: 64MB. This relatively large chunk size has a few benefits:

1. it reduces clients' need to interact with the master node, because reads and writes on the same chunk require only one initial request to the master;
2. it reduces the network overhead by keeping a persistent TCP connection to the chunkserver over a longer time, rather than establishing multiple connections;
3. it reduces the size of metadata saved on the master, which helps improve the overall system performance.

Each chunk has replicas across the entire GFS. By default, there are three replicas. The benefit of replicas is two-fold. First, this redundancy provides reliability. In case of one or even two replicas fail, the data is still safe. Second, replicas located in different servers provide more flexible access to clients. When reading data, a client reads from a certain replica that is 1) has less network delays to the client; and 2) resides on a server with less load.

A.1 Data Flow in GFS

The chunk replication mechanism of GFS requires careful handling when writing data onto the disk. Specifically, writing to multiple replicas should not take multiple times longer. GFS uses two approaches to speed up data write on multiple replicas:

1. decouple the expensive data flow and inexpensive control flow;
2. data flows in a pipelined fashion along a chain of chunkservers.

Data flow starts from the client who has a write task, following a linear chain of chunkservers. To make best use of the machine's network bandwidth, each machine forwards the data to the "closest" machine in the network topology that has not received it. This forwarding step happens immediately after a chunkserver receives some data. Each chunkserver puts received data in the buffer without immediate commit; rather, it only commits until all chunkservers finish receiving data, and receives a commit instruction.

Control flow communicates different information between different nodes:

1. master tells client which chunkserver holds the current lease for the chunk and the locations of the other replicas;
2. client sends a commit instruction to the "primary" of the multiple chunkservers when all chunkservers finish receiving data;
3. the "primary" chunkserver sends commit instruction to all the secondary chunkservers; and
4. the "primary" chunkserver replies to the client that all replicas are successfully committed.

The benefit of the decoupling of data flow and control flow is that control flows become light-weighted so that they are more tolerant to failures of any kind. Data flow is heavy-weighted, but this operation is relatively simple: it happens in an "all or nothing" fashion. Data is committed only after all chunkservers successfully receive the full data to write. This data flow paradigm is shown in Figure 4.

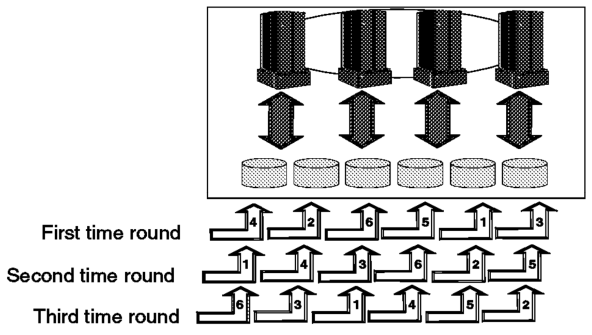

Fig. 5. Balanced Random striping. Disk order is not enforced from the first to the second to the third round, but the overall load across multiple disks is kept balanced.

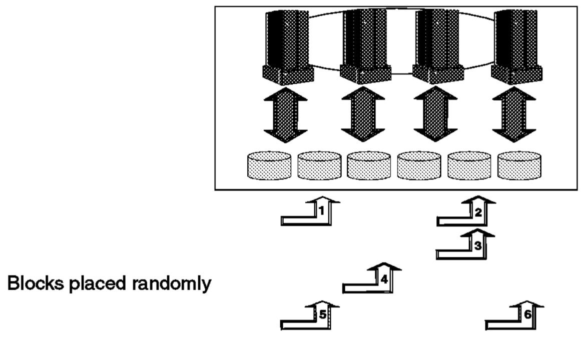

Fig. 6. Random striping. Each data block is randomly placed on a disk. Load balance is not ensured by this striping scheme.

B. Stripes in GPFS

GPFS adopts a decentralized architecture (Section II-B), so files are striped across all disks in the file system [2], [3]. Each file is divided into equal sized blocks, and consecutive blocks are placed on different disks in a round-robin fashion by default. The choice of block size is very different from that of GFS though: GPFS allows block size between 16KB and 1MB, with 256KB being the default size (compared to the 64MB chunk size by GFS).

To ensure maximum parallelism when reading a large file, GPFS issues I/O requests in parallel to as many disks as necessary. Similarly, data buffers that are no longer being accessed are written to disk in parallel. This practice is enabled by the property of GPFS that all nodes in the cluster have equal access to all disks II-B; it also achieves reading or writing data from/to a single file at the aggregate data rate supported by the underlying disk subsystem and interconnection fabric.

The round-robin fashion to place file blocks requires equal size and performance among all disks to achieve best disk utilization and performance. A non-uniform disk configuration requires a trade-off between throughput and space utilization: maximizing space utilization means placing more data on larger disks, but this reduces total throughput. GPFS allows the administrator to make this trade-off by specifying whether to balance data placement for throughput or space utilization.

Besides round-robin, GPFS provides another two striping options, mainly to ease the strict disk order enforced by round-robin option: 1) balanced random striping, and 2) random striping. Round-robin achieves best performance but it also suffers from the longest time to re-stripe the files in any unexpected event. Balanced random differs from RoundRobin in that each data block is written to a disk selected randomly. However, the same as in the round-robin method, the same disk is not selected until all the disks within the stripe group have been used. Figure 5 illustrates this striping method. With random striping, each block of data is written on a disk selected randomly. No surprise, this method does not assure that the traffic will be balanced among all the disks. Balanced random and random striping approaches are usually used when re-striping is expected to happen frequently.

C. *Stripes in Lustre*

Striping in Lustre follows a similar pattern as it is in GPFS, in that files are divided into multiple stripes to achieve aggregated performance. This section we address two major differences between them though.

The first difference is that Lustre supports locating file blocks based on the available free space on each object storage target (OST) [4], [9]. More specifically, when the free space across OSTs differs by more than a specific amount (17% by default), the metadata server (MDS) then uses weighted random allocations with a preference for allocating objects on OSTs with more free space. This can reduce I/O performance until space usage is rebalanced again.

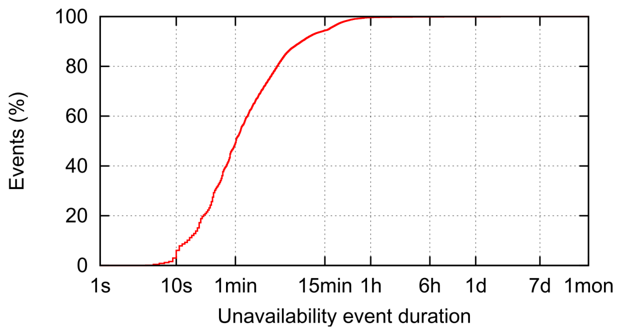

Fig. 7. Cumulative distribution function of the duration of node unavailability periods.

The second difference is that Lustre uses a large block size than GPFS. In contrast to the 16KB to 1MB in GPFS, Lustre supports a 512KB to 4GB block size.

D. *Discussion on Partitioning Schemes*

Among the three surveyed file systems, there is a big difference regarding the file access patterns between GFS and the other two: GPFS and Lustre. In GFS, most files are mutated by appending new data rather than overwriting existing data. A much larger block size helps to reduce the overload of asking and exchanging the file locks. While in GPFS and Lustre, most file operations are normal creating and reading tasks, resulting in the design choice of smaller block sizes.

GFS also puts more emphasis on data replications. Particularly, GFS itself manages the replications. The GPFS and Lustre do not focus on data replications though. They mostly use Redundant Array of Independent Disks (RAID) and leave the data replication job to RAID.

IV. Achieve an Even Higher Performance

We survey the efforts researchers have made to achieve an even better performance on these three file systems. The better performance here refers to characteristics such as throughput, scalability, etc.

A. *GFS: Toward a Global Scalability*

To achieve great scalability, a good handling of failures is necessary. In the real application of GFS, many storage server nodes are called a cell. A typical cell may comprise many thousands of nodes housed together in a single building or a set of co-located buildings.

To characterize the failures and availability of a GFS cell, we start from the availability of single storage nodes. A storage node becomes unavailable when it fails to respond positively to periodic health checking pings. Nodes can become unavailable for a large number of reasons. For example, a storage node or networking switch can be overloaded; a node binary or operating system may crash or restart; a machine may experience a hardware error; or the whole cluster could be brought down for maintenance. The vast majority of such unavailability events are transient and do not result in permanent data loss. In fact, statistics show that less than 10% of events last longer than 15 minutes (see Figure 7). For this reason, GFS typically waits 15 minutes before commencing recovery of data on unavailable nodes.

Two measurements are used to measure the stability of the GFS: average availability and mean time to failure (MTTF). Given a cluster of N nodes, average availability is defined as following:

$$A_N = \frac{\sum_{N_i \in N} uptime(N_i)}{\sum_{N_i \in N}(uptime(N_i) + downtime(n_i))} \tag{1}$$

, where $uptime(N_i)$ and $downtime(N_I)$ refer to the length of time a node N_i is available or unavailable. MTTF is defined as following:

$$MTTF = \frac{uptime}{numberoffailures}. \tag{2}$$

A.1 Data Replication and Chunk Placement

When a node failure causes the unavailability of a chunk within a stripe, we initiate a recovery operation for that chunk from the other available chunks remaining in the stripe. Distributed file systems will necessarily employ queues for recovery operations following node failure. These queues prioritize reconstruction of stripes which have lost the most chunks.

To minimize the effect of large failure bursts in a single failure domain, we also consider a rack-aware policy. A rack-aware policy is one that ensures that no two chunks in a stripe are placed on nodes in the same rack. Research has shown that using a rack-aware placement policy increases the stripe MTTF by a factor of 3 typically.

The researchers also used a Markov model to validate some findings. Two interesting finds are:
1. improvements of component failure rates below the node layer of the storage stack do not significantly improve data availability. For example, a 10% reduction in the disk failure rate increases stripe availability by less than 1.5%. On the other hand, cutting node failure rates by 10% can increase data availability by 18%.
2. Replicating data across multiple cells greatly improves availability because it protects against correlated failures. It also greatly increase the inter-cell bandwidth requirement, so there is still trade-offs to make.

A.2 Google Services Built on Top of the GFS

Because of the extreme scalability of GFS, other services become available on top of the GFS. Corbett et al. [6] demonstrated a database system in in Google. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. Chang et al. [7] demonstrated Bigtable, a distributed storage system for managing structured data. It is able to scale to a very large size: petabytes of data across thousands of commodity servers. Even though many Google applications place diverse demands on Bigtable, Bigtable is still able to provide high-performance solutions for all these services.

B. GPFS to Achieve Better Performance

GPFS has demonstrated great performance in high scalability, fast scan, and high throughput. There are many research of GPFS on real systems, for example, Yu et al. [10] demonstrated a highly scalable GPFS file system with satisfactory overall performance; Andrews et al. [11] researched from both theoretical and practical sides of the performance of GPFS file system in an inter-state scale. Further, Freitas et al. [12] reported performance of GPFS file system to scan 10 billion files in 43 min-

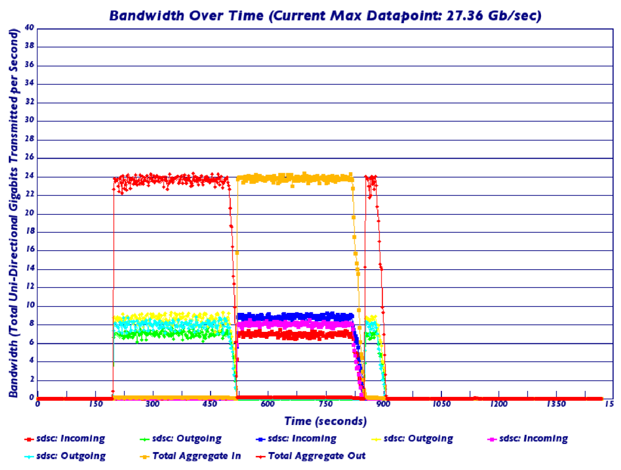

Fig. 8. Measured WAN performance at SC'04 for sequential reads and writes.

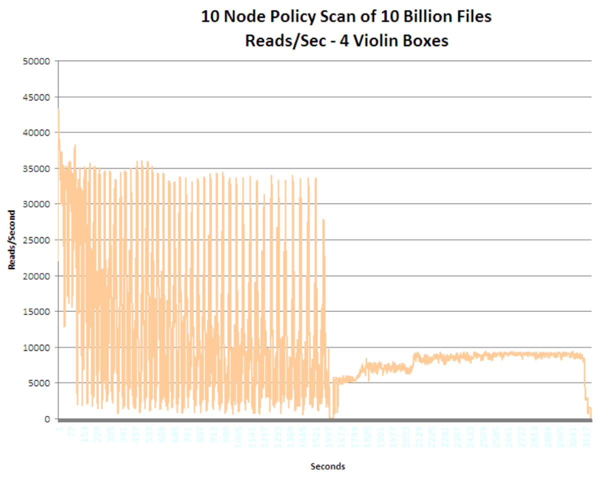

Fig. 9. Aggregate read operations per second.

utes. We elaborate these results in the following subsections.

B.1 GPFS to achieve global scalability

Andrews et al. [11] have demonstrated the high scalability of the GPFS file system in two consecutive SuperComputing (SC) conferences. In these demonstrations, data centers across the US continent are connected using the GPFS file system. These data centers include: San Diego Supercomputing Center (SDSC) in San Diego, CA, the National Center for Supercomputing Applications (NCSA) in Urbana, IL, and the conference location in Phoenix, AZ and Pittsburgh, PA. In the SC'03 conference, the researchers used a 10Gb/s connection at the conference center, and the GPFS achieves around 9Gb/s read/write speed working on storage in San Diego. This is about 90% percent of the connection bandwidth. In the SC'04 conference, the researchers used a 40Gb/s connection at the conference center, with GPFS file system connecting to storage in both SDSC and NCSA sites. They achieved around 27Gb/s sustaining speed, which is about 67% of the connection bandwidth. Figure 8 plots the bandwidth overtime in this test. Given the large physical distance (from the west coast to the east coast) of these experiments, GPFS has demonstrated great scalability.

B.2 GPFS to Achieve Fast Scan

Distributed file systems are good at providing big throughput, but are traditionally bad at operations on many small files. Freitas et al. [12] experimented the use of solid state drives (SSDs) to store the metadata of a file system, and reported satisfactory result on scanning 10 billion files on a GPFS file system. It took 43 minutes in this test.

The scan over 10 billion files takes 2 steps. In the first step, it parallelly traverses all directories. Each processor takes care of a sub-tree. When this phase is complete, the full path to every file and that fileâĂŹs inode number is stored in a series of temporary files. The second step is sequential scan. It begins by assigning the temporary files to the processes running on each node. Each set of temporary files contains a range of inodes in the file system. The filesâĂŹ inodes are read directly from disk via a GPFS API, thus providing the filesâĂŹ attributes such as owner, file size, atime and it includes the filesâĂŹ extended attributes. Figure 9 shows the number of reads per second in this test.

The first step takes about 20 minutes. This step is highly input operations intensive, because it reads many random locations in the file system. This step of scan greatly benefits from the underlying SSD storage that keeps all the metadata. The second step takes about 23 minutes. It reads the temporary files created by the first phase. It is essentially bandwidth bound.

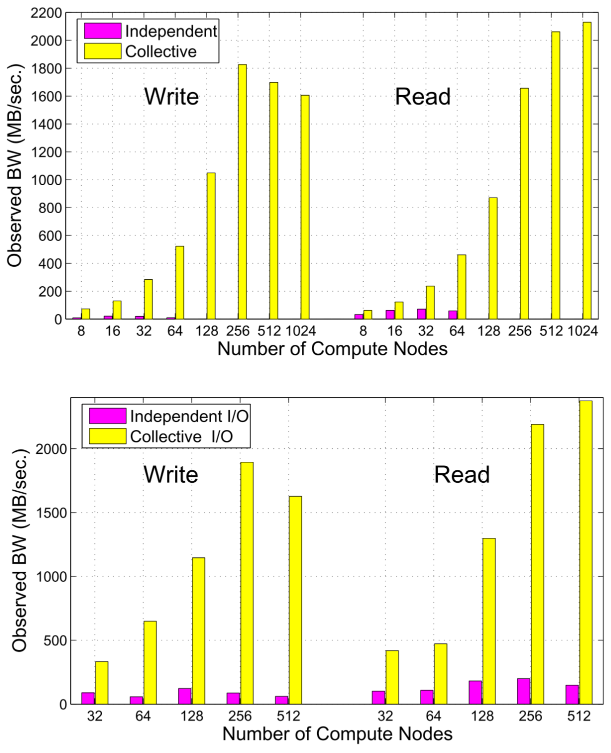

Fig. 10. Improvement on bandwidth of GPFS file system after applying collective I/O.

B.3 GPFS to Achieve High Bandwidth

Ross et al. [10] used MPI I/O on top of GPFS to achieve good read/write performance on an IBM Blue Gene supercomputer. They used the ROMIO MPI implementation, and tuned it to work best with GPFS. For example, they implemented collective buffering in MPI I/O, which rearranges and aggregates data in memory prior to writing to files to reduce the number of disk accesses. For scientific applications, collective buffering is effective for achieving scalable I/O. They also replaced a few MPI implementations based on the characteristics of the GPFS system. For example, they replaced the use of the point-to-point functions with MPI_Alltoallv, which can utilize up to 98% of the peak bandwidth within a compute node. They also replaced the MPI_Allgather with an MPI_Allreduce, which performs much better for short and medium sized messages. The results show that the achieved bandwidth with collective I/O is significantly improved, as shown in Figure 10.

C. *Higher Performance on Lustre File System*

The Lustre file system receives most intensive research because of its open source nature. For example, Crosby [13] characterized the performance of Lustre file system on a real supercomputer system; Xie et al. [14] specifically characterized output performance with respect to a number of system parameters; Schwan [4] and Henschel et al. [15] demonstrate implementations of Lustre to achieve the best performance on real systems; Lofstead et al. [16] specifically researched interference effects measured on two real systems; and Shipman et al. [17] summarized real lessons learned to achieve high performance on a very large scale Lustre file system.

To focus our discussion on the core performance issues, this subsection first identifies a few parameters that affect the performance of Lustre file system in a real production environment, and then provides a few demonstrations of Lustre file system to achieve both high scalability and performance.

C.1 Performance Characteristics of the Lustre File System

Crosby et al. [13] characterized the performance of Lustre file system on a real system: a Cray XT5 supercomputer at the Oak Ridge National Lab. Striping is an important mechanism in Lustre to achieve high throughput III-C, so it is important to test the I/O performance as a function of the stripe sizes. The researchers performed read and write tests on data files with size varying from 32MB to 5GB on the real system, and plotted their performance as shown in Figure 11. It shows that both write and read performance significantly degrades when the stripe count is greater than 32, with 16 or 4 probably being the sweet spot. These tests also show that a larger sized stripe, for example 32MB other than 1MB, is helpful to achieve a better performance.

C.2 High Scalability of Luster File System

Researchers have also investigated the capability of Lustre file system to achieve a great scala-

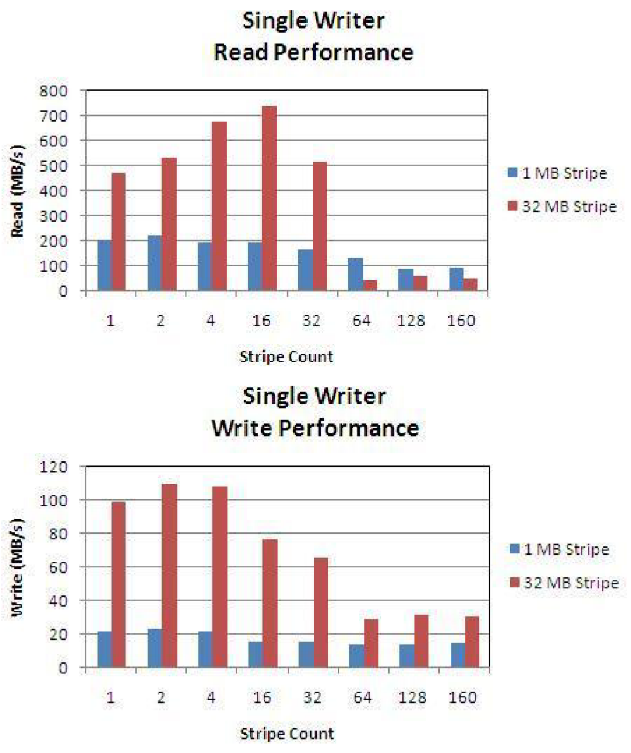

Fig. 11. Read and write performance of Luster file system on a real system as a function of stripe count.

bility. Henschel et al. [15] demonstrated a Lustre file system over a 100Gbps wide area network of 3,500km. These researchers compared the same setups of hardware and software on a local machine room, as well as spanning from Seattle to Indianapolis. Evaluations are performed on both benchmark test suits as well as real-world applications. The results are shown in Figure 12.

These evaluations show a 20% to 30% degradation of throughput when the file system is set to cross the country. This result is satisfactory considering the many technical difficulties associated with a WAN and the complexity of a distributed file system. These results serve as a concept demonstration that a Lustre file system with very big scales is possible.

V. Conclusion

This paper surveys performance factors of three popular distributed file systems: the Google File System (GFS), IBM General Parallel File System (GPFS), and Lustre File System. We especially focused on the 1) different architecture design, 2) file partitioning scheme, and 3) efforts and demonstrations to achieve higher performance using these three file systems. Architecture wise, both GFS and Lustre adopt a master-server mode, whereas GPFS adopts a decentralized architecture. File partition scheme is more similar among these three file systems, with a different in the choice of chunk size. Higher performance is achieved by all three file systems, including high robust to failures, high throughput, and high scalability to span thousands of miles.

References

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The google file system,” in *ACM SIGOPS operating systems review*. ACM, 2003, vol. 37, pp. 29–43.

[2] F Schmuck and R Haskin, “GPFS: A Shared-Disk File System for Large Computing Clusters,” *Proceedings of the First USENIX Conference on File and Storage Technologies*, , no. January, pp. 231–244, 2002.

[3] Jason Barkes, Marcelo R Barrios, Francis Cougard, Paul G Crumley, Didac Marin, Hari Reddy, and Theeraphong Thitayanun, “Gpfs: a parallel file system,” *IBM International Technical Support Organization*, 1998.

[4] Philip Schwan, “Lustre: Building a File System for 1,000-node Clusters,” *Proceedings of the Linux Symposium*, pp. 401–409, 2003.

[5] Daniel Ford, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan, “Availability in Globally Distributed Storage Systems,” *Operations Research*, pp. 61–74, 2010.

[6] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J J Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, “Spanner : Google ’ s Globally-Distributed Database,” *Proceedings of OSDI 2012*, pp. 1–14, 2012.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, “Bigtable: A distributed storage system for structured data,” *7th Symposium on Operating Systems Design and Implementation (OSDI ’06), November 6-8, Seattle, WA, USA*, pp. 205–218, 2006.

[8] Erich Strohmaier, “Top500 supercomputer,” in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006, SC ’06, ACM.

[9] Peter J Braam, “The lustre storage architecture. cluster file systems inc. architecture, design, and manual for lustre, november 2002,” .

[10] Hao Yu, Ramendra K Sahoo, C Howson, G Almasi, JG Castaños, Manish Gupta, José E Moreira, JJ Parker, TE Engelsiepen, Robert B Ross, et al., “High performance file i/o for the blue gene/l supercomputer,” in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*. IEEE, 2006, pp. 187–196.

KEY RESULTS OF THE DEMONSTRATION. PERCENTAGE OF THEORETICAL PEAK IS GIVEN IN PARENTHESIS.

	Compute nodes to local storage over 100G	**From Seattle compute nodes to Indianapolis storage**
Round trip time (ping)	0.24 ms	50.5 ms
TCP iperf, one stream	9.8 Gbps (98%)	9.8 Gbps (98%)
TCP iperf, 30 streams (peak)	98 Gbps (98%)	96 Gbps (96%)
IOR, 1 client, 8 servers	1.2 GB/s (96%)	1.2 GB/s (96%)
IOR, 30 clients, 16 servers	9.6 GB/s (77%)	6.5 GB/s (52%)
8 Applications concurrently (peak)	8.8 GB/s (70%)	6.2 GB/s (50%)
8 Applications concurrently (sustained)	7.9 GB/s (63%)	5.6 GB/s (45%)

Fig. 12. Scalability test of Lustre File System. This test compares the same software and hardware performing either on the local site, or through a WAN connection.

[11] Phil Andrews, Bryan Banister, Patricia Kovatch, Chris Jordan, and Roger Haskin, "Scaling a global file system to the greatest possible extent, performance, capacity, and number of users," *Proceedings - Twenty -second IEEE/Thirteenth NASA Goddard Conference on Mass Storage Systems and Technologies*, pp. 109–117, 2005.

[12] Richard Freitas, Joseph Slember, Wayne Sawdon, and Chiu Lawrence, "Gpfs scans 10 billion files in 43 minutes," *San Jose, CA, USA*, 2011.

[13] Lonnie D Crosby, "Performance Characteristics of the Lustre File System on the Cray XT5 with Respect to Application I / O Patterns .," pp. 1–6, 2009.

[14] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki, "Characterizing output bottlenecks in a supercomputer," *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2012.

[15] Robert Henschel, Stephen Simms, David Hancock, Scott Michael, Tom Johnson, Nathan Heald, Thomas William, Donald Berry, Matt Allen, Richard Knepper, Matthew Davy, Matthew Link, and Craig A. Stewart, "Demonstrating Lustre over a 100Gbps wide area network of 3,500km," *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2012.

[16] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf, "Managing variability in the io performance of petascale storage systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[17] Galen M. Shipman, David A. Dillow, Sarp Oral, Feiyi Wang, Douglas Fuller, Jason Hill, and Zhe Zhang, "Lessons learned in deploying the world's largest scale Lustre file system," *CUG '10 Proceedings of the Cray User Group meeting*, pp. 1–10, 2010.