Large Scale Distributed File System Survey

Yuduo Zhou
Indiana University Bloomington
yuduo@indiana.edu

ABSTRACT

Cloud computing, one type of distributed systems, is becoming very popular. It has demonstrated easily processing very large data over commodity clusters is possible with correct programming model and infrastructure. One critical issue here lies in the file system (FS) [1]. In this report, I reviewed a number of outstanding distributed file systems (DFS).

KEY WORDS

Cloud computing; data-intensive computing; distributed file system

1. INTRODUCTION

Since the beginning of computing, there are always problems too big for the largest machines to solve. Even with today's powerful supercomputers like IBM Blue Gene or Blue Water. With advanced communication technology, computer scientists can aggregate numbers of machines into computing cluster to effectively unbounded computation power and storage capacity to solve big problems. Thus MapReuce framework [2] and cloud computing is gaining rapid popularity now.

Though numbers of cloud computing systems are broadly used, few of them can deliver satisfied result without the integration with a DFS. As the computation ability increasing, the requirements for a DFS are becoming stricter. Comparing to local FS, many requirements are different and needed to be re-considered when designing a DFS.

- The fault tolerance feature must be well-implemented. The possibility of hardware failure in a 500 nodes cluster is almost 40% even if a single node's error rate is only 0.1%. How fast the data can be recovered after any failure becomes one of the most important requirements here.
- Files stored in DFS are huge by traditional standards. It's very common that most files' size exceed GB level. It's crucial how FS will handle huge files. Some FS will divide files into blocks. The advantage by doing this is downgrading the size of data handled by one operation from several GBs to several MBs. On the other hand, it requires additional mapping procedure for every operation, which may cause performance drop.
- Most files in DFS are in write-once-read-many pattern. With respect to this fact, many DFS provide optimized function for file writer and reader. Few of them also have efficient function to edit an arbitrary position in an existing file. Some DFS don't even provide function to change any existing file.
- Metadata starts to play a key role in data management. Since most DFS claims they support millions of files, it's not possible to efficiently retrieve the information on any given file simply by traversing every node directly. Due to this reason, most DFS assign a certain node as the central, which maintains the metadata of all files stored in the system. The retrieval for file information will become much faster via the metadata list.

- Data consistency is redefined under the scope of clusters. Some loosely consistent systems also have good performance. However, it still needs to be handled well when multi-clients are operating on one file simultaneously.

2. GOOGLE FILE SYSTEM

The Google File System (GoogleFS) is introduced in 2003 to meet the rapidly growing demands of Google's data processing needs [3]. With broadly usage within Google ever since then, the GoogleFS is proved to have good performance, scalability, reliability, and availability. Since it is optimized for Google's core data storage and usage needs, apart from the features mentioned above, it is designed with the following assumptions:

- Most files only need appending new data, rather than overwriting the whole file;
- Workloads primarily consist two kinds of reads, large streaming reads and small random reads;
- Workloads also have many large, sequential writes that append data to files. Small writes at arbitrary position in a file are also supported, but do not have to be efficient;
- High sustained bandwidth is more important than low latency

Files are divided into fixed-size chunks (64 MB) with a 64 bit chunk handle, and distributed to the nodes across the cluster, which is called chunk server. Clients send data request to the master node, which maintains metadata of all chunks and the mapping from file to chunks. The master returns the metadata requested to the client. Then the client is enabled to connect to the chunk server directly for data transfer.

However, since the cluster size can beyond 1000 nodes and all requests go to the master first, which is only a single node. The master may be overwhelmed by simultaneously requests. In this condition the master can become a bottleneck and severely increase the total overhead. To overcome this, the GoogleFS stores metadata only in master's memory, instead of the hard disk. Although the processing speed is increased, size of the whole system is limited by the memory on the master node.

All the data in GoogleFS is triple replicated. Whenever a chunk server is down, the master can always redirect data requests to the replicas, until the node is back online. If the master fails, the system can easily choose another node to generate a metadata list by scanning over all chunk servers and work as master. GoogleFS also equipped with a carefully designed locking function that can handle multi-operation to one same chunk simultaneously.

With all these outstanding features, GoogleFS is considered to be one of the most powerful DFS. However, there still exist some drawbacks within it.

2.1. Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is an open-source version of GoogleFS from Yahoo! Inc. [4]. HDFS follows write-once-read-many pattern and doesn't provide functions to change an existing file. Other than this, it is designed to have similar functions and architectures with GoogleFS. So I will not look into HDFS deeper. However, I will still introduce some variant of HDFS later.

3. Variants of HDFS

The HDFS adapts a successful architecture from the GoogleFS. Since it is an open-source project, HDFS is altered into many different variants with different focuses.

3.1. Ring File System

The Ring FS (RFS) is claimed as a scalable DFS that doesn't have a single point of failure [5]. Since both HDFS and GoogleFS rely on only one master node or name node, it is a potential failure point. Besides, a typical GoogleFS master node can handle a few thousand requests per second [6]; it can be overloaded by the massive parallel applications. This is the reason for the development of RFS.

The RFS is the same as HDFS except for it has multi master nodes act as meta servers. Each meta server maintains part of the metadata by using a Distributed Hash Table (DHT). The key point is each meta server triple replicates its data and keeps it in other meta servers. The metadata can always recover as long as at least one replica survives.

In [5] they claimed the RFS performs better in fault tolerance, scalability and throughput than HDFS. But this doesn't balance the lost in performance or efficiency by adding all additional features. First, both HDFS and GoogleFS monitor the master node closely. Second, the failure possibility on a master node is very low. Third, they provide very high speed recovery method. Therefore, it will not gain much by adding additional masters and hash calculations to HDFS/GoogleFS. On the contrary, the system speed will drop severely due to the redundant processing.

3.2. Efficient Distributed File System

The Efficient Distributed File System (EDFS) is a semi-centralized DFS [7]. Similar to RFS, it focuses on the potential failure point from the single master node or name node. Also, they blame TCP used by HDFS/GoogleFS is a slow protocol, so they replace it with their own protocol, which doesn't have a name yet.

EDFS also has many name nodes, and a frontend server to manage sessions and forward requests to name nodes via hashing. On the other hand, they create a protocol contains user client, light weight front end server, name node servers, resource allocator, block servers and resource monitors.

In [7], they showed EDFS transferred data faster than GoogleFS. However, they didn't tell where the reason of this improvement was. Was it from the faster protocol or from the workload distribution across many name nodes? Another interesting point is they claimed the single master node in GoogleFS to be a failure point so they adapted many name nodes schema. But they never mention the single frontend server they used is a single failure point, too.

3.3. GreenHDFS

The energy-conservation of the extremely large-scale, commodity data centers has become a priority problem, especially when the whole world is trying to go green. In 2010, a variant of HDFS called GreenHDFS is proposed in [8], which focuses on energy consuming issue in DFS.

In GreenHDFS, the data node is categorized into two zones, the cold zone and the hot zone. Hot zone consists of files that are being accessed currently or newly created. Performance is the greatest importance here so the energy savings are traded-off for high performance. The cold zone consists of files

with low accesses. Files in cold zone are moved from hot zone by File Migration policy. For optimal energy savings, the servers in cold zone are in a sleeping mode by default.

Each file in GreenHDFS is associated with temperature. A file is in hot zone when it's created, but its temperature decreases if it's not accessed frequently. When its temperature is lower than a threshold, it's moved to the cold zone. Similarly, a file in cold zone is moved to hot zone if it's accessed frequently.

GreenHDFS has a straight-forward goal and a simple design to achieve it. In [9] they declare the GreenHDFS is capable of achieving 24% savings in energy costs. However, moving files between servers and putting servers into sleep mode will definitely do harm to the overall performance. It's would be great if a balance point between performance and energy-saving can be found for GreenHDFS.

3.4 QuantcastFS

QuantcastFS (QFS) is an open-source DFS from Quantcast. It was designed as an alternative to Apache Hadoop's HDFS, intended to deliver better performance and cost-efficiency for large-scale processing clusters. Similar to GoogleFS and HDFS, QFS includes one metaserver and a certain number of chunkservers, usually one chunkserver per node. Besides, QFS also includes a client component is the interface point that presents a file system API to other layers of the software. It makes requests of the metaserver to identify which chunk servers hold (or will hold) its data, then interacts with the chunk servers directly to read and write.

QFS uses Reed-Solomon error correction to meets the needs of fault tolerance. Fault tolerance can be done with only 50% data expansion, a considerable space reduction comparing to the 3 relication in HDFS. In their 20 TB data IO test, QFS is claimed to be 75% faster in writing and 46% faster in reading. However, Quantcast doesn't claim this result is representative for QFS or HDFS performance in general.

4. General Parallel File System

The General Parallel File System (GPFS) is IBM's parallel, shared disk file system for cluster computers and supercomputers [10]. It's a centralized DFS and its shred-disk architecture enables GPFS to achieve extreme scalability [11]. It supports up to 4096 disks, with the maximum size 1TB each, with a total scale of 4 PB. Big files are divided into blocks, by default 256 KB (configurable from 16 KB to 1MB). Small files are stored into sub-blocks which are 1/32size of an ordinary block. GPFS also supports large directory that can contain millions of files. Extensible hashing is used for fast locating files within a directory. GPFS also applies a prefetching buffer for multi-reading and write-behind method for multi-writing. Besides, Distributed Locking is adapted to synchronize accesses to shared disks.

GPFS dynamically elects metanodes for centralized management of file metadata as well as each node has its own log stored in GPFS. GPFS also chooses one node as the allocation manager, which maintains free space statistics about all allocation regions. Loosely up-to-date via periodic message in which each node reports the net amount of disk space allocated or freed since last message.

For the fault tolerance, GPFS uses a different mechanism from GoogleFS or HDFS. If a node fails, GPFS will try to restore the metadata being updated by the failed node and release resources held by

it (tokens). GPFS will also appoint replacements for any special roles played by the failed node (metanodes or allocation manager). If the metanode fails, GPFS will create a newmetanode. However, this new metanode will not issue any new tokens until the log is recovered.

Comparing to GoogleFS and HDFS, the structure of GPFS is more complicated. GPFS also fully supports POSIX, which makes it capable for more subtle operations. However, the fault tolerance in GPFS is not as strong since the data itself is only duplicated by RAID on the same node, not triple replicated as in HDFS and GoogleFS, although the metadata can be recovered easily.

5. Global File System

The Global File System (GFS) and GFS2 is also a shared disk file system for Linux computer cluster. GFS and GFS2 differ from a traditional DFS because they allow all nodes to have direct concurrent access to the same shared block storage. GFS and GFS2 can also be used as local file system [12]. Comparing to GoogleFS/HDFS, GFS is fully POSIX-compliant, meaning applications don't have to be rewritten to use GFS. It is also the first native 64-bit cluster FS on Linux for huge workloads, while GoogleFS and HDFS both base on a native FS.

In GFS, all nodes function as peers. No server or client roles. GFS enables several servers connected to a storage area network (SAN) to access a common, shared file store at the same time with standard UNIX/POSIX file system semantics. It is a journaling FS. Each cluster node is allocated its own journal. Changes to the file system metadata are written in a journal and then on the file system like other journaling file systems. In case of a node failure, file system consistency can be recovered by replaying the metadata operations. Optionally, both data and metadata can be recorded.

GFS is often confused as the GoogleFS, but they don't share a similar structure at all. GFS is more like a local file system extended to a cluster level, while the GoogleFS is designed for cluster only. The complicated structure of GFS provides all POSIX functions. On the contrary, the centralized structure of GoogleFS turns everything into a simpler state with fewer functions.

6. Sector

The Sector is a DFS that can be deployed over a wide area and allows users to ingest and download large datasets from any location. It's designed as the basis FS for Sphere Compute Cloud [13]. Sector consists of a master, a security server, and a number of slaves. Files are divided into Sector slices and each slice is a saved in a slave's native FS as a file. Hence Sector relies on native FS and can be interoperate with it if necessary.

The attracting part in Sector is the security server connected to the master, which doesn't appear in any other FS in this paper. It is used to maintain user accounts, file access information, list of IP addresses of slave nodes and so on. Given a client's request for accessing a file, the master will check with the security server to see whether this client is legal and if it has permission to get the data using SSL. If so, the master will request the needed slave to open a connection with the client starting to transfer data. Slaves only listen to the master, so the data security is guaranteed.

The data in Sector is triple replicated, similar to GoogleFS, therefore any failed slave can be recovered easily. Given a master failure, the metadata can be re-constructed without difficulty by simply

scanning through all slaves' native FS. Discarding the TCP for data transfer, Sector employs UDP for faster speed. For message passing, User Defined Type [14] is adapted. Sector also includes a reliable library called group messaging protocol. By employing these methods, Sector is claimed to be faster than GFS and very reliable.

Table 1 A Summary of Some Differences between File Systems

Design Decision	GoogleFS	HDFS	Sector	GPFS	GFS
Datasets Divided Into Files Or Blocks	Blocks	Blocks	Files	Blocks	Both
Protocol For Message Passing	TCP	TCP	Group Messaging Protocol	Not Mentioned	TCP
Protocol For Data Transferring	TCP	TCP	UDP	Not Mentioned	MPI For NAS
Replication Strategy	Replicas Created At The Time Of Writing	Replicas Created At The Time Of Writing	Replicas Created Periodically By System	RAID-Replicated	No
Security Model	Not Mentioned	None	User-Level And File-Level Access Controls	Not Mentioned	None

7. Conclusion

In this report, I reviewed the GoogleFS and its open-source version HDFS, and introduced a number of alternatives to GoogleFS, like GFS, GPFS, and Sector. Moreover, some variants of GoogleFS/HDFS, EDFS, RFS, GreenHDFS, and QFS are also reviewed based on their features. A summary of some differences between these file systems is shown in Table 1.

As the computation ability increases significantly, the bottleneck for processing large scientific data starts to appear in data management and transfer. Hence a robust file system with outstanding performance, scalability, reliability and availability is broadly demanded. Another aspect worth noting is that besides performance and scale, the energy efficiency starts drawing more attention. It should be one hot topic in future.

8. References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, In Proc. of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec. 2004.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. SIGOPS Oper. Syst. Rev., 37(5):29–43, 2003.
[4] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. The ApacheSoftware Foundation, 2007.
[5] A. Verma, S. Venkataraman, M. Caesar, R. Campbell. Efficient Metadata Management for Cloud Computing applications – University of Illinois, Urbana-Champaign, Technical Report, January 2010
[6] M. K. McKusick and S. Quinlan. GFS: Evolution on Fastforward. Queue, vol. 7, no. 7, pp. 10–20, 2009.
[7] D. Fesehaye, R. Malik, K. Nahrstedt. EDFS: A Semi-Centralized Efficient Distributed File System. (Extended Abstract), Middleware 2009, Yr. 2009.
[8] R. T. kaushik and M. Bhandarkar. Greenhdfs: Towards An Energy-Conserving, StorageEfficient, Hybrid Hadoop Compute Cluster. HotPower, 2010.
[9] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System. cloudcom, pp.274-287, 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 2010
[10] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System For Large Computing Clusters. In Proceedings of the 2002 Conference on File and Storage Technologies (FAST), pages 231–244. USENIX, Jan. 2002.
[11] C. Mohan, Inderpal Narang. Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment. VLDB 1991: 193-207.
[12] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The Global File System. In Proceedings of the 5th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 319–342, College Park, MD, 1996.
[13] Y. Gu, R. Grossman. Sector and Sphere: The Design and Implementation of a High Performance Data Cloud. Theme Issue of the Philosophical Transactions of the Royal Society A: Crossing Boundaries: Computational Science, E-Science and Global E-Infrastructure, 28 June 2009, vol. 367, no. 1897, page 2429-2445.
[14] Yunhong Gu, Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks. Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.