Distributed Metadata Management Scheme in HDFS

Mrudula Varade[*], Vimla Jethani[**]

[*] Department of Computer Engineering, R.A.I.T.
[**] Department of Computer Engineering, R.A.I.T.

Abstract- A Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably and to stream those data sets at high bandwidth to user applications. Metadata management is critical to distributed file system. In HDFS architecture, a single master server manages all metadata, while a number of data servers store file data. This architecture can't meet the exponentially increased storage demand in cloud computing, as the single master server may become a performance bottleneck. Comparative study of a metadata management scheme is done. There is three of techniques sub-tree partitioning, hashing and consistent hashing of metadata management scheme. Out of these three schemes consistent hashing is the best techniques which employs multiple NameNodes, and divides the metadata into "buckets" which can be dynamically migrated among NameNodes according to system workloads. To maintain reliability, metadata is replicated in different NameNodes with log replication technology, and Paxos algorithm is adopted to keep replication consistency.

Index Terms- Hadoop, HDFS, Distributed File System, Metadata Management, File Management System.

I. INTRODUCTION

With the rapid development of Internet, the amount of data is growing exponentially, and the large-scale data storage and processing has become a problem. Cloud computing is one of the most popular solutions to meet the demand. Cloud computing provides decreased cost of hardware resource and increased equipment utilization. Users can access all kinds of Internet services and virtual computing power through lightweight portable devices rather than traditional Pc. Cloud storage is a key issue for cloud computing, and metadata management plays a critical role in Cloud storage. Metadata is the data that describes the structure of a file system, such as hierarchical namespace, file and directory attributes and mapping of file to data chunks. Although the size of metadata is relatively small in the file system, metadata operations can take up over 50 percent of the whole file system operations. So the metadata management is critically important for file system performance.

There are various distributed file systems like PVFS (Parallel Virtual File System), Lustre, GFS (Google File System), HDFS (Hadoop Distributed File System).The Google File System (GFS) [1] and Hadoop Distributed File System (HDFS) [2] are the most common file system deployed in large scale distributed systems such as Face book, Google and Yahoo today.

PVFS [3] is an open source RAID-0 style parallel file system for clusters. It partitions a file into stripe units and distributes these stripes to disks in a round robin fashion. PVFS consists of one metadata server and several data servers. All data traffic of file content flows between clients and data server nodes in parallel without going through the metadata server. The fatal disadvantage of PVFS is that it does not provide any fault-tolerance in its current form. The failure of any single server node will affect the whole file system.

Lustre is a shared disk file system. Commonly used for large scale cluster computing. It is an open-standard based system with great modularity and compatibility with interconnects, networking components and storage hardware. Currently, it is only available for Linux.

Google file system (GFS) [1] is a scalable distributed file system that supports the heavy workload at the Google website and runs on a cluster with inexpensive commodity hardware. In GFS, a single master node is used to maintain the metadata and the traffic of high volume of actual file contents are diverted to bypass the master to achieve high performance and scalability. GFS takes an aggressive approach to provide fault tolerance, in which three copies of data are stored by default. GFS is tailored to meet the particular demands for Google's data processing and is not a general-purpose file system.

Hadoop is a Distributed parallel fault tolerant file system. It is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. Hadoop DFS stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files are "write once" and have strictly one writer at any time. Hadoop is a top-level Apache project being built and used by a global community of contributors, written in the Java programming language.

In a distributed system even if we decide to deploy dedicated high performance machines which are really costly, faults or disruptions are not frequent. So forerunners like Google decided to use commodity hardware which is ubiquitous and very cost effective , but to use such hardware they have to make a design choice of treating faults/ disruptions as regular situation and system should able to recover from such failures. Hadoop developed on similar design choices to handle faults. So comparing lustre, pvfs which system assumes faults are infrequent and needs manual intervention to ensure continued services on other hand Hadoop turns out to be very robust and fault tolerant option. Hadoop ensures that few failures in the system won't disrupt continued service of data through automatic replication and transfer of responsibilities from failed machines to live machines in Hadoop farm transparently. Though it's mentioned that GFS has

same capabilities since it's not available to other companies those capabilities cannot be availed.

A. *Introduction to Hadoop*

Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the Map Reduce paradigm. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data.

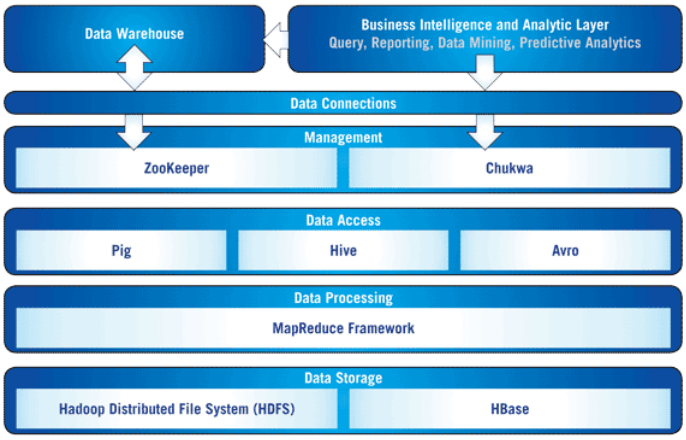

Figure 1: Hadoop System

The table 1 shows the components of hadoop. Hadoop is an Apache project; all components are available via the Apache open source license. Yahoo! has developed and contributed to 80% of the core of Hadoop (HDFS and MapReduce). HBase was originally developed at Powerset, now a department at Microsoft.

Table 1.Hadoop project components

HDFS	Distributed file system Subject of this paper!
MapReduce	Distributed computation framework
HBase	Column-oriented table service
Pig	Dataflow language and parallel execution framework
Hive	Data warehouse infrastructure
ZooKeeper	Distributed coordination service
Chukwa	System for collecting management data
Avro	Data serialization system

Hive was originated and developed at Facebook. Pig, Zookeeper, and Chukwa were originated and developed at Yahoo! Avro was originated at Yahoo! and is being co-developed with Cloudera. HDFS is the file system component of Hadoop. While the interface to HDFS is patterned after the UNIX file system, faithfulness to standards was sacrificed in favour of improved performance for the applications at hand.

B. *Hadoop Distributed File System*

The Hadoop Distributed File System (HDFS) [7]is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

HDFS stores file system metadata and application data separately. HDFS architecture consists of NameNode, DataNode, and HDFS Client. A HDFS Cluster may consist of thousands of DataNode and tens of thousands of HDFS clients per cluster, as each DataNode may execute multiple application tasks concurrently. The main features of HDFS are that, it is highly fault tolerance, suitable for applications with large data sets. The below figure 2 shows the Hadoop Distributed File System Architecture:

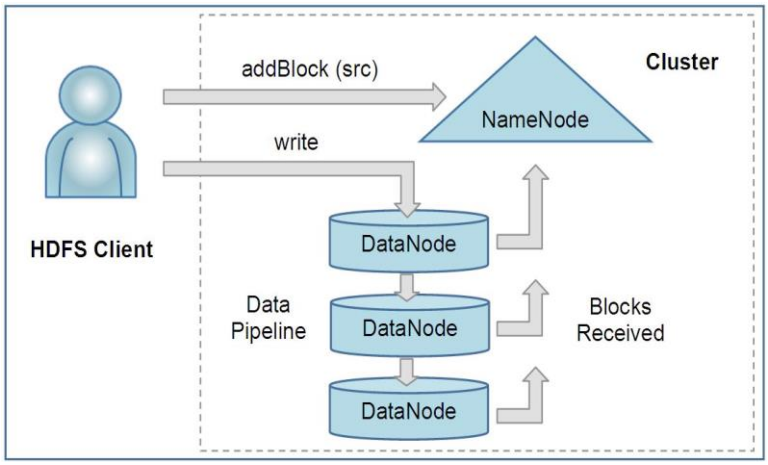

Figure 2: HDFS architecture

HDFS [2] is master/slave architecture and consist of single NameNode, a master server that manages the file system namespace and regulates access to files by clients. HDFS namespace is a hierarchy of files and directories. These files and directories which records attribute like permissions, modification, access time namespace and disk space quotas.

A file is split into one or more blocks and set of blocks are stored in DataNodes. [2] A DataNodes stores data in the files in its local system and it does have any knowledge about HDFS file system. It stores each block of HDFS data in a separate file.

HDFS cluster has a single name node that manages the file system namespace. The current limitation that a cluster can contain only a single name node results in the following issues:

1. Scalability: Name node maintains the entire file system metadata in memory. The size of the metadata is limited by the physical memory available on the node. This results in the following issues:

a. Scaling storage – while storage can be scaled by adding more data nodes/disks to the data nodes, since more storage results in more metadata, the total storage file system can handle is limited by the metadata size.

b. Scaling the namespace – the number of files and directories that can be created is limited by the memory on name node.

To address these issues one encourages larger block sizes, creating a smaller number of larger files and using tools like the hadoop archive (har).

2. Isolation: No isolation for a multi-tenant environment. An experimental client application that puts high load on the central name node can impact a production application.

3. Availability: While the design does not prevent building a failover mechanism, when a failure occurs the entire namespace and hence the entire cluster is down.

A single NameNode manages file system namespace, determines the mapping of file to blocks, and regulates access to files. In HDFS, all metadata is kept in the memory of the single NameNode, so it may become performance bottleneck as metadata number increases.

II. METADATA MANAGEMENT SCHEMES

To distribute metadata among multiple servers some techniques are used like Sub-Tree Partitioning, Hashing technique and Consistent Hashing.

A. Sub-Tree Partitioning

The **Sub Tree Partitioning** [4][6] is used in Ceph file system and Coda file system. The key design idea is that initially, the partition is performed by hashing directories near the root of the hierarchy, and when a server becomes heavily loaded, this busy server automatically migrates some subdirectories to other servers with fewer loads. It also proposes prefix caching to efficiently utilize available RAM on all servers to further improve the performance. This approach has three major disadvantages. First, it assumes that there is an accurate load measurement scheme available on each server and all servers periodically exchange the load information. Second, when an MS joins or leaves due to failure or recovery, all directories need to be rehashed to reflect the change in the server infrastructure, which, in fact, generates a prohibitively high overhead in a petabyte-scale storage system. Third, when the hot spots of metadata operations shift as the system evolves; frequent metadata migration in order to remove these hot spots may impose a large overhead and offset the benefits of load balancing.

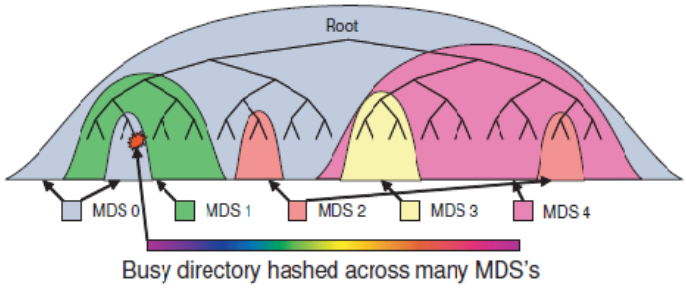

Figure 3: Sub-Tree Partitioning

In sub tree partitioning, namespace is divided into many directory sub trees, each of which is managed by individual metadata servers. This strategy provides a good locality because metadata in the same sub tree is assigned to the same metadata server, but metadata may not be evenly distributed, and the computing and transferring of metadata may generate a high time and network overhead.

B. Hashing Technique

Hashing technique [10] is used in Lustre, zFs file system. Hashing technique uses a hash function on the path name to get metadata location. In this scheme, metadata can be distributed uniformly among cluster, but the directory locality feature is lost, and if the path is renamed, some metadata have to migrate. However, a serious problem arises when an upper directory is renamed or the total number of MSs Changes the hashing mapping needs to be re-implemented, and this requires all affected metadata to be migrated among MSs. Although the size of the metadata of a file is small, a large number of files may be involved. In particular, the metadata of all files has to be relocated if an MS joins or leaves. This could lead to both disk and network traffic surges and cause serious performance degradation. The hashing-based mapping approach can balance metadata workloads and inherently has fast metadata lookup operations, but it has slow directory operations such as listing the directory contents and renaming directories; in addition, when the total number of MSs Changes, rehashing all existing files generates a prohibitive migration overhead.

As the above mention two techniques has the main drawback that metadata are not evenly distributed in the system, to overcome these consistent hashing technique is used.

C. Consistent Hashing

Consistent hashing [9] is proposed hash method used in Amazon Dynamo. In basic consistent hashing, the output range of the hash function is treated as a ring. Not only the data is hashed, but also each node is hashed to a value in the ring. Each node is responsible for the data in the range between it and its predecessor node. In consistent hashing, the addition and removal of a node only affects its neighbor nodes. An optimization of consistent hashing is the introduction of "virtual node". Instead of mapping a physical node to a single point in the ring, each physical node is assigned to multiple positions, each of which is called a virtual node. With virtual node, data and workload is distributed over nodes more uniformly.

A single NameNode manages file system namespace, determines the mapping of file to blocks, and regulates access to files. In HDFS, all metadata is kept in the memory of the single NameNode, so it may become performance bottleneck as metadata number increases. So in HDFS, we changed the single NameNode architecture to multiple NameNodes, and the author has proposed a metadata management scheme.

1) System Design Architecture

The Figure 4 shows the architecture of metadata management system [9]. The system mainly consists of four components: Client, multiple NameNodes, NameNode Manager and Database. Client exposes interfaces to access metadata. To improve system performance, some recently used metadata will be cached in Client. NameNode is responsible for managing metadata and dealing with metadata request from Client.

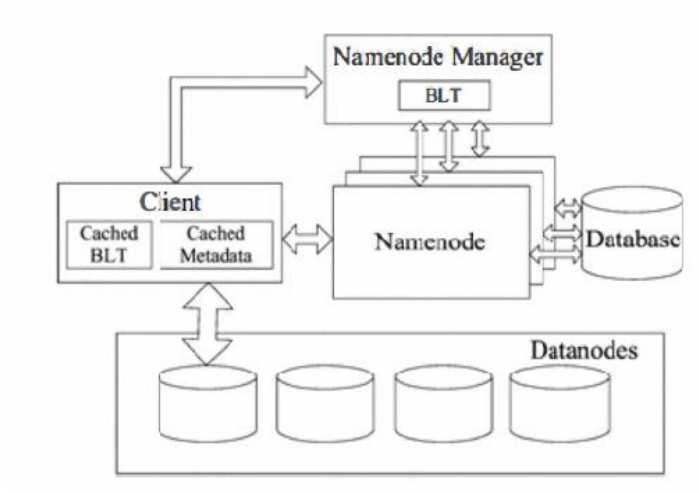

Figure 4: Metadata Management System

The metadata in this paper consists of directory metadata and file metadata. Directory metadata includes a hierarchical namespaces and directory attributes, and file metadata includes file attributes and the mapping from file to data blocks. Generally, a metadata is a tuple as below:

GLOBAL_ID	USER_ID	PARENT_GLOBAL_ID	NAME	BLOCK_PTR	OTHER_META

Figure 5: Metadata format

GLOBAL_ID is the global unique identifier that is invariable once the path is created. USER_ID is the identifier of user that created the path. PARENT_GLOBAL_ID is the GLOBAL_ID of the parent directory of the path. OTHER_META saves other information, such as permission, last access time and update time. BLOCK_PTR is the pointer to the file data blocks. The updated metadata in NameNode is persisted into database periodically. NameNode Manager provides a route for Client to get the target NameNode. Besides, it manages the metadata distribution and load balancing among NameNodes by periodically receiving heartbeat message from NameNodes [9].

2) Metadata Partitioning

Consistent hashing ring is divided in Q equally sized parts and each is called"bucket".Metadata is partition using buckets and evenly distributed across NameNodes. Mapping of path's from metadata to bucket is like consistent hashing First hash USER_ID and PARENT_GLOBAL_ID of the path to yield its position p. Walk clockwise to find 1st bucket with position larger than p.

3) Metadata Access

To organize namespace hierarchy they have adopt hash table. The figure 5 shows the NameNode data structure. For example we want to access to path /A/B/C/filename

- Client gets user_id and global_id of path as Parent_Global_ID
- Computes the consistent hashing result
- Then client see the cached BLT to find out bucket_id and NameNode I
- Sends the request to NameNode i in form of <bucket_id, user_id, parent_global_id, filename> then that NameNode I see its bucket array by bucket_id.

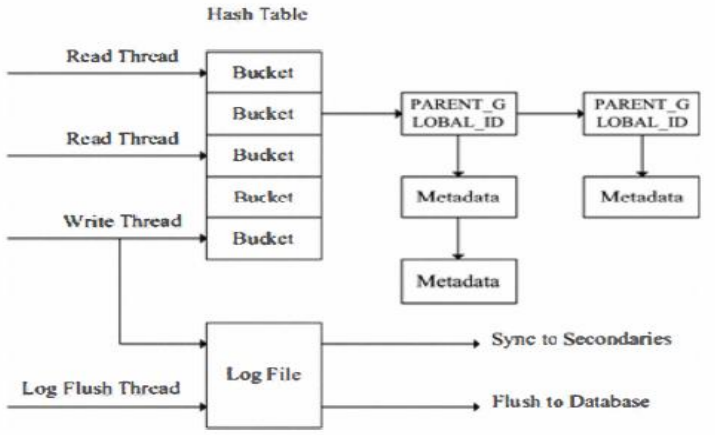

Figure 5: NameNode Data Structure

In the paper [9] they have also proposed the solution to metadata safety in memory. The solution is "Log Replication". As the metadata is periodically persisted into database, they have got the newest metadata by applying the latest log records on the last-persisted metadata in database. So they have just replicated the latest log records rather than in-memory metadata [9]. To avoid log records consensus problems between primary and secondary's, Paxos algorithm is used [9].

Failure detection is implemented by heartbeat mechanism. Failure handling includes metadata recovery, bucket re-contribution. In recovery all metadata managed by the failed primary is recovered. After the metadata in failed primary has been recovered, NameNode manager will contribute the buckets to other NameNode by modifying BLT.

A NameNode can be added to or removed from the cluster without system restarting, and the metadata can be redistributed to keep load balance.

III. COMPARISON OF METADATA MANAGEMENT TECHNIQUES

Since scalability is the main issue in HDFS and single NameNode has to operate all metadata operations; the entire load is on one single NameNode that can affect the efficiency, reliability and scalability of the system. Multiple NameNode are introduced in the architecture to distribute the load, but metadata distribution evenly among multiple NameNode is important.

Consistent hashing technique when compared with other two techniques we got the findings as shown in below table:

Table 2: Comparison of Techniques

	Consistent Hashing	Hashing	Sub-Tree partitioning
Performance	High	Low	Medium
Load Balancing	Yes	Yes	No
Reliability	High	Low	Low
Scalability	Highly Scalable	Moderately Scalable	Moderately Scalable

In Consistent Hashing technique the performance is high as compare to other two techniques; since the distribution of metadata and routing of metadata request is effectively done using consistent hashing which leads to improvement in system performance. In this technique the data bucket is divided into equal size which is further evenly distributed over the NameNodes which leads to efficient load balancing because of even distribution. Log method is used for storing and prefetching of metadata. As the logs are stored into bucket look up table which is stored at client cache and similarly replicated to NameNodes memory as well. Due to log replication fault-tolerance is increased which causes high reliability. Consistent hashing consist the insertion of metadata or removal of the same without disturbing the cluster and it also redistribute the load which leads to proper load balancing which increases scalability.

In Hashing technique the performance is low since hashing destroys the locality of metadata which causes the opportunity to prefetching and storing the metadata in bucket. The hashing-based mapping approach can balance metadata workloads and inherently has fast metadata lookup operations, but it has slow directory operations such as listing the directory contents and renaming directories. In addition, when the total number of MSs Changes, rehashing all existing files generates a prohibitive migration overhead. In hashing the hash function uses the search-key of the bucket. This search key is unique. Due to the uniqueness of search key in hashing dependency is generated which leads to low reliability.

In Sub-Tree partitioning the performance is medium compare to hashing technique. As sub-tree partitioning uses N-nary data structure in which the dependency is formed over root node and on parent node. Thus reliability decreases. We can say that compare to two techniques consistent hashing is better technique.

IV. CONCLUSION

We have seen the components of hadoop and the hadoop distributed file system in brief. As compare to other file system HDFS is a highly fault tolerance system. The main drawback of HDFS was its single NameNode which handles all metadata operations. In this the drawback is overcome by introducing multiple NameNodes in the system. The other problem arises is to handle metadata operations among multiple metadata servers. In this paper we have compared three techniques Sub-Tree partitioning, hashing technique and consistent hashing. To distribute the metadata evenly among the metadata server they have adopted consistent hashing technique. As compare to other techniques consistent hashing evenly distributes load to the server and consumes less memory. BLT provides an efficient metadata retrieval mechanism for Client to find the target NameNode. To guarantee metadata availability under cluster failure, log replication technology with Paxos algorithm is adopted. In addition, system performance benefits from metadata caching and prefetching.

REFERENCES

[1] S. Ghemawat, H. Gobioff, S. Leung. “The Google file system,” In Proc. of ACM Symposium on Operating Systems Principles, Lake George, NY, Oct 2003, pp 29–43.

[2] Konstantin Shvachko, et al., “The Hadoop Distributed File System,” Mass Storage Systems and Technologies (MSST), IEEE 26th Symposium on IEEE, 2010, http://storageconference.org/2010/Papers/MSST/Shvachko.pdf.

[3] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. “PVFS: A parallel file system for Linux clusters,” in Proc. of 4th Annual Linux Showcase and Conference, 2000, pp. 317–327.

[4] M. Satyanarayanan, J. 1. Kistler, P. Kumar, et ai, "Coda: A highly available file system for a distributed workstation environment", IEEE Transactions on Computers, 1990, 39(4):447-459.

[5] Y.Zhu, H.Jiang, J.Wang, and F.Xian, "HBA: Distributed Metadata Management for Large Cluster-Based Storage Systems", IEEE Trans. Parallel and Distributed Systems, June 2008, vol.19, no.6, pp.750- 763.

[6] S. A. Weil, S. A. Brandt, E. L. Mille, et ai, "Ceph: A Scalable, High-Performance Distributed File System", In Proceedings of the 7th symposium on Operating systems design and implementation, 2006, pp. 307-320.

[7] http://hadoop.apache.org/docs/r0.20.0/hdfs_design.html

[8] Karger, D., Lehman, E., Leighton, T., Panigahy, R., Levine, M. , and Lewin, D, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web", In Proceedings of the Twenty-Ninth Annual ACM Symposium on theory of Computing, ACM Press, New York, 1997, pp. 654-663.

[9] Bing Li, Yutao He, Ke Xu, “Distributed Metadata Management Scheme in Cloud Computing “, In Proceedings of IEEE in PCN&CAD CENTER, Beijing University of Post and Telecommunication, China, 2011.

[10] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, Ethan L.Miller ,”Dynamic Metadata Management for Petabyte-scale File Systems “, In Proceedings of IEEE University of California, 2004.

AUTHORS

First Author – Mrudula Varade, M.E.(Computer Pursuing), Department of Computer Engineering, R. A. I. T.
Email: mvarade8@gmail.com
Second Author –Vimla Jethani, M.E.(Computer), Department of Computer Engineering, R. A. I. T.
Email: vimlajethani@gmail.com