ISSN:0975-9646

Survey on Metadata Management Schemes in HDFS

Abhinay Gupta , Raviraj Gundety, Vivek Fernando, Neeraj Iyer, Beatrice.S

Department of Computer Engineering
Xavier Institute of Engineering ,Mumbai, India

Abstract **- Hadoop provides a reliable shared storage and analysis system.The storage is provided by Hadoop Distributed File System(HDFS). Hadoop is a popular open source implementation of mapreduce, a powerful tool designed for deep analysis and transformation of very large data sets. Managing metadata is very important in a distributed file system like HDFS. In this paper comparative study of three techniques of metadata management namely sub-tree partitioning, hashing and consistent hashing is presented. After detailed analysis we have come to the conclusion that consistent hashing is the best of the three techniques as it makes use of multiple namenodes instead of just one namenode currently available in HDFS, thus improving its performance.**

***Keywords*— HDFS, distributed file system, metadata management system.**

I. INTRODUCTION

Hadoop was created by Doug Cutting, the creator of Apache Lucene. Hadoop has its origins in Apache Nutch, an open source web search engine which itself a part of the Lucene project. The name “Hadoop” is given by the project creator’s kid’s stuff toy yellow elephant [1]. Building a web search engine from scratch was an ambitious goal. So Mike Cafarella and Doug Cutting runs a dedicate operation team with the support of software required for crawling and indexing websites goal, and introduces the search engine called Nutch. Nutch started in 2002, and working crawler search system quickly emerged. In early in 2005, the Nutch developers had a working MapReduce implementation in Nutch with the help of Google’s paper, published on Map reduce function in the year 2004 and further in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop [1]-[2].

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

Hadoop is a collection of related subprojects that fall under the umbrella of infrastructure for distributed computing. These projects are hosted by the Apache Software Foundation, which provides support for a community of open source software projects. Although Hadoop is best known for MapReduce and its distributed filesystem (HDFS, renamed from NDFS), the other subprojects provide complementary services, or build on the core to add higher-level abstractions. The subprojects, where they sit in the briefly here:

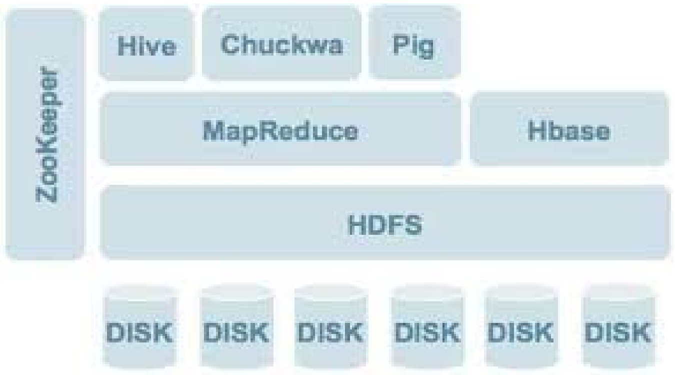

Fig I. Hadoop System

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

Table I. Hadoop project components description

The fig.1 and table.1 shows the components of hadoop and description of the same. 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

Hadoop enables applications to work with thousands of nodes and petabytes of data. Hadoop was inspired by Google's MapReduce and Google File System (GFS). Hadoop is a top-level Apache project being built and used by a global community of contributors, written in the Java programming language. Yahoo! has been the largest contributor to the project, and uses Hadoop extensively across its businesses [1]-[2].

II. HDFS

1 Introduction

The Hadoop Distributed File System (HDFS)[1][2] 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 relaxes a few Portable Operating System Interface (POSIX) requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

2 Key Features of HDFS

- *Scale-Out Architecture* - Add servers to increase capacity.
- High Availability - Serve mission-critical workflows and applications.
- Fault Tolerance - Automatically and seamlessly recover from failures.
- Flexible Access - Multiple and open frameworks for serialization and file system mounts.
- Load Balancing - Place data intelligently for maximum efficiency and utilization.
- Tunable Replication - Multiple copies of each file provide data protection and computational performance.
- Security - POSIX-based file permissions for users and groups with optional LDAP integration.

3 Assumptions and Goals

3.1. Hardware Failure

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

3.2. Streaming Data Access

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

3.3. Large Data Sets

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A Map/Reduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

3.4. Computation

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

3.5. Portability across Heterogeneous Hardware and Software Platforms

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

4 Architecture

The Hadoop Distributed File System (HDFS)[2] 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:

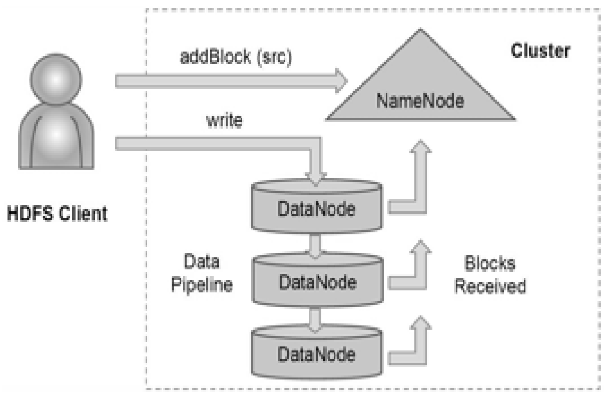

Fig.II. HDFS architecture

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.

5 *Limitations*

HDFS[7] 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.

III METADATA MANAGEMENT SCHEMES

In Distributed Metadata Management the meta data is distributed among the various data nodes or servers.There are several techniques to manage distributed meta data such as Sub-Tree partitioning, Hashing and consistent hashing .

1 *Sub Tree Partitioning*

The Sub Tree Partitioning [3][4] 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. Some Common

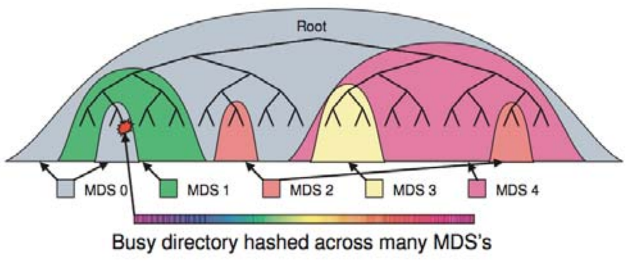

Fig III. Subtree partitioning

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.

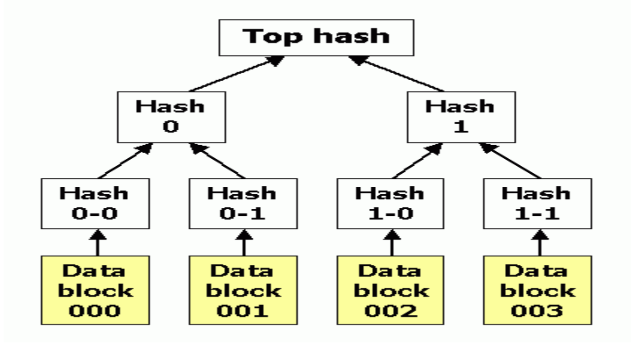

Fig IV. Hashing Diagram

2 *Hashing Technique*

Hashing technique [6] 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.

3 *Consistent Hashing*

Consistent hashing [5] 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.

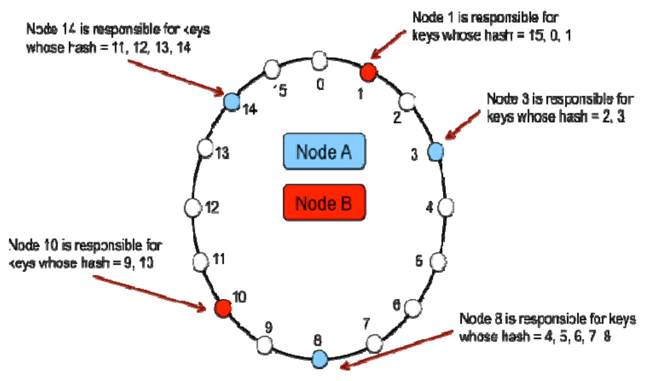
Node 14 is responsible for keys whose hash = 11, 12, 13, 14
Node 1 is responsible for keys whose hash = 15, 0, 1
Node 3 is responsible for keys whose hash = 2, 3
Node A
Node B
Node 10 is responsible for keys whose hash = 9, 10
Node 8 is responsible for keys whose hash = 4, 5, 6, 7 8
15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Fig V. Consistent Hashing

IV COMPARISON OF TECHNIQUES

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

Table II. Comparison of meta data management schemes

From the table II, it can be noticed that consistent Hashing technique has higher performance as compared to other two techniques; as the distribution of metadata and routing of metadata request is effectively done. 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. Storing and prefetching of metadata is done by log method. The logs are stored into bucket look up table which is NameNodes'memory to log replication fault-tolerance is increased which in turn stored at client cache and which leads to proper load balancing which increases scalability.

Consistent hashing consists of the insertion of metadata or removal of the same without disturbing the cluster and it also redistributes the load which leads to proper load balancing which in turn increases scalability.

Hashing technique gives low performance 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 change, 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 comparing above techniques, consistent hashing is better technique.

V. CONCLUSION

It has been discussed the hadoop system and its architecture in brief. We have explained hadoop distributed file system including its features, goals and limitations in detail. HDFS is highly fault tolerant. The drawback of HDFS having only a single namenode is overcome by making use of multiple namenodes in the system. Distributing metadata evenly among multiple metadata servers is another issue for which we have done comparative study of three schemes used for metadata management namely sub-tree partitioning, hashing and consistent hashing techniques. Consistent hashing is better than other two techniques as it distributes the load relatively evenly to the server and consumes comparatively less memory

REFERENCES

[1] http://en.wikipedia.org/wiki/Apache_Hadoop
[2] Mrudula Varade*, Vimla Jethani** "Distributed Metadata Management Schemes in HDFS", IEEE AMY 2013, www.ijsrp.org/research-paper-0513/ijsrp-p1770.pd
[3] 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.
[4] 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
[5] 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.
[6] 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.
[7] http://hortonworks.com/blog/thinking-about-the-hdfs-vs-other-storage-stechnologies/.