International Journal of Computer Applications (0975 – 8887)
Volume 86 – No 2, January 2014

Improving Data Transfer Rate and Throughput of HDFS using Efficient Replica Placement

Neha M Patel
PG Student, BVMCE
V.V Nagar-India- 388120

Narendra M Patel
Associate Prof. BVMCE
Dept. of Computer Engg.

Mosin I Hasan
Asst. Prof. BVMCE
Dept. of Computer Engg.

Mayur M Patel
Asst. Prof. CSPIT
Dept. of Computer Engg.

ABSTRACT

In last half decade, there is tremendous growth in the network applications; we are experiencing an information explosion era. Due to which large amount of distributed data being managed and stored. To handle these types of data, application uses distributed file system. Advantages of DFS are increased availability and efficiency. Generally some parameters like scalability, reliability, transparency, fault tolerance and security are considered while making DFS. Some open challenges are still there in DFS like fault tolerance in various conditions, optimized architecture of DFS, Synchronization, consistency and replications. Goal of this paper is study evolution of DFS from the history; current state of the art design & implementation of the DFS and propose new approach for efficient replica placement in Hadoop DFS which can improve throughput and data transfer rate.

General Terms

Distributed File System

Keywords

NFS, AFS, GFS, XtreemFS, HDFS.

1. INTRODUCTION

A distributed file system is a client/server-based application that allows clients to access and process data stored on the server. In other words, we can say Distributed file system consists of software residing on network servers and clients that transparently links shared folders located on different file servers into a single namespace for improved load sharing and data availability. When a client device retrieves a file from the server, a file appears as a normal file on the client machine, and the user is able to work with the file in the same ways as if it were stored locally on the workstation. When the user finishes working with the file, it is returned to the server, which stores the now-altered file for retrieval at a later time. The flow of this research papers is: section two gives overview of different Distributed File System and related work, section three elaborates designing issues of distributed file system and comparison between different file system on the basis of various factors, and section 4 describes Hadoop DFS read write operation., and at the end we propose a proof of concept for writing replicas on Hadoop DFS using parallel approach. Finally, conclusion of this paper is outlined.

2. RELATED WORKS

Russell Sandberg‘s team at Sun Microsystems laboratory has developed Network File System, it is most widely used distributed file systems in the UNIX world. Upon releasing the first versions of NFS in 1985, SUN made public the NFS protocol specification [10] which allowed the implementation of NFS servers and clients by other vendors. After that AFS [15][16] developed for distributed workstation environment that has been under development at Carnegie Mellon University since 1983. At Google Inc’s Research Laboratory, Google File System is developed by Sanjay Ghemawat, Howard Gobi off and Shun- Tak Leung. GFS [13] shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and are accessed by a comparable number of client machines. GFS [13] cluster consists of a single master and multiple chunk servers and is accessed by multiple clients.

As object oriented picks the world, a new file system evolved, named as XtreemFS [18]. It has been specifically designed for Grid environments as a part of the XtreemOS [18] operating system. As an object based design, it is composed of clients, OSDs and metadata servers that are also responsible for keeping replica locations. As a result of reverse engineering of GFS, Hadoop Distributed File System [1] evolved. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS [20] provides high throughput access to application data and is suitable for applications that have large data sets. An HDFS [1][20] cluster consists of a single Name Node, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of Data Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes. The Data Nodes are responsible for serving read and write requests from the file system’s clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node. In Table 1, we have compared different DFS based on different designing issues.

Table 1: Comparison of Distributed File System

File System Design Issues	NFS	AFS	GFS	XFS	HDFS
Scalability	Limited	√	√	√	√
Reliability	X	√	√	√	√
Flexibility	X	Limited	√	√	√
Transparency	X	Limited	√	√	√
Fault Tolerance	Limited	Limited	√	√	√
Security	√	√	√	Limited	Limited

Architecture	Centralize	Decentralize	Decentralize	Decentralize	Decentralize
Process	Stateless	Stateless	State-full	State-full	State-full
Naming	CFS	CMS	CMS	CMS	CMS
Synchronization	X	WORM	WORM	WORM	WORM

3. DESIGNING ISSUES IN DFS

Design issues in DFS are scalability[2][4], reliability[2][4], flexibility[2][4], transparency[2][4], fault tolerance[2][4], security[2][4], architecture[2][4], process type[2][4], naming[2][4], synchronization[2][4], In this section we will discuss about all these issues.

3.1 Scalability

Now a day, more and more users or client are getting connected with system. Hence, we require an efficient system to handle those users or client. For that at any point of time we may require to scale up our system. If we follow the centralize architecture at the time of designing of DFS then it would require more administration to scale up the DFS but if we use decentralize architecture then it can be easily managed by administrator. In Hadoop DFS design, we found that we can increase number of data nodes but we can’t increase number of name node which leads single point of failure.

3.2 Reliability

Reliability means that data should be available when and where it required on time without alteration or modification and without any errors. If we use replication to provide reliability, all the replicas must be consistent with its content. Hadoop DFS creates multiple replicas for a single block. By default a file’s replication factor is 3 but we can change it.

3.3 Flexibility

To achieve flexibility in the DFS, we must decide that whether we want to manage activities like memory management, process management and resource management or not. If we want to manage all these things as it requires then we can use micro kernel approach. Otherwise, we can use monolithic kernel approach in which kernel does all things by its own. Hadoop DFS provides this flexibility.

3.4 Transparency

While designing a DFS, some degree of transparency should be achieved. Like file name does not reveal the file’s physical storage location, client should see uniform namespace. Hadoop DFS design provides transparency like location, access, replication, and concurrency.

3.5 Fault Tolerance

Because of hardware or software failure in distributed-file systems, these systems have to provide a fault-tolerant capability so as to tolerate faults and to try to recover from these faults. There are techniques like Replication and Redundant Arrays of Inexpensive Disk (RAID) available to provide redundancy for fault tolerance. Traditionally, distributed-file systems have relied on redundancy or high availability. In general, file systems replicate at the server-level, directory-level, or file-level to deal with processor, disk, or network failures. Redundancy allows these systems to operate easily and continuously despite partial failure at the cost of maintaining replicas in the file system. By providing decentralized system, we can avoid one point of failure of the DFS .Hadoop DFS do not use data protection mechanism such as RAID to make data durable instead the file contents are replicated on many DataNodes which provides high fault tolerance. DataNodes send heartbeats to the NameNode to confirm that DataNode is alive and blocks replicas it hosts are available.

3.6 Security

To achieve a security in any system, we should focus on mainly three aspects i.e. Confidentiality, Integrity and Availability. Confidentiality can protect our system from unauthorized access, Integrity can identify and protect our data against corruption, and Availability avoids situations like failure of system. To achieve Confidentiality we can use authentication techniques, using message digest we can achieve integrity. Hadoop DFS doesn’t provide any dedicated security mechanism but as it open source we can put security mechanism as per our requirement.

3.7 Architecture

There are mainly four types of architecture (i.e. Client - Server, Parallel, Centralize and Decentralize) used to design a DFS, which are mainly describes goal of file system. Hadoop DFS has Master /Slave with single NameNode and many DataNodes decentralize-parallel architecture.

3.8 Process

In distributed file service, file servers processes can be stateless or stateful. Stateless file servers do not store any session state and every client request is treated independently. While state-full servers, do store session state and keep track of which clients have opened which files, current read and write pointers for files, which files have been locked by which clients, etc. Hadoop DFS is a stateful file server.

3.9 Naming

While designing a DFS, we should consider whether all machines and processes should have the exact same view of the directory hierarchy or not. DFS should provide Location transparency with location independence and access transparency. Location transparency means path name of a file gives no hint to where the file is located. Hence, files can be moved without their names changing. In access transparency, applications and users can access remote files just as they access local files. To facilitate this, the remote file system name space should be syntactically consistent with the local name space. In Hadoop DFS entire namespace and file’s metadata are handled by NameNode it uses CMS approach for naming.

3.10 Synchronization

More than two users share same file at that time it is necessary to maintain semantics of reading and writing of file to avoid consistency problems. There are various ways to provide synchronization of files. One can use file locking system, but administration of the locking system can be handled by either client or server. We can use hybrid approach also. Another alternative is to use atomic transactions. To access a file or a group of files, a process first executes a begin transaction primitive to signal that all future operations will be executed indivisibly. When the work is completed, an end transaction primitive is executed. If two or more transactions start at the same time, the system ensures that the end result is as if they were run in some sequential order. All changes have an all or nothing property [10].Hadoop DFS use write once read many approach.

3.11 Consistency & Replication

In DFS, to maintain a consistency caching is used either on server side or client side. Caching is performed to improve system performance. There are four places in a distributed system where our data can be held: On the server's disk, Cache in the server's memory, in the client's memory, on the client's disk. For high availability of data, replication is the primary mechanism. Hadoop DFS replicate each file block on different DataNode, default replication factor is three in HDFS.

4 HADOOP DFS ARCHITECTURE

The proposed architecture of a distributed file system is designed to run on commodity hardware. It has many similarities with existing distributed file systems and HDFS [1][20]. However, the differences from other distributed file systems are significant. Proposed DFS is designed to achieve high throughput and data transfer rate. It has master/slave architecture. A Proposed DFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes. The NameNode and DataNodes are responsible for serving read and write requests from the file system's clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the NameNode. The NameNode and DataNodes are pieces of software designed to run on machines. Hadoop run on commodity hardware and supports many platforms like GNU/Linux, Windows, Mac operating system (OS). Proposed DFS will be built using the Java language; any machine that supports Java can run the Name Node and DataNode software. Usage of the highly portable Java language means that Proposed DFS can be deployed on a wide range of machines. The NameNode is the arbitrator and repository for all DFS metadata and DataNodes are responsible for reading and writing HDFS blocks to actual files on the local filesystem and communicate with other DataNode for replication. Our system is designed in such way, when client writes a file, data blocks replication are written on DataNodes in parallel manner instead of pipeline.

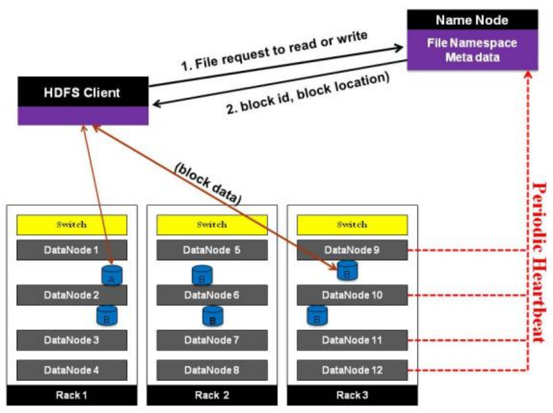

Fig 1: Hadoop DFS Architecture

4.1 Hadoop DFS file read and write operation

Client writes data on Hadoop DFS by creating a file and writing data on that file. Client first requests a NameNode to write data on a file. NameNode checks the permissions and assigned unique block ID and list of DataNodes to host replicas. Now, client directly communicates with DataNodes. A single block is replicate on many other DataNodes through pipeline. After a data block is written on all replication DataNode through pipeline client can request for next block to write. Fig. 2 shows HDFS file write operation.

Client request a NameNode to read a file. NameNode sends list of blocks and locations of each block replica. When reading a content of a block, the client tries the closet replica first. If the read attempt fails, the client tries the next replica in sequence

4.2 Hadoop DFS file write operation using our Parallel approach

Creation and writing of a file is faster than the Hadoop DFS file. NameNode (NN) never writes any data directly on DataNodes (DN). Only manages the namespace and inode .In our approach single block is written on three different DataNodes, Assume its DN1, DN2 and DN3.

1. Client request NameNode to write a file.
2. Client first receives list of DataNodes to write and to host replicas of a single block.
3. Client first writes a block to DN1.
4. Once a block is filled on DN1, DN1 creates thread and request to DN2 and DN3 for creating replicas of a desired block in parallel.
5. Once block is written on DN2 and DN3, they send acknowledgement to DN1.
6. After getting acknowledgement from both DN2 and DN3, DN1 sends acknowledgement to client. If DN1 fails to receive acknowledgement from any of DN2 or DN3, it resend same block.
7. Finally client sends acknowledgement to NameNode that block is successfully written on three different nodes

Fig.3 shows writing a file using parallel approach.

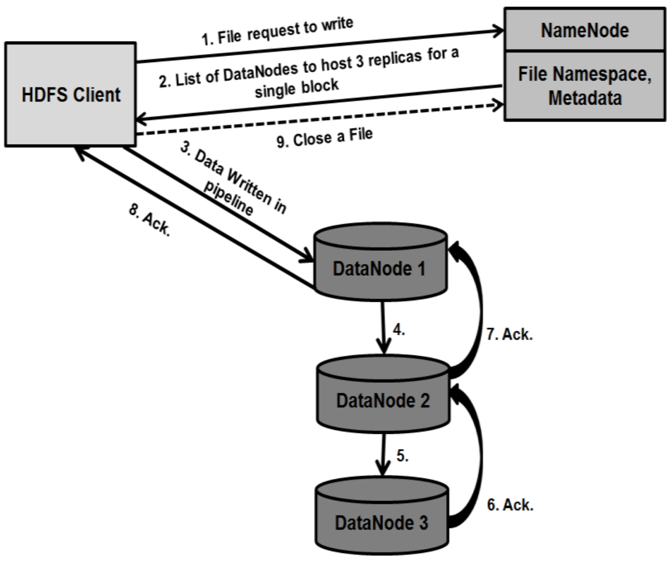

Fig.2 Writing a File on Hadoop DFS using pipeline approach

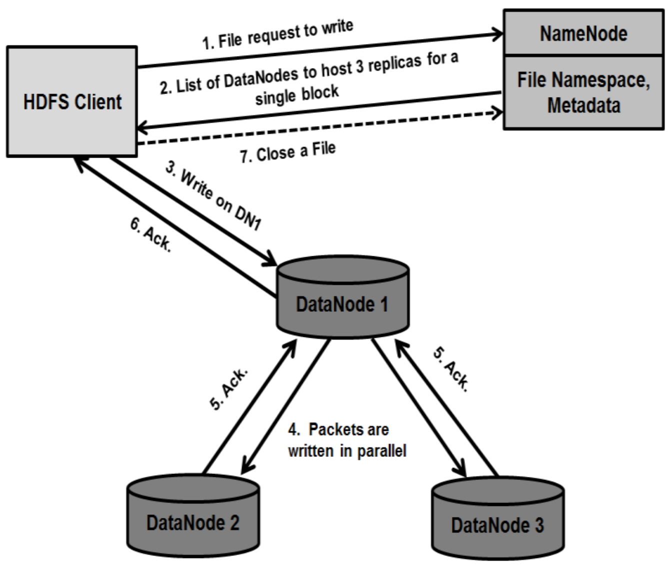

Fig.3 Writing a File on Hadoop DFS using our parallel approach

4.3 Advantages of Proposed DFS

The proposed design of DFS has many advantages such as it improve data transfer rate and throughput, because replicas are written in parallel fashion instead of pipeline. Here replicas on other two DataNodes are created in same time, so large volume of data are transfer, which increase data transfer rate and number of bytes written per second are more than pipeline approach, which increase throughput.

4.4 Limitation of Proposed DFS

Compare to HDFS our proposed DFS may have increased overhead on first DataNode when two other DataNodes read data in parallel from its buffer. It may increase communication cost on first DataNode.

5 CONCLUSION AND FUTURE WORK

Data replication is a technique commonly used to improve data availability. In HDFS each block is replicated on different nodes. In our approach replica factor is three. In this paper we present new approach for efficient replica placement on HDFS that can increase throughput by replicating data blocks in parallel and also large volume of data are transfer which can increase data transfer rate.

In the future, we plan to implement this proposed approach and compare the results with existing approach. We also plan to integrate other Hadoop component and evaluate results and identify architectural bottlenecks of higher level HDFS designs.

6 REFERENCES

[1] Lin Weiwei, Liang Chen and Liu Bo. A Hadoop-based Efficient Economic Cloud Storage System. PACCS at Wuhan, China - July 2011. IEEE Conference Publication.

[2] Mahesh Maurya, Chitvan Oza and Prof. Ketan Shah. A Review of Distributed File System. ICAET at Nagapattinam, India – May 2011. CiiT International Journals Conference Publication.

[3] MARTIN PLACEK and RAJKUMAR BUYYA. Taxonomy of Distributed Storage Systems. The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne- July 2006.

[4] Tran Doan Thanh, Subaji Mohan, Eunmi Choi, SangBum Kim, Pilsung Kim. A Taxonomy and Survey on Distributed File Systems. NCM at Geongju, Korea - September 2008. IEEE Conference Publication.

[5] Song Guang hua, Chuai Jun na, Yang Bo Wei, Zheng Yao. QDFS – A Quality Aware Distributed File Storage Service Based on HDFS. IEEE-CSAE at Shanghai, China - June 2011. IEEE Conference Publication.

[6] Debessay Fesehaye, Rahul Malik, Klara Nahrstedt. EDFS - A Semi- Centralized Efficient Distributed File System. Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware Article No. 28 Springer – Verlag, New York, USA – 2009.

[7] Fabio Kon. Distribute File Systems Past, Present and Future A Distributed File System for 2006. March 1996.

[8] M Satyanarayanan. A Survey of Distributed File Systems. February 1989. Tech. Rep. CMU-CS-89-116, Pittsburgh, Pennsylvania.

[9] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in adistributed file system. ACM Transactions on Computer Systems, 6:51–81, 1988.

[10] Design and Implementation or the Sun Network Filesystem by Russel Sandberg , David Goldberg , Steve Kleiman , Dan Walsh , Bob Lyon.

[11] Debessay Fesehaye, Rahul Malik, Klara Nahrstedt. A Scalable Distributed File System for Cloud Computing.

[12] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," , Incline Village, NV, 2010, pp. 1-10.

[13] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google file system," in SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, 2003, pp. 29-43.

[14] Philippas Tsigas, "AFS Report," Department of Computing Science, Chalmers University of Technology, Göteborg, Sweden, Lecture 2010.

[15] John H. Howard, "An Overview of the Andrew File System," in Proceedings of the USENIX Winter Technical Conference, Dallas TX, 1988.

[16] http://www.openafs.org/

[17] http://research.google.com/gfs.html

[18] http://www.xtreemfs.org/

[19] http://hadoop.apache.org/