2014 IEEE Fourth International Conference on Big Data and Cloud Computing

Fault Tolerant Erasure Coded Replication for HDFS Based Cloud Storage

Aye Chan Ko
University of Computer Studies, Mandalay
Myanmar
Ayechankomm86@gmail.com

Wint Thida Zaw
University of Computer Studies, Mandalay
Myanmar
wintthida@gmail.com

***Abstract*— Businesses and individuals move their data to the cloud because fault-tolerant data storage is becoming more important. Currently fault-tolerance cloud storage file systems are available and being used widely. Hadoop Distributed File System (HDFS) has been widely adopted to build cloud storage systems. The default storage policy in cloud file systems has become triplication (triple replication), implemented in the HDFS and many others. Triplication has been favoured because of its ease of implementation, high performance, and reliability. The storage overhead of triplication is a concern; we present the HDFS along with how fault tolerance is achieved by means of erasure coded replication. The placement of the replicas is critical to HDFS reliability and performance, the core concept of the consistent hashing is applied in this work. To evaluate the performance of our HDFS with erasure coded replication scheme, we focus on least storage space consumption and good storage space utilization. We conduct the experiment on original HDFS and HDFS with erasure coded replication. The experimental results show that our scheme can save storage space and utilization is significantly better in erasure coding.**

Keywords—cloud storage; consistent hashind; erasure coded replication; fault tolerant; HDFS.

I. INTRODUCTION

The availability of cloud storage services is becoming a popular option for consumers to store data that is accessible via a range of devices, such as personal computers, tablets, and mobile phones. Business and individuals applications are moving to the cloud, and it became a big challenge that how to make sure the information which storages in clouds are reliable. Private cloud storage is a type of storage mechanism that stores an organization's data at in-house storage servers by implementing cloud computing and storage technology. Private cloud storage is similar to public cloud storage in that it provides the usability, scalability and flexibility of the storage architecture. But unlike public cloud storage, it is not publicly accessible and is owned by a single organization and it's authorized external partners [8].

Cloud storage is becoming a popular business paradigm, e.g. Amazon S3, ElephantDrive, Gigaspaces, etc. Small companies that offer large Web applications can avoid large capital expenditures in infrastructure by renting distributed storage and pay per use. Ample research on cloud storage file systems [3, 5, 7] has been done but no detailed study on the aspect of fault-tolerant mechanisms of cloud storage is yet to be done at the moment of writing.

. Apache Hadoop is an open source software framework created by Doug cutting and Michael J. Cafarella [1]. The Hadoop Distributed File System has been widely adopted to build cloud storage systems. It provides reliable storage and high throughput access to large-scale data by Map/Reduce parallel applications [2]. The primary objective of HDFS is to store data reliably even in the presence of failures. An uniform triplication policy (i.e. three replicas for each file) is used in HDFS for improving data locality and ensure data availability, and fault tolerance in the event of data and disk failures

One of the widely used methods to improve data reliability is distribution of data with redundant information on several storage devices. As a result, if a storage node fails, enough information can still regenerate the failed node or reconstruct the original file. In the distributed cloud storage environment, reliability can be defined as tolerance to storage node failure while availability means promptly access to the original file and the storage efficiency is the amount of redundant information stored in the system. Replication is a process where a whole object is replicated some number of times, thus providing protection if a copy of an object is lost or unavailable. In the case of replicating a whole object, the overhead would be 100% even for a single replica. Erasure coding [11] is a process where data protection is provided by slicing an individual object in such a way that data protection can be achieved with greater storage efficiency that is some value less than 100%.

Erasure codes [4] are space-efficient schemes to encode data into redundant fragments to protect against erasures of some of the fragments. Such erasure codes have been used traditionally in communication systems and more recently, in storage systems as a way to protect data against node crashes. Erasure codes provide space-optimal data redundancy to protect against data loss. A common use is to reliably store data in a distributed system, where erasure-coded data are kept in different nodes to tolerate node failures without losing data.

In this paper, we present the framework of HDFS with erasure coded replication scheme and analyze how fault tolerance is achieved by means of erasure coded replication. This paper is organized as follows: Section II discusses the related work. HDFS is presented in Section III. Proposed fault tolerant erasure coded replication scheme for HDFS based cloud storage framework is presented in Section IV.

978-1-4799-6719-3/14 $31.00 © 2014 IEEE
DOI 10.1109/BDCloud.2014.69

In Section V, system evaluation and results discussion are provided. Section VI draws conclusions from the work conducted.

II. Related Work

This section reviews the previous literatures concerning with distributed file systems for cloud storage which introduce general architectural issues affecting performance, and reliability. Maheswaran et al. [11] proposed and implemented a new set of erasure codes on Hadoop HDFS to overcome the limitation of high repair cost of Reed-Solomon codes. Their study showed a reduction of approximately 2x on the repair disk I/O and repair network traffic. However, this coding requires 14% more storage compared to Reed-Solomon codes.

In [14], the author presented the benefits and drawbacks of existing fault-tolerant file systems by defining criteria on which fault-tolerant file systems can be graded. Three file systems: Apache Hadoop File System (HDFS), GlusterFS and XtreemFS that are architecturally different have been compared on network related fault-tolerance in this paper. A total of six criteria each of which is associated with a phase in a network transaction have been identified and used for comparison. According to the analysis, HDFS is overall very capable of delivering fault-tolerance for file storage. GlusterFS is an robust file system which is the only file system to receive the good grade on all six criteria. The XtreemFS file system architecture is not quite as resilient as the two other tested file systems.

RomanusIshengoma [12] designed and implemented HDFS+, an erasure coding based Hadoop Distributed File System for improving both space efficiency and I/O performance of the HDFS while preserving the same data reliability level. The performance of the proposed scheme is compared with the HDFS. The experimental results showed that their proposed scheme can save storage space while outperforming the original scheme in write performance. Their scheme provided the same read performance as the original scheme as long as data can be read from the primary DataNode even under single-node or double-node failure.

III. Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware [2]. 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 is designed to reliably store very large files across machines in a large cluster [6]. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The default replication factor in HDFS is 3. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time. HDFS has master/slave architecture. The architecture of Hadoop distributed file system [10, 13] including daemons is illustrated in Figure 1. There are three major daemons that can make up a standard Hadoop cluster. They are Master Daemons, Client Daemons and Slave Daemons. Each Daemon has its own participants. The pair of daemons with its related candidates can be found as follows:

- **Master daemons:** Primary NameNode, Secondary NameNode and JobTracker
- **Slave daemons:** DataNodes and TaskTrackers
- **Client daemons:** Clients

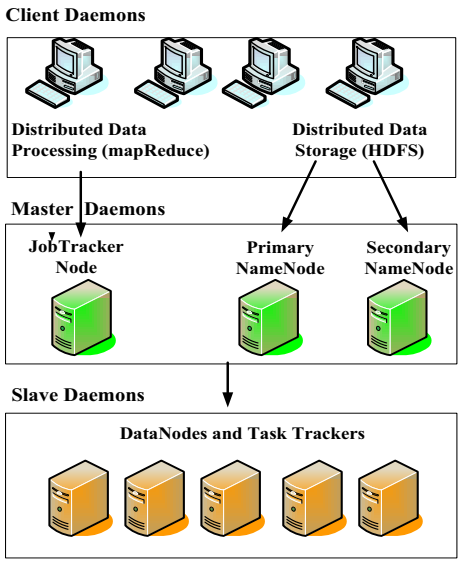

Figure 1. Conceptual Model of Hadoop Distributed File System.

IV. Fault Tolerant Replication Framework

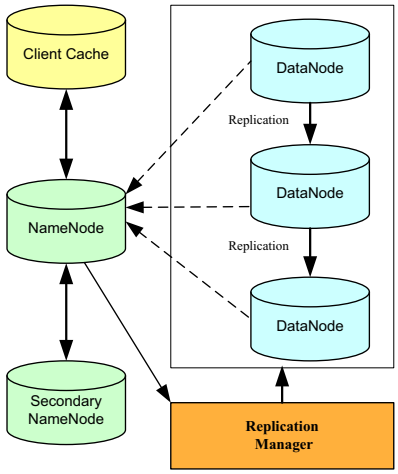

Figure. 2. Framework of Fault Tolerant Replication for HDFS.

The conceptual framework of fault tolerant replication scheme for HDFS based cloud storage is illustrated in Figure 2. Replication manager is responsible for encoding, replication and reconstruction of lost data. The processes of erasure coded replication and placement of replica (Data Distribution) in HDFS cluster are shown in Figure 3.

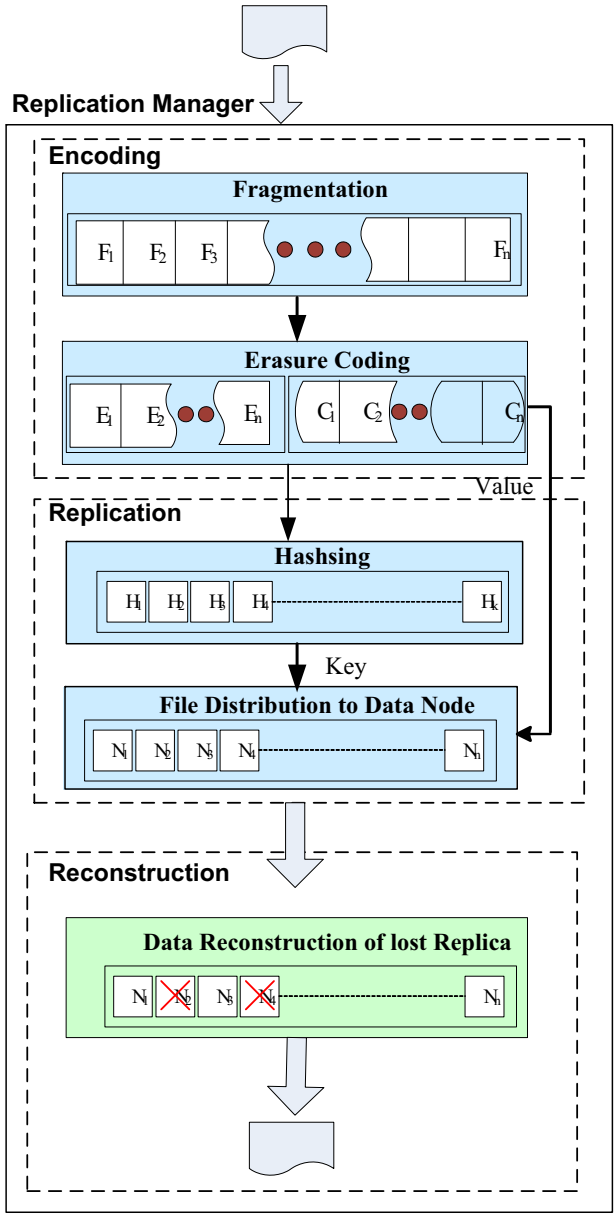

Figure 3. Processes of Replication Manager.

The algorithm of erasure coded replication works for a file as presented in Figure 4.

Algorithm: Erasure Coded Replication

Input:
F, an input file for replication;
Output:
RF, a reconstructed file after replication;
Method:
(1) **encode**( );// file encoding
(2) **replicate**(); // encoded file replication
(3) **reconstruct**( );// file reconstruction from replicas
(4) **return** *reconstructed_file*;

procedure encode ( )
(1) readFile( );// Reads the contents of *F* in byte by byte manner}
(2) InformationDispersalEncoder( ); //Cauchy Reed Solomon Encoding Process for the file
(3) save_fragment_metadata( ); //store the metadata of encoding fragments of file *F* to candidate VM

procedure replicate() // To save fragments after encoding
(1) **for each** fragment *frag* ∈ *fragList* {
(2) **get_candidateVM**( ); // get candidate VM from VM pools with ring topology for file replication
(3) distributeFragToVM(); // send fragment to the candidate VM and replicate
(4) }

procedure reconstruct( ) // file reconstruction by collecting the replicated data over replica VMs
(1) readMetadataFile( ); // read the information from a meta data file specified by a file.
(2) readFragment( ); // read each fragment specified by the fragment-id
(3) decode ( );
(4) writeDecodedContent( ); // reconstruct file collecting blocks

procedure decode( ) // decode the list of packets by erasure coding.
(1) getDigestforPackets ();
InformationDispersalDecoder ( ); //Cauchy Reed Solomon Decoding Process

Figure 4. Replication Algorithm.

A. *Erasure Coded Replication*

Reliability is usually obtained through redundant nodes in distributed storage systems. The simplest way of providing redundancy might be by repetition. Recently, both academic and industrial storage systems have addressed this issue by relying on erasure codes to tolerate component failures. Erasure codes provide space-optimal data redundancy to protect against data loss. A common use is to reliably store data in a distributed system, where erasure-coded data are kept in different nodes to tolerate node failures without losing data.

In an erasure coded system (Figure 5), a total of n=k+m disks are employed, of which k hold data and m hold coding information. The act of encoding calculates the coding information from the data, and decoding reconstructs the data from surviving disks following one or more failures. Storage systems typically employ Maximum Distance Separable (MDS) codes, which ensure that the data can always be reconstructed as long as there are at least k disks that survive the failures.

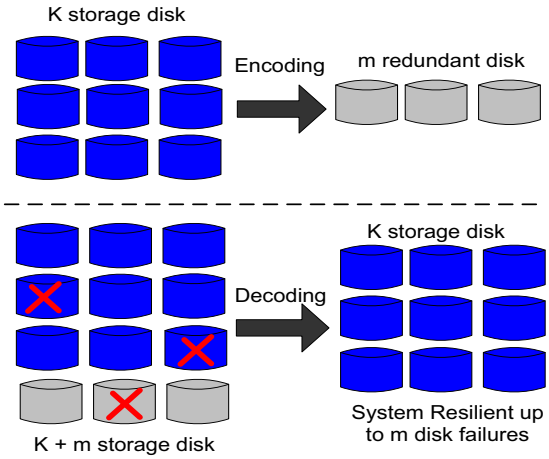

Figure 5. Data Resiliency.

B. *Data Distribution Strategie*

The placement of the replicas is critical to HDFS reliability and performance, the core concept of the consistent hashing [9] is applied in data distribution. While running collections of caching machines some limitations are experienced. A common way of load balancing ***n*** cache machines is to put object ***0*** in cache machine number ***hash(0) mod n***. But this will not work if a cache machine is added or removed because ***n*** changes and every object are hashed to a new location. This can be disastrous since the originating content servers are flooded with requests from the cache machines. Hence consistent hashing is needed to avoid swamping of servers.

Consistent hashing maps objects to the same cache machine, as far as possible. It means when a cache machine is added, it takes its share of objects from all the other cache machines and when it is removed, its objects are shared between the remaining machines. The main idea behind the consistent hashing algorithm is to associate each cache with one or more hash value intervals where the interval boundaries are determined by calculating the hash of each cache identifier. If the cache is removed its interval is taken over by a cache with an adjacent interval. All the remaining caches are unchanged.

Consistent hashing is based on mapping each object to a point on the edge of a circle. The system maps each available machine to many pseudo-randomly distributed points on the edge of the same circle. Consistent hashing specifies how keys are to be assigned to nodes and how a node can discover the value for a given key by first locating the node responsible for that key. It maps the items to corresponding nodes where node's identifier is the hashing the node's IP address and the key's identifier is hashing the key. According to the example illustrated in Figure 5, there are two nodes A and B and three objects 1–3 initially.

The objects 3 and 1 are mapped to node A, object 2 to node B. When a node leaves the system, data will get mapped to their adjacent node (in clockwise direction) and when a node enters the system it will get hashed onto the ring and will overtake objects.

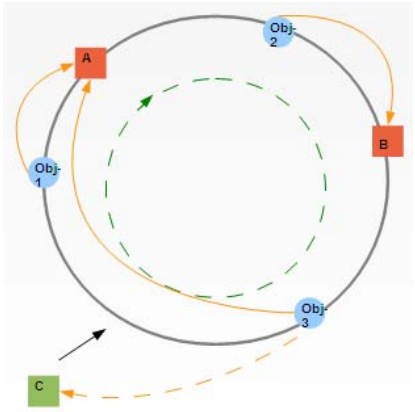

Figure 6. Demonstration of Consistent Hashingd

The illustrated procedure for encoding data replicas can be seen in Figure 6. To put the encoded blocks (***n***) in the storage pool, the core concept of the consistent hashing is applied. A consistent hash function is the mapping of items to corresponding nodes. A node's identifier is defined by hashing the node's IP address***, Hash(IP)***, and the identifier of a key, which is an item, is produced by hashing the key, ***Hash(key)***. During the process, *<key, value>* is submitted to a node of the ring which has the associated key. Each encoded block is in the form of $<Hash(n_{id}), n_i>$. n_{id} is the identifier of the blocks.

V. EVALUATION

The performance of proposed system is evaluated on commodity Linux cluster with virtual machines. Cluster (testbed) is composed of virtual machines. The VMs are interconnected via a 1-gigabit Ethernet. Each host machine runs Windows 7 Ultimate.

In the Hadoop cluster implemented, one VM stands for NameNode, Secondary NameNode, DataNode, JobTracker and TaskTracker. The other VMs stand for DataNodes and TaskTrackers. Table I displays the experimental setup to test the performance of original HDFS and HDFS with erasure coded replication scheme.

TABLE I. PARAMETERS OF TESTBED

Testing Environment	System Specification
Commodity Linux VMs Cluster	Cluster- 8VMs
Hosts Specification	Intel ® Core™ i7- 2600 CPU @ 3.40GHz, 4 GB RAM, 1TB Hard Disk, 1Gigabit Ethernet
	Intel® Core™ i5-3337U CPU @ 1.80GHz, 4GB RAM, 500GB Hard Disk, 1Gigabit Ethernet
VMs Specification	VM1 Intel ® Core™ i7- 2600 CPU @ 3.40GHz, 1024MB RAM, 50 GB Hard Disk NN, SNN, DN, JT, TT
	VM2 to VM8 Intel ® Core TM i5-3337U CPU @ 1.80GHz DN, TT
Software Components	Storage System • Hadoop 0.20.2 • VMware®Workstation 9.0.2 build-1031769 • Ubuntu 12.04 LTS

NN=NameNode SNN=Secondary NameNode
JT=JobTracker DN=DataNode TT=TaskTracker

We implemented the algorithm shown in Figure 5. Enwiki dataset [15] is used to evaluate the performance of proposed system. We tested performance of writing and reading files of different file sizes.

A. Storage Consumption

Erasure codes are space-efficient schemes to encoded data into redundant fragments to protect against erasures of some of the fragments. Such erasure codes have been used traditionally in communication systems and more recently, in storage systems as a way to protect data against node crashes. It can also make the storage systems reliable.

In the experiment, we kept the fault tolerant level at 2 and the obtained result is as shown in Figure 7.

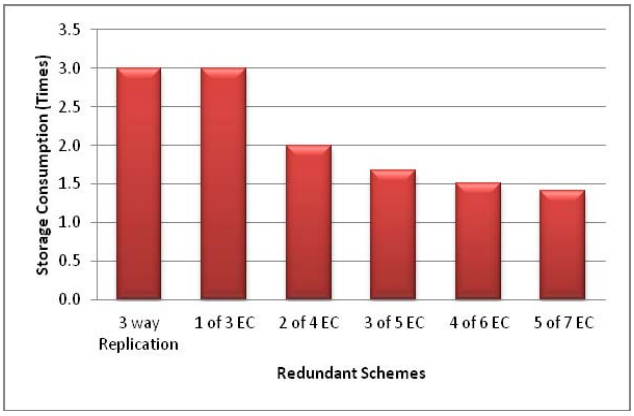

Figure 7. Storage consumption (times) vs. redundant schemes.

We found that, by using 3 way replication, the storage consumption is 3 times and by using 3 of 5 erasure coded replication, the storage consumption is only 1.6 times. Hence, 3 of 5 erasure coded replication can save data written over the network onto the storage while tolerating the same number of faults.

The amount of occupied storage space depends on data redundant schemes. There can be variants even with erasure coding techniques according to the configuration shown in Table II.

TABLE II. DIFFERENT CONFIGURATIONS OF REDUNDANT SCHEMES

Redundant Scheme	3-way	1 of 3EC	2 of 4 EC	3 of 5 EC	4 of 6 EC	5 of 7 EC
n total nodes	3	3	4	5	6	7
k data nodes	1	1	2	3	4	5
m coding nodes	0	2	2	2	2	2

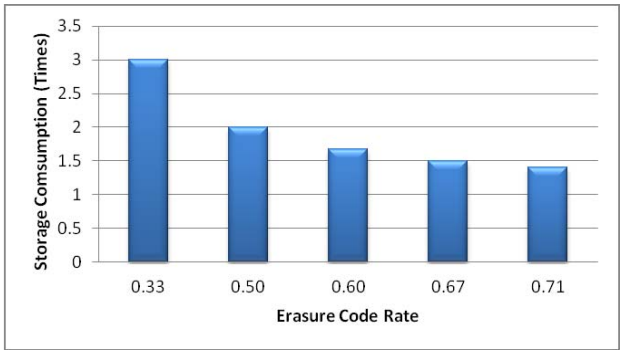

Figure 8. Storage consumption (times) vs. erasure code rate.

The relationship between the rate of encoding and the storage consumption can be seen in Figure 8. When the erasure code rate is increased the storage consumption is decreased.

B. Storage Space Utilization

Storage space utilization is a measure of how well the available data storage space in an enterprise is used. It measures how full the space is compared to its capacity. Utilization rates can be assessed in terms of both actual use and predicted use.

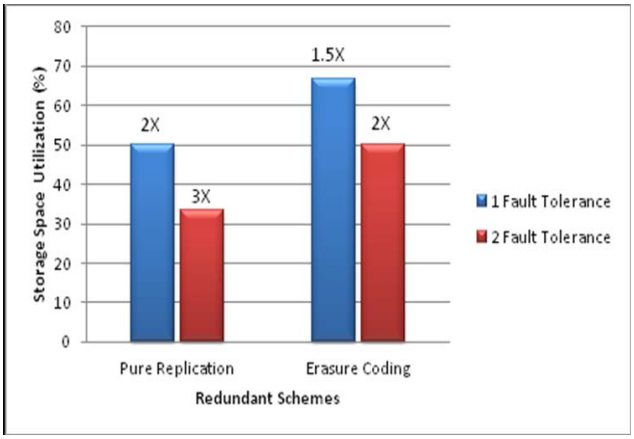

Figure 9. Storage space utilization vs. redundant schemes

The experimental results of storage capacity utilized by two different redundancy schemes: pure replication and erasure coding are illustrated in Figure 9. Beside it compares the storage space utilization for two kinds of fault tolerant level. Obviously, to stand for 1fault tolerance, it needs only 1.5x storage overhead in erasure coding. But the storage space utilization of erasure code is 50%, and 30% in pure replication. Therefore the utilization is significantly better in erasure coded replication.

VI. CONCLUSION

Fault-tolerance is an important aspect in cloud storage because it concerns the robustness of the data that is stored. HDFS has become popular due to its reliability, scalability, and low-cost storage capability. HDFS is designed to operate on commodity hardware components, which are prone to failure. Files are triplicated (triple replication) to guarantee high data reliability. To reduce storage overhead, the default storage policy in cloud file systems has become triplication (triple replication), implemented in HDFS and many others. Triplication has been favored because of its ease of implementation, good read and recovery performance, and reliability.

The storage overhead of triplication is a concern, cloud file systems are transitioning from replication to erasure codes. In this paper, we implemented HDFS with default storage policy and HDFS with an erasure coding. We compared the performance of the HDFS and HDFS with erasure coded replication scheme. The experimental results show that the HDFS with erasure coded replication can save up to 33 % of space while outperforming the original HDFS scheme. We also found that the storage space utilization for any fault tolerant level is significantly better in erasure coding than pure replication.

REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org

[2] D. Borthakur, The hadoop distributed file system: Architecture and design, 2007, Hadoop Project Website, 11, 21.

[3] M. C. Chan, J. R. Jiang, and S. T. Huang, Fault tolerant and secure networked storage, In proceeding of the 7th International Conference on Digital Information Management, ICDIM 2012. pp. 186-191, 2012.

[4] A. G. Dimakis, K. Ramchandran, Y. Wu and C. Suh, A survey on network codes for distributed storage, In proceedings of the IEEE, vol. 99, no. 3, pp. 476--489, March 2011.

[5] J. Evans, Fault Tolerance in Hadoop for Work Migration, 28-03-2011, 2011.

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

[7] F. Hupfeld, T. Cortes, B. Kolbeck et al., The XtreemFS architecture - A case for object-based file systems in Grids, Concurrency Computation Practice and Experience, vol. 20, no. 17, pp. 2049-2060, 2008.

[8] http://www.techopedia.com/definition/26847/private-cloud-storage

[9] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, & D. Lewin, 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, May, 1997, pp. 654-663

[10] S. Konstantin, HairongKuang, R. Sanjay, C. Robert, The Hadoop Distributed File System‖, In proceedings of SST2010, May 2010.

[11] S. Maheswaran, A.Megasthenis, P. Dimitric, G. Alexandros, Dimakis, V. Ramkumar and C. Scott, XORing Elephants: Novel Erasure Codes for Big Data, In proceedings of the 39th international conference on Very Large Data Bases, pp. 325-336, 2013.

[12] F. RomanusIshengoma, HDFS+: Erasure Coding Based Hadoop Distributed File System, International Journal of Scientific & Technology Research 2.9 (2013). pp.190-197

[13] The Apache Software Foundation, ―Welcome to Hadoop Distributed File System,‖ http://hadoop.apache.org/hdfs.

[14] S. Verkuil, A Comparison of Fault-Tolerant Cloud Storage File Systems, In proceedings of the 19th Twente Student Conference on IT, June 24th, 2013, Enschede, The Netherlands.

[15] Wikimedia dump service:
http://dumps.wikimedia.org/enwiki/20140102/