Novel Approach for Distributed File System with Multiple Layers of Fault Tolerance

Keshav Gupta
Assistant Professor
School of Computing Science and Engineering
Galgotias University
Uttar Pradesh, India
keshavgupta101@gmail.com

Jaspal Kaur Saini
Assistant Professor
School of Computing Science and Engineering
Galgotias University
Uttar Pradesh, India
sainijassi87@gmail.com

***Abstract*—Numerous distributed file systems have been implemented for individuals and companies still users continue to face problems in using these systems and it is a still a big challenge for the researchers. In this paper, a novel Fault Tolerant Distributed File System (FTDFS) with multiple levels of fault tolerance is presented. At first level we are trying to filter faulty nodes/clients before they enter into system or we can say register with the server. At second level, to avoid failure we are maintaining chunk replicas so that if a client possessing file chunks gets disconnected, the file can still get downloaded. For security purpose, AES algorithm has been used for encryption and decryption. Many different concepts, various implementations and design strategies have been demonstrated by a survey along with different distributed systems for file sharing. Testing and performance evaluation of the newly developed FTDFS system is presented that improves load balancing by distributing the file over the network and increases the overall system performance.**

Keywords—FTDFS, DFS, AES

I. INTRODUCTION

A file system is used to store files permanently and organizing them in such a way so that they can be easily searched and accessed. On the other hand a distributed file system is a network of file systems, in order to provide the file storage among a group of connected computers. In a general configuration of a distributed file system, there are several mainframes and workstations connected by a LAN. A DFS is implemented as part of the operating system of each of the computers connected to network. The main drawback found in the previous file systems was the lack of the scalability. Also security and reliability is quickly becoming a necessary feature of file storage systems.

This paper proposes a novel Fault Tolerant Distributed File System (FTDFS) with multiple levels of fault tolerance. The FTDFS provides storage for secure files, high scalability, local independence and transparency. The users can upload files. The uploaded file is first encrypted to provide greater security. After encryption file will be divided into various chunks depending on size of file based on given chunk size. The chunks are replicated to increase reliability of the system. The chunks created and replicas will be distributed among all the clients registered provided original chunk and its replica will not be stored on same client. After a successful upload a key will be generated and will be sent to the client which will be named including client ID and name of the file uploaded. While downloading the file, client will send this key to the server. Server will get client ID from the key, directly go to the path specified by the clients where chunks are stored and fetch the chunks related to the file. If a particular chunk is missing or client is unavailable, replicated chunks are retrieved.

After that all the chunks are merged together with the help of a Batch file which was created during the creation of chunks. The retrieved file will be decrypted and will be sent back to the client.

The paper is organized as: Section 2 elaborates distributed file systems. The proposed Hierarchical Distributed File Systems (HDFS) with multiple layers for fault tolerance is explained in section 3. Section 4 shows the simulation steps and experimental results. The system is analyzed on the basis of scalability, reliability by connecting a very high number of nodes in the section 5. Section 6 concludes the paper.

II. LITERATURE REVIEW

Users can store and access remote files like a local computer would with help of DFS but from any computer in the network [1]. It distributes the files from one computer to other computers. A Distributed File System is basically a distributed implementation of the classical time sharing model which is constructed on a file system where it allows one or many users to share the files and storage resources [2]

DFS supports the common operations effectively on the same kind of sharing resources. It supports especially the operations for the files physically dispersed among the different sites of a distributed system. If it is to be seen from the inheritance point of view, a DFS is a system which inherits the property of both the File System and Distributed System. A DFS provides commonly used files operations such as read, copy, rename, delete and share. These operations can be used in both local sites of DFS as well as in other sites through authorization access. Distributed file system is used as polymorphic subclass that constitutes its processors or the resources distributed mechanism that is derived from Distributed System [2].

A Distributed Operating System can be seen as a collection of interconnected computers which appears to the users or the

clients as a single system. If any system doesn't work, or crashes, it won't affect the working of the other systems interconnected. It manages the data i.e. the files and migrate the data/file from one site to other sites.

A. Fault tolerance in DFS

Alexandru Costan proposed an approach relying on replication techniques and based on monitoring information to be applied in distributed systems for fault tolerance. Their approach uses both active and passive strategies to implement an optimistic replication protocol [3]. Marieta Nastase and others proposed a Fault Tolerance scheme using a Front-End Service for Large Scale Distributed Systems which is based on a set of replicated services running in a fault-tolerant container and a proxy service able to mask possible faults, completely transparent to a client [4]. Sunil Chakravarthy, Chittaranjan Hota proposed a Secure Resilient High Performance File System for Distributed Systems that has the potential to cope with increasing number of participants and ensures strong file encryption [5]. Swastisudha Punyatoya gave Generic Algorithm-Based Fault Diagnosis for Distributed Systems which results in decrease in time and no. of messages passed in order to diagnose the fault [6]. Ioan Petri, proposed a Quorums Systems as a Method to Enhance Collaboration for Achieving Fault Tolerance in Distributed Systems, 2009[7]. Bharath Balasubramanian gave a Coding-Theoretic Approach for fault tolerance in Distributed systems which uses a fusion based fault tolerance scheme [8].

B. Replication in DFS

Bin Cai, Changsheng Xie, Guangxi Zhu proposed an Effective Distributed Replication File System for Small-File and Data-Intensive Application [9]. It works with a single metadata server and multiple storage nodes, deploys whole-file replication scheme at the file level, and tracks what storage node a file is replicated on. Anna Hat, Xiaowei Jin, Jo-Han So0 provided a survey about Algorithms for File Replication in a Distributed System which included File Replication by Using a Single-Host Sequential Update Algorithm, File Replication by Using a Single-Host Concurrent Update Algorithm, File Replication by Using a Multiple-Host Update Algorithm and Algorithm in the System without File Replication[10]. Yan Chen, Randy H. Katz and John D. Kubiatowicz gave a Dynamic Replica Placement policy for Scalable Content Delivery [11]. Gyuwon Song, Suhyun Kimz and Daeil Seo also proposed a Replica Placement Algorithm for Highly Available Peer-to-Peer Storage Systems[12] in which they develop a replica placement algorithm which Exploits the availability pattern of each individual peer.

III. PROPOSED WORK

The Rapid Application Development methodology was chosen to implement the client-server model for DFS because it works well when the work can be broken into manageable chunks like working on various functions and classes in this project. It follows the Client-server architecture. It shows how all clients are linked to each other and to the server. There can be any number of clients connected to the server and any client can upload any no. of files into the system subjected to the secondary storage at client nodes. Also multiple clients are able to download files at the same time.

A. The Proposed Fault Tolerance Policy

Fault Tolerant HDFS provides dual level fault tolerance. At first level it tries to remove faulty nodes/clients before they enter into the system by registering with the server. This is done by maintaining a record of system properties of all the clients like CPU speed, Signal Strength and Battery power. Their values are stored on a scale of 1-10. We have decided a threshold value of 5. If it is more than threshold, clients are able to register with the server else appropriate error message is shown. Thus in FTDFS only clients with higher reliability and availability are present.

At second level, if a client possessing file chunks gets disconnected which was registered successfully in past, replication policy is used to get the lost chunk from some other client, so that file can be downloaded successfully. Thus making FTDFS a highly reliable and fault tolerant distributed file system.

B. Replication Strategy

In order to avoid system failures due to any client disconnection possessing file chunks, replication policy is used. After replication, replicas are placed at different client nodes. There are many replica placement policies [8] [9] [10] [11] which we studied for improving quality of service and availability. A random replica placement policy is used because of very low overhead requirement and good performance. Many distributed systems like FARSITE [17] also use the same replica placement policy. The drawback of random replica placement policy to ignore node availability was also overcome by ensuring that only clients with higher availability are able to register to the server by keeping record of system properties of clients like CPU speed, Signal Strength and Battery power.

C. Encryption Algorithm

For the purpose of security, numerous algorithms have been proposed. The Advanced Encryption Standard is the replacement of DES published by the National Institute of Standards and Technology (NIST) in Federal Information Processing Standard (FIPS). It's the characteristics of AES: ease of implementation, cost effective and security that attracts the researcher to use the AES encryption and its enhanced version for image security[15][16]. We used AES which is symmetric key algorithm that constitutes of four types of transformations:

1. SubByte Transformation: To provide non linearity in the cipher, each byte in the state matrix is replaced with sub byte using 8 bit substitution box.
2. Shift Row Transformation: This transformations shift the rows of state with predetermined offset.
3. Mix Column Transformation: to increase the confusion-diffusion, the column of the state is combined with polynomial.
4. Add Round Key: Exclusive-or operation is performed between byte of state and sub key.

AES_ENCRYPTION(file, secret key)

Step 1: Read the input file.
Step 2: Convert the file into 16*16 blocks.
Step 3: for each block repeat the steps 4 to 6.
Step 4: [Pre initial Round]
4.1: AddRoundKey (state, roundkey)
Step 5: for round=1 to Nr-1 [(Nr-1) Rounds]
5.1: SubBytes (state, s_box)
5.2: ShiftRows(state)
5.3: MixColumns(state, poly_mat)
5.4: AddRoundKey (state, roundkey)
Step 6: [Final Round]
6.1: SubByte (state, s_box)
6.2: ShiftRows(state)
6.3: AddRoundKey (state, roundkey)
Step 7: Combine all encrypted blocks to constitute cipher file.

Fig. 1. AES Encryption

IV. SIMULATION STEPS

A. *Uploading file on FTDFS*

The proposed fault tolerant HDFS is a distributed file system where a node in the network can upload a file on the system and file will be distributed over the network in form of chunks. Figure2depicts steps for uploading a file on FTDFS which are as follows:

1. In the initial step the authenticity of the user is checked. Only authenticated users are allowed to upload a file on the server. An authorized user is the one who is registered with the server.
2. In the next step, a unique ID of the client is generated using the system clock (system clock ensures the uniqueness of the ID).
3. The user uploads the file on the server using a secured TCP/IP connection (TCP + SSL + TCP).
4. The uploaded file is now encrypted using the DES algorithm.
5. The uploaded file is then divided into fixed size chunks and a batch file is maintained to store the information about these chunks.
6. The chunks of the uploaded file are replicated and are distributed on the clients. The replication of chunks is done in a way such that no clients hold the replica of its own original chunks. This ensures fault tolerance within the system.
7. After successful distribution of the chunks on the clients, the client is given a key file to retrieve the file.

B. *Downloading file from System*

Figure2 depicts steps for downloading a file from FTDFS which are as follows:

1. In order to download a file, client will select the respective key and send it to the server. The file corresponding to that key is downloaded.
2. During the download request from client, the key provided by the client is broken down to get the corresponding ID.
3. The server now fetches the chunks from the path specified by the client corresponding to that ID.
4. If a chunk is not found or the client possessing that chunk gets disconnected, a replica is selected.
5. When all chunks are available, server merges them with the help of the batch file and we get the original file but in encrypted form.
6. Now the server decrypts that file and send the original file back to the client.

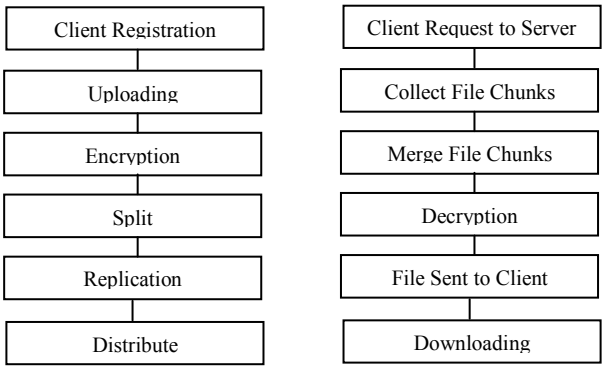

Fig. 2. Uploading and Downloading File on FTDFS

V. PERFORMANCE ANALYSIS

The proposed FTDFS provides security, file replication, scalability and better performance. It is analyzed on the basis of scalability, reliability by connecting a very high number of nodes. Figure 3 illustrates the performance of the proposed system. It is clear from the graph that maximum no of faults at any point of time are less than 3. Hence the FTDFS is highly scalable and reliable.

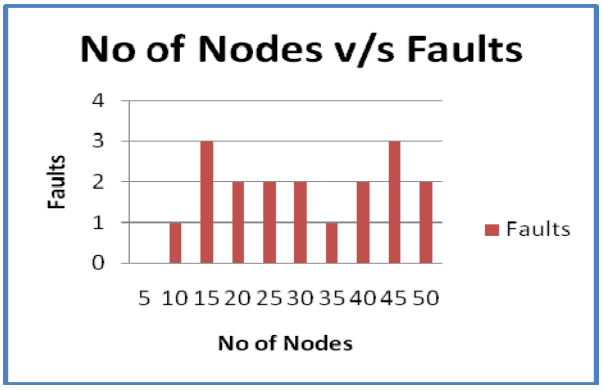

Fig 3. : No of nodes v/s faults

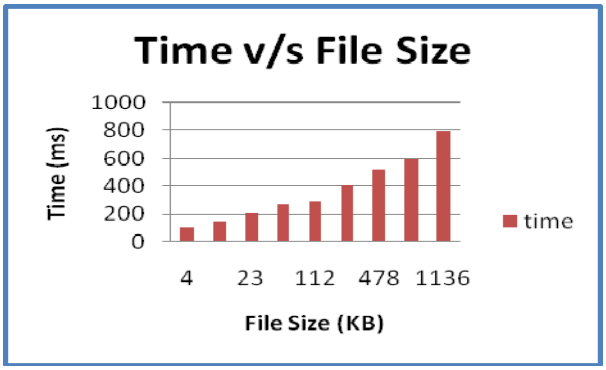

Fig. 4. Time v/s File Size

Figure 4 shows the performance of the FTDFS. It represents the average time to upload the file against time (in milliseconds)

VI. CONCLUSION

Replication plays the crucial role in achieving availability and reliability. If any file/chunk is not available due to any reason, its replica stored at some other location may be used for the same. The proposed FTDFS minimizes the number of faults. No faulty node will be able to register to the server. Various performance parameters CPU speed, signal strength and battery are analyzed. The replicas of file chunks are maintained. If a client possessing a file chunk gets disconnected, it is still able to retrieve the file with the help of replica present on another machine which leads to the multilevel fault tolerance. We have maintained the replicas of the file chunks on other clients as well. So if a client possessing a file chunk gets disconnected, it is possible to retrieve the file with the help of replica present on other machine leading to dual level fault tolerance.

In Future, FTDFS can be transformed into a system where out of n chunks, if some k chunks are available, the original file can be created. It can be further developed for multiple servers in case anyone goes down the system would keep running.

REFERENCES

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," In proceedings of the 19th ACM symposium on Operating systems principles, New York, USA,2003, pp. 29-43.

[2] Tomasz Mista, "Java Distributed File System," M. Sc Project dissertation - COMP5200, London,UK, 2011.

[3] Alexandru Costan, Ciprian Dobre, Florin Pop, Catalin Leordeanu, Valentin Cristea, "A fault tolerance approach for distributed systems using monitoring based replication," In proceedings of the 6th IEEE International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, August 2010.

[4] Marieta Nastase, Ciprian Dobre, Florin Pop, Valentin Cristea, "Fault Tolerance using a Front-End Service for Large Scale Distributed Systems," In proceedings of the 11th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 2009.

[5] Sunil Chakravarthy, Chittaranjan Hota, "Secure Resilient High Performance File System for Distributed Systems," In proceedings of the IEEE International conference on Computer and Communication Technology, Allahabad, India, September 2010.

[6] Swastisudha Punyatoya, "GA-Based Fault Diagnosis Algorithm for Distributed Systems", M. Tech thesis, National Institute of Technology, Rourkela, July 2011.

[7] Ioan Petri, "Quorums Systems as a Method to Enhance Collaboration for Achieving Fault Tolerance in Distributed Systems", Informatica Economica, Academy of Economic Studies - Bucharest, Romania, February 2009, pp.68-75.

[8] Bharath Balasubramanian, "A Coding-Theoretic Approach for fault tolerance in Distributed systems", UT electronic thesis and dissertation, August 2012.

[9] Bin Cai, Changsheng Xie, Guangxi Zhu, "An Effective Distributed Replication File System for Small-File and Data-Intensive Application," In proceedings of the 2nd IEEE International conference on Communication system software and middleware and workshops, Bangalore, India, January 2007.

[10] Anna Hat, Xiaowei Jin, Jo-Han So0, "Algorithms for File Replication in a Distributed System," In proceeding of 13 IEEE conference on local computer networks, Minnesota, October 1988.

[11] Yan Chen, Randy H. Katz and John D. Kubiatowicz, "Dynamic Replica Placement for Scalable Content Delivery," In proceedings of the First International Workshop, IPTPS Revised Papers, Cambridge, MA, USA, March 2002 , pp. 306-318.

[12] Gyuwon Song, Suhyun Kimz and Daeil Seo, "Replica Placement Algorithm for Highly Available Peer-to-Peer Storage Systems," In proceedings of the First International Conference on Advances in P2P Systems, Sliema, Malta, October 2009.

[13] Russel Sandberg, "The Sun Network Filesystem: Design, Implementation and Experience," Sun Microsystems,Inc. Technical Report, (1985).

[14] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System: Architecture and Design," In proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST), Lake Tahoe, Nevada, USA, 6 – 7 May 2010.

[15] Saini, J.K , Verma, H.K," A hybrid approach for image security by combining encryption and steganography"In proceedings of the 2nd IEEE International conference on image information processing(ICIIP), Shimla, India,2013,pp.607-611.

[16] Jaspal Kaur Saini and Kriti Saini. Article: New Advance Encryption Standard to Analyze Encrypted Image Quality. *International Journal of Computer Applications* 74(7):1-5, July 2013.

[17] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, Roger P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," In proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002.