A Flexible and Adaptable Distributed File System

S. E. N. Fernandes[1], R. S. Lobato[1], A. Manacero[1], R. Spolon[2], and M. A. Cavenaghi[2]
[1]Dept. Computer Science and Statistics, Univ. Estadual Paulista UNESP, São José do Rio Preto, São Paulo, Brazil
[2]Dept. Computing, Univ. Estadual Paulista UNESP, Bauru, São Paulo, Brazil

Abstract— *This work describes the development of a flexible and adaptable distributed file system model where the main concepts of distributed computing are intrinsically incorporated. The file system incorporates characteristics such as transparency, scalability, fault-tolerance, cryptography, support for low-cost hardware, easy configuration and file manipulation.*

Keywords: Distributed file systems, fault-tolerance, data storage.

1. Introduction

The amount of stored data increases at an impressive rate, demanding more storage space and compatible processing speeds. Aiming to avoid complete data loss from failures or system overloads, it became usual to adopt the distributed files model [1] [2].

Therefore, a distributed file system (DFS) is a system where files are stored along distinct computers, linked through a communication network. Even though several DFS are capable of attending several characteristics, such as access/location transparency, performance, scalability, concurrency control, fault-tolerance and security, to attend them simultaneously is complex and difficult to manage. Another important aspect to consider is that when one characteristic has its complexity increased, the remaining ones may be negatively affected. This explains why most of the DFS are developed aiming at fulfilling specific scenarios [2] [3] [4].

This paper proposes a novel model for a flexible DFS, named FlexA (**Flex**ible and **A**daptable Distributed File System), that can be adapted to the environment where it is being used. This flexibility allows for DFS features to be adapted or even replaced by other including but not limited to the cryptography algorithm, level of replication, application programming interfaces, move some tasks from servers to clients and several configurations of software and hardware.

In the following sections we start with a brief description of other DFS in use, focusing on ones that are the basis for the model presented here. Then we focus in the description of the proposed model, including its main characteristics and architecture. Results from the model evaluation are presented next, finishing with conclusions drawn from this evaluation and directions for future work.

2. Related work

Among the several existing DFSs, this work focused on exploring the key features of some models of DFSs based on traditional designs and some newer systems, allowing to extract features for the development of a DFSs that has characteristics such as high performance, fault-tolerance and easiness of use.

2.1 Network File System

Network File System (NFS) [2] [3] is a DFS based on remote procedure calls (RPC) providing a convenient medium to applications through a virtual layer (Virtual File System - VFS) that enables a transparent access to NFS components [5] [6] [7].

2.2 Andrew File System

Andrew File System (AFS) was designed aiming scalability to several users. In order to achieve this, aggressive cache policies are implemented on the client side, as well as efficient techniques for consistency [2] [3].

2.3 Google File System

Google File System (GFS) operates on an architecture composed by parallel server clusters. GFS is distinguished by the serialization and file distribution directly to chunk servers that are the actual storage nodes, without the need for additional accesses to the main server, called "master" [1].

2.4 Tahoe - The Least-Authority Filesystem

Tahoe-LAFS is a DFS in the user space, where file sharing occurs through a sequence of characters manipulated by the Uniform Resource Locator (URL). This form of sharing allied to a decentralized security model, based on individual access control, allowed Tahoe-LAFS to manage directories and files as independent objects, which can be referenced by several processes using different names [8].

2.5 Another systems

Besides the systems just presented, other works seek to establish different priorities for their DFS models, such as SPRITE [9], CODA [10], IBM General Parallel File System [11], Ceph [12], XtreemFS [13], HDFS [14], Red Hat Global File System [15] and GlusterFS [16]. Among these DFSs, the

choice of NFS, AFS and GFS is justified by extensive documentation available, allowing to explore problems commonly encountered in the development of DFSs. As regards the Tahoe-LAFS, its importance for this project was motivated by his development in an open-source project, providing a model for access to files by *upload/download* and the use of *the Principle of Least Authority* [17] to distribute files.

3. A model for DFS

This work describes a DFS model that incorporates the important characteristics of NFS, AFS, GFS and Tahoe-LAFS. It is expected to work in a controlled environment, offering support to heterogeneity, flexibility, easier file management, fast and secure cryptographic mechanisms, and fault-tolerance tools.

The main characteristics of this model are listed in this section.

3.1 Adaptability and Flexibility

Adaptability allows clients to become part of the server group for helping in the provision of distributed files. Through this feature, resources of the client station such as disk space can be shared. Also, the client can become a host server completely. The concept of flexibility comes from the possibility of making changes in FlexA to adjust the scenario utilized, in other words, providing a means to modify their functionality by replacement or adjustment of the model components. The components that can be modified are designed to be independent of the set, among them can be emphasized the modification or replacement of the cryptographic algorithm, the changing levels of replication and the adaptation interface for other applications such as, for example, using the file system through a web browser.

3.2 Access Control

Our model uses a set of hash sequences to generate the encryption key file and its variations, providing two levels of access: read-write and read-only. The methodology used for this process is adapted from the cryptographic security model of the Tahoe-LAFS, which follows the Principle of Least Authority. With this model, the files and directories are managed directly by the user through independents handlers, which are responsible for identifying and providing the permissions for that type of file/directory, replacing the traditional fields of "login" and "password". As a result, our model enables users to interact with DFS without the need of system administrator privileges [8].

3.3 Low Cost Hardware

The specification of this our model was based on an open language, widely used in operating system implementations. It also expected that the client stations are in charge of most of the process involved on serving files to the distributed system. This inversion of which side does the processing, allows for several gains in the server side, making it more reliable (less failures due to overloads) and less expensive. Moreover, the rapid advances in of-the-shelf hardware for the client has brought better conditions for the user to effectively use these resources.

File storage is managed by the local file systems, making it easier to adapt to individual operating systems. This also makes model implementation, management and manipulation easier [1] [8].

3.4 Fault-Tolerance

Fault-tolerance is achieved through the division of a file into smaller blocks (chunks), which are transferred to a set of servers. This process allows the client to work with distributed chunks, thus avoiding compromising data integrity due to isolated problems in specific servers.

File chunks are stored in two groups of servers: write and replica. The former is composed of a fixed set of three servers in charge of receiving or providing file chunks, and is capable of reading and writing operations. The latter is composed of replicas of the first group and allows only reading operations of file chunks. Consistency is achieved by modifications initiated by servers from the first group [1] [3] [8].

The use of replicas is optional, enabling the control of chunks availability according to the demand for such chunks.

For agreement and consistency purposes, it is necessary for the write group to have two of the three servers active. All transfer operations only occur under this condition.

3.5 Performance

FlexA tries to minimize the number of interactions with the servers, bringing most of the operations to the client side. Accessing a local file improves the overall performance. To achieve this, the DFS uses a load/update model where the client transfers the file chunks to the local file system before executing operations on the file.

The use of a cache in the client prevents future interactions on files already transferred. In the situation where the file's version becomes outdated, the client will be warned about the availability of a newer version in the servers group.

Chunks are transferred in parallel from different servers, using variable sized blocks. The smallest size of a block is 4096 bytes.

4. Project overview

4.1 Architecture

Differently from the conventional client-server model, FlexA, as shown in Figure 1, eliminates the concept of a main server that would be in charge of manage all requests to the files.

This architecture resembles a peer-to-peer architecture, enabling clients to interact with storage groups without the

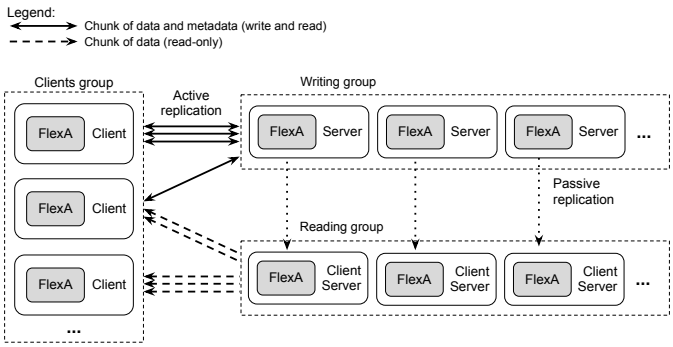

Fig. 1: FlexA architecture

presence of a central manager. Another point is that, due to the access control model, a certification centre is not necessary, since each client is responsible for controlling the access permissions to its own files, without the intervenience of managers or super-users.

In this model, computers belonging to the DFS are administered in specific groups, which can be of three types: reading, writing and clients. The first, called writing group or primary servers, comprises computers with active server process, and it is responsible to manage and store files with their metadata. The second group, called replicas or secondary servers, consists of computers that can be clients and servers at the same time, creating a backup set of primary servers for assistance in case of overload or failures. The third group is formed by the client workstations, which are responsible for encryption and distribution chunks of the file to the primary servers.

Files can be read in parallel from several storage computers, either from the write or the replica group. The transfer of new or update files can only be performed by computers in the write group.

Network latency is minimized by the elimination of intermediary servers and the addition of concurrent access to different sets of servers. These characteristics also allow the prevention of possible bottlenecks in the transfer process.

As stated before, the migration of most of the operations to the client side, including cryptography and file partitioning, also provides gains in the model's performance. Indeed, this also reduces the usage of hardware resources in the storage nodes.

4.2 Security

The access to files is determined by the client's handler, which can be read-write or read-only. Each handler has two keys: one cryptographic and other for validation.

The "write" cryptographic key (WK) is given by *WK = SHA256Trunc(Key)*, where *Key* is a 16 bytes sequence randomly generated, and the function *SHA256Trunc* is the hash 256 bits (32 bytes) with output encoded in Base64 and truncated in 32 bytes. The "read" cryptographic key (RK) is given by *RK=sha256Trunc(WK)*. Each file is encrypted using its RK and the symmetric AES 256 bits algorithm in the CBC (*Cipher Block Chaining*) mode [18]. This allows modifying a *WK* to a *RK*, but not the other way round. These steps are illustrated in Figure 2.

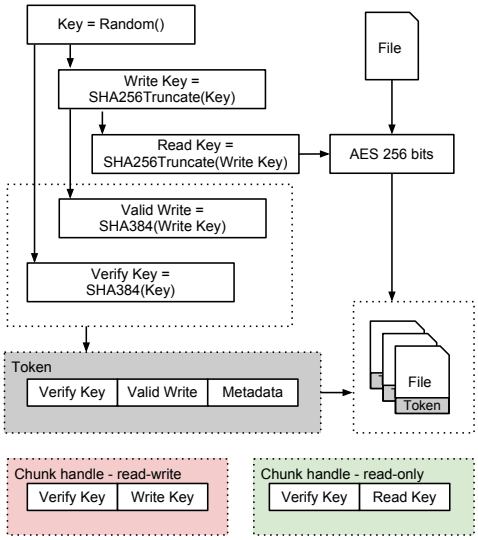

Fig. 2: FlexA security

Two other keys determine the validation process: *Verify Key* (*VK*) and *Valid Write* (*VW*), both in 48 bytes. The *VK* is present in the client handler and in the token sent to the servers. It is used to certify the communication between nodes and it is generated by *VK = SHA384(Key)*. The *VW* is also generated by the client, but only the servers have its copy. It is used to verify the validity of the client's *WK*. This key is given by *VW = SHA384(WK)*. If the result from the client's *VW* is equal to the stored chunk's *VW*, the file modification is executed in the write group.

File decryption can be performed only through the keys generated by the client. This is true independently of the communication channel in use or the server security level. This process produces three chunks, each one with one token containing file attributes, *VK* and *VW* and the handlers that allow the user to change the permissions.

4.3 Client

Three separated modules make up the client process, as shown in Figure 3: *Collector* manages the data input, *Synchronizer* manages outward data transmission, and *Communicator* identifies which hosts are active and which are in the replica group or are clients.

In each client there is a reduced part of the storage server database. This part contains some information about the availability and locality of file chunks. This local database helps for a fast search/recovery of data, without the need of performing these operations in the storage servers.

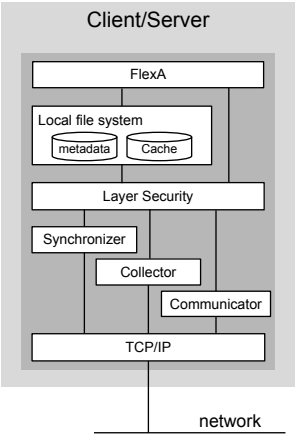

Fig. 3: FlexA structure

4.4 Storage

The servers, in the DFS model proposed here, have the function of store file chunks, validate client requests and execute replication between the groups of servers. Since there is no specific server in charge of the communication management or acting as a central unit, all servers follow architecture close to the client's model.

The chunk organization is performed through the local database, which provides the file properties with its routing table and keys for verification and validation.

4.5 Communication

Communication between the components of FlexA occurs with TCP/IP protocol and persistent connections. Each file chunk is transferred this way.

The *Communicator* module scans the network searching for active hosts and looking at what functions they execute. This operation enables the interaction between clients and storage groups. Periodically, the identified hosts exchange messages in order to keep their status updated.

The user performs the choice between a client or storage station, whilst the definition of the write servers group is made independently. The independent definition of write servers guarantees that the needed number of storage servers is satisfied.

In case of a failure in the servers of the write group, it is possible to relocate stations from the replica group through an election process. Specifically, a Bully algorithm is used to determine a new candidate to the write group through the priority of each member of the replica group. Once the new station is defined, all active process in FlexA are informed of the new member [19].

4.6 Synchronization

Chunk synchronization occurs automatically through client interactions with the write group. These interactions are propagated to the replica group and the active clients are informed about new updates.

Indexers present in each local database are used to identify if the copies in cache are obsolete in relation to the write group. The update of these copies, however, is only performed when a user tries to use them.

Since each station keeps a record of the status of every other station, services can be re-established quickly in case of failures or a host crash.

The consistency of the chunks is determined by its version in the database on each station, which is controlled by write group. All clients and the read group are notified about every operation that modifies a file in a server. Periodically, stations communicate with the write group to update its data base.

The entry of a new station needs a token from the write group, that token contains information about the actual state of all stations.

5. Performance evaluation

5.1 Micro-benchmarks

Performance evaluation used a set of computers with Intel Pentium Dual E2160 - 1,8 GHz processor, 2GB of RAM memory, hard disk of 40GB at 7200RPM, Ubuntu Linux operating system with 64-bit and interconnected by Ethernet 100 Mbps full-duplex using a 3Com switch. To do the performance tests, we used operations of reading and writing (upload/download) of files of 1MB, 5MB, 10MB, 25MB and 50MB. These files seeking include the variation of data found in the academic environment, which can be text files, music, photos, videos and applications.

5.2 Evaluation

The evaluation process considered five cases of access to the servers. Each one contains sequential access of writing and reading for each one the clients simultaneously.

For each test scenario, levels of interactions were used 12 times for each size in each read and write operation, totalising 120 interactions for each client. In tests with more than 1 client, simultaneous interactions followed the same read/write sequences with same sized files.

For evaluation and comparison of performance we used Tahoe-LAFS, because it is precursor of FlexA, and NFS because it is a client-server DFS without additional layers.

For the comparison between DFSs, the tests were considered with 5 concurrent clients using 5 types of files to read and write operation.

For the write operation, as shown in Figure 4, it is possible to verify that the NFS in environment with multiple requests becomes slower, affecting the overall system performance. Even in an environment small-scale, centralized architectures may represent a factor of performance degradation and risks of failure.

The write operation using FlexA and Tahoe-LAFS has similarity, because the logic of this operation (encryption,

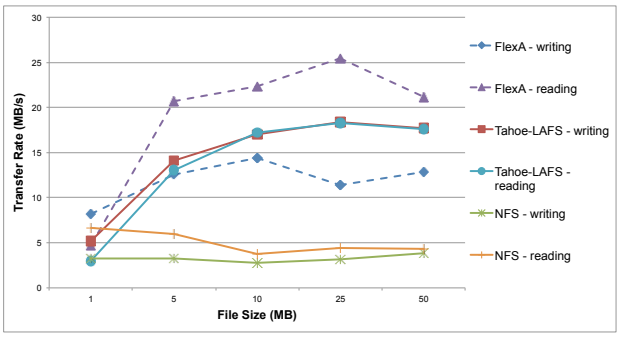

Fig. 4: Rate of reading and writing files

split and distribution) is similar, differing in some aspects such as the encryption without using RSA asymmetric keys in FlexA, in the splitting the file in 66% compared with 50% for Tahoe-LAFS and distribution using TCP/IP compared to the HTTP in Tahoe-LAFS. However, the write operation using Tahoe-LAFS is faster than FlexA during the evolution of files, being less optimized for files smaller than 4MB, to which FlexA is slightly faster than Tahoe-LAFS (8MB/s in FlexA and 5MB/s in Tahoe-LAFS). The reason for FlexA be significantly slower than Tahoe-LAFS for writing files over 5MB is justified by the large number of divisions and distributions of chunks of files that FlexA does for set of three servers, totaling 66% of the file for each computer against 50% of a file in Tahoe-LAFS.

Considering the read operation, showed in Figure 4, FlexA has the highest rate of evolution for the transfer. Because of its number of divisions of the file and the process of choice for distributed chunks, the read operation in FlexA can uses all the available servers of the writing group. This situation does not happen with the Tahoe-LAFS which is limited to a few servers to restore the file again.

The fact that the client station is responsible for data processing (division and cryptography) causes a small part of the consumption of the hardware resources compared with NFS, but much smaller compared to Tahoe-LAFS, as is shown in Figure 5.

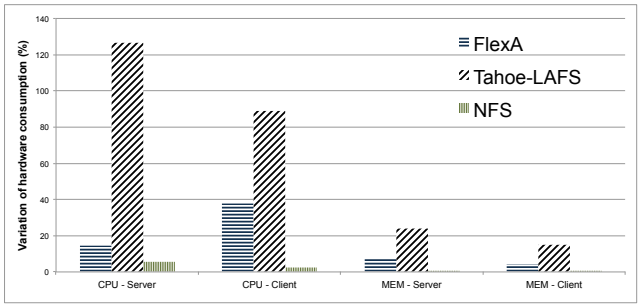

Fig. 5: Variation of hardware consumption

The advantage of our DFS model is to permit a larger quantity of simultaneous operations to big client files, while the storage server only concentrates on receiving and organizing the chunks.

Another point implemented in our model is the direct replication by the client, allowing more control and security for the user if failures occur in some servers.

6. Conclusion

The developed model is a way of expressing the possibility of aggregating the characteristics of others DFS and making it functional.

The simplicity of its construction brought flexibility to the utilization of computational resources, a more efficient use of the client's processing power and less conditions for an overload in the servers.

The adaptation of the model of decentralized permissions allowed more independence for system administrators and a quicker cryptography process. This is due to the smaller number of steps that are used to achieve an acceptable security level.

It brought more data reliability with the use of cryptography directly on the files, avoiding cases where security failure or fault of the storage set compromises data integrity.

The use of storage groups guarantees availability of the files and presents conditions for fault tolerance by distributing data among several nodes.

FlexA model was constructed to create an environment favouring the client, to provide a DFS that is easier to implant and use in normal conditions, and offer some of characteristics that are found in most file systems.

In this evaluation, focusing only on the technical upload/download the files, there is a similarity between FlexA and Tahoe-LAFS. FlexA has better read rate than Tahoel-LAFS, but Tahoe-LAFS has the best writing performance.

The best performance in FlexA in read operation is explained by the division factor of the files in order to simultaneously use 3 servers, which therefore creates greater consumption in the writing process than Tahoe-LAFS.

At this stage, this research work presents the components that are needed for an operational use in a controlled environment. Among the next steps of the research are the enlargement of the model's functionality, aiming at providing a more convenient way of interacting with the operating system, support for efficient chunk compression, optimize cryptography, support mobile devices, and other characteristics.

Acknowledgment

The authors would like to thank Brazilians foundations FAPESP 2012/02926-5, Fundunesp and CAPES for their financial support.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, pp. 29–43.

[2] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. Addison-Wesley, 2005.

[3] S. D. Pate, *Unix Filesystems: Evolution, Design, and Implementation*. Wiley Publishing, 2003.

[4] H. H. Huang and A. S. Grimshaw, "Design, implementation and evaluation of a virtual storage system," *Concurr. Comput. : Pract. Exper.*, vol. 23, pp. 311–331, March 2011.

[5] A. Batsakis and R. Burns, "Cluster delegation: high-performance, fault-tolerant data sharing in nfs," in *Proceedings of the High Performance Distributed Computing, HPDC-14*. 14th IEEE International Symposium, 2005, pp. 100–109.

[6] S. Shepler, "Nfs version 4 design considerations, rfc 2624," Tech. Rep., 1999.

[7] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network file system (nfs) version 4 protocol, rfc 3530," Tech. Rep., 2003.

[8] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, ser. StorageSS '08.ACM, 2008, pp. 21–26.

[9] M. Nelson, B. Welch, and J. Ousterhout, "Caching in the sprite network file system," *SIGOPS Oper. Syst. Rev.*, vol. 21, pp. 3–4, 1987.

[10] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Trans. Comput. Syst.*, vol. 10, pp. 3–25, 1992.

[11] M. R. Barrios, J. Barkes, F. Cougard, P. G. Crumley, D. Marin, H. Reddy, and T. Thitayanun, "Gpfs: A parallel file system." IBM Cor., Tech. Rep., 1998. [Online]. Available: http://www.redbooks.ibm.com/

[12] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06, 2006, pp. 307–320.

[13] F. Hupfeld, T. Cortes, B. Kolbeck, E. Focht, M. Hess, J. Malo, J. Marti, J. Stender, and E. Cesario, "Xtreemfs - a case for object-based file systems in grids," *Concurrency and Computation: Practice and Experience*, vol. 20, 2008.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010.

[15] RedHat, "Red hat global file system," 2011. [Online]. Available: http://www.redhat.com/software/rha/gfs/

[16] Gluster, "Glusterfs," 2011. [Online]. Available: http://www.gluster.org

[17] A. H. Karp, "Enforce pola on processes to control viruses," *Commun. ACM*, vol. 46, no. 12, pp. 27–29, 2003.

[18] M. Dworkin, "Special publication 800-38a: Recommendation for block cipher modes of operation: Methods and techniques," NIST - National Institute of Standards and Technology, Tech. Rep., 2001.

[19] A. S. Tanenbaum and M. V. Steen, *Distributed systems: principles and paradigms*. Pearson Prentice Hall, 2007.