Availability in the Flexible and Adaptable Distributed File System

Danilo C. M. Segura
Matheus D. C. Oliveira
Thiago K. Okada
Renata S. Lobato
Aleardo Manacero
Computer Science and Statistics
Paulista State University - UNESP
Rio Preto, Brazil
Email: aleardo@ibilce.unesp.br

Lúcio R. Carvalho
Federal Technical Institute
Catanduva, Brazil
Email: luciorodrigocarvalho@gmail.com

Roberta Spolon
Computing Dept
Paulista State University - UNESP
Bauru, Brazil
Email: roberta@fc.unesp.br

***Abstract*—The goals of a Distributed File Systems (DFS) may vary broadly. It is impossible to design a DFS attaining every desirable characteristic, such as, transparency, performance, privacy, reliability, and availability, for example. In this paper we describe the improvements achieved with the availability and performance offered by a DFS named FlexA (*Flex*ible and *A*daptable Distributed File System), which already proposed an architecture that could provide data security and flexibility. Modifications included a new approach to provide file replication and procedures to prevent system overloads. Details about the modifications introduced to FlexA, as well as results achieved with them, are provided. These results indicate that FlexA can be an important option among the known DFS.**

I. Introduction

Current developments in computer applications led to an increase in the demand for data storage and file management. This growth has been accompanied by a large use of networks and distributed systems, either in small or large scale. The efficient use of distributed systems is linked to how files are managed by their associated distributed file systems.

Distributed File Systems (DFS) have been explored since the 80s, with the introduction of NFS. Unfortunately, the wide range of goals aimed by these systems, such as scalability, fault-tolerance and simplicity, imply that most of them can provide just few of these goals.

Here we present improvements in an user-level DFS, named FlexA (from **Flex**ible and **A**daptable DFS) [1]. FlexA provides scalability, fault-tolerance, and privacy, even when implemented with low-cost hardware, being easily configured and maintained. The improvements presented here are concerned with the system's availability, through server's recovery and overload avoidance, and performance, by load balancing.

In the remaining text we present a description of similar efforts, an overview of FlexA and the description of the proposed improvements. Results achieved are also presented and allow us to claim that FlexA is an interesting alternative as an user-level distributed file system.

II. Related work

Several distributed file systems have been proposed along the years. We will not discuss many of these proposals due to two main reasons: lack of further application/usage and lack of strict addressing issues related to availability. In the next few lines we describe the most representative DFSs that deal with availability, briefly identifying how they approach this issue.

a) Google File System: GFS[1] operates on an architecture composed by one server (master), parallel server clusters (chunk server) and clients. The master serializes the requests, which are served, without further accesses to the master, by the chunk servers. In this architecture, the availability is provided through the replication of all chunks, although the master may represent a serious constraint on this [2], [3].

b) Hadoop File System - HDFS: HDFS was introduced to improve the reliability of distributed systems through file replication [4]. It uses a single node

[1]Not to be confused with Red Hat's GFS (Global File System), which has several drawbacks.

(*Namenode*) to manage files across a cluster of servers. Files are replicated in a structure aimed to reduce latency, although its files are usually large (Gbytes). The major problem with HDFS is the possibility of Namenode's failure, when a whole cluster would become unavailable. Current versions try to overcome this allowing for a passive mirror of the cluster's Namenode.

c) Andrew File System: AFS, which was developed by IBM and Carnegie Mellon University, works with two distinct processes: the server, called Vice and the client, known as Venus. To provide the availability, this system offers a specific cache in the client, which reserves a space in disc for the files [5]–[7].

d) Deceit: Similar to NFS, using a client-server model. This DFS replicates files in another servers (named secondary servers), aiming to ensure availability [8].

e) Lustre: This system separates metadata management from the input and output service. There are three essential elements in its architecture: clients (called Object Storage Client), servers that storage objects (Object Storage Servers) and servers that storage metadata (Metadata Services). A copy of the metadata is sent to other servers to ensure availability, with other servers continuing to work in case of client failure [9]–[11].

f) Tahoe-LAFS: The architecture of Tahoe-LAFS uses three types of nodes: clients, storage servers and a central component called introducer. The availability of files in Tahoe is ensured through a mechanism of file division, where each file is split among the storage servers (usually at least 3 servers capable to recover a file). One interesting characteristic of this DFS is that all the treatment of integrity and privacy are restricted to client nodes [12].

III. THE FLEXA DFS

Our work is based in the FlexA (**Flex**ible and **A**daptable distributed file system) [1], which is an attempt to incorporate in a single system all the important characteristics of NFS, AFS, GFS and Tahoe-LAFS. It eliminates the need for a main server (such as the master in GFS or the introducer in Tahoe) through a peer-to-peer architecture, depicted in Figure 1, where client nodes interact directly with the storage servers. File storage is provided by two group of servers, a *reading group*, renamed as replica or secondary servers, where only data is found, and a *writing group*, renamed as primary servers, where data and metadata are stored. Availability is provided in FlexA through a mechanism similar to the one presented by Tahoe-LAFS, that is, replicating chunks of each file through few servers.

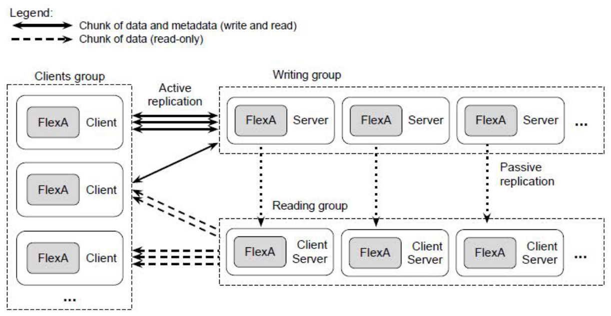

Figure 1. FlexA's architecture.

The operation of FlexA, including how files are distributed among servers, is based in the Client nodes. To read a file a client requests it and receives information about where the file chunks are stored from the primary servers (writing group). The client can retrieve the chunks either from primary servers or from secondary servers (reading group). It must be noted that a client may become a secondary server to some files.

To save a file, however, a client can only upload file chunks to the primary servers, not being allowed to do this directly to secondary servers (that is why they were called reading group). This operation consists in splitting the file into three chunks and copying two of them to each of the primary servers (Figure 2). Therefore, each file can be reassembled if at least two servers can be reached.

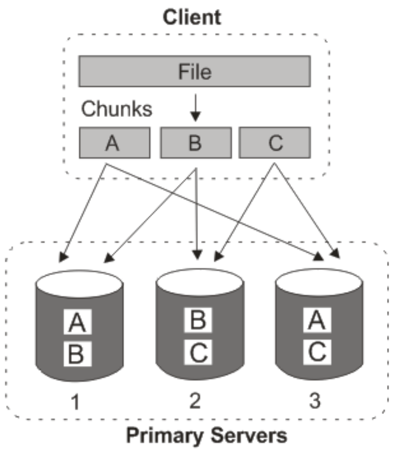

Figure 2. File distribution scheme in FlexA.

IV. Improving FlexA's availability and performance

Besides providing availability through the chunks mechanism, the original proposal of FlexA was dependent of primary servers. Availability was restricted to pieces of file chunks present in the primary servers, since the "reading group" was not fully implemented. With this design a file could be retrieved even if one primary server was down. In the event that two of the primaries crashed during the same time interval, the files addressed by such servers could not be retrieved.

To circumvent this dependance we modified FlexA's architectural model, where availability is mainly provided by a group of replica servers. In this model, the primary servers become mostly metadata servers, with each primary storing only one file chunk of each file. The replica (secondary) servers stored copies of chunks present in the primaries, allowing a broader reachability for these chunks. This enables file retrieval even when two primary servers are down, since a single server could address chunks in the replica servers through the metadata stored in it.

The operation of the adapted FlexA is based in three major modules (Figure 3): Collector (to receive requests), Synchronizer (to keep file coherence) and Communicator (to actually transfer data). To improve the FlexA's fault tolerance we modified how these modules interact. This means modifications in several management procedures including:

- Turning the primary servers mostly metadata servers;
- Turning the reading group into replica servers;
- Storing only one copy of each file chunk in the primary/metadata servers;
- Storing at least two copies of each file chunk in different replica servers;
- Creating a primary server replacement mechanism to temporarily turn a replica server into a primary (replacing a crashed one);
- Electing a replica server as an additional primary server to avoid overloading primaries;
- Using a load balance factor to distribute replicas as well as elect a replacement (when necessary).

These modifications are described in the following sections.

A. *Upload process*

In its new architecture FlexA distributes files in a different fashion. Firstly, a file chunk is now written

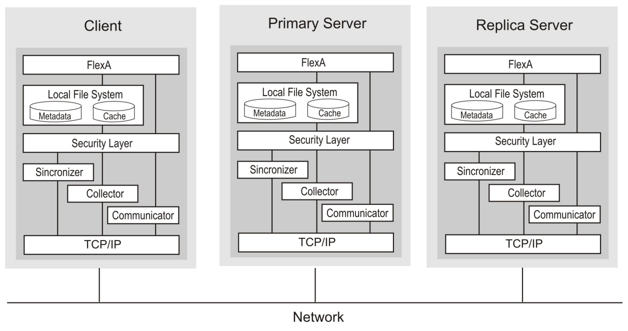

Figure 3. Client and Primary/Replica Server modules.

to a single primary server, instead of two, with each chunk going to a different server. Each primary replicates its chunk to some replica servers, at the same time it synchronizes the metadata with the other primaries. This modification allows for a faster save operation, cutting by half the number of transfers needed before the file can be safely considered saved. These operations involve:

1) Encrypting the file using user's cipher key;
2) Splitting the file in three chunks, if it is larger than 10Mbytes;
3) Retrieving a list of active primary servers;
4) Sending each resulting chunk to three different primary servers (this demands at least three active primary servers);
5) Having each primary server to choose at two replica servers to send the received chunk;
6) Having the primary servers to synchronize themselves in order to update their metadata tables.

Operations 1 through 4 are performed by the client node, reducing the amount of work done in the servers. As one can see, the major constraint is to have at least three active primary servers during step 3. It is assumed here that the whole system will always have more than 3 hosts, providing enough hosts to the client. In the event that momentary crashes reduce the number of active primary servers to less than 3 machines, the upload process will delay until an election mechanism designates a new primary.

The whole procedure is exemplified in Figure 4. In this figure, a file is split in chunks A, B and C; the primary server 1, after receiving chunk A, sends it to replica servers 1 and 3. The primary server 2 sends its chunk to replica servers 2 and 3, while primary server 3 sends to replica servers 2 and 4.

To prevent that a host gets a large amount of files we used a load balancing scheme to select where the replicated chunks will be stored. In this scheme each primary

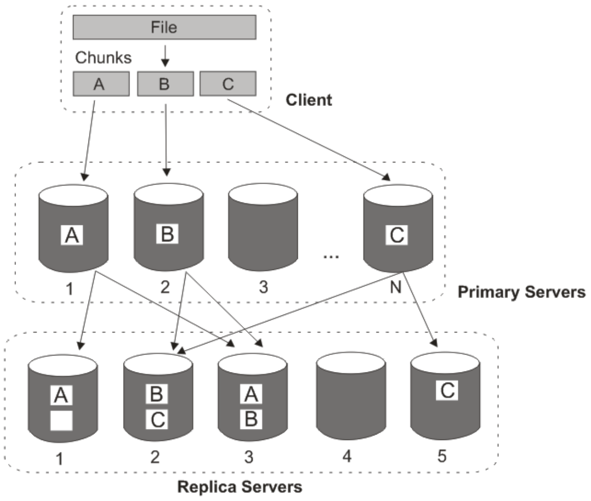

Figure 4. New file distribution in FlexA.

server creates a list of candidates by the application of the equation 1. This equation determines the amount of unused resources that each host has, considering hard disk space (capacity to store the file) and communication channels occupation (capacity to quickly respond for file requests). Each node with an idleness above a given threshold is inserted in the candidates list. When the list is complete, the primary server randomly chooses two hosts to become replica servers for that chunk file. This final step avoids that all primary servers choose the same replica servers.

$$\textit{Idleness} = 1 - \left(\frac{HD}{3} + \frac{DL}{3} + \frac{UL}{3}\right) \tag{1}$$

Where:
$HD = \frac{\textit{Disk space used}}{\textit{Disk capacity}}$ and
$DL = \frac{\textit{Volume of outgoing traffic}}{\textit{Channel capacity}}$ and
$UL = \frac{\textit{Volume of incoming traffic}}{\textit{Channel capacity}}$

Currently, the threshold used to select candidates is set to 0.8 (or 80% of idleness). This threshold is automatically updated if not enough hosts could offer such availability. When this occurs, the threshold is reduced by 0.1 (10%) from the previous value, until a feasible threshold is achieved. Following this policy enables FlexA to balance the load among all servers, always looking for hosts that are more ready to respond.

B. Download process

When a client wants to retrieve a given file, it requests the file's metadata from one of the active primaries. This request implies in the following sequence of events:

1) The primary locates the metadata for that file, including the location (IP addresses) of every chunk stored for that file;
2) The primary verifies the communication channel load for every host storing a file chunk;
3) It sorts the list of hosts based on the communication latency (smaller to higher), trying to balance the network traffic;
4) The primary returns the sorted list to the client;
5) The client can read each file chunk using this list to improve download time.

In order to effectively use the file, the client has to reassemble the chunks and to decipher the file contents. It must be observed here that the ciphering process is used in FlexA as a way to provide authentication. This means that all files are visible to all users, but since they are encrypted only users that possess the file's encryption key are able to read it.

C. Managing availability

The main goal in our work was to improve the system's availability. This was reached attacking three issues: file replication, system's reachability, and server's availability. File replication was already described in the previous sections and is basically performed through the replica servers. System's reachability involves access to hosts and software components and does not demand special techniques to be solved, as explained next. Server's availability, on the other hand, is achieved minimizing either server's downtimes or overloads.

a) Managing reachability: In a distributed system there are several failures that can affect its reachability. Among these failures we considered network disconnections, host crashes and DFS crashes. As expected they demand different reactions from the system.

Firstly, a network disconnection implies that some hosts will become unreachable. The detection and the treatment of such event is left to the network management protocol. FlexA will have to deal only with the treatment of file consistency, which may become a problem when a host gets temporarily disconnected and some of the file chunks stored there get outdated. The treatment of such occurrences is performed at host initialization, where timestamps for each chunk in that host are verified against the timestamps of that chunk in a primary. If the host has an older timestamp, that chunk is updated from the newest copy.

A DFS crash is understood here as when one of the FlexA's components becomes unresponsive. This is recovered by a daemon mechanism that probes the system periodically. When a given component is missing, that daemon restarts it. With this mechanism only local availability may be affected by the failure of a single component, and even this can be fixed shortly.

A host crash has different impacts depending on its type. Clients and replica servers do not demand any special treatment. If a client crashes, its recovery is left to the user. Files that were opened in that client are considered lost and the system keeps their last update as their current versions. If a replica server crashes, the system simply ignores it for file requests and updates the necessary chunks when the host is back to work.

b) Managing downtimes: When a primary server crashes, implying in a downtime, FlexA enters in a state where files may become unavailable if a second server fails. This occurs because primary servers are the hosts in charge of managing the files metadata, what may turn a file unavailable. The recovery from a primary server crash is done in three phases:

1) **Crash detection**, where a peer-probing mechanism detects a primary server's crash by periodically sending, to its peers, a message containing results from previous probing rounds. The content of all messages (sent/received) is compared to find problems. In the case that a message from one server is not received, the field corresponding to its answer is marked as null in the next probing message. If that field remains null during three rounds, then the host is marked as crashed.
2) **Election**, where one of the remaining servers starts an election mechanism by a request to one of replica servers. This host starts the election using the ring algorithm [13] to select a host that has the largest idleness among all replica servers. The idleness is calculated by equation 1. The election mechanism is concluded when the message containing the current best option reaches back the host that started it, which sends a message to the primary that requested the election informing the elected host.
3) **Replacement**, when the elected host is requested to become the new primary. The elected host will create a metadata database and synchronize it with the metadata tables from the other primary servers. After doing the synchronization the new primary will begin to act as primary starting a primary collector module.

c) Managing overloads: A different scenario appears when a server becomes overloaded, what reduces its efficiency to answer to client requests. To solve this FlexA adds an additional server in the primary pool if a primary is considered as overloaded. The metric to calculate the primary server's load is given by equation 2, and considers disk usage (space and access), main memory usage, and network traffic.

$$\mathit{Load} = (DU + DA + MEM + NT) \qquad (2)$$

where DU, DA, MEM and NT are scores given by the occupation of the resources described by Table I.

Table I
OCCUPATION SCORES FOR THE RESOURCES USED TO DEFINE A SERVER'S AVAILABILITY

Resource	Score		
	0-50%	51-75%	76-100%
Disk space used (DU)	0.5	1.0	3.0
Disk access activity (DA)	0.5	1.0	3.0
Memory used (MEM)	0.5	1.0	2.0
Network used (NT)	1.0	1.0	3.0

The whole process involves up to five phases:

1) **Data collection**, performed periodically, currently a 10 minutes period, by each server;
2) **Determination of server's load**, where each server uses equation 2 to determine its score and stores it in a "load index" list. The load is tagged as *normal* if the score is lower than 5.0, or it is tagged as *overloaded* otherwise;
3) **History evaluation**, where the load index list is evaluated and an election process is started if the past three loads were classified as *overloaded*, or returns to the data collection phase otherwise;
4) **Election**, where a secondary server is requested to start the same election procedure used for server crashes, returning the IP of the elected host to the primary server that requested the election;
5) **Server's addition**, where the replica server just elected receives a message to start its Collector module and to synchronize its metadata database to the primary's database in order to become an additional primary server.

V. PERFORMANCE EVALUATION

To evaluate the performance of FlexA, considering both file access time and server recovery, we conducted several tests using a cluster of 16 hosts split in two groups and linked through a 1Gb/s network. The first group (called cluster A) contained hosts running Intel Pentium Dual E2160 processors, with 2 Gbytes of RAM and 40 GBytes of disk capacity. The second group (called cluster B) contained hosts running Intel Core i7-3770 processors, with 16 Gbytes of RAM and 500 Gbytes of hard disk. All machines, including the virtual machines (VM), run Debian Linux 7.1.0.

A. Measuring access time

Access time was measured considering operations of reading and writing (download/upload) files with sizes of 1MB, 5MB, 10MB, 25MB, 50MB, 100MB and 200MB. The choice for such sizes was based on typical content stored in a file system, which includes text files, music, photos, videos and applications. To evaluate its performance we compared FlexA results with Tahoe-LAFS (it was the basis for FlexA) and NFS (simple and widely available). Comparisons against GFS, among other systems were not performed due to its restrictive availability.

The results presented here are for a scenario with 16 clients, distributed over hosts in cluster A (by the virtualization[2] of up to four clients in each host). The servers were distributed over cluster B. Each run was repeated 20 times in order to accommodate statistical variations.

The servers implemented in cluster B presented the following configuration:

– **Original FlexA -** 3 primary servers
– **Adapted FlexA -** 3 primary and 5 replica servers
– **Tahoe-LAFS version 4 -** 1 introducer and 7 storage servers acting as helpers
– **NFS version 1.9.2-1 -** 1 server

The plot in Figure 5 shows the average time spent reading files in these DFS, partially summarized in table II, while attending 16 clients. As one can see, NFS has a higher throughput for smaller files, what is expected due to its simple client-server model. FlexA has a better throughput for files larger than 25 Mbytes, which is a typical size for multimedia data. It is interesting to observe that for files larger than 10 Mbytes, which are split in three chunks, the FlexA's throughput has a noticeable improvement.

The performance of these systems for writing operations is shown in Figure 6, while table III partially summarizes the average time spent to write these files. In this case NFS also performed better for smaller files, although FlexA now has a better throughput even for the 25 Mbytes files. Although FlexA's throughput also decreases for larger files, the rate of reduction is smaller than the other DFS evaluated.

[2]Each VM was configured with one processor, 1Gb of RAM, and 10Gb of disk.

Table II
AVERAGE TIME, IN SECONDS, SPENT TO READ A FILE FOR 16 SIMULTANEOUS CLIENTS.

	1 MB	10 MB	25 MB	100 MB	200 MB
Original FlexA	1,22	1,21	1,45	5,59	23,26
Adapted FlexA	1,12	1,21	1,38	4,57	22,14
NFS	0,14	0,28	0,65	9,35	242,37
Tahoe-LAFS	0,91	25,48	55,27	107,57	1073,42

Table III
AVERAGE TIME, IN SECONDS, SPENT TO WRITE A FILE FOR 16 SIMULTANEOUS CLIENTS.

	1 MB	10 MB	25 MB	100 MB	200 MB
Original FlexA	0,48	1,21	2,50	17,35	50,57
Adapted FlexA	0,17	0,55	1,29	5,99	16,42
NFS	0,12	0,54	1,81	30,39	93,33
Tahoe-LAFS	0,61	1,67	4,79	74,68	556,99

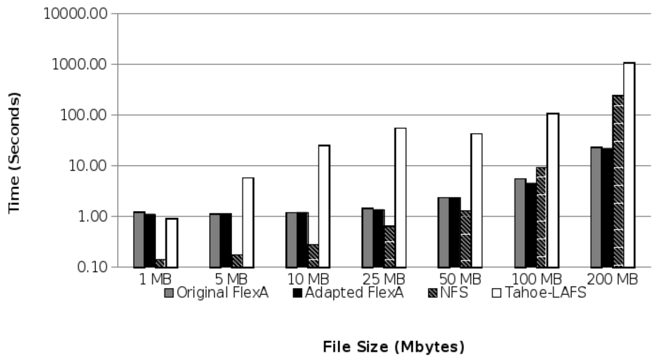

Figure 5. Measured delivery times for reading (download) operations, with 16 clients.

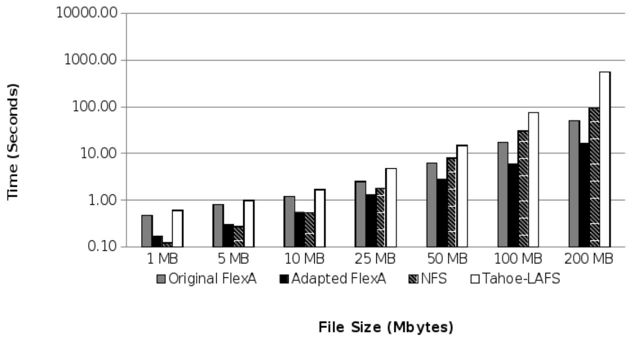

Figure 6. Measured delivery times for writing (upload) operations, with 16 clients.

B. Measuring FlexA's reaction to overloads

A different set of tests involved measuring how FlexA can detect an overloaded primary server and react to this status. The test presented here involved 3 primaries and up to 36 additional hosts, where 4 of them were acting only as replica servers. The primary servers have, respectively, disks with 26%, 43%, and 28% of occupation. Since all the remaining parameters used to calculate the server's load were kept in a low usage, the overload status was dependent on the disk usage.

As shown in Table IV, the tests involved repeated writing operations from 16 and 32 clients, which wrote files with sizes of either 50MB or 100MB. In these scenarios FlexA considered that server 2 was in an overload status only for 32 clients writing 100MB files. In this case the disk occupation reached 76%, making its load score higher than 5.0.

Table IV
REACTION TO SIMULTANEOUS WRITING OPERATIONS WITH PRIMARY SERVERS (P1, P2 AND P3) DISKS INITIALLY WITH 26, 43, 28% OF OCCUPATION RESPECTIVELY

Number of clients	File size	Final % of disk use (P1, P2, P3)	Status Status
16	50 MB	35, 52 and 37	Normal
16	100 MB	44, 59 and 45	Normal
32	50 MB	43, 59 and 45	Normal
32	100 MB	59, **76** and 60	Overload

When the situation shown at line 4 of Table IV is reached the system has to starts the search for a replica server that can acts as primary. In the tests the replica servers (S1, S2, S3 and S4) had, respectively, 57%, 23%, 29% and 28% of disk occupation, resulting in the election of server S2 as the new primary server.

C. Measuring server's recovery

The current version of FlexA has its availability depending on how frequently a server crashes and how quickly it responds with a replacement. While the latter is defined by the system's implementation, the former depends on the hardware and how is its usage. Statistically speaking, distributed systems present failure behavior following certain patterns. From the patterns and parameters presented by Schroeder and Gibson [14] we can expect that a system running current hardware would have around 4 failures per server per year (hardware type D).

As described earlier, files will be unavailable every time that a primary server crashes. This unavailability lasts during the whole recovery process, including detection, election and replacement phases. Therefore, in order to have an estimation for system's availability we measured how long this process takes for different configurations. The results for a configuration with four replica servers are presented in table V, and allow us to conclude that the recovery process is constrained by the detection phase. The averages below were obtained with 200 induced crashes (removing the network cable of a server).

Table V
TIME SPENT IN THE RECOVERY OF PRIMARY SERVERS (IN SECONDS)

	Detection	Election	Replacement	Total
Average	13.00773	0.06935	0.00784	13.01577
Standard Deviation	0.00176	0.00352	0.05466	1.05539

As a remark, tests involving the variation in the number of replica servers allowed to conclude that the election is not a problem, even for a large number of hosts. The time spent in the election process using 32 hosts was only 50ms higher than the fastest measured time (less than 20% above the average) and its growth curve implies that this time would remain under a second even for few hundred servers.

Therefore, if a server crashes the system will take less than 14s in average to recover and have one of the replica servers working as its replacement. This is a very short time if one remembers that it is expected only four crashes per server per year. Combining this result with the fact that FlexA regularly works with three primary servers we have:

Time to replace a server	< 14s
Number of servers	3
Number of server failures/year	4
Time spent recovering from server failures/year	168s
% of time without failures/year	99.999%

This result can be considered excellent. Clients using files stored in FlexA may have access to them almost all the time, supposedly with very few and short downtimes. This is a direct result of replicating the metadata among at least three primary servers, allowing the system to respond if at least one still active.

VI. CONCLUDING REMARKS

The results presented in the previous section show that the modifications made to FlexA, in order to improve its availability and performance were quite successful. Its speed improved from the former version besides the additional processing to forward file's chunks over the replica servers. In another direction, the use of parameters such as storage capacity and network latency as metrics for balancing the file replication and access proved to be advantageous, although this could not be directly measured because the original version did not implement file replication.

Looking at our major goal, that was availability, it is clear that the process to recover from a server crash was highly efficient, besides simple. Our contribution resides in providing a distributed mechanism for failure detection and recovery that is simple, fast and efficient. Even the detection phase, which is the bottleneck of this process, is quite fast. This avoids that the remaining servers get overloaded by a long period with less than three hosts working as metadata providers. The process of replacing the crashed server is also simple and usually fast, since it consists simply of copying the metadata to the elected host.

In the same direction, allowing that a replica server become a primary server during a temporary overload condition also improved availability. The continuous monitoring of the server's load can be done without a significant overhead. Servers have only to collect simple data to calculate a simple load index, electing a secondary host to act as a primary if a sustained overload condition is observed.

Overall, the new FlexA is faster than the version without replica servers and Tahoe-LAFS, although being slower than NFS for smaller files. It should be noted, however, that NFS does not provide a stronger mechanism for file availability (does not provide replicas) and does not work for larger files, which are usual nowadays. This allows us to conclude that FlexA has a good overall performance and provides a strong file availability.

ACKNOWLEDGMENT

The authors would like to thanks FAPESP for the grants that partially allowed this work.

REFERENCES

[1] S. Fernandes, R. Lobato, A. Manacero, R. Spolon, and M. Cavenaghi, "A flexible and adaptable distributed file system," in *Proceedings of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, ser. PDPTA 2013, Las Vegas, NV, USA, 2013, pp. 258–263.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM, 2003, pp. 29–43.

[3] A. Osadzinski, "Gfs: Evolution on fast-forward," *Communications of the ACM*, vol. 53, no. 3, pp. 42–49, 2010.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. of IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

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

[6] J. Williams, E.H., N. Sullivan, J. Rusnak, J. Menges, D. Ogle, R. Floyd, and W. Chung, "The andrew file system on os/2 and sna," in *Proceedings of TRICOMM '91*, 1991, pp. 181–191.

[7] J. H. Howard, "An overview of the andrew file system," in *Proceedings of the Usenix Winter Technical Conference*. Usenix Association, 1988.

[8] A. Siegel, K. Birman, and K. Marzullo, "Deceit: a flexible distributed file system," in *Management of Replicated Data, 1990. Proceedings., Workshop on the*, 1990, pp. 15–17.

[9] S. Jian, L. Zhan-huai, and Z. Xiao, "The performance optimization of lustre file system," in *7th Intl Conf on Computer Science Education (ICCSE)*, 2012, pp. 214–217.

[10] J. Logan and P. Dickens, "Towards an understanding of the performance of mpi-io in lustre file systems," in *IEEE Intl Conf on Cluster Computing*, 2008, pp. 330–335.

[11] W. Yu, R. Noronha, S. Liang, and D. Panda, "Benefits of high speed interconnects to cluster file systems: a case study with lustre," in *20th Intl Parallel and Distributed Processing Symposium, IPDPS*, 2006, pp. 8 pp.–.

[12] 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. New York, NY, USA: ACM, 2008, pp. 21–26.

[13] E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes," *Commun. ACM*, vol. 22, no. 5, pp. 281–283, May 1979.

[14] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks*, ser. DSN '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 249–258.