Fault-Tolerance and Load-Balance Tradeoff in a Distributed Storage System

Estudio de la Interdependencia entre Tolerancia a Fallas y Balance de Carga en un Sistema de Almacenamiento Distribuido

Moisés Quezada Naquid, Ricardo Marcelín Jiménez and Miguel López Guerrero
Department of Electrical Engineering , Universidad Autónoma Metropolitana - Iztapalapa
09340 - México City, Mexico
moises@arte.izt.uam.mx, {calu, milo}@xanum.uam.mx

Article received or September 1, 2008; accepted on September 25, 2009

Abstract. In recent years distributed storage systems have been the object of increasing interest by the research community. They promise improvements on information availability, security and integrity. Nevertheless, at this point in time, there is no a predominant approach, but a wide spectrum of proposals in the literature. In this paper we report our findings with a combination of redundancy techniques intended to simultaneously provide fault tolerance and load balance in a small-scale distributed storage system. Based on our analysis, we provide general guidelines for system designers and developers under similar conditions.
Keywords: IDA, storage schemes, failures, recovery, simulations.

Resumen. En los últimos años los sistemas de almacenamiento distribuido han sido objeto de un gran interés por parte de la comunidad de investigadores. Estos sistemas prometen mejoras en cuanto a integridad, seguridad y disponibilidad de la información. Sin embargo, hasta este momento no existe un enfoque predominante, aunque hay diversas propuestas en la literatura. En este artículo reportamos los resultados de nuestras investigaciones con una combinación de técnicas de redundancia que tienen el propósito de proveer simultáneamente tolerancia a fallas y balance de carga en un sistema de almacenamiento distribuido de pequeña escala. Con base en nuestro análisis proporcionamos líneas directrices generales para diseñadores y desarrolladores de sistemas similares.
Palabras clave: IDA, esquemas de almacenamiento, fallas, recuperación, simulaciones.

1 Introduction

Trends confirm the ever-increasing demand for information services. However, successful deployment of various of such services largely relies on storage resources that must include, among their key features, fault tolerance, privacy and load balance. Thus, storage has become a driving force of research and development and, furthermore, distributed storage appears to be the ideal support for many applications.

A distributed storage system is a collection of interconnected storage devices that contribute with their individual capacities to create an extended system offering improved features. The importance of this emerging technology has been underlined in recent research works [14, 7]. Although the simplest function of such a system is to spread a collection of files across the storage devices attached to a network, desirable attributes of quality must also be incorporated.

In a previous paper [8] we introduced a collection of space-redundancy techniques, called storage schemes. They are intended to support robust recording of subsequent global states in a distributed computation. In the present work we apply these ideas to a storage network in order to achieve load balance. As for fault tolerance, we complement our design with an erasure coding technique, namely the Information Dispersal Algorithm (IDA) due to Rabin [9]. Both approaches have been independently used in the past in a number of applications, but to the authors' best knowledge, there are no studies reporting their combined performance in order to provide distributed storage services. In this paper we report our experiences with this combined framework. In particular, we evaluate the effect of diverse parameters on the system's mean time to failure.

The remaining of this paper is organized as follows. In Section 2, we present an overview of related works. For the sake of clarity, in Section 3 we provide a summary of all relevant concepts. The system description and the simulation framework are presented in Section 4 and Section 5, respectively. We detail assumptions and experiments in Section 6. In Section 7 we discuss how relevant factors are interrelated and how they affect global performance. We end this paper with some conclusions and comments on our current work directions, in Section 8.

2 Related work

In recent years, distributed storage has attracted the attention of several groups. However, in spite of the large amount of proposals found in the literature, only a few of them share common elements with the kind of system described in this work. Furthermore, some systems have been announced, but they are still under development and comparisons are not possible at this point in time. This is the case of the Celeste system which was proposed by Sun Microsystems and is based on peer-to-peer (P2P) networking technologies [3]. Some other systems have been developed with a specific application in mind. This is the case of Bigtable [4], which was tailor-made for Google product support and therefore, related information is scarce. Probably, the Cleversafe [6] project is the most related with our proposal. It is also based on a proprietary version of IDA and was released as free software under the GPL license.

A common feature found in distributed storage is the usage of information redundancy in order to provide fault tolerance and, particularly, integrity. In this context, the simplest strategy consists of file replication. This technique is used, for instance, in PAST [11], which was conceived as a global-scale storage system. Designers considered file replication as the key element to achieve high availability and robustness. Another proposal based on file replication is Farsite [1]. Here, clients contribute with their individual capacities in order to support a collective repository. In contrast, there exist systems like OceanStore [7] or Intermemory [5], where information redundancy is implemented using file fragmentation and error correcting codes. In this approach, a file to be stored is split into several blocks. Afterwards, each block is transformed using a particular coding technique (e.g., Reed-Solomon, network codes and fountain codes). In the last step, the coded blocks are allocated to a given set of storage devices.

The selection of a particular strategy for providing information redundancy has a great impact on operational issues such as cost and management. Replication offers a very simple retrieval mechanism, but it might require an excessive cost on storage space. In contrast, block coding can provide similar levels of availability, using a very limited amount of storage. Nevertheless, tracking all pieces that make up each and every "puzzle" may become a tremendous challenge. Recent studies [10] suggest that when devices offer a long-term stable service, systems might profit from "conservative" block coding. On the other side, systems where devices offer intermittent service should be built on the basis of "aggressive" replication. Intermediate solutions, with combined replication and block-coding techniques, are also suggested in order to facilitate information retrieval and tracking.

We end this section by mentioning that load balance is an issue that has been addressed in the last years, but only for big scale storage systems. Many recent works on P2P networks show that balance is achieved using uniform random allocation. Over the long term, with this simple mechanism, the load statistics of individual devices can be fit to normal distributions [2].

3 Background concepts

Let us start by providing a formal definition of a storage scheme. Let $V=\{c_1,c_2,...,c_v\}$ be a set of v storage devices, from now on called *nodes*. A *storage scheme* is an ordered collection $B=\{B_0,B_1,...,B_{b-1}\}$ of b subsets of V, that is used for distributed storage operation in a cyclic fashion. That is, subsequent storage requests i and $i+1$ are handled by the subsets $B_{i \bmod b}$ and $B_{(i+1) \bmod b}$, respectively. The subsets, that we call *committees*, are created from the set of nodes according to an arbitrarily defined rule. More formally, such a rule can be specified by means of an indicator function $\delta : V \times B \rightarrow \{0,1\}$ so that for $i=1,...,v$ and $j=0,...,b-1$, the function $\delta(c_i, B_j)$ equals 1 if

$c_i \in B_j$ and 0 otherwise. The following observations are in order:

a) A node may belong to one or more committees. Furthermore, if each node belongs to the same number of committees, the summation $\sum_{j=0}^{b-1} \delta(c_i, B_j)$ equals a constant quantity r for all i.

b) Different committees from B may not have the same number of member nodes. However, if all committees have the same number of nodes, the summation $\sum_{i=1}^{v} \delta(c_i, B_j)$ equals a constant quantity k for all j.

Some properties of storage schemes are worth mentioning. A storage scheme is *balanced* if the summations provided in previous observations a) and b) lead to constant values (i.e., parameters k and r simultaneously exist). This is a desirable property for a storage scheme since it guarantees load balance. Another related concept is that of saturation. A storage scheme is *unsaturated* if the number of committees each node belongs to is less than the total number of existing committees. In the case of a balanced system, this means that $r < b$. This property guarantees that no node will be permanently working. Table 1 shows a particular example where 6 nodes make up a storage scheme with 6 committees. Each node is part of $r = 5$ committees and each committee has $k = 5$ elements. Such scheme is both, balanced and unsaturated.

The storage scheme considered in this work is created as follows. Assume we have a set V of v nodes, then B consists of all the subsets of size k, selected from V. Thus, the total number of committees that make up B equals the combinatorial number $\binom{v}{k}$. Let us call this policy "all k out of v", or *kv* for short. Notice that any node is part of $r = \binom{v-1}{k-1}$ committees, but does not participate in $\binom{v-1}{k}$ committees. Therefore, *kv* is a balanced and unsaturated storage scheme. The resulting set of committees is scheduled to work in a round-robin fashion. In paper [8] this approach was used to store subsequent global states in a distributed computation. Active components were also assumed to work as storage nodes. During the *i*-th storage step, each component takes a synchronized "snapshot" of its local state. Then it forwards its information to some node in $B_{i \bmod b}$. Since *kv* is unsaturated, it is guaranteed that, when a component crashes, there is at least one committee that remains unaffected. Thus, the computation can be resumed from the last global state recorded in the unaffected committee.

Table 1. The "all 5 out of 6" storage scheme

	C₁	*C₂*	*C₃*	*C₄*	*C₅*	*C₆*
B₀	✓	✓	✓	✓	✓	
B₁	✓	✓	✓	✓		✓
B₂	✓	✓	✓		✓	✓
B₃	✓	✓		✓	✓	✓
B₄	✓		✓	✓	✓	✓
B₅		✓	✓	✓	✓	✓

The idea of a balanced system is intended to reflect a fair load assignment. Meanwhile, the idea of an unsaturated system introduces a redundant collection of committees. We call *space redundancy* an excess of physical devices either organized as redundant parts, or as spare parts.

Besides storage schemes and space redundancy, this work proposes to use the Information Dispersal Algorithm (IDA) in order to provide information integrity [9]. Let F be a file to be stored in a *kv* storage scheme. By using IDA, F is transformed into k files, called *dispersals*, each of size $|F|/m$, where $k > m > 1$. Dispersals are sent to the next scheduled committee so that each node in the committee receives one of the k dispersals. From the algorithm properties it is granted that if up to $k - m$ dispersals are lost, the original file can be reconstructed from the m surviving dispersals. It is worth mentioning that not only can the file F be recovered, but also any missing dispersal can be recovered, provided that m dispersals remain available.

A very important matter that we address in this work is the quest of what we could call a "good" combination of IDA parameters (k, m). Let us recall that a given combination, say $(5,3)$, means that an initial file F is transformed into 5 dispersals and can be recovered from any 3 of them. It also means that each dispersal is $1/3$ the size of F and, therefore, there is an excess of information, or information redundancy, equal to $2/3$.

Table 2 shows all possible combinations of IDA parameters for $k = 2,...,7$ and $m = 1,...,6$. Each entry represents a 2-tuple (k, m), with its corresponding redundancy $(k-m)/k$. If we read by rows, the rightmost entry represents the combination supporting the biggest number of losses or missing dispersals, for fixed k. Nevertheless, the price to pay is the excess of redundant information which, in the extreme case $(k,1)$, implies that each dispersal is actually a replica of F. If we read by columns instead, each column represents all the possibilities that may withstand the same number of losses, i.e. the *i*-th column (from left to right) describes all combinations that support up to i losses. It is clear that a given entry supports the same number of losses that any lower entry in the same column, but at the price of a higher redundancy.
We will focus our work on systems featured by the second column, which means that we parameterize IDA in order to support up to 2 losses.

Table 2. IDA parameter combinations and their redundancy levels

(2,1) 1					
(3,2) 1/2	(3,1) 2				
(4,3) 1/3	(4,2) 2/2	(4,1) 3			
(5,4) 1/4	(5,3) 2/3	(5,2) 3/2			
(6,5) 1/5	(6,4) 2/4	(6,3) 3/3	(6,2) 4/2	(6,1) 5	
(7,6) 1/6	(7,5) 2/5	(7,4) 3/4	(7,3) 4/3	(7,2) 5/2	(7,1) 6

4 System description

In this work we consider a low-cost fault-tolerant distributed storage system. The structure of this proposal is illustrated in Fig. 1. It consists of the following components:

- Clients: These are the users that generate service requests such as storage, retrieval and deletion, using standard communication procedures.

- Network gateway: This component serves as the interface between the clients and the storage network. All requests arrive at the gateway which, in turn, dispatches appropriate information to other system components in order to fulfill a service.

- Storage network: It is a collection of $v+s$ storage nodes, connected through a Fast Ethernet switch. Initially, *v* nodes are configured to work as *active nodes* organized according to a *kv*+IDA storage scheme. Actual storage takes place in these components. The remaining *s* nodes are *spares* that do not take part in any ordinary storage operation. They are intended to replace an active node when it fails.

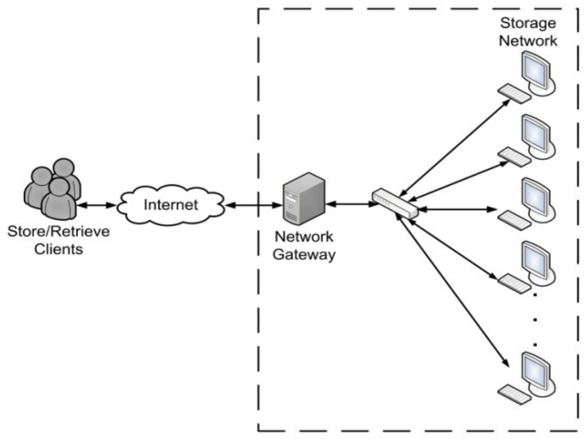

Fig. 1. System architecture of the distributed storage network

In addition to the already-mentioned client operations, the system has to carry out management and recovery procedures. Management is related to the support of client's accounts. Recovery is triggered when an active node crashes. In the latter case a spare node is immediately selected in order to replace the one that failed. Let us recall that each active node stores the dispersals of the corresponding committees it belongs to. Thus, all pertinent information from other active nodes has to be retrieved in order to fully reconstruct the information previously available in the node to be replaced. We also consider that a repair procedure is in effect. It is possible to repair a node that crashed to become fully operational again. A repaired component returns to the system as a spare node. While recovery is in process, client services and management are temporally put off.

In spite of the recovery procedure, there are two possible conditions that may lead to a system break-down or *collapse*. First, when the spare node in charge of recovery is unable to retrieve the minimum number of dispersals from a committee. Second, when another active nodes crashes, but there are no more available spares.

5 Simulation framework

We want to estimate the *system's mean time to failure* (MTTF), this is the expected time that the system takes to go from start to collapse. Clearly, this characteristic depends on several parameters including the underlying *kv* scheme features, the IDA parameters, the number of spare nodes, and also individual node's lifetime and repair time.

In order to carry out our assessment, we developed a discrete-event simulation model. Our simulations are based on two types of entities: ordinary nodes and a supernode. We provide Figure 2 and Figure 3 to show the state-machine pseudo code of the corresponding entities.

An *ordinary node*, or node for short, represents either an active or spare node. When simulation starts, v nodes are selected to work as active nodes whereas the remaining s become spare nodes. In either case individual lifetimes are modelled using random variables. When a node fails, it is taken out of the system to be repaired. Repairing time is also assumed to follow another exponential random distribution. Once the repair finishes, the repaired node becomes part of the spare stock. When an active node fails, a spare node is selected and is given the task of recovering the information in the node that failed. It is important to note that, during this operation, the selected spare node may also crash. Furthermore, it might be the case that failures lead to a shortage of spare nodes. As we described before, there are two possible conditions to declare collapse: either, a dispersal reconstruction is impossible, or a required spare is unavailable.

The *supernode* is a theoretical resource that simplifies our description and models the (otherwise) distributed control. Its operations include node's initialization, *kv* scheme installation, creation of committees, monitoring, failure detection, trigger and supervision of the recovery procedures. The supernode distributes the recovery procedures among available active nodes. Once each one of them has finished the task it was in charge, the supernode selects a spare node to receive and store the recovered information. Finally, the supernode notifies to all components of the system, when a collapse occurs.

6 Assumptions and experiment design

Based on the description previously given, we programmed a set of experiments using the OMNeT++ [13] simulation tool. We considered the following assumptions: *i*) the system is working at its maximum capacity, i.e. each node has a total capacity of 100 GB and this amount is available for storage, *ii*) the time to recover the contents of a node is linearly dependent on its capacity, *iii*) transmission time is negligible compared to the processing time (see Ref. [12] for a thorough discussion on this issue), *iv*) the lifetime of any ordinary node is modelled by independent and identically distributed (i.i.d.) exponential random variables and, *v*) repair times are also represented by i.i.d. exponential random variables.

We recognize that there might be other sources of failure, such as faulty power supplies or other electronic components that may shorten the system's MTTF. Nevertheless, we only focus on aspects that may affect information integrity, according to our definition of system collapse.

Regarding the underlying communications network, we assume that although communication failures may affect service availability, information integrity is always preserved. This is a realistic assumption due to the error control procedures commonly performed by logical-link and transport layers of the protocol stack. They ensure error-free data transfer and, therefore, network failures do not affect data integrity.

For a fixed committee size k equals to 5, our experiment design included 4 parameters: a) initial number of active nodes, b) initial number of spares, c) node's lifetime and d) node's repair time. Each parameter was tested under 3 different levels, which means that we performed 81 different experiments. Each experiment was repeated 10,000 times with different initial conditions (i.e., seeds for random number generation). All results are reported with a 99% confidence interval.

```
state N0
<1>   upon reception of INIT_ACTIVE/INIT_SPARE from programmer
<2>         lifetime = random_function1(lambda1);
<3>         send CRASH to myself in lifetime
<4>         state = N1/N2
state N1
<5>   upon reception of HELP_RECOVER(j,B) from super
<6>         for each node in committee B
<7>               send NEED_DISPERSALS(j,B) now
<8>         committee_members[B]=0
<9>         available_dispersals[B]=0
<10>  upon reception of NEED_DISPERSALS(j,B) from l
<11>        send DISPERSALS(j,B) to l now (?)
<12>  upon reception of DISPERSALS(j,B) from l
<13>        committee_members[B]++
<14>        available_dispersals[B]++
<15>        if (committee_members[B]==k)
<16>        then  if ( available_dispersals[B] ≥ m)
<17>              then  send RECOVERED(j,B) to super now
<18>              else  send UNRECOVERED(j,B) to super now
<19>  upon reception of DOWN from l
<20>        committee_members[B]++
<21>        if (committee_members[B]==k)
<22>        then  if ( available_dispersals[B] ≥ m)
<23>              then  send RECOVERED(j,B) to super now
<24>              else  send UNRECOVERED(j,B) to super now
state N1/N2
<25>  upon reception of CRASH from myself
<26>        repairtime = random_function2(lambda2);
<27>        send CRASH to super now
<28>        send REPAIRED to myself in repairtime
<29>        state = N3
state N2
<30>  upon reception of CONTENTS(j) from super
<31>        myself replaces j
<32>        state = N1
state N3
<33>  upon reception of REPAIRED from myself
<34>        lifetime = random_function1(lambda1);
<35>        send REPAIRED to super now
<36>        state = N2
<37>  upon reception of NEED_DISPERSALS(j,B) from l
<38>        send DOWN to l now
```

Fig. 2. State machine description of an ORDINARY NODE

```
state S0
< 1>    upon reception of START from programmer
< 2>        select v nodes to be in Actives
< 3>        select s nodes to be in Spares
< 4>        send INIT_ACTIVE to each node in Actives now
< 5>        send INIT_SPARE to each node in Spares now
< 6>        start clock
< 7>        state = S1
state S1
< 8>    upon reception of CRASH from j
< 9>        if j in Actives
<10>        then    dismiss j from Actives
<11>                for each committee B where j worked
<12>                    select() a coordinator node in Actives
<13>                    send HELP_RECOVER(j,B) to coordinator now
<14>                recover[j]=0
<15>        else    dismiss j from Spares
<16>    upon reception of RECOVERED(j,B) from l
<17>        recover[j]++
<18>        save B in C[j]
<19>        if (recover[j]==number of committees where j worked)
<20>        then    if (j'=find_spare())
<21>                then    send CONTENTS(j, C[j]) to j' now (?)
<22>                        insert j' in Actives
<23>                        j' replaces j
<24>                else    send COLLAPSE to each node
<25>                        send REPORT to super now
<26>                        state = S2
<27>    upon reception of UNRECOVERED(j,B) from l
<28>        send COLLAPSE to each node
<29>        send REPORT to super
<30>        state = S2
<31>    upon reception of REPAIRED from l
<32>        insert l in Spares
state S2
<33>    upon reception of REPORT from super
<34>        report elapsed time
<35>        stop
```

Fig. 3. State machine description of the SUPERNOD

7 Results and analysis

Figure 4 shows the results of an experiment that we identify as case “a”. The figure shows the corresponding histogram obtained from sample data and we compare it to a fitted exponential *pdf* in solid line. The mean value equals 4,981.60 years, the longest MTTF obtained in our tests. In contrast, Fig. 5 shows the results of another experiment that we call case “b”. Again, we show the histogram and a fitted exponential *pdf* in solid line. This time, the mean value equals 8.85 years, the shortest MTTF recorded. As it can be seen, the exponential distribution is a good fit for the system’s MTTF.

Table 3 summarizes the mean-time-to-failure values for each experiment carried out in this work. It is divided in 3 main vertical and 3 main horizontal sections, corresponding to the values of active and spare nodes under testing, respectively. Each of the 9 possible combinations of active and spare nodes is further divided in order to represent the full set of parameters including node’s lifetime and repair time. Results are shown in bold face type. Those with labels “a” and “b” represent the results of the cases already introduced. We will now proceed to systematically analyze the complete set of results.

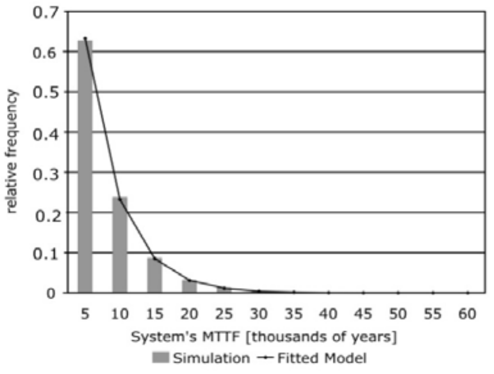

Fig. 4. Longest MTTF

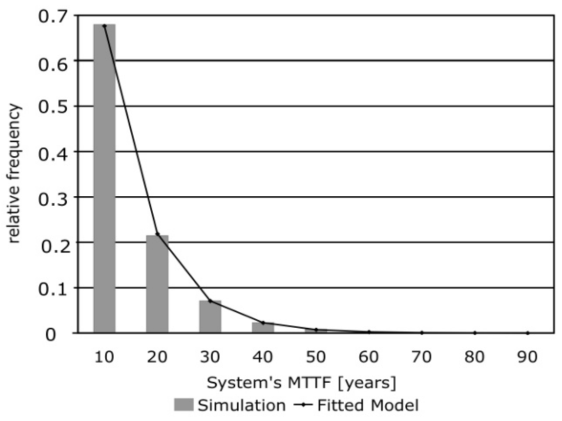

Fig. 5. Shortest MTTF

Our first remark is about the number of active nodes. Let us read Table 3 by rows. We will find that for any fixed combination of spares, lifetime and repairing time values, those systems having 6 active nodes offer the best available performance. As we just mentioned, we assumed a fixed committee size k equals to 5, therefore using only 5 active nodes prevents an unsaturated *kv* storage scheme. It means that there is a single committee involved on every storage operation. A missing node affects this committee and the whole storage system. In contrast, a bigger number of active nodes enables load balance, since it is granted that any node does not work in, at least, one committee. Nevertheless, it is known from reliability theory, that a bigger number of active nodes also accelerates the overall failure rate. Let us recall that recovery is a critical operation where the system becomes vulnerable. Indeed, collapse happens during recovery, when the system is unable to cope with a number of failures beyond its capacity. Therefore, the optimal number of active nodes seems to be a trade off between load balance and the pace at which the system is able to deal with failures.

Table 3. Experimental results

active nodes											
5				***6***				***7***			
spares	*lifetime (hours)*	*repair (hours)*	*MTTF (years)*	*spares*	*lifetime (hours)*	*repair (hours)*	*MTTF (years)*	*spares*	*lifetime (hours)*	*repair (hours)*	*MTTF (years)*
1	5000	5	**25.89**	1	5000	5	**91.68**	1	5000	5	**59.58**
		10	**25.92**			10	**82.28**			10	**32.92**
		20	**25.21**			20	**38.53**			20 **b**	**8.85**
	10000	5	**185.50**		10000	5	**672.22**		10000	5	**400.75**
		10	**183.13**			10	**575.06**			10	**170.33**
		20	**181.95**			20	**192.70**			20	**36.47**
	20000	5	**1277.92**		20000	5	**4875.28**		20000	5	**2493.63**
		10	**1291.20**			10	**3652.66**			10	**782.19**
		20	**1289.37**			20	**864.89**			20	**148.71**
2	5000	5	**25.67**	2	5000	5	**90.70**	2	5000	5	**70.56**
		10	**25.80**			10	**91.99**			10	**70.62**
		20	**26.00**			20	**90.08**			20	**66.17**
	10000	5	**179.27**		10000	5	**676.45**		10000	5	**529.61**
		10	**182.36**			10	**679.46**			10	**502.92**
		20	**181.68**			20	**670.28**			20	**492.14**
	20000	5	**1295.09**		20000	5	**4889.68**		20000	5	**3697.95**
		10	**1284.74**			10	**4929.65**			10	**3689.86**
		20	**1268.90**			20	**4973.56**			20	**3479.42**
3	5000	5	**25.48**	3	5000	5	**93.12**	3	5000	5	**71.40**
		10	**25.29**			10	**91.26**			10	**69.98**
		20	**25.93**			20	**91.23**			20	**68.86**
	10000	5	**181.20**		10000	5	**685.69**		10000	5	**523.27**
		10	**179.05**			10	**675.29**			10	**531.73**
		20	**180.39**			20	**695.05**			20	**515.81**
	20000	5	**1291.61**		20000	5	**4924.65**		20000	5	**3805.19**
		10	**1318.80**			10 **a**	**4981.60**			10	**3758.70**
		20	**1279.99**			20	**4841.71**			20	**3787.54**

Our second remark is about the impact of repair time on system’s performance. We will now read Table 3 by columns. Let us call an *entry set* to the 3 consecutive results corresponding to any fixed combination of active nodes, spares and lifetime; these are the smallest squares enclosing results in Table 3. Compare now 2 entry sets corresponding to two different spares. We will realize that for 2 or more spares, the impact of repairing time is marginal. This result implies that a sufficient spare supply is more important than a potentially slow repair procedure. Our third remark is about the number of spares. It is closely connected with our previous comment. Again, we propose to compare entry sets on the same column. For those entries having the same parameter values but the spares, we will find out that, within the limits of accuracy, entries do not change for either 2 or 3 spares. In other words, from the system´s point of view, 3 spares seem to be a useless excess of redundancy as they provide the same support achieved with only 2 spares.

Based on our last two remarks, we suggest to simplify the reading of Table 3. Let us focus on the sections corresponding to 2 spares only. For the entry sets in this part of the table we have already noticed that the influence of repair time on system's performance, is rather negligible. Thus, we can dismiss this parameter and represent the 3 consecutive results that make up an entry set by its mean value. Once we have finished this pre-processing, we are prepared to present our final remark, about the importance of lifetime. Our results show that for a given set of storage nodes, a twofold increase their individual lifetime produces about a sevenfold increase on the overall system's MTTF.

8 Conclusions and future work

In this work we presented a performance study intended to evaluate the mean time to failure of a distributed storage system. We tested a particular approach that makes use of both space and information redundancy. An advantage of this combination stands on the fact that both are parameterized techniques, therefore, they allow us to experiment with different amounts of redundant resources.

System operation can be briefly described as follows. A set of autonomous stations with storage capacities, called storage nodes, is connected through a fast Ethernet switch. Initially nodes are classified in active or spare nodes. Subsets of active nodes, called committees, are scheduled to work according to a distributed procedure called storage scheme. The committees that make up our proposal are all the subsets of active nodes having a fixed size.

When a storage request arrives at the system, a given committee is called according to a fixed and cyclic order. A file to be stored is transformed into a certain number of dispersals using Rabin´s IDA. Each member of the selected committee is in charge of storing one of the resulting dispersals. Recall that the original file, or any of its dispersals, can be rebuilt provided that a given amount of redundant information remains available. If an active node crashes, two actions take place. First, a distributed control starts the recovery procedure using the surviving active nodes. Then, one spare replaces then missing element and stores the recovered information. Second, the missing element undergoes a repair procedure. Once it becomes operational again, it is regarded as a spare node.

We tested the effect of 4 different parameters on the system’s performance. These parameters are the number of active nodes, the number of spares, the individual node’s lifetime and repair time. Our study mainly shows that the number of active components defines a compromise between load balance and the overall failure rate. As for spares, they are important up to a certain operational point where their availability compensates the repair procedure. Beyond this point, an excess of spares does not pay back any further improvement. Nevertheless, the most influential parameter turned out to be the node’s lifetime. Also, it is worth mentioning that even under the worst combination of parameters, our design renders a mean time to failure longer than the summation of individual lifetimes.

We have settled the basis of a construction method for distributed storage emphasizing on the reutilization of local resources. This approach revaluates the infrastructure already deployed and allows small groups to build up their own solutions according to their particular needs. For future research, we are already studying the best approach to scale up our system. Preliminary work suggests that we may borrow some ideas from P2P networks, to build a federation of local storage systems.

References

1. **Adya, A., Bolosky, W. J., Castro, M., Cermark, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M. & Wattenhofer R. P. (2002).** FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. *5th Symposium on Operating Systems Design and Implementation (OSDI),* Boston, USA, 1-14.
2. **Bhagwan, R., Moore, D., Savage, S. & Voelker, G. M. (2003).** Replication strategies for highly available peer-to-peer storage. In Schiper, A., Shvartsman, A.A., Weatherspoon, H., Zhao, B.Y. (Eds.) *Future Directions in Distributed Computing* (153-158). New Jersey: Springer.
3. Celeste: An Automatic Storage System (s.f.). Retrieved from http://hub.opensolaris.org/bin/view/Project+celeste/WebHome
4. **Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. E.(2006).** Bigtable: A Distributed Storage System for Structured Data. *7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, USA, 205-218.
5. **Chen, Y., Edler, J., Goldberg, A. V., Gottlieb, A., Sobti, S. & Yianilos, P. N. (1999).** A Prototype implementation of Archival Intermemory. *4th ACM Conference on Digital Libraries*, California, USA, 28-37.
6. Cleversafe (s.f.). Retrieved from http://www.cleversafe.org
7. **Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C. & Zhao, B. (2000)** OceanStore: An Architecture for Global-Scale Persistent Storage. *ACM SIGPLAN Notice, 35 (11)*, New York, USA, 190-201.
8. **Marcelín-Jiménez, R., Rajsbaum, S. & Stevens, B. (2006).** Cyclic Storage for Fault-tolerant Distributed Executions. *IEEE Transactions on Parallel and Distributed Systems*, 17(9), 1028-1036.
9. **Rabin, M. O. (1989).** Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance. *Journal of the ACM*, 38(2), 335-348.
10. **Rodrigues, R. & Liskov, B. (2005).** High Availability in DHT's: Erasure Coding vs Replication. *Peer-to-Peer Systems IV.* 4th *International Workshop on Peer-to-Peer Systems*, *Lecture Notes on Computer Science* , 3640, 226-239.
11. **Rowstron, A. & Druschel, P. (2001).** Storage management and caching in PAST, a large-scale, persistent, peer-to-peer storage utility. *ACM SIGOPS Operating Systems Review*, 35(5), 188-201.
12. **Quezada-Naquid, M., Marcelín-Jiménez, R. & López-Guerrero, M. (2007).** Service Policies for a Storage Services Dispatcher in a Distributed Fault-Tolerant Storage Network and their Performance Evaluation. *Canadian Conference on Electrical and Computer Engineering (CCECE '07)*, Vancouver, Canada, 34-40.
13. OMNeT++: Discrete Event Simulation System (s.f.). Retrieved from http://omnetpp.org/
14. **Yianilos, P. & Sobti, S. (2001).** The Evolving Field of Distributed Storage. *IEEE Internet Computing*. 5 (5), 35-39

Moisés Quezada Naquid

Received the B.Sc. degree in Electronics Engineering in 2005 and the M.Sc. in Information Technologies in 2007, both from the Metropolitan Autonomous University (UAM, Mexico City). Currently, he is an Associate Professor with UAM. His research interests are distributed computing, telecommunication networks and digital systems.

Ricardo Marcelín Jiménez

Received his B.Sc. in Electronics Engineering from the Metropolitan Autonomous University (UAM, Mexico City) in 1987, the M.Sc. in Computer Eng. from the National Polytechnic Institute (CINVESTAV-IPN) in 1992 and the PhD degree in Computer Science from the National Autonomous University of Mexico (UNAM) in 2004. He is a full professor with tenure at the Department of Electrical Engineering at UAM. His research interests are in the theory and practice of distributed computing, specially issues related to coordination and fault tolerance. Dr. Marcelín-Jiménez is a SNI member, level I

Miguel López Guerrero

Received his B.Sc. in Mechanical-Electrical Engineering in 1995 and the M.Sc. in Electrical Engineering in 1998, both with honors from the National Autonomous University of Mexico. He received his Ph.D. in Electrical Engineering from the University of Ottawa in 2004. Currently, he is an Associate Professor with the Metropolitan Autonomous University (Mexico City). His research interests span several aspects of telecommunication networks including network traffic modeling, medium access control and mobility-related studies.