Long-term Digital Archiving Based on Selection of Repositories Over P2P Networks

Tiago Vignatti, Luis C. E. Bona, Marcos S. Sunye
Federal University of Paraná
Department of Informatics
{vignatti, bona, sunye}@inf.ufpr.br

André L. Vignatti
University of Campinas
Institute of Computing
vignatti@ic.unicamp.br

Abstract

The importance of digital information is constantly increasing in the last years. Such information often needs to be preserved for a long-term and this is the responsibility of digital archiving systems. This paper proposes a reliable replication model of immutable digital content to be used in long-term archiving systems. The archiving system is modeled as a set of storage repositories where each repository has an independent fail probability assigned to it. Items are inserted with a reliability that is satisfied by replicating them in subsets of repositories. Through simulation, we evaluated three different proposed strategies to create replicas. It is also proposed a completely distributed archiving system using this model over a structured peer-to-peer (P2P) network. The communication between the nodes (repositories) of the network is organized in a distributed hash table and multiple hash functions are used to select repositories that will keep the replicas of each stored item. The system is evaluated through experiments in a real environment. The proposed model and the algorithms, combined with the structured P2P scalability made possible the construction of a reliable and totally distributed digital archiving system.

1. Introduction

The goal of digital archiving systems is to preserve large volumes of data that need to be stored safely for a indefinitely long period of time. Digital libraries, Internet applications such as email, photo sharing and homepages are some examples of services that need these systems. Archiving systems can be built using specialized hardware but, in many cases, this solution can be economically unfeasible. An alternative is to replicate the information in multiple storage repositories, consisting of conventional and low cost computers [2]. Peer-to-Peer (P2P) networks appear as a promising approach to organize systems with these characteristics, since they are highly scalable for the distribution and retrieval of data. However, the mechanisms of replication available in most P2P networks are not sufficient to ensure the preservation of data over a long period of time. In the digital archiving environment, it is important to observe that we are not interested in replicas updates, since we are only dealing with *read-only and static* objects [11]. When a replica is created, it becomes immutable in the system.

The main contribution of this work is the creation of a completely distributed P2P archiving system. In this system, the repositories are organized by a *distributed hash table* (DHT) [16, 14, 19, 20] and *multiple hash functions* are used as mechanisms for replication. The choice of structured P2P, instead of non-structured, is motivated by its scalability regarding the number of nodes. Moreover, in many cases, the search algorithms of non-structured P2P networks cannot locate rare items, which is unacceptable in the our context. Multiples hash funcions were used to perform a selection on specific set of repositories, a feature not present in non-structured P2P model using DHT. The system was evaluated through experiments in a real world environment.

The P2P digital archiving system motivates the definition of a model for data replication. Thus, we propose a model for replication where a reliability metric is associated with each repository. This metric denotes the probability that the data are not lost or damaged in a given period of time. Furthermore, each item (digital information) needs to be stored with a *desired reliability* that reflects the importance of the item, allowing high or low durability (preservation time) depending on its importance. To ensure the desired reliability of an item, several *replicas* of the item are created in the repositories, and the number of replicas needed to preserve an item is determined by the reliability metric of the repositories. This allows an optimization of the network resources usage, compared to other systems where the number of replicas is fixed [10, 15], however, more elaborated strategies for replication should be used in this case. One of the contributions of this work is to present and compare three different strategies for the reliable replication prob-

lem.

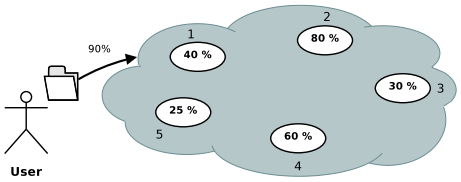

Figure 1: Repositories labeled with independent reliabilities.

As an example, Figure 1 shows a network with five repositories, identified by 1, 2, 3, 4 and 5; with reliability of 40%, 80%, 30%, 60% and 25% respectively. Suppose that a user wants to insert an item in the network with a *desired reliability* of 90%. A simple strategy would replicate this item in the order of identifiers of repositories, until it achieves the desired reliability. Thus, with one replica in the repository 1, the item has guaranteed a reliability of 40%. Adding a replica in the repository 2, the reliability guaranteed would be $1 - (0.6 \times 0.2) = 88\%$, but still not reaching the desired reliability of 90%. With an additional replica in the repository 3, the reliability guaranteed would be $1 - (0.6 \times 0.2 \times 0.7) = 91.6\%$, therefore reaching the desired reliability of the item.

Related Work: In digital preservation and long-term archiving environments, the information is not updated or removed [11]. When a replica is created, it is *immutable* in the system. Thus, in this work we are not interested in strategies and mechanisms for replicas update.

Traditional solutions for backup or data storage, such as replicated file systems and RAID disks [8] do not provide a degree of autonomy as the P2P networks do. Unlike the proposed P2P archiving system of this work, these systems use a centralized control to manage the content that needs to be replicated. P2P systems for file sharing such as Kazaa, eDonkey and Gnutella [12] focus on searching resources in dynamic collections, and are not focused on the reliability of the information. In such systems, a file is replicated every time it is copied into a new node.

CFS [3] and PAST [17] are not able to dynamically change the number of replicas since they employ replication in the neighborhood. The number of replicas cannot exceed the size of the neighbors list, since the number of neighbors is tied with the DHT protocol. Any digital archiving system built on these systems could not be able to achieve the desired degree of replication required to preserve their information. Other forms of storage that use DHT as OceanStore [7] and Glacier [1] consider a simpler model where their nodes have equal probability of failure. BRICKS [15] consider availability instead reliability, associating a single fail probability to all nodes in the network.

The LOCKSS (*Lots of Copies Keep Stuff Safe*) [10] uses a P2P network where computers are controlled by autonomous organizations to preserve the information for a long period of time. It uses a complex scheme of audit to detect and repair the damage in the replicas. Unlike our model, the LOCKSS treats its repositories with a single probability of failure, which does not exactly model real environments. Also, LOCKSS considers a fixed number of replicas for its items.

Contributions and Results Obtained: This paper presents a model for reliable replication of data in digital archiving systems with immutable data. To cope with the scalability, we create a structured P2P network for the reliable archiving, implemented over the model proposed. We also present three strategies of reliable replication. Simulations were performed to compare the replication strategies and experiments in real environments have been made to establish the P2P digital archiving system.

In the proposed model, independent probabilities of failure are associated with each storage repository. These probabilities allow items to be stored based on their preservation time needs. Moreover, the repositories have different storage capacities, i.e., the repositories are *heterogeneous*. Thus, due to limited storage capacity, we aim to insert the maximum number of items in the network; always satisfying the desired reliability of the items.

To maximize the number of items inserted in the network, we designed the strategies with two (heuristic) goals: balance the load between the repositories and, at the same time, minimize the total number of replicas created. This is justified by two facts: (i) balance the load, as well as minimize the replicas created, avoids the overhead in the repositories. Furthermore, (ii) if the load balancing is not performed, a repository can easily becomes completely filled and, consequently, the number of repositories that can be selected decreases, thus decreasing the number of available options that meet the desired reliability of an item. This, in turn, implies the creation of a larger number of replicas to satisfy the desired reliability of an item and hence the total number of items to be inserted in the network decreases due to the limited capacity of the repositories.

In the case of heterogeneous repositories that we consider, balancing the load and minimizing the replicas do not necessarily imply the maximization of the number of items inserted in the network due to different capacities of each repository. Even so, we use these two goals, hoping to approximate the above arguments. Indeed, in Section 2.3, the experimental results of the simulations confirm that these are good goals for the heterogeneous case.

In Section 2.2, we present three strategies for creating replicas, which were empirically stressed in Section 2.3. All these strategies aim to optimize the number of replicas created and the load balancing. However, each strategy is based on different arguments to justify its use. The

Randomized strategy has the load balancing as the main motivation, justified by theoretical results of ball-and-bins [13]. Also, in this strategy, we obtained a theoretical bound in the number of replicas created. The *Greedy over Subset* strategy has the minimization of replicas created as the main motivation, but adjustments are made to perform a good load balancing. The *Ideal Subset* strategy solves the problem without giving priority to minimizing the replicas or balancing the load, acting as a "mean" between the two goals.

Through simulations, in Section 2.3, we found that among the three strategies presented, the Ideal Subset strategy proved to be the most effective in the creation of replicas and load balancing, and also obtaining the highest number of inserted objects in the network. This strategy has a computational complexity that could not be feasible in practice if not carefully formulated.

Due to high scalability for the distribution and retrieval of data, structured P2P networks appear as natural candidates to implement the model proposed. Thus, in Section 3, we present a reliable P2P archiving system, which is the main contribution of this work. The structured P2P networks have a difficulty inherent in the method of routing information which makes difficult the selection of specific nodes (repositories) for the storage. However, this problem was overcome by the use of *multiple hash functions*, which is the main contribution of this section. We describe in detail the algorithms that use multiple hash functions for the basic operations of the network. The implementation of the P2P digital archiving system was evaluated in a real environment, where digital objects were inserted in a network to stress the preservation time in function of their importance.

Organization: In Section 2 we present the model of replication for archiving systems and the strategies used for creating replicas. Moreover, we compare the strategies through simulations. In Section 3, we present the reliable P2P archiving system and use a real environment for evaluation. The conclusion and future work are presented in Section 4.

2. Model and Proposed Strategies

The model is composed by a set N of $|N| = n$ items (digital objects). All items have identical size, without loss of generality, equal to 1. Each item i has associated a probability $0 < r_i < 1$, called the *desired reliability* of the item. Furthermore, we have a set $M = \{\ell_1, \ldots, \ell_m\}$ of $|M| = m$ repositories, where each repository ℓ_j has associated a *storage capacity* c_j and a probability $0 < p_j < 1$, called the *reliability* of the repository. To determine this reliability, we can consider some parameters such as the number of bugs, the vulnerabilities of the system, human factors, the frequency at which the machines are repaired, among others. The determination of the desired reliability of an item is not in the scope of this work.

The reliability of the subset $S \subseteq M$ is defined as $1 - \prod_{\ell_j \in S}(1-p_j)$, which denotes the probability of at least one repository in S does not lose data in a given time interval.

We define the problem as follows. Items arrive one by one, i.e., initially there is no item, and the items arrive as time passes. After an item i arrives, we have to choose a subset S_i of repositories where each repository in S_i receives a copy of the item (*replica*) to satisfy the desired reliability r_i. In other words, we select $S_i \subseteq M$ such that

$$1 - \prod_{\ell_j \in S_i} (1 - p_j) \geq r_i. \quad (1)$$

In addition, each repository in S_i must have enough free space to receive the copy. The *load* of a repository is the number of replicas assigned to it. The replicas are never updated, i.e., they are immutable. The objective of the problem is to maximize the number of items inserted in the network, satisfying the desired reliability of items and the capacity of the repositories.

As argued before, we focus on two goals: (i) on the one hand, we want to minimize the total number of replicas created, i.e., minimize $\sum_{\forall i} |S_i|$. On the other hand, (ii) at the same time, the replicas should be placed so as to balance the load. To measure the load balacing, we evaluate two metrics: the *makespan*, i.e., the load of the most loaded repository, and *standard deviation* of the repositories load. Note that the minimization of the makespan and the standard deviation of the loads are sufficient measures to ensure that all repositories have a balanced load.

A naive approach would treat both goals in an independent way, for example, first solving the minimization of the replicas and then balancing the load. This is not a good approach, because the number of replicas created depends directly from repositories which had allocate the replicas. As an example, we illustrate a situation where we first solve the problem of replicas and then solve the load balancing. Suppose an instance where the repository ℓ_j has reliability p_j and all items have desired reliability less than p_j. It suffices to create a single replica of each item in ℓ_j. However ℓ_j will be overloaded, and when we start the phase of balancing the load, we have to remove items from ℓ_j. In this way, we have to select other repositories to accommodate new replicas of these items to meet their desired reliability, lying again on the problem that we thought was solved before the load balancing.

2.1. Equivalent Definition

Next, we rewrite the desired reliability constraint to ease the handling of the problem of replicas creation. We present an equivalent definition of the desired reliability constraint, replacing the product by a summation. We rewrite the de-

sired reliability constraint as follows:

$$1 - \prod_{\ell_j \in S_i} (1 - p_j) \geq r_i \equiv \prod_{\ell_j \in S_i} (1 - p_j) \leq 1 - r_i$$
$$\equiv \prod_{\ell_j \in S_i} e^{\ln(1-p_j)} \leq e^{\ln(1-r_i)}$$
$$\equiv e^{\sum_{\ell_j \in S_i} \ln(1-p_j)} \leq e^{\ln(1-r_i)}.$$

But

$$e^{\sum_{\ell_j \in S_i} \ln(1-p_j)} \leq e^{\ln(1-r_i)} \iff \sum_{\ell_j \in S_i} \ln(1-p_j) \leq \ln(1-r_i).$$

As $0 < p_j, r_i < 1$, then the value of the logarithm function is negative. Thus, for clarity of notation, we redefine the problem variables. Let $a_j = -\ln(1 - p_j)$ and $b_i = -\ln(1 - r_i)$. Therefore, the desired reliability constraint can be rewritten in the following equivalent way,

$$\sum_{\ell_j \in S_i} a_j \geq b_i. \tag{2}$$

That is, for an item i we have to select $S_i \subseteq M$ such that the sum of a_j exceeds b_i. Equation 2, besides being easier to handle, is equivalent to the Equation 1.

2.2. Strategies for Replicas Creation

In what follows, we present three strategies for the creation of replicas. In all strategies, we do a selection over a set of repositories. When doing a selection on M, the complexity of the worst case is linear in m, which is a good theoretical bound. In practice, however, a selection on the set of all machines of the network is infeasible. Thus, in the strategies described below, we denote by $M^o \subseteq M$ the set of machines that are available based on a feasible number of machines that we can select in practical situations. For each item that arrives to be inserted, we consider that M^o is selected at random from M; in practice, the way that M^o is selected may depends on the system features. Note that the strategies presented are generic, i.e., can be used in various situations of reliable replication systems. Therefore, the size of M^o may depends on the features of the real-world situation that we are considering.

It is worth noting that in all strategies, when a replica is assigned to a full repository, we ignore it and randomly select another repository of the considered set.

Randomized Strategy: Here, we present the first proposed strategy to solve the problem. Algorithm 1 shows the details.

In this subsection, we use $m = |M^o|$ and assume that the reliabilities of the repositories are uniformly distributed in an interval. Formally, the values p_j is chosen according to the *continuous uniform distribution* in the interval $[a, b]$, where $a > 0$ and $b < 1$. We assume that the expected

```
begin
    S = ∅
    while reliability of S is less than r_i do
        choose ℓ_j ∈ M^o uniformly at random
        S = S ∪ {ℓ_j}
    return S
end
```

Algorithm 1: Randomized

number of reliable repositories (in a specified time interval) is at least $8 \ln m$; note that this is not a strong assumption to the problem, since the expected fraction $\frac{8 \ln m}{m}$ of reliable repositories goes to 0 when m goes to infinity. For example, for $m = 1000$, we assume that the expected fraction of reliable repositories is $\frac{8 \ln 1000}{1000} \approx 0.05$, i.e., we assume that, in expectation, 5% of repositories are reliable.

Let X_j be a random variable that is equal to 1 if the repository ℓ_j is reliable in the time interval specified, 0 otherwise. Let $X = \sum_{j \in M^o} X_j$ be the random variable of the total number of reliable repositories in a given time interval. As we assume that the reliabilities of the repositories are distributed according to the continuous uniform distribution, then $E[X] = \frac{m(a+b)}{2}$. In a given time interval, X can be less than $E[X]$, however, with high probability, it cannot be much less than the expected value, as shown in Lemma 2.1.

Lemma 2.1. *Let X_j be a random variable that is equal to 1 if the repository ℓ_j is reliable, 0 otherwise. Let $X = \sum_{\ell_j \in M^o} X_j$ be the random variable of the total number of reliable repositories in a given time interval. Then $Pr(X < \frac{1}{4}E[X]) < \frac{1}{m^2}$.*

Proof. Note that X is a sum of Poisson random variables. Therefore, we can apply a known Chernoff bound [13], obtaining $Pr(X < (1 - \frac{3}{4})E[X]) \leq e^{-\frac{9E[X]}{32}} \leq m^{-2}$, where the last inequality follows from the fact that $E[X] \geq 8 \ln m$. □

Lemma 2.1 tells us that with high probability (i.e., probability greater than $1 - \frac{1}{m^2}$), a fraction of $\frac{a+b}{8}$ of the repositories are reliable. That is, with high probability, when selecting a repository uniformly at random, we have probability at least $\frac{a+b}{8}$ that it is a reliable repository. Thus, by choosing k repositories uniformly at random, the probability that one or more repositories are reliable is at least $1 - \left(1 - \frac{a+b}{8}\right)^k$ and we want that this probability be greater than the desired reliability r_i. Thus, by choosing $k = \left\lceil \frac{8}{a+b} \ln\left(\frac{1}{1-r_i}\right) \right\rceil$ repositories uniformly at random and place the replicas on them, the desired reliability of the item i is satisfied with high probability, as the Theorem 2.2 claims.

Theorem 2.2. *If item i places $k = \left\lceil \frac{8}{a+b} \ln\left(\frac{1}{1-r_i}\right) \right\rceil$ replicas in k repositories chosen uniformly at random then, with high probability, the desired reliability of i is satisfied.*

As an example of Theorem 2.2 application, suppose that the reliability of the repositories are uniformly distributed between 50% and something close to 100%. Thus, an item with desired reliability of 95% needs to choose $\left\lceil \frac{8}{0,5+1} \ln \left(\frac{1}{1-0,95} \right) \right\rceil = 16$ repositories uniformly at random to place its replicas.

Regarding the load balancing, we note that in Algorithm 1, a replica always chooses a repository uniformly at random. That is, Algorithm 1 simulates the balls-and-bins process, well studied in the area of randomized algorithms and the existing results can be used in our problem [13]. The results of Theorems 2.3 and 2.4 can be easily obtained from the results of balls and bins and therefore the demonstrations will be omitted.

Theorem 2.3. *If n balls are thrown independently and uniformly at random on m bins, then the load of the most loaded bin is bounded by $2e\frac{n}{m} + 2\log m$ with high probability.*

Theorem 2.4. *Let $n \geq m \log m$. If n balls are thrown independently and uniformly at random on m bins, then the load of the most loaded bin is bounded by $\frac{n}{m} + \sqrt{8\frac{m}{n}\log m}$ with high probability.*

Theorems 2.3 and 2.4 are related, respectively, to a small and a large number of balls (in our problem, they are the items). However, in practical situations, it is likely that $n \geq m \log m$ and, in this case, Theorem 2.4 tells us that with high probability the most loaded repository have a constant number of items in addition to the optimal balance.

Greedy over Subset Strategy: Suppose we want to minimize the total number of replicas without worrying about the load balancing. Thus, using the equivalent definition of Section 2.1, it suffices to solve the integer linear program (ILP) below

$$\begin{aligned} \min & \sum_{\ell_j \in M^o} x_j \\ \text{s.t.} & \sum_{\ell_j \in M^o} a_j x_j \geq b_i \\ & x_j \in \{0,1\} \qquad \forall \ell_j \in M^o \end{aligned}$$

The ILP described can be optimally solved by sorting the values a_j in non-increasing order and taking the values in such order until the sum of the selected values reaches b_i. This greedy strategy, besides being very simple and efficient, optimally solves the ILP as, in a given step, we have no advantage in selecting a value less than the value in the sorted order. Algorithm 2 shows the details of this strategy.

Note that this strategy accumulates the replicas on the repositories in M^o with higher reliability, which is not good for the load balancing. We know that M^o is randomly chosen in each insertion of an item, but this do not suffices to improve the load balancing because if M^o is large, we lie again in the problem of accumulating the replicas in repositories with higher reliability. Moreover, if M^o is small, we do not have many options to choose or there are not enough repositories to satisfy the desired reliability of an item. In Section 2.3, we evaluate several sizes of M^o.

```
begin
    sort M^o in non-increasing order according to the values a_j
    S = ∅
    t = 1
    while Σ_{ℓ_j ∈ S} a_j < b_i do
        S = S ∪ {ℓ_t ∈ M^o}
        t = t + 1
    return S
end
```

Algorithm 2: Greedy over Subset

Ideal Subset Strategy: To create the replicas, we select the subset $S \subseteq M^o$ that provides the reliability that is closest to the desired reliability of the item. That is, we choose $S \subseteq M^o$ that minimizes $1 - \prod_{\ell_j \in S}(1-p_j) - r_i$. Using the equivalent definition of the problem, we need to solve the following ILP

$$\begin{aligned} \min & \sum_{\ell_j \in M^o} a_j x_j - b_i \\ \text{s.t.} & \sum_{\ell_j \in M^o} a_j x_j \geq b_i \\ & x_j \in \{0,1\} \qquad \forall \ell_j \in M^o \end{aligned}$$

Note that, if the solution value is equal to 0 then there is a subset of values that together sums exactly b_i; if the solution value is greater than 0, then there is no such subset. Thus, if we solve this ILP, then we could also solve the SUBSET-SUM decision problem, which is NP-complete[4]. Therefore, as the ILP is an optimization problem, then it is NP-hard; assuming P $\neq$ NP, no polynomial algorithm can solves such problem. However, there is a *dynamic programming* algorithm which solves this problem in pseudo-polynomial time, but in practice is satisfactory for the vast majority of instances [4]. The details of the algorithm are omitted.

The solution of the above ILP do not necessarily provides a good solution to minimize the number of replicas created. Nevertheless, the Ideal Subset strategy is motivated by the fact that in practical situations it is expected to not create too many replicas and to select different subsets for each item, so that the distribution of replicas in the repositories balance their loads. Note that, if we had not used the equivalent definition, we are not able to model this case as a subset sum problem, and then we need a real exponencial algorithm to solves this, which turns out to be an unfeasible alternative to this case.

2.3. Simulation and Evaluation of the Strategies

Through experiments we compare the three strategies. In the evaluation, we used a simulator that implements the algorithms. All experiments were performed 10 times and the average was collected. In all experiments we consider a set M with 100 repositories. The size of the subset considered in the simulations reflects the size of the subset M^o defined in Section 2.2, and for each item inserted, M^o is randomly selected in M.

In the experiments of number of replicas created, standard deviation and makespan we assume uncapacitated repositories.

Number of Replicas Created: In this experiment, we compare the number of replicas created in each strategy. The simulation was executed for repositories with the average reliability ranging from 50%, 60%, 70% and 80%, with 6% of standard deviation. We performed the insertion of 1000 items, all items with desired reliability of 99%.

Figure 2 shows the number of replicas created according to the size of M^o. In the Randomized strategy, the change in the size of M^o is almost negligible in the result. This is not true when considering repositories with bounded capacity, as the experiment of number of items inserted below. Note that in some cases the line starts only after a given size of M^o. The reason is that for a M^o of small size, the desired reliability is not satisfied in the insertion of some items.

As the size of the subset of repositories grows, the number of replicas required to achieve the desired reliability decreases. This decrease is higher in the Greedy Over Subset, which aims the minimization of replicas created, always creating them in repositories with the greatest reliabilities. The Ideal Subset strategy also has a decreasing curve, but less than the Greedy Over Subset and greater than the Randomized strategy.

Standard Deviation: Figure 3 illustrates the second experiment, showing the standard deviation of the repositories loads in function of the size of the subsets. We used an average reliability of 60% for repositories, with 6% of standard deviation (the same experiment was repeated for repositories with other reliability values and the result was proportionally the same).

Just as in the previous experiment, the Randomized strategy is almost not influenced by the size of M^o. In the other strategies, we see that, as M^o increases, the standard deviation also increases. The Greedy Over Subset is the strategy with the worst load balancing.

Makespan: Figure 4 illustrates the third experiment where the makespan (i.e., the load of the most loaded repository) is plotted as a function of the size of M^o. The network setup for this experiment is the same of the first experiment.

As M^o grows, the makespan also grows in Greedy Over Subset and Ideal Subset strategies, but the latter has a lower growth. This behavior is due to the fact that some repositories are always more demanded. In Greedy Over Subset strategy the most demanded repositories are those with greater reliability. Repositories with low reliability are the most demanded in Ideal Subset strategy, as repositories with this feature are used to avoid the "waste" of reliability.

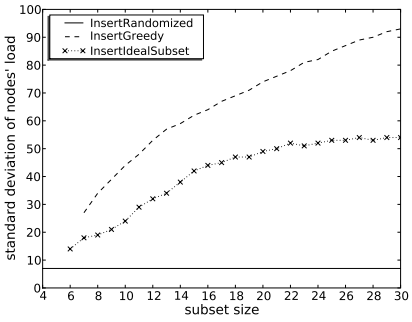

Figure 3: Standard deviation of the loads of the repositories in function of the size of M^o.

Number of Items Inserted: For repositories with homogeneous storage capacity, the makespan is an important metric of evaluation because it evaluates the number of items inserted. The strategy with the greatest makespan implies in the smallest number of items inserted. However, in most real-world environments, the storage capacity of the repositories differ in several orders of magnitude. In such a network, the makespan do not necessarily indicates the same behavior of the homogeneous case. The goal of this experiment is to measure the number of items inserted in heterogeneous repositories. The items were inserted until one fail in creating replicas due to the lack of space in some repository. The items have size of 35 MB and desired reliability of 99%. The storage capacity of the 100 repositories varies between 100 MB and 100.000 MB, and their average reliability is 67% (standard deviation of 17%).

Depending on the strategy used, changing the size of M^o causes different results: the greater the M^o, greater is the number of possible solutions and consequently, greater is the number of items inserted. Table 1 shows the insertion of items depending on the size of M^o for the three strategies proposed. Ideal Subset strategy has the best results.

	10	20	40	100
Randomized	18569	30228	32464	34138
Greedy over Subset	17544	28182	30891	33899
Ideal Subset	20535	31094	35399	39107

Table 1: Insertion of the items in function of the size of the subset.

Discussion of the Results: Among the three strategies, the Greedy Over Subset creates the lowest number of replicas. On the other hand, it has the highest makespan. The Ran-

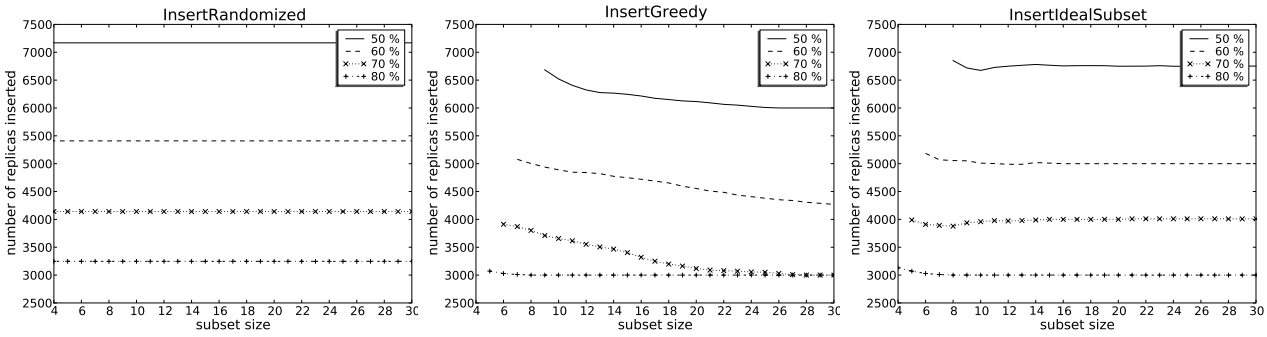

Figure 2: Number of replicas created in function of the size of M^o.

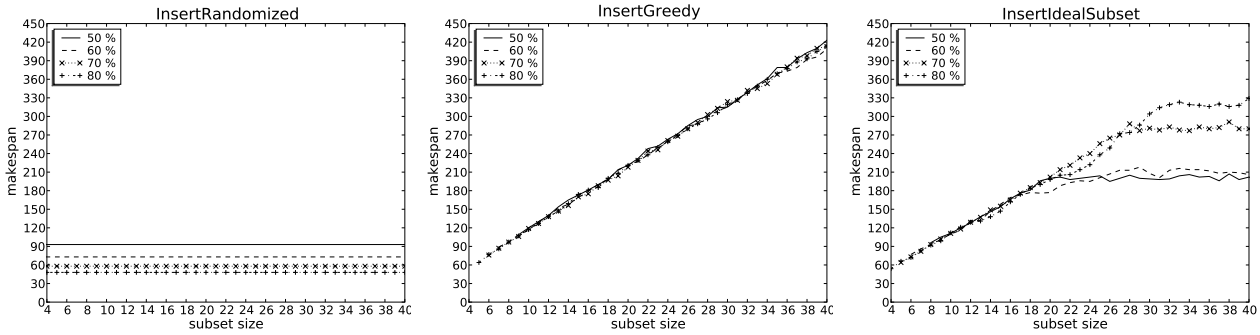

Figure 4: Makespan in function of the size of M^o.

domized strategy is the strategy with the best load balancing of replicas, but it is the worst in the number of replicas created. The "mean" between minimizing the number of replicas and balancing the load is obtained with Ideal Subset strategy. Moreover, in a set of repositories with heterogeneous storage capacities, the Ideal Subset strategy fill the free spaces of the repositories better than the other strategies and insert the greatest number of items.

3. A Peer-to-Peer Digital Archiving System

The model proposed in Section 2 is designed in a generic way and can be implemented on any distributed mechanism for organizing the storage repositories. In particular, structured P2P using DHT appears as natural candidate as it is highly scalable for data distribution and retrieval. However, a difficulty inherent in structured P2P networks is the accurate selection of nodes (repositories) to store the replicas. It is not trivial to select a specific subset of nodes using the routing method from DHTs. Therefore, the implementation of the digital archiving system needs to define an architecture that accommodates all the features that the model of Section 2 requires. Thus, we present a scheme of selection of nodes using *multiple hash functions*, which allows the selection of a particular set of nodes.

3.1. Architecture

Structured P2P Networks and DHT: The system routes the messages of the network through structured P2P networks, using DHTs. The choice of structured P2P, instead of non-structured, is motivated by its scalability regarding the number of nodes. A problem of non-structured P2P networks is that they often use brute force algorithms to perform the search ("flooding") and are more suitable for popular content. Moreover, in many cases, the search algorithms of non-structured P2P networks cannot locate rare items, which is unacceptable in the context of digital archiving where the objects are equally popular [9].

DHTs have a problem with the transient population, i.e., maintaining the structure of the routing tables is relatively expensive in *churn* situations. However, the machines transiency in organizations that intend to preserve digital documents is not as frequent when compared with machines used in traditional applications in the non-structured architecture [10]. Therefore, the necessary adjustments in the topology of the structured networks does not overload the archiving system in case of churn.

Specific Selection of Repositories: The strategies proposed in Section 2.2 assume that is possible to do a selection on specific repositories. However, the DHT by itself does not provide the mechanisms of selection of a specific node,

due to its method of index keys. So if we are interested to store the content in a given set of repositories, we must provide a mechanism that simulates the process of specific selection. To perform the selection of specific nodes, we propose the use of *multiple hash functions*, as explained below.

A digital object consists of a *key*, which is the identifier of the object; a *value*, which is the content of the object; and a parameter of *desired reliability*, which is the reliability that should be achieved when inserting the object in the network. Let $h_1, h_2, \ldots, h_r$ be the r *hash functions*. The hash functions have global visibility, i.e., they are the same for all nodes. Given the key k of a digital object, we apply k to the hash functions, i.e., $h_1(k), h_2(k), \ldots, h_r(k)$. Each of the r generated hash maps to a node in the network. Thus, for each object to be inserted, we get r nodes where we can place replicas (i.e., the set M^o). From this set of r nodes, we use a strategy of replica creation (e.g., the strategies of Section 2.2) to define the subset of these r nodes that receives the replicas. It is not difficult to obtain a family of such hash functions. One way is to use a single hash function h and append a number $i = 1, \ldots, r$ to the key of the object, which is used as argument to the function h. For instance, if the object key is the string *foo*, then $h(foo1), h(foo2), \ldots, h(foor)$ would give us r hashes of this object.

Figure 5 illustrates the selection of repositories performed by three different objects. In the figure, `object_a`, `object_b` and `object_c` are keys, and the dotted lines denotes the set of nodes associated with each key after applying $r = 6$ hash functions. As we have 6 hash functions, the resulted hashes maps to 6 nodes of the network. From each subset of nodes associated with an object, the strategy of creating replicas is applied to determine the nodes that receive the replicas. The black circles represent the repositories that have been chosen to put the replicas.

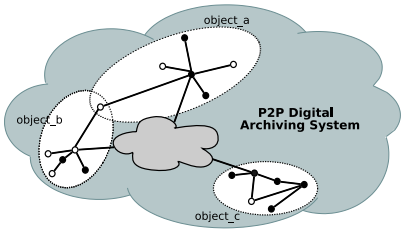

Figure 5: Subsets of repositories associated with their respective digital objects.

It is worth noting that performing a selection among *all* nodes of the network (or equivalently, using m hash functions) is not a good approach because, in this case, an instance would have the size of the network, which is unfeasible in real world situations. On the other hand, considering a subset of small size is a feasible option of implementation even in face of the scalability of the network. Thus, based on the results of Section 2.3, we can choose the optimal size of the subset and, therefore, decide how many hash functions to use.

The main advantage of using multiple hash functions is the ability to select specific nodes. Without this, the strategies proposed in Section 2.2 could not be implemented in structured P2P networks using DHT. Moreover, an advantage of using the strategy of multiple hash functions is the ability to use any DHT protocol to route messages in the network, unlike the strategies for replication in P2P using the *neighborhood* or the *path* [6], which are tied to the protocol. Multiple hash functions also allow flexibility for digital objects to have different numbers of replicas, which is not possible in the *symmetric* strategy of replication [5]. Another feature is the easy retrieving of a given replica without necessarily retrieve another one previously, making it easy to develop a retrieving algorithm. In the *correlated hash* strategy [6], all the keys of a given object are correlated with the first key, thus not allowing this feature.

The selection of repositories proposed above involves no centralized information about the location of the stored replicas. An alternative would be the use of a super-node that had a "directory" which could be consulted about the information of the exact location of replicas of a given object. However, this super-node would be a contention point. Our system avoids super-nodes, adopting a completely distributed approach where no information is centralized.

3.2. Algorithms for System Operation

To operate the P2P digital archiving system, we need to define some basic operations of the system. In this section, we present the algorithms `insert` and `retrieve`, used respectively for the insertion and retrieving of a digital object in the system. These algorithms implement multiple hash functions discussed earlier. It is important to note that both algorithms are executed locally in each node. For example, a user who wishes to insert or retrieve an object contacts any node of the network, which in turn initiates the process of routing the message of the DHT.

input: key, $value$, $reliability$ r_i
begin
  $M^o = \emptyset$
  for $i = 1$ *to* r **do**
    $M^o = M^o \cup \{\ell_{h_i(key)}\}$
  $S =$ `insertion_strategy`(M^o, r_i)
  foreach $s \in S$ **do**
    $j \leftarrow$ hash function number of s
    put$(h_j(key), value)$
end

Algorithm 3: insert (key, value, reliability)

To insert an object, the routine `insert(key, value, reliability)` is used, as shown in Algorithm

3. When inserting an object in the network, the desired reliability of the object is previously chosen by the user. Initially, M^o starts empty. The first loop selects the subset M^o of size r associated to the key of the object; $\ell_{h_i(key)}$ is an abuse of notation which denotes the node pointed by the i-th hash of the key. In this loop, we implicitly save the values i used for each node; this will be used later. After that, the function `insertion_strategy`(M^o, r_i) is executed, which returns the subset $S \subseteq M^o$ of nodes that will receive the replicas. The function `insertion_strategy` can be replaced by any strategy of replication, for example, those presented in Section 2.2. In our implementation, the reliability of the nodes are stored in the DHT. The last loop is where the insertion occurs. The value j denotes the hash function number of the object s considered; as stated previously, these values were saved in the first loop. The routine `put(`h_j`(key), value)` puts a replica of the object in the chosen location.

The implementation of the algorithm takes care of not assigning more than two replicas of the same object in one node.

```
input: key
begin
    for i = 1 to r do
        value ← get(h_i(key))
        if value is not null then
            return value
    return −1 /* not found                                    */
end
```

Algorithm 4: retrieve (key)

To retrieve the object, the user performs the function `retrieve(key)`, where `key` is the key of the object to be retrieved, as shown in Algorithm 4. The idea of the algorithm is to search in all r nodes for a replica of the item, using the hash functions for that. There are two cases where the DHT does not return the object: when the node is not present or the node does not contain a replica of the object.

3.3. Experimental Results

The P2P archiving system was implemented and evaluated through experiments carried out in a real world environment. The implementation uses the *Overlay Weaver* environment to build networks [18]. Overlay Weaver provides great flexibility in the choice of DHT protocols and other high-level services implemented as overlay networks. In particular, this environment supports Chord, Pastry, Tapestry and Kademlia. In our experiments we use Chord. It is worth noting that Overlay Weaver has itself a mechanism of replication that has been turned off for the evaluation of our experiments. The experiments were conducted in a network with 12 nodes; this quantity is enough to validate the ability of long-term archiving and the mechanism of creation of replicas.

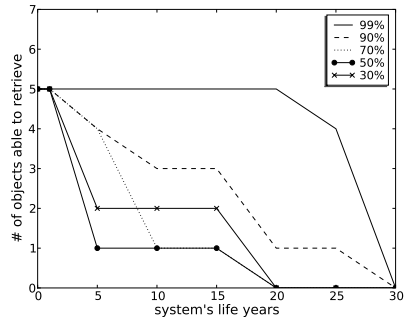

Figure 6: Number of objects able to retrieving information depending on the age of the system.

The reliability of each node was considered to be the probability of the node do not lose information during the period of 1 year. Thus, we can simulate the state of the network regarding the preservation of information over the years. In our experiments, we do not evaluate changes in the reliability of the nodes over the years.

The experiment starts with a network where various objects are inserted. Objects are inserted with different reliabilities. For the replication of objects we used the Ideal Subset strategy, with the size of the subsets equal to 6. Initially all nodes are reliable (*online*). When a node becomes unreliable, it loses its information and is disconnected from the network (*offline*). We purposely did not implement a strategy to recover offline nodes because we want to stress that objects with high desired reliability are preserved longer.

Figure 6 shows the results. Of the total of 12 nodes, 2 of them have 30% of reliability, 2 have 50%, 2 have 70%, 3 have 80%, and the remaining 3 have 90% of reliability. Initially, in the first year, we include a total of 25 objects, divided into 5 groups, each group containing objects with desired reliability respectively, 30%, 50%, 70%, 90% and 99%.

After the first year of existence of the system, a node of reliability equal to 30%, a node with 50% and another with 90% failed. Even with these fails, all the 25 objects were able to be retrieved. At the end of the fifth year of the system, two nodes (30% and 70% of reliability) get disconnected from the network. In the fifth year, it was not possible to retrieve the 3 objects with 30% of desired reliability, 4 objects of 50%, 1 of 70% and 1 of 90%. In the tenth year, three additional nodes gets *offline*. They are nodes with reliability of 70%, 80% and 90%, respectively. It was unable to locate a total of 3 objects of 30% of desired reliability, 4 objects of 50%, 4 of 70% and 2 of 90%. At the end of the fifteenth year, a node with 80% of reliability was disconnected and the same objects of the tenth year were possible to be retrieved. In the twentieth year, 1 node of 50% was disconnected and only one node of 80% and another of 90% left

the system. In this age of the system, it was able to locate all objects of 99% of desired reliability and 1 object with 90%; all other objects were lost. In the twenty-fifth year, the node of 80% failed and it was still possible to retrieve 4 objects of 99% and 1 of 90%. In the thirtieth year the last node were disconnected from the network and no further replicas existed.

4. Conclusions and Future Work

The experiments conducted in this work demonstrate the ability of the system proposed to preserve information for a long period of time. The importance for the preservation of each digital object - measured in the model by the desired reliability - impacts on different lifetime of this object. Objects that were inserted with a high desired reliability had a greater lifetime. Thus, through experiments, we can conclude two main characteristics of the P2P system and the replication model proposed:

Independence of object preservation: different information requires different storage time. Collections of photos, journals and articles may need few years of storage; other information such as digital objects in museums and libraries requires hundreds of years. Our system allows flexibility in the choice of lifetime of objects to be preserved;

Optimization of the storage resources: storage repositories may suffer many types of damages on their contents, so each repository has a different reliability. Allowing each storage repository with a parameter capable of measuring the independent probability of failure is the closest way to model real networks. This approach allows the flexibility in the time of preservation of each object and therefore different numbers of replicas for them, impacting on a better usage of network storage repositories.

Future works include the implementation of a complete digital preservation system. In order to this, the system should be concerned with the auditory of the replicas and other threats such as software obsolescence, not considered in this work. Furthermore, very little emphasis was given to the retrieving of the items. Possibly, we can use our model of digital archiving on systems such as LOCKSS and BRICKS, and evaluate their feasibility is also an interesting future work.

References

[1] Glacier: highly durable, decentralized storage despite massive correlated failures. In *In proceedings of NSDI'05*, Berkeley, CA, USA, 2005. USENIX Assoc.

[2] B. Cooper and H. Garcia. Creating trading networks of digital archives. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2001.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[5] A. Ghodsi, L. O. Alima, and S. Haridi. Symmetric replication for structured peer-to-peer systems. In *DBISP2P*, pages 74–85, 2005.

[6] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance evaluation of replication strategies in DHTs under churn. In *In proceedings of MUM '07*. ACM, 2007.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *In Proceedings of ASPLOS-IX*, New York, NY, USA, 2000. ACM.

[8] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, and L. Shrira. Replication in the harp file system. *SIGOPS*, 1991.

[9] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *In proceedings of SIGMETRICS '02*. ACM, 2002.

[10] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Trans. Comput. Syst.*, 23, 2005.

[11] V. Martins, E. Pacitti, and P. Valduriez. Survey of data replication in P2P systems. Technical report, INRIA, 2006.

[12] D. S. Milojicic, V. Kalogeraki, R. Lukose, and K. Nagarajan. Peer-to-peer computing. Technical report, HP Labs, 2002.

[13] M. Mitzenmacher and E. Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. 2005.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the SIGCOMM '01*, New York, NY, USA, 2001. ACM.

[15] T. Risse and P. Knezevic. A self-organizing data store for large scale distributed infrastructures. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, Washington, DC, USA, 2005. IEEE Computer Society.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2001.

[17] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, 2001.

[18] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay weaver: An overlay construction toolkit. *Comput. Commun.*, 2008.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the SIGCOMM '01*, pages 149–160, 2001.

[20] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, January 2004.