Redundancy in an heterogeneous distributed storage system

[Extended Abstract]

Manuel Cabral
Superior Technical Institute
Technical University of Lisbon
Lisbon, Portugal
manuel.cabral@ist.utl.pt

ABSTRACT

The goal of digital preservation is the accurate rendering of digital content over time. To do this it is necessary, among other things, to be able to store digital information reliably, that is, without data losses. This cannot be achieved without using redundancy. The GRITO project aims to develop a distributed storage system which allows the harnessing of spare resources of grids both dedicated to preservation or with other main purposes. Due to the usage of different grids, it is likely that the storage system's nodes will be very heterogeneous and have different characteristics. This means that many of the failures of the system's components will be correlated. The goal of this work is to study redundancy strategies suitable for such an heterogeneous environment and to implement them in a GRITO.

To achieve this, a system was developed which allows the analysis of a redundancy strategy's performance using simulation, its implementation in the real system and monitorization of that system to adapt the strategy to its real behaviour. We also propose strategies which take correlated failures into account and models that can be parametrized to represent the failures of a real system. Both the simulation system and strategies support different failure models, allowing the most suitable one to be used for each specific system. Experimental results show that the performance of strategies which take correlated failures into account is much superior and that their automatic adaptation can improve their performance.

Keywords

Digital preservation, data grids, redundancy, correlated failures, simulation, introspection

1. INTRODUCTION

Digital preservation combines policies, strategies and actions to ensure access to digital content regardless of the challenges of media failure and technological change. The goal of digital preservation is the accurate rendering of authenticated content over time.

To achieve this it is necessary, among other things, to be able to store digital information reliably, that is, in a way which will prevent data losses. The GRITO project attempts to lower the cost of this by developing a distributed storage system which allows the harnessing of spare resources of grids in different Portuguese institutions.

The most important requirement in digital preservation storage systems is that the data stored there is not lost. Being able to read it quickly is not usually very important and read acesses are rare. Also, after the data is stored, it is not usually necessary to change it, which means that there won't be any random-writes to the data, eliminating the problem of replica consistency.

Due to the usage of different grids, it is likely that the storage system's nodes will be very heterogeneous, with major differences in characteristics like software configuration or physical location. Because of this, it is expected that many of the failures of the system's components will be correlated, for example, a worm outbreak could cause failure of multiple components with similar software configurations.

Redundancy is a fundamental tool to achieve reliable data storage in a distributed system. This paper describes the study of redundancy strategies to be used in a storage system like the one described earlier. A redundancy strategy defines which type of redundancy to use (replication or erasure coding) and how to react to each system state. For instance, a redundancy strategy may consist on using replication and creating a new replica in a randomly chosen storage resource when the number of file replicas is lower than three.

The goal of this work is to study and implement redundancy strategies in a GRITO system. This means developing a set of strategies, a method to compare their performance and a redundancy system to be used with GRITO. These strategies should take into account correlated failures.

We have developed a system to analyse and implement redundacy strategies in an environment with correlated failures. This solution includes three components:

Simulation Using Serapeum, a simulator which allows the long-term performance of a redundancy strategy to be analysed. Different failures models can be used to generate the simulations' failures, allowing the most adequate model to be used for each particular system.

Implementation Using a redundancy system which allows that strategy to be implemented in a real system using the same code as the simulation. This means that strategies only have to be implemented once for testing and real usage. This system is designed to work with different types of data grids and we have developed support for the iRODS data grid, which is used by GRITO.

Introspection Using a mechanism which measures the real behaviour of the system during execution. These measurements can be used to adapt the models used to predict failures and to implement strategies which automatically adapt to the real system.

As mentioned before, the redundancy strategies used on a system such as this should take correlated failures into account since, otherwise, they may become useless if all replicas are placed in nodes with a high probability of failing simultaneously. In this paper we propose and implement strategies that do this and that can use different models to predict failures. We also propose and implement models which can be parametrized to represent the failures of a real system.

In section 2 we present related work in this area, in section 3 we describe this problem in more detail, including a list of the threats to a digital preservation system. In section 4 we give an overview of our solution. In sections 5, 6 and 7 and are described Serapeum, the redundancy system and the algorithms we developed, respectively. The failure models that we propose are also described in these sections. Finally, in section 8 we present results obtained from our simulations and in section 9 the conclusions of our work.

2. RELATED WORK

In this section we describe relevant aspects of a distributed storage system which can contribute to reduce data loss and lower its cost.. Also, we give an overview of some storage systems designed for preservation or with similar requirements.

2.1 Relevant aspects

2.1.1 Redundancy method

If data is stored in a single component, it will be lost when that component fails, which is very likely to happen in the long lifetime of preservation systems. Therefore, most of these systems take advantage of a basic attribute of digital information: that it can be copied without any loss of information. This means that several copies of the data can be stored across many components.

Using redundancy greatly lowers the probability that data will be lost, since it becomes necessary that more than one component fails simultaneously for the information to be lost. This is one of the main techniques used to ensure data longevity, and has been used for a long time in RAID disk arrays, which replicate data across several disks or use parity schemes to ensure that data is not lost when one of the disks fail [7]. Unfortunately, the hardware required to store a large amount of data this way is expensive, and it still doesn't protect the data against something that affects all the components in the same location, such as a natural disaster. Because of this, it is safer to replicate the data across systems with different properties, such as physical location.

Two of the most widely used methods for redundancy are replication, which means storing several copies of the data, and parity schemes such as the one in RAID. Erasure codes provide redundancy without requiring the high storage overhead of replication [22]. An erasure code divides an object into m fragments and recodes them into n fragments, where $n > m$. Then, the original object may be reconstructed from any m fragments of those n. The storage cost will be increased by a factor of $\frac{n}{m}$.

For example, if we have a file with a size of 10Mb and encode it using an erasure code with $m = 2$ and $n = 6$, we will have 6 fragments with 5Mb each, and we will only need any 2 of those 6 fragments in order to reconstruct the file. The total storage required for this would be $6 * 5 = 10 * \frac{6}{2} = 30Mb$.

Normal replication can be seen as a special case of erasure coding, with $m = 1$ and $n = replicas$. Likewise, parity schemes such as the one used in RAID can be seen as a erasure code with $n = x + 1$ and $m = x$.

The advantages of erasure coding over normal replication are:

- Allows a larger flexibility placing the files, since it can be divided into several fragments.
- Has been shown to allow a much larger MTTF (Mean Time To Failure) for the same storage overhead.
- Can prevent data theft, since it is necessary for an attacker to retrieve several fragments in order to reconstruct the file [24].

The disadvantages are:

- Changes to the file require its re-encoding and are therefore very costly. Fortunately, this is not an issue in preservation systems.
- It's necessary to have some mechanism to detect corrupted or failed fragments, or it will be necessary to try many different combinations of fragments for the correct original file to be reconstructed.
- In case a fragment is corrupted or fails, more effort will be required to reconstruct that fragment when the system is recovering. Instead of just copying the file

from a replica, the missing fragment will have to be computed again from the original file.

- Retrieving the file requires access to several components in order to obtain the required number of fragments. This problem can arise in the above case, when the system is recovering a missing fragment, if no copy of the original file is maintained.

2.1.2 Auditing

Auditing helps to detect latent faults which can otherwise take a lot more time or be impossible to detect. This can allow the system to recover faster and reduce the chance of losses. For example, faults that cause data loss may only be detected when the data is accessed, which can be done by the audit system periodically. This is especially important in digital preservation systems where there are usually few accesses to the data by clients.

The use of erasure codes can also make auditing more difficult, since no two nodes will store the same file [24].

2.1.3 Diversity

Failures in distributed systems are far from independent. Diversifying the properties of the components helps limit the number of simultaneous failures in the system and can be used to design a replication strategy which is more likely to survive a large correlated failure, such as in the case of a worm outbreak. There are several properties which can be diversified.

Physical location Placing nodes in different geographic locations can limit the number of simultaneous failures in case of a power loss or natural disaster.

Software Operating systems and other software components can have vulnerabilities which leave the system subject to attack or infection due to worms or viruses. Also, diversifying software can help prevent vendor lock-in.

Hardware Likewise, similar hardware components can have flaws which cause it to fail under certain conditions. Vendor lock-in can also be a problem if the hardware stops being supported and is impossible to replace.

Administration If replicas are administered independently it becomes harder for a single person to compromise the entire system, either through human error or attack.

Storage media Also using offline media for backup means that not all the components will be connected to a single network, the Internet.

Funding Diversifying the sources of funding for a digital preservation system means that it will be more likely to stay in operation even if there are problems in the organizations which support it and have more budget stability.

2.2 Self-Management

Self-managing systems automatically adapt their behaviour to the environment and working conditions, such as available bandwidth and probability of node failures. These factors can be hard to predict and change a lot with time, especially in complex systems such as distributed systems. Also, managing the intricate trade-offs involved when configuring such a complex system is a difficult job which few people can perform, especially if the conditions are constantly changing. For example, the replication level in a distributed storage system may be inadequate if it is static and it will also be very difficult for an administrator to know which level to choose if it is configurable [6].

The *Autonomic Computing* initiative [13] has the goal of creating self-managing systems. Autonomic systems should be *self-configuring*, *self-healing*, *self-protecting* and *self-optimizing* [12]. In data storage systems this can mean, for example, automatically choosing an adequate replication strategy (self-configuration), creating a new replicas when existing ones fail (self-healing), discovering compromised hosts (self-protection) and adjusting failure models as failures are detected (self-optimizing).

Early elements self-management, also called introspection, can be found in *AutoRAID*. This system provided different redundancy modes and switched between them depending on the workload [23].

The system proposed in [12] designs data storage systems based on user requirements (e.g. reliability and price), failure model parameters (e.g. failure frequency and correlation) and available protection techniques (e.g. backups and erasure codes). The system then provides a data protection design which meets the user requirements for the failure model provided, using the specified protection techniques.

Much of the research on self-managing systems comes from *p2p* (peer-to-peer) systems, since they have to deal with a massive number of unreliable nodes [8]. In *Total Recall* [6] users simply set a level of availability for each file. The system then automatically monitors the hosts' availability to predict their future availability at multiple time scales. This information is used to manage the redundancy and repair strategies, that can also depend on other factors, such as file size. *Oceanstore* [15] uses information about servers' workloads to adjust the number of data replicas and global usage trends to optimize itself.

Other systems with self-managing capabilities include *FARSite* [1] and *WiND* [3].

2.3 Distributed storage systems

The SafeStore system [14] stores data both in a local server and across several autonomous Storage Service Providers (SSPs). The local server is used as a cache and write buffer. The data is stored redundantly in the SSPs, which should be chosen in a way that minimizes the probability of correlated failures. For example, they should have different geographic locations and belong to different organizations. The SSPs can be outsourced or belong to the data owner. An audit system is used to detect corrupted data and to limit the effect that poorly-run SSPs can have in the system. Erasure

coding is used for both inter-SSP and intra-SSP redundancy.

The Phoenix system [11] replicates data across hosts with different characteristics, such as OS or provided services. The motivation for this is the observation that, nowadays, the Internet is very vulnerable to epidemics such as worms. Most of the times, these epidemics are only able to spread to nodes which have a certain characteristic; for example, running a flawed Web server software which contains a vulnerability. Because of that, it is argued that *Internet catastrophes result from shared vulnerabilities* [11].

Glacier [10] uses massive redundancy to prevent data loss on large scale correlated failures. It's argued that it is very difficult to create an accurate failure model for large scale systems which can predict all the rare catastrophes that can affect them. Therefore, the system should be protected against any type of correlated failures. Glacier uses erasure coding for redundancy. The redundancy level depends on the required durability for the system and the maximal correlated failure fraction (the maximum number of nodes which is assumed to be possible to fail at the same time).

LOCKSS [16] is a system where several independent peers collaborate with each other to preserve content. The techniques used for this imitate those used by librarians for physical documents.

The Google File System (GFS) is a distributed file system used by Google [9]. Google's clusters usually consist of commodity hardware, therefore GFS is designed to handle common failures. The GFS architecture consists in a single master and several chunkservers. The master stores all the file system metadata, which includes the mapping of the files to chunks and which chunkservers store each chunk. The master communicates with chunkservers periodically to give them instructions and collect their state. To read or write a file, clients only contact the master to know which chunkservers they should interact with. All the data transfer is done directly between the clients and chunkservers. Having a single master allows for sophisticated replication and chunk placement decisions. Currently, GFS only uses simple replication, but the use of erasure codes is being studied.

3. PROBLEM DESCRIPTION

As described earlier, our work consist in studying redundancy strategies to be used in a distributed storage system using data grids. Therefore, we will seek to answer the question "Which strategy is the best to be used in a certain system?". This raises two new questions:

- Which redundancy strategies should we consider using?
- What will be the behaviour of system X if strategy Y is used?

If we can answer these two questions, then we can compare several strategies and decide which is best for each case. To do this, we must first define a model of such a storage system and identify the threats to it.

3.1 System model

To be able to simulate the behaviour of a distributed storage system, we must first identify its components and how they interact with each other. The main components we have identified are:

Files The information which must be stored. If replication is used, a file can have one or more replicas. They also have a size and an unique identifier.

Storage resources Anything that can be used to store files, such as disks or RAID arrays. Resources have a capacity and an unique identifier.

Resource network Connects the storage resources. A resource network can be represented as a graph where the edges are links and the vertices are storage resources or connection points. These connection points can represent routers or networks which work as "clouds", such as the Internet. Each link has a capacity in each direction, which defines the speed at which files can be transfered through it. We choose to ignore link latency since this usually is not important for file transfer speed, as long as the files are of significant size. We also ignore the protocol stack which is used in the network, therefore the link capacity should be the actual capacity which can be used to transfer files.

We have also identified other things which affect the behaviour of a distributed storage system: the failure scenario, which is the set of all failures which occur during the system's lifetime and the redundancy strategy, which are the actions that are performed by the system in reaction to a certain state.

We have identified two types of failures which can affect the storage resources: unavailability and data loss. Unavailability means that the resource becomes temporarily or permanently unreachable, which can be caused, for example, by a power failure. Data loss means that the storage resource looses data which is part of the replication strategy, such as when a file replica becomes corrupted. Both these failures can happen simultaneously. For instance, when a server's disk fails and needs replacement, the server will be unavailable for some period of time and the disk's data will be lost.

There are also failures which can affect the resource network, such as links becoming unavailable or slower.

3.2 Threats

There are several threats to distributed storage systems which can hamper their ability to reliably store files.

3.2.1 Component errors

There can be several types of problems with system components: *(i) Storage media* can fail with time, losing either all data through disk crashes or some of it due to *bit rot*. The later may not be detected until there is an attempt to access the data. *(ii) Hardware components* can suffer transient recoverable failures, such as a power loss, or permanent irrecoverable ones, such as a power supply unit burning out. *(iii) Software components* can have bugs which can cause

failures, for example, the firmware in hard drives. *(iv) The communication network* can also fail and communication become unavailable or severely hampered. Some causes for this are failures in network components or congestion. *(v) Network services* on which the system is dependent in order to work may fail. For example, name resolvers such as DNS may not work correctly or even be discontinued in the future [19, 5, 4].

3.2.2 Obsolescence

Obsolescence can affect both hardware and software components. *Hardware components* can become obsolete and not be able to communicate with other system components or to be replaced in case of failure. Removable storage media, in particular, is only useful while a reader for it is available, becoming obsolete when it is not. *Software obsolescence* often manifests itself as format obsolescence, when the bits of a file can still be read but not decoded by any existing application. Another example of this problem is when software is stored for preservation but the existing hardware is unable to execute it.

3.2.3 Human operation errors

Human errors are inevitable [18] and human operators will always be involved with systems. It has been shown that operators can be the leading cause of failure in different types of systems [17]. People often delete by mistake data which is still needed. Besides, humans can cause failures in other components such as hardware (accidentally disconnecting a power cable) or software (uninstalling a needed software library).

3.2.4 Natural Disasters

Natural disasters such as earthquakes, floods or fires can cause failures in many components simultaneously. For example, an earthquake may cause a data center to be destroyed or a wide-scale power failure.

3.2.5 Attacks

Attacks can have several goals: data destruction, denial of service, theft, modification of data. Some of the motivations for the attacks are political, ideological or financial reasons. They can be done both by outsiders and insiders who have full access to the system; they can be illegal or legal; and they can be quick and short-term or continued, long term attacks by dedicated individuals or organizations. Also, the fact that many Digital Preservation systems are connected to the internet makes them vulnerable to automatic attacks by *worms* or viruses.

3.2.6 Management errors

Management can cause DP systems to fail. The primary resource needed for DP is money [20], therefore there is a high risk of *economic failure*. Costs with power, cooling and administration make digital information more vulnerable to lack of money than paper one. The budget for digital preservation systems may change a lot or even be completely cut off at some point. *Organizational failures* are also possible. An organization responsible for preserving content may dissapear or cease to be responsible for it. In this case, it should be possible to transfer the administration of the system to some other organization or copy the data entirely to some other system.

3.2.7 Threat visibility and independence

It cannot be assumed all that failures will be visible immediately. Many times a fault goes unnoticed for a long time. For example, a file format may only be discovered to be obsolete when it is necessary to access it, or a sucessful data corruption attack may never be discovered.

It's also incorrect to assume that failures are usually independent [5]. A power failure will cause many components who use it to fail at the same time; a worm or virus may affect only computers that are running a specific operating system; and a natural disaster like a blizzard will affect many nodes in a certain geographic area no matter what is done. The probability of being affected by a fault can also depend on characteristics of the component. For example, a computer running windows may be more vulnerable to viruses. It can even vary with time. A component in Florida is more likely to fail during hurricane season and long cold winters can cause frequent power failures in certain areas.

3.3 Replication and erasure coding

Erasure coding is a very promising tool which can be used for redundancy. However, it is a lot more complex than simple replication, and most data grids nowadays don't support it yet. Because of this, it would be unenviable for us to build a system supporting erasure coding in the time we have available. Therefore, this paper only focuses on redundancy strategies that use simple replication.

4. PROPOSED SOLUTION

We have developed a system to analyse and implement redundacy strategies in an environment subject to correlated failures. This solution has three components: a simulator to analyse the long term performance of a strategy, a redundancy system to implement it and mechanisms to adapt the strategy to the real system's behaviour. We have also developed redundancy strategies that take into account correlated failures and models to represent them.

The behaviour of a distributed storage system is too complex to simply be analysed analytically, and implementing a real life prototype is not feasible, since the lifetime of digital preservation storage systems is usually very large. Therefore, we have developed Serapeum, a simulator which enables the analysis of the behaviour and efficiency of a distributed storage system in different failure scenarios using a certain replication strategy.

We have also developed a redundancy system which allows the use of these replication algorithms in real life grid systems. For now, this system can be used with the iRODS grid technology (although with some limitations, due to the iRODS APIs still being very recent) but it can easily be extended to work with other grids. Also, the replication algorithms are defined in the same way for this system and for Serapeum, which means that it is only necessary to implement them once for testing and real life usage. Altough iRODS has mechanisms which would allow such a system to be implemented in the grid itself, we have chosen to implement it as a separate process. This makes integration with

other grid technologies easier and provides greater fault tolerance.

Due to the large number of correlated failures that exist in these systems, we propose algorithms which attempt to take advantage of the system's diversity by taking these correlations into account, thereby reducing the probability of data loss. This requires modelling of the failures which may affect the system, which is out of the scope of our work. Therefore, we have chosen to propose a way in which to represent these models and develop algorithms which can work with models developed later on. We define two different types of models: "failure models", which are used to generate failure scenarios for Serapeum simulations and "predictive models", which are used by replication algorithms to know which nodes are more likely to fail simultaneously and place replicas according to that. We also propose introspection mechanisms which can sometimes be used to compensate for incorrect models.

The replication algorithms work by receiving a view of the grid's state and returning a set of operations to be performed on the grid. The grid's state is characterized by the components described in 3.1: the files (their identifier, size and replica location), the storage resources (their identifier and capacity) and the resource network. The latter is optional, since algorithms can make decisions about replica placement without knowledge about the capacity of the links between storage resources. The grid's state can be affected by the performed operations or by external causes, which include failures. Currently, algorithms return two types of operations: replica creation (through a file copy) and replica deletion. Another useful operation could be copy cancelation, however, it is not supported yet, since the iRODS grid technology doesn't also support this as of now. We support most of the failures described earlier, namely temporary and permanent storage resource unavailability and data loss, however we still don't support failures in the resource network.

We have chosen to create a system in which decisions regarding replication are taken centrally. This means that there is a central system which collects all the information from the system and makes decisions based on it. We have taken this option because it allows replication strategies to be implemented much easier than in a distributed system. Also, it appears that the main advantages of a distributed approach, such as the ability to use computational resources from several nodes, aren't very relevant in a storage system used for digital preservation, where immediate data availability is not critical. A centralized system may also be less intrusive, since it will only be necessary to change the software configuration of one node to install it.

5. SERAPEUM

Serapeum allows the simulation of the behaviour of a distributed storage system with centralized replication decision-making. It is built in Java and can be used both to study properties of replication algorithms and the planning of a real life deployment of such a storage system.

As input, Serapeum receives a description of the system components defined in 3.1, the failures which will occur during the simulated lifetime, the replication algorithm to be used, the simulation time and which statistics should be retrieved from the simulation's data. Some of this information is passed as XML files and other simply as parameters. As of now, it is not possible to simulate the ingestion of additional files during the system's lifetime.

	Physical location	Software
Resource 0	Lisbon	Windows
Resource 1	Lisbon	Unix
Resource 2	Lisbon	Unix
Resource 3	Stockholm	Windows

Table 1: Values of attributes for each resource

With this information, two system states are created: a "real state", which represents the simulated state of the system, and a "visible state", which is available to the replication algorithm. As would happen in a real system, the "visible state" is passed on to the replication algorithm, which returns a set of operations that are then simulated and affect both states. This is done cyclically until the simulation time is over. While this is happening, information is also being retrieved to generate the required statistics. When the simulation time is over, these statistics are returned.

We have implemented the retrieval of statistics regarding the number of file losses, the total bandwidth usage and average replica number through time. However, the retrieval of other statistics is easy, since they are simply implemented as classes containing functions that are called when relevant events happen, such a new replica being created.

5.1 Failure models

Although it is possible to define the failures for a simulation one-by-one, this is usually done with failure models. We have developed a tool which generates lists of failures from these models. In this section are described the failure models which we developed to test our algorithms and that may in the future be better parametrized to represent the real behaviour of failures. Each of these models can represent one or more causes for failures.

5.1.1 Independent failures

This model is used to generate failures for a single resource. Both data loss and unavailability failures can be generated, the latter being either permanent of temporary. The time between failures can be generated using an exponential or Weibull distribution.

This model can be used to model the failure and replacement of disks which, according to [21], follow a Weibull distribution, causing temporary unavailability and total data loss in a storage resource.

5.1.2 Attribute-based failure model

This failure model is an extension of the one proposed in [11]. In this model, several attributes can be defined, such as "physical location". Each attribute may have several possible values, in this case, these could be "Lisbon" or "Stockholm". Each storage resource is then assigned a value, as illustrated in table 1.

	Natural disaster		Power loss	
	Frequency	Effects	Frequency	Effects
Lisbon	100 years	Table 3	6 months	Table 3
Estocolmo	400 years	Table 3	2 years	Table 3

Table 2: Threats affecting the "physical location" attribute

P	Data loss	Return	Recovery time
5%	-	-	Permanent
35%	Yes	Non-simultaneous	4 months
60%	No	Simultaneous	1 month
2%	Yes	Non-simultaneous	4 months
98%	No	Simultaneous	6 hours

Table 3: Effects of a natural disaster (top) and power loss (bottom)

As represented in table 2, a set of threats to these values is defined, which consists on the frequency of events caused by those threats and their effects on the system.

When an event occurs, it has an affect on the system resources which have the value that caused the event. For instance, in this example, we define that a natural disaster affects the resources with the "Lisbon" value for "physical location" every 100 years. Table 3 , for example, represent the possible effects of a natural disaster and power loss. The first column is the probability of that effect affecting each resource, the second whether data loss occurs in the affected storage resources or not, the third if their return is simultaneous or not (for example, when a power loss occurs, the resources will typically be available again at the same time, which doesn't happen when several nodes are damaged and must be replaced) and the fourth represents the average time until the affected resources are again available. When an event occurs, each affected resource will suffer one of these effects according to its probability.

5.1.3 Other correlated failures

This model can be used to add unexpected correlated failures to a simulation. It has two parameters: the failure rate λ and the correlation level $0 < c \leq 1$. The time between events that cause failures follow an exponential distribution of parameter λ and the number of nodes that fail in a single event follows a geometric distribution of parameter c. The nodes which are affected are then selected randomly.

Table 4 shows the probability that a certain number of resources will fail due to a single event in a system with a total of 8 resources and when $c = 0.6$. Also represented is the time between failures which affect that number of nodes when $\lambda = 1\ month$.

Number	P	Time (months)
1	60.00%	1.67
2	24.00%	4.17
3	9.60%	10.42
4	3.84%	26.04
5	1.54%	65.10
6	0.61%	162.76
7	0.25%	406.90
8	0.10%	1017.25

Table 4: Failure probability of a number of resources

6. REDUNDANCY SYSTEM

A replication system has been implemented as a service which interacts with data grids through specific drivers. The replication systems' replication algorithms are the same as Serapeum's, which means that an algorithm can be evaluated using Serapeum and immediatly be put to use in a real system.

The drivers enable the replication system to obtain information about the system's resources and files and to perform operations on the data grid, such as replica creation and deletion. A driver is responsible for detecting changes on the grid's state (such as a new file being added) and updating the replication system's view of that state through function calls. That view will then be used by the replication algorithm to generate a set of operations, such as replica creation, which will be performed on the grid by the driver.

We have partially implemented a driver for iRODS data grids using its Java API (Jargon). This driver supports replica creation, the detection of new files in the system and, thanks to the work developed in [2], of failures of storage resources. Due to limitations in Jargon, it still doesn't support replica deletion. However, work in Jargon and iRODS is in progress, which means that a complete driver can be expected soon.

7. STRATEGIES

Replication algorithms receive the storage system's state and return a set of operations to be performed on it, which can be the creation of new replicas through a file copy from one resource to another or their deletion.

The algorithms proposed in this work can be divided into two main groups: those that use a fixed and variable number of replicas. In both cases, heuristics are used for replica placement and, in the latter, also to decide the replication level. The most important component of these heuristics are metrics based on the predictive models, which attempt to measure the probability of file loss. These metrics are described in the next section.

In our way to describe predictive models, we assume that there are events which cause the simultaneous failure of one of more storage resources. Given a certain set of storage resources, a predictive model should be able to return two things: the average time between events which cause data loss in all those resources and the average time between events which cause data loss in all those resources exclusively, that is, in which no other resource suffers data loss. Also, they should be able to return the average time until the first recovery of a failed resource. The predictive models we have implemented are described in 7.2

7.1 Metrics

Metrics which can be used to make decisions on replica placement include the usage of storage resources, the available bandwidth and the predicted mean time between file

losses. We have defined two ways in which to estimate the latter.

The first is based on the work developed in [11] and is called "time between failures" (TBF). Using one or more predictive models, the TBF for a certain file is calculated as the average time between events which cause the failure of all the resources containing that file. To calculate this, it is simply necessary to ask each failure model for the frequency of events causing data loss in those resources and add those frequencies.

The problem with this metric is that it simply measures if there's nothing in common between *all* the resources in the set. For example, if storage resources could have one of the operating systems "Windows", "Linux" and "Solaris", this metric would have the same value if two of them were "Windows" and one "Linux" or if each of them had a different operating system, since there would be no event which affected two operating systems at the same time.

Therefore, we have defined a new metric: the "time between double failures" (TBDF). For some file, this metric measures the average time until a sequence of two events which cause data loss in all resources containing that file, in which the second failure occurs before any resource affected by the first failure has time to recover. For instance, if a file is stored in the resources {A,B,C}, this metric includes not only the failures which affect {A,B,C}, but also sequences of two failures which affect first a subset of the nodes (such as {A,B}) and then the rest of them (in this case, {C}), however, only those sequences where C fails before either A or B can recover are relevant.

7.2 Predictive models

The predictive models we have developed are related to the failure models described earlier, since they allow the prediction of the failures generated by them.

The independent failures predictive model simply returns the average of the distribution being used to generate the failures. A particular case of this is the modelling of disk failures and replacement, which according to [21], follows a Weibull distribution.

The attribute based predictive model looks for common attribute values in the given set of nodes and, using information regarding the threats to those attributes and their effects, calculates the average time between events which cause data loss on all the given nodes, either exclusively or not.

Finally, the other correlated failures predictive model can use a hipergeometric distributions to calculate the time between failures of a set of nodes with a certain size. An example of this is given on table X, where the time between both exclusive and not exclusive failures with data loss is represented when $\lambda = 1\ month$, $c = 0.6$ and there's a total of 8 resources.

7.3 Introspection

We have implemented two introspection mechanisms which change the redundancy system's behaviour according to the failures observed in the storage system.

Resources	Non-exclusive	Exclusive
1	4.82	13.33
2	25.85	116.67
3	83.94	583.33
4	193.27	1822.92
5	352.46	3645.83
6	550.40	4557.29
7	775.05	3255.21
8	1017.25	1017.25

Table 5: Average time in months between failures with data loss of a number of resources

The first allows the adjustment of the predictive models being used so that they better reflect the failures measured in the system. We have only implemented a proof of concept of this, which allows the adjustment of the "average time between failures" parameter of independent failure predictive models. An initial value for this parameter is defined, and also a confidence value for that prediction. As the system runs and failures are detected, we also define a confidence value for the observed failures, which is calculated as $c = \frac{observation\ time}{observation\ time + initial\ confidence\ value)}$, the new prediction for the average time between failures parameter is then calculated as: $observed\ frequency \cdot c + initial\ prediction \cdot (1-c)$.

The other introspection mechanism allows Serapeum to be used as part of the replication strategy. With this mechanism, simulations are run periodically using the adjusted models to compare the performance of several replication algorithms and parameters in this new scenario. Different metrics can be used to compare the performance, for instance, one may define a requirement for maximum data loss and define the best algorithm as the one which complies to that requirement and uses the least total bandwidth.

8. RESULTS

In this section we present the results of three experiments. In the first, we demonstrate how Serapeum can be used for planning by comparing the efficiency of different replication algorithms in an idealized system. In the other two, we demonstrate how our introspection mechanisms adjust a system's replication strategy to its real working conditions.

8.1 Simulated scenario

The system used in these experiments consists of 14 storage resources with a capacity of 500Gb. These resources are connected to a single network by links with bidirectional capacity of 100Mbit/s. The stored collection has 208 files of 5GB each, for a total size of 1TB. This large file size was chosen because it allows simulations to be ran faster. Different failure models are used to generate the failure scenarios for each experiment.

8.2 Comparison of replication algorithms

Several failure models are used in this experiment:

- Independent failure model parametrized to represent disk failures and replacement

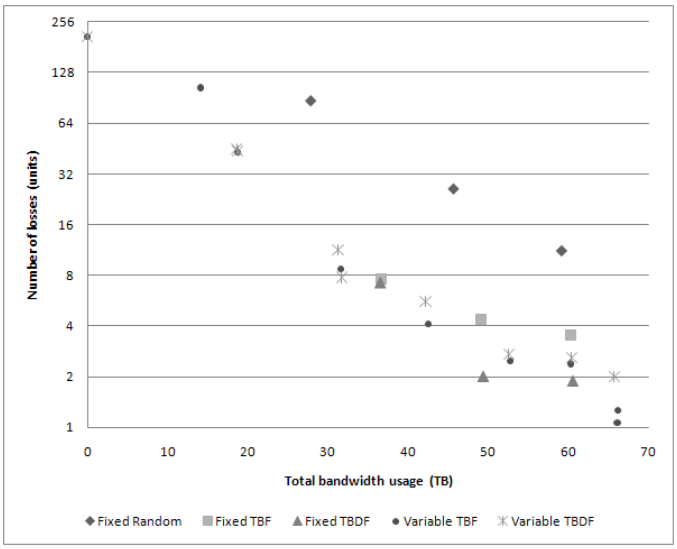

Figure 1: File losses and bandwidth usage

- Independent failure model which generates unavailability failures without data loss for each storage resource with an average frequency of 1 year and return time of 12h
- Attribute based failure model to generate correlated failures. This model is parametrized with three attributes: "room", "operating system" and "software services". The storage resources are spread across three different rooms and each has one of three possible operating systems. For each operating system there are four possible software services which the resources may or not support. The threats to any room are natural disasters, power losses, communication failures and physical attacks, each of which has different effects on the storage resources. The threats to the operating systems and software services are *worms* or viruses, and have different frequencies for each of them.

We have simulated the behaviour of the system for 50 years and measured the number of file losses and the total bandwidth usage when operating with different algorithms and replication levels. The less the file losses and bandwidth usage are, the more efficient an algorithm is. 100 simulations were ran for each case.

Both fixed and variable replica number algorithms were tested, using the TBF and TBDF heuristics. Also tested was a fixed replica number algorithm which places replicas randomly. Values of 3, 4 and 5 replicas were used for the fixed algorithms and several replication thresholds for the variable ones. The predictive models used by the algorithms were parametrized in the same way as the ones used to generate the failures and an additional "other correlated failures" model was added for the variable replica number algorithms. This is done to increase the number of replicas created by these algorithms, which is otherwise very low.

Figure 1 shows the number of file losses and total bandwidth usage for each algorithm. As expected, the random algorithms are outperformed by all others, which clearly shows the advantage of taking into account correlations between failures in the replication strategy. The performance of the variable and fixed replica number algorithms is simular, however, the former allow a greater granularity defining the replication level. In the fixed algorithms, the TBDF heuristic achieves better results than TBF for 4 and 5 replicas. When using a variable number of replicas, however, the TBF heuristic seems to slightly outperform it.

Number of resources	Time between failures
2	6 months
3	1 year
4	2 years
3	5 years
2	10 years
Total: 14	Average: 40 months

Table 6: Average time between resource failures

It's interesting to notice that, while adding a 5th replica exponentially decreases the number of losses experienced by the random algorithm, it doesn't have much effect in the other ones.

8.3 Predictive model adjustment

In this experiment we show how predictive models adjust to the observed failures. We generate independent failures in the 14 resources with average time between failures as described in table 6. However, the predictive models initially being used by the replication algorithm predict the failure frequency to be the same for all nodes (in this case, 40 months, which is the average of values in table 6). As failures are observed in the system, these models will be adjusted.

Figure 2 shows the average of the predicted values for each set of resources in table 6 throughout time. The confidence value for the inicial prediction is different in each of them, respectively, 1 year and 3 years. As we can see, the higher the confidence in the initial prediction, the less variation there is as failures are observed. Also, due to a larger sample size, the values for failures which are more common converge faster, which demonstrates the difficulty of using introspection to model rare events such as natural disasters.

Figure 3 shows the average file losses over 100 simulations with and without this introspection mechanism. As we can see, there is a slight improvement when introspection is used, even with such a simple mechanism as this one.

8.4 Replication strategy using Serapeum

In this experiment, we test the mechanism which allows automatically running simulations and changing the replication algorithm or parameters. We generate independent failures for all nodes with a frequency of 1 year, however, an initial value of 3 years is used on the predictive models. We set a threshold of 2 file losses per year which, according to the initial prediction, can be achieved with 3 replicas. However, 3 replicas will no longer be enough for this if the failure frequency is 1 year.

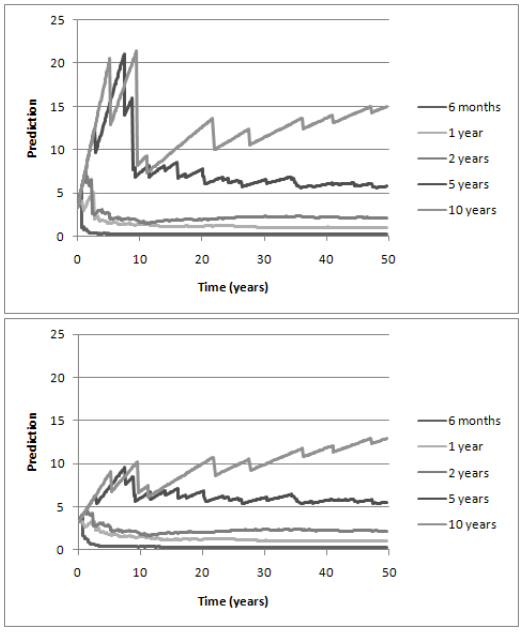

Figure 2: Prediction with confidence of 1 and 3 years

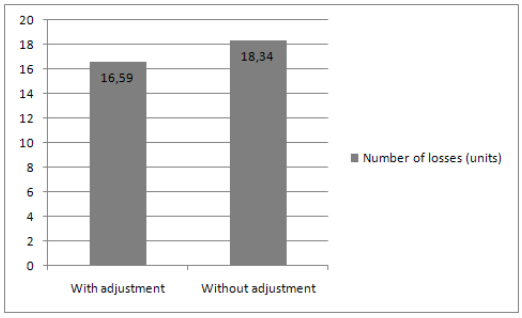

Figure 3: File losses with and without introspection

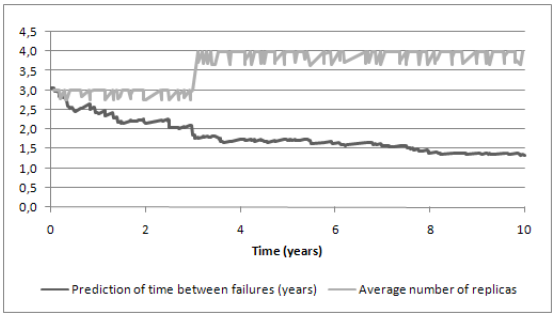

Figure 4: Automatic change in replication parameters

Figure 4 shows the adjustment of predictive models throughout time and also the average number of replicas in the system. As more failures are detected, the prediction gets closer to reality. Yearly, simulations are ran which, in this case, compare the performance of using 3 or 4 replicas. Initially, 3 replicas are enough to fulfill the file loss threshold and require less total bandwidth than 4, therefore that option is considered more efficient. However, as the models are adjusted and more failures are generated in the simulations, the results show that 3 replicas aren't enough, which causes the replication level to increase to 4 in year 3.

9. CONCLUSIONS AND FUTURE WORK

We developed a simulator that can be used to help planning the implementation of a storage system based on data grids, study new replication strategies and even be used as part of a replication strategy, automatically changing the replication algorithm and parameters according to the observed system behaviour.

We also developed a redundancy system which allows the implementation of these strategies in a real system. This system can be used with data grids, particularly, the iRODS data grid.

We have proposed replication strategies which take into account the correlations between failures in the storage resources of a data grid. These strategies have shown during testing to have a much better performance in terms of file losses and bandwidth usage than ones that simply place replicas randomly.

Our simulator and strategies are able to work with different failure models, so that the best ones may be used for each particular system. We developed failure models which may be parametrized to represent the failures of a real system. Also, we have implemented introspection mechanisms which can help compensate for some incorrect models.

However, there is still a lot of work that can be done. Most importantly, studying the use of strategies based on erasure coding, better modelling of the failures which affect grid systems, mechanisms which allow different grid technologies to interact so that multiple grid technologies may be used simultaneously by the replication system and further usage of introspection, for instance, to automatically detect network topology or similarities on the software configuration of the storage resources.

10. REFERENCES

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 1–14, New York, NY, USA, 2002. ACM.

[2] G. J. B. Antunes. GRITO: Utilização de clusters GRID para um sistema de preservação digital. Master's thesis, Instituto Superior Técnico, Portugal, 2008.

[3] A. Arpaci-dusseau, R. Arpaci-dusseau, J. Bent, B. Forney, S. Muthukrishnan, F. Popovici, and O. Zaki. Manageable storage via adaptation in wind. In *In Proceedings of IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid' 2001*, pages 169–177, 2001.

[4] M. Baker, K. Keeton, and S. Martin. Why traditional storage systems don't help us save stuff forever. *1st IEEE Workshop on Hot Topics in System Dependability*, June 30, 2005.

[5] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale. A fresh look at the reliability of long-term digital storage. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 221–234, New York, NY, USA, 2006. ACM.

[6] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total recall: system support for automated availability management. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[7] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. Raid: High-performance, reliable secondary storage, 1993.

[8] I. T. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *International Workshop on Peer-to-Peer Systems*, 2003.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press.

[10] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 143–158, Berkeley, CA, USA, 2005. USENIX Association.

[11] F. Junqueira, R. Bhagwan, A. Hevia, K. Marzullo, and G. M. Voelker. Surviving internet catastrophes. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 4–4, Berkeley, CA, USA, 2005. USENIX Association.

[12] K. Keeton and J. Wilkes. Automating data dependability. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 93–100, New York, NY, USA, 2002. ACM.

[13] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[14] R. Kotla, L. Alvisi, and M. Dahlin. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142. USENIX, 2007.

[15] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*. ACM, 2000.

[16] P. Maniatis and D. S. H. Rosenthal. The lockss peer-to-peer digital preservation system. *ACM Transactions on Computer Systems*, 23:2005, 2005.

[17] U. of California at Berkeley. Recovery oriented computing: A new research agenda for a new century. In *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, page 247, Washington, DC, USA, 2002. IEEE Computer Society.

[18] J. Reason. *Human error*. Cambridge University Press, 1990.

[19] D. S. H. Rosenthal, T. S. Robertson, T. Lipkis, V. Reich, and S. Morabito. Requirements for digital preservation systems: A bottom-up approach. *D-Lib Magazine 11*, 2005.

[20] C. Rusbridge. Excuse me... some digital preservation fallacies? http://www.ariadne.ac.uk/issue46/rusbridge/, 2006.

[21] B. Schroeder and G. A. Gibson. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, page 1, Berkeley, CA, USA, 2007. USENIX Association.

[22] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338, London, UK, 2002. Springer-Verlag.

[23] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 90–106. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[24] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, 2000.