INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

GRITO: Sistema de medições para suporte à preservação digital num ambiente heterogéneo

Manuel Bertrand Cabral

Dissertação para obtenção do Grau de Mestre em

Engenharia de Redes de Comunicações

Júri

Presidente: Prof. Dr. Luís Eduardo Teixeira Rodrigues

Orientador: Prof. Dr. José Luis Borbinha

Vogais: Prof. Dr. Pável Pereira Calado

Dezembro de 2008

Resumo

A preservação digital consiste em assegurar que objectos em formato digital se mantêm acessíveis ao longo do tempo. Para isto, é necessário utilizar um sistema de armazenamento que não sofra perdas de dados, o que só é possivel com a utilização de redundância. O projecto GRITO visa construir um sistema de armazenamento para preservação digital usando grids dedicadas à preservação e recursos excedentários de grids cujo propósito principal seja outro. Isto implica um ambiente heterogéneo, em que os nós podem ter características distintas e onde existem muitas falhas correlacionadas. O objectivo deste trabalho é estudar estratégias de redundância para um ambiente com falhas correlacionadas e a sua implementação num sistema GRITO.

Para isto, desenvolveu-se um sistema que permite analisar o desempenho de uma estratégia através de simulação, implementá-la no sistema real e monitorizá-lo para permitir a adaptação da estratégia a ele. Além disso, são propostas estratégias que têm em conta as falhas correlacionadas e modelos que podem ser parametrizados para representar as falhas que ocorrem num sistema. Tanto o sistema de simulação como as estratégias podem usar diferentes modelos de falhas, permitindo assim a utilização do que melhor se adapte a um sistema específico. Os nossos resultados mostram que as estratégias que têm em conta as correlações entre falhas obtêm melhor desempenho que as que não o fazem e que a adaptação automática das estratégias permite melhorar o seu desempenho em certos casos.

Palavras-chave: Preservação digital, grid de dados, redundância, falhas correlacionadas, simulação, introspecção

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

Agradecimentos

Ao Professor Rodrigo Rodrigues, pela sua orientação e interesse neste trabalho, tanto antes como depois de ter saído de Portugal e ao Professor José Borbinha também pela sua orientação e pelas óptimas condições proporcionadas para realizar este trabalho. Ao Gonçalo Antunes e José Barateiro pela ajuda e convívio ao longo deste trabalho.

Aos meus amigos, pelo apoio e bons momentos ao longo dos anos, sem os quais a minha vida seria seguramente muito diferente agora.

Aos meus colegas de curso e também grandes amigos, que tornaram este percurso muito, muito mais fácil. Ao Luís por tudo pelo que já passámos, ao André por me ter salvo a vida umas quantas vezes e pela pachorra para me aturar, ao Tiago pelo companheirismo e visão iluminada das coisas e ao Fred por estar disposto a ajudar em qualquer altura. Ainda ao Nizar, Matos, Edu, Luís Santos e restantes colegas pela sua amizade e ajuda ao longo do curso.

Finalmente, à minha família, em especial os meus irmãos, pais e avós, que sempre me apoiaram incondicionalmente.

Índice

Lista de Figuras

Lista de Tabelas

Lista de acrónimos

API	Application Programming Interface
DoS	Denial of Service
IE	Internet Explorer
IIS	Internet Information Services
GFS	Google File System
iRODS	Integrated Rule-Oriented Data System
LOCKSS	Lots of Copies Keep Stuff Safe
MTTF	Mean Time To Failure
P2P	Peer to Peer
RAID	Redundant Array of Inexpensive Disks
DoS	Denial of Service
SRB	Storage Resource Broker
XML	eXtensible Markup Language

1 Introdução

A preservação digital consiste em assegurar que conteúdos em formato digital se mantêm acessíveis ao longo do tempo. Para isto ser possível, é crítico utilizar um sistema de armazenamento que minimize as perdas de dados. Contudo, os suportes de armazenamento digital existentes hoje em dia (como discos, CDs ou fita) não são infalíveis, pelo que se torna necessário armazenar os dados com alguma redundância em vários deles. A utilização de redundância permite que os dados sobrevivam à falha de um suporte de armazenamento mas, por outro lado, faz com que seja mais dispendioso armazená-los, já que estes vão necessariamente ocupar mais espaço de armazenamento.

A abordagem do projecto GRITO a este problema passa pela criação de um sistema de armazenamento para preservação digital que utilize recursos de *data grids*. Este sistema será capaz de utilizar não só recursos de *data grids* instaladas especificamente para a preservação, mas também recursos excedentários de *data grids* cujo propósito principal seja outro.

A utilização de diferentes *grids* faz com que o ambiente em que um sistema GRITO vai operar possa ser muito heterogéneo. Por exemplo, alguns nós das *grids* podem ter configurações de software e localizações físicas distintas. Isto faz com que haja uma grande correlação entre as falhas que afectam o sistema. Por exemplo, um *worm* pode causar a falha num curto espaço de tempo de vários nós com configurações de software semelhantes. As falhas correlacionadas são particularmente destrutivas, pois podem afectar um grande número de nós simultaneamente [22].

Para se conseguir ter armazenamento fiável num sistema destes, é necessário utilizar uma estratégia de redundância. Esta consiste em definir que tipo de redundância utilizar e que acções tomar perante um determinado estado do sistema. Por exemplo, pode-se definir como estratégia de redundância utilizar replicação e, sempre que um ficheiro tiver menos que três réplicas no sistema, criar uma nova num recurso de armazenamento escolhido aleatoriamente, copiando o ficheiro de um outro que contenha uma das réplicas.

O objectivo deste trabalho é estudar e implementar estratégias de redundância de dados num sistema GRITO. Assim, é necessário desenvolver um conjunto de estratégias, uma maneira de comparar o seu desempenho e um sistema de redundância para utilização no GRITO. Estas estratégias devem ter em conta as falhas correlacionadas que afectam os componentes do sistema, já que estas ocorrem em grande número.

Neste trabalho foi desenvolvido um sistema para análise e implementação de estratégias de redundância para preservação digital num ambiente com falhas correlacionadas. Esta solução inclui três componentes:

- Simulação, usando o Serapeum, um simulador que permite analisar o desempenho de uma estratégia de redundância a longo prazo. As simulações podem ser feitas utilizando diferentes modelos para gerar as falhas, permitindo que se utilize o modelo que melhor se adapta a cada sistema em particular.
- Implementação, usando um sistema de redundância que permite implementar essa estratégia no sistema real usando o mesmo código da simulação, só sendo assim necessário implementá-la uma vez para teste e utilização real. Este sistema é desenhado para poder funcionar com diferentes tipos de *grids* de dados. Neste trabalho, foi desenvolvido suporte para a tecnologia de *grid* iRODS, que será inicialmente usada pelo GRITO.
- Introspecção, através de um mecanismo que permite o ajuste dos modelos utilizados às falhas observadas no sistema e o acesso das estratégias de redundância a essas medições. Utilizando este

mecanismo, pode-se implementar uma estratégia que se adapta às condições observadas no sistema.

Como referido anteriormente, as estratégias de redundância utilizadas num sistema destes devem ter em conta as falhas correlacionadas já que, caso contrario, poderão ser inúteis caso coloquem todas as réplicas de um ficheiro em nós com uma probabilidade elevada de falhar simultaneamente. Por isto, neste trabalho são propostas estratégias que têm em conta as correlações entre as falhas e que tentam minimizar a perda de ficheiros ao longo do tempo. Também estas estratégias podem utilizar diferentes modelos de falhas para prever as falhas correlacionadas. Neste trabalho são ainda propostos e implementados alguns modelos que poderão ser parametrizados pare representar as falhas afectam que um sistema específico.

Nesta tese apresenta-se o desenho e implementação do sistema, estratégias de redundância e modelos de falhas referidos acima. Utilizando o Serapeum, é feita uma comparação do desempenho das estratégias de redundância propostas ao de uma que não utilize conhecimento sobre a correlação das falhas. É também feita uma análise da influência de alguns factores no desempenho destas estratégias, como a correcção dos modelos de falhas utilizados. Demonstra-se ainda o funcionamento dos mecanismos de introspecção.

Os resultados obtidos experimentalmente mostram que ter em conta as falhas correlacionadas pode reduzir drasticamente o número de perdas de ficheiros num sistema, mas que isto apenas é possível utilizando modelos de falhas adequados. Mostra-se também que a utilização de introspecção pode ser usada para ajustar alguns modelos de falhas incorrectos e adaptar as estratégias de redundância às condições de funcionamento do sistema. Assim, a introspecção também pode contribuir para reduzir a quantidade de perdas de ficheiros que ocorre num sistema.

No capítulo 2 é apresentado o estado da arte nesta área, no capítulo 3 é feita uma descrição mais detalhada do problema que será abordado. No capítulo 4 é apresentado o sistema proposto e em 5 as estratégias e modelos de falhas implementados. No capítulo 6 são apresentadas as experiências efectuadas e resultados obtidos e, finalmente, em 7 são apresentadas as conclusões e o trabalho futuro a ser desenvolvido.

2 Estado da Arte

Neste capítulo são apresentados aspectos relevantes de um sistema de redundância para reduzir a sua perda de dados e custo de operação. Além disso, são também apresentados alguns sistemas de armazenamento distribuídos para preservação digital e outros que foram desenhados com requisitos semelhantes em mente.

2.1 Aspectos relevantes

2.1.1 Mecanismo de redundância

Se um ficheiro for armazenado num único recurso de armazenamento, este perder-se-á caso este recurso falhe, o que é muito provável que aconteça durante o tempo de vida longo de um sistema de preservação. Devido a isto, a maioria destes sistemas toma partido de uma característica básica da informação digital: poder ser copiada sem qualquer perda de informação. Isto significa que diversas cópias dos ficheiros podem ser guardadas em vários componentes.

Usar redundância reduz muito a probabilidade de existir perda de ficheiros, já que se torna necessário que existam falhas simultâneas em vários componentes para que um ficheiro seja perdido. Esta é uma das técnicas principais utilizadas para garantir a longevidade de dados, e é usada há muito tempo em sistemas RAID, que replicam dados em vários discos ou utilizam esquemas de paridade para assegurar que os dados não se perdem quando há a falha de um dos discos [10]. Infelizmente, o hardware para armazenar uma grande quantidade de dados desta forma é caro, e continua a não proteger os dados contra um evento que afecte todos os componentes que se encontram numa mesma localização física, como um desastre natural. Devido a isto, é mais seguro que os dados sejam replicados em vários sistemas com características diferentes, como por exemplo, a localização física.

Um dos grandes problemas num sistema replicado é manter a coerência entre as réplicas existentes quando existem actualizações aos dados, que podem ocorrer simultaneamente. Já foram desenvolvidas várias técnicas para resolver este problema, que envolvem um compromisso entre consistência (garantia de que os clientes vão ver um ficheiro como é suposto) e disponibilidade (quanto tempo é necessário até os clientes poderem aceder ao ficheiro) [29]. Contudo, na preservação digital os ficheiros não têm que ser actualizados e não é critico que exista disponibilidade imediata [21], pelo que estas técnicas não serão discutidas aqui.

2.1.1.1 “Erasure coding”

Duas das técnicas de redundância mais utilizadas são a replicação, que envolve armazenar várias cópias dos dados, e esquemas de paridade como os utilizados no RAID. Utilizando *erasure codes* pode-se ter redundância utilizando menos espaço de armazenamento que na replicação [34]. Um*erasure code* divide um ficheiro em m fragmentos e transforma-os em n fragmentos, onde $n > m$. O objecto original pode então ser reconstruído a partir de quaisquer m fragmentos desses n. Isto faz com que o espaço utilizado para armazenar o ficheiro aumente um factor de $\frac{n}{m}$.

Por exemplo, se um ficheiro de 10MB for armazenado utilizado um “erasure code” com $m = 2$ e $n = 6$, serão criados 6 fragmentos com 5MB cada, e apenas serão necessários quaisquer 2 fragmentos desses 6 para reconstruir o ficheiro. O espaço total de armazenamento necessário para isto seria $6*5 = 10*\frac{6}{2} = 30MB$.

A replicação normal pode ser vista como um caso particular de “erasure coding” em que $m = 1$ e $n =n^o$ *de replicas*. Esquemas de paridade como os do RAID também podem ser vistos como um *erasure code* com $n = x + 1$ e $m = x$.

A utilização de *erasure coding* tem as seguintes vantagens em relação á replicação normal:

- Permite uma maior flexibilidade na colocação os ficheiros em recursos de armazenamento, já que estes podem ser divididos em vários fragmentos.
- Tem um MTTF (tempo médio até perda) maior utilizando o mesmo espaço de armazenamento
- Pode ajudar a prevenir roubo de dados, já que um atacante tem que conseguir ter acesso a vários fragmentos para poder reconstruir o ficheiro.

Da mesma forma, as desvantagens da sua utilização são:

- Quando um ficheiro é alterado, é necessário voltar a calcular os seus fragmentos, o que torna esta operação muito dispendiosa. Felizmente, isto nao constitui um problema em sistemas de preservação digital.
- É necessário ter um sistema para detectar fragmentos corrompidos, ou será necessário tentar combinações diferentes de fragmentos para reconstruir o ficheiro original quando um deles se encontra corrompido.
- Caso um fragmento esteja corrompido ou falhe, será mais dispendioso reconstruí-lo para o sistema está a recuperar. Em vez de simplesmente copiar o ficheiro de uma réplica, o fragmento em falta terá que ser de novo calculado a partir do ficheiro original.
- A velocidade dos acessos é limitada pelo fragmento a que se demora mais tempo a aceder.
- Para obter um ficheiro do sistema, é necessário aceder a diferentes componentes para obter o número necessário de fragmentos. Este problema pode levar a uma latência maior nas operações quando um sistema está a recuperar um fragmento em falta.
- A implementação de *erasure codes* é normalmente mais complexa, pelo que pode introduzir bugs no sistema que levem à perda de dados.

2.1.2 Auditorias

As auditorias ajudam detectar falhas latentes que, caso contrário, poderiam não ser detectadas durante bastante tempo. Isto pode ajudar o sistema a recuperar mais rapidamente e a reduzir a probabilidade de haver perdas. Por exemplo, falhas que causem perdas de dados podem só ser detectadas quando há uma tentativa de aceder aos ficheiros, o que pode ser feito pelo sistema de auditoria periodicamente. Isto é especialmente importante em sistemas de preservação digital, onde normalmente os clientes raramente tentam aceder aos dados [6].

Muitas vezes, o armazenamento de dados pode ser delegado para uma entidade externa. Quando isto é feito, o dono dos dados deve ser capaz de auditar os sistemas dessa entidade para assegurar que os

dados estão a ser guardados devidamente, especialmente porque donos de sistemas de armazenamento desonestos podem não notificar o dono dos dados caso ocorra a perda destes. Isto significa que o sistema de auditoria deve ser desenhado de forma a não permitir que uma entidade externa esconda perdas de dados [19].

A utilização de *erasure codes* pode também tornar a auditoria mais difícil, já que nenhuns dois nós irão guardar ficheiros semelhantes [37].

2.1.3 Diversidade

As falhas que ocorrem nos sistemas distribuídos não são independentes umas das outras [6]. Diversificar as propriedades dos componentes ajuda-nos a limitar o numero simultâneo de falhas que ocorrem no sistema e pode ser usado para desenhar uma estratégia de redundância que permita ter um número baixo de perdas de dados, mesmo quando ocorre um evento que causa um numero elevado de falhas simultâneas, como por exemplo, um *worm*. Existem várias propriedades de um sistema que podem ser diversificadas [26].

- Diversificar a localização física pode limitar o número de falhas que ocorre caso haja uma falha de electricidade ou um desastre natural.
- Software, como o sistemas operativo, tem vulnerabilidades que fazem com que os sistemas possam ser atacados ou infectados por *worms* ou vírus. A diversificação do software também pode ajudar a prevenir a dependência de um fornecedor.
- Da mesma maneira, componentes de hardware podem ter problemas que causem a sua falha em certas condições. A dependência de um fornecedor também pode ser um problema caso o hardware deixe de ser suportado ou se torne impossível de substituir.
- Se as réplicas forem administradas independentemente, torna-se mais difícil que uma única pessoa ou organização comprometa todo o sistema, tanto devido a erros humanos como a ataques.
- Fazer backup para diferentes suportes de armazenamento, como fita, faz com que nem todos os componentes estejam ligados a uma única rede, a Internet.

2.1.4 Introspecção

Os sistemas com capacidade de introspecção (ou auto gestão) adaptam automaticamente o seu comportamento ao ambiente e condições em que funcionam, como a largura de banda disponível e probabilidade de existirem falhas. Estes factores podem ser difíceis de prever e variar muito com o tempo, especialmente em sistemas complexos como os sistemas distribuídos.

Gerir os compromissos envolvidos quando se configura um sistema complexo é um trabalho difícil, especialmente se as condições de funcionamento mudarem constantemente. Por exemplo, o nível de replicação num sistema de armazenamento distribuído pode não ser adequado caso seja estático. Mas mesmo que seja configurável, será também muito difícil para um administrador saber qual o nível apropriado [8].

A *computação autonómica* [18] tem o objectivo de criar sistemas com capacidades de auto-gestão. Os sistemas autonómicos devem ser auto-configuráveis, auto-regeneradores, auto-protectores e auto-optimizantes. Em sistemas de armazenamento de dados isto pode significar, por exemplo, escolher uma automaticamente

estratégia de replicação adequada (auto-configuração), criar automaticamente novas réplicas quando há falhas (auto-regeneração), detectar nós comprometidos (auto-protecção) e ajustar os modelos de falhas á medida que falhas são detectadas (auto-optimização). Para atingir estes quatro objectivos, um sistema deve ser consciente de si próprio e do ambiente em que opera, de forma a conseguir ter uma vista do seu estado actual. Ele deve então monitorizar esse estado e, se necessário, ajustar-se para atingir um estado desejável. Isto significa que um sistema com capacidades de auto-gestão deve ter conhecimento sobre os seus recursos disponíveis, componentes, características de performance desejáveis, estado actual e o estado da ligação com outro sistemas [32].

Alguns elementos primitivos de introspecção podem ser encontrados no AutoRAID. Este sistema fornecia diferentes modos de redundância e alternava entre eles conforme a carga [36],

O sistema proposto em [17] desenha sistemas de armazenamento de dados baseado em requisitos do utilizador (como fiabilidade e preço), modelos de falhas (como frequência das falhas e correlação) e técnicas de protecção disponíveis (como backups e *erasure coding*). Assim, o sistema fornece um desenho de um sistema que deve satisfazer os requisitos do utilizador para o modelo de falhas fornecido, utilizando as técnicas de protecção especificadas.

Muito do trabalho feito em sistemas com capacidades de introspecção vem dos sistemas *p2p* (peer-to-peer), já que estes têm que lidar com um número massivo de nós pouco fiáveis [13]. No *Total Recall* [8] os utilizadores limitam-se a definir um nível de disponibilidade desejado para cada ficheiro e o sistema monitoriza automaticamente os nós existes para prever a sua disponibilidade futura. Esta informação é utilizada para gerir as estratégias de redundância e reparação do sistema, que também podem depender de outros factores, como o tamanho dos ficheiros. O *Oceanstore* [20] utiliza informação sobre a carga dos servidores para ajustar o número de réplicas e sobre tendências de utilização global para se auto-optimizar. Por exemplo, este sistema consegue detectar conjuntos de nós que normalmente falham juntos e espalhar as réplicas de um ficheiro por grupos diferentes [35]. Este mecanismo foi testado em [22] onde se concluiu que, apesar de conseguir identificar conjuntos de nós para 99% das falhas, o 1% para o qual não funcionava incluia as falhas que afectam um número superior de nós. Ou seja, as falhas mais raras e mais destructivas, como as causadas por desastres naturais, não conseguem ser previstas com mecanismos deste género.

Outros exemplos de sistemas com capacidades de introspecção são o *FARSite* [2] e *WiND* [4].

2.1.5 Inércia

Um sistema que funciona rapidamente também falha rapidamente, especialmente quando é atacado. Como os sistemas de preservação digital não têm normalmente necessidade de um funcionamento muito rápido, podem ser desenhados de forma a que o seu estado não se altere muito rapidamente, independentemente do esforço que é aplicado sobre eles [21]. Isto faz com que um sistema tenha uma probabilidade menor de falhar completamente quando, por exemplo, há um ataque, já que dá mais tempo aos administradores do sistema para o detectar e tomar acções correctivas.

Obviamente, existem certos tipos de ataques e falhas que não podem ser atrasados, e por vezes é vantajoso que os sistemas de preservação digital actuem rápidamente, como quando estão a recuperar de uma falha. Contudo, este principio pode ser utilizado em partes do sistema para, por exemplo, limitar a

velocidade a que ficheiros podem ser apagados acidentalmente por um administrador.

O sistema LOCKSS utiliza este principio através de técnicas que permitem limitar a velocidade a que um atacante pode comprometer todo o sistema após comprometer um único nó.

2.2 Sistemas existentes

Nesta secção são apresentados sistemas de armazenamento distribuídos para preservação digital ou com requisitos semelhantes.

2.2.1 SafeStore

O SafeStore [19] armazena dados num servidor local e em vários Fornecedores de Serviços de Armazenamento (SSPs) autónomos. O servidor local é utilizado como uma cache e buffer de escrita. Os dados são armazenados nos SSPs com redundância, que devem ser escolhidos de forma a minimizar a probabilidade de falhas correlacionadas. Por exemplo, será útil que os SSPs tenham diferentes localizações geográficas e que pertençam a organizações diferentes. Os SSPs tanto podem pertencer ao dono dos dados como a terceiros. É utilizada uma interface restrita para aceder aos SSPs, o que limita o dano que pode ser causado por utilizadores pouco cuidadosos ou por atacantes, impedindo-os de apagar ou alterar dados livremente.

Um sistema de auditoria é usado para detectar dados corrompidos e limitar o efeito que SSPs de fraca qualidade podem ter no sistema. A maioria do trabalho de auditoria é realizado pelo SSP auditado, reduzindo assim a quantidade de recursos necessária no sistema. Como os SSPs podem pertencer a terceiros, o algoritmo utilizado foi desenhado para impedir que estes emitam respostas falsas nas auditorias.

"Erasure codes" são utilizados tanto para redundância inter-SSP e intra-SSP. O nível de redundância inter-SSP depende do nível interno utilizado por cada SSP, o que permite atingir uma maior durabilidade. Esta durabilidade pode ser ainda aumentada se os SSPs que o seu nível de redundância interno seja controlado pelo sistema.

2.2.2 Phoenix

O Phoenix [16] replica dados em nós com diferentes características, como o sistema operativo e os serviços fornecidos por ele. A razão para isto é que, hoje em dia, a Internet está muito vulnerável a epidemias como *worms* que, na maioria das vezes, apenas se espalham para nós que têm uma certa característica em comum, como correrem um servidor *web* que contem uma vulnerabilidade. Portanto, é dito que *as catástrofes na internet resultam de vulnerabilidades partilhadas* [16].

Para replicar dados desta forma, são dados atributos aos nós. Por exemplo, um nó pode ter os atributos *Windows*, *IIS* e *IE*, que se referem ao sistema operativo do nó, o seu servidor *web* e o seu *browser*. Isto pode ser feito automaticamente utilizando ferramentas como o *nmap*. Com esta informação, são utilizadas heurísticas para criar *cores*. Um *core* é um grupo de nós com configurações disjuntas, ou seja, em que não exista nenhum atributo que seja partilhado por todos os nós do *core*. As réplicas dos ficheiros são colocadas em todos os nós de um único *core*. Desta forma, mesmo que uma catástrofe afecte todos os nós com

um certo atributo, pelo menos um dos nós do *core* não será afectado e uma cópia do ficheiro sobreviverá. Quando um nó que falhou retorna, ele irá recuperar os dados de um dos nós que não foram afectados.

2.2.3 Glacier

O Glacier [15] utiliza replicação massiva para prevenir perdas de dados quando ocorre um grande número de falhas correlacionadas. Como é muito difícil criar modelos precisos para prever as catástrofes que podem causar falhas em sistemas de grande escala, tenta-se proteger o sistema contra todos os tipos de falhas correlacionadas. O Glacier foi desenhado apenas como sistema de armazenamento para durabilidade, sendo assumido que existe uma outra camada de armazenamento com um nível de redundância mais baixo para fornecer acesso aos dados aos clientes.

O Glacier utiliza *erasure codes* para redundância. O nível de redundância depende dos requisitos definidos de durabilidade do sistema e da fracção máxima de falhas correlacionadas (o número máximo de nós que se assume que podem falhar simultaneamente). Ambos estes parâmetros são configuráveis pelo administrador do sistema.

É assumido que os nós do sistema formam uma rede *overlay* que é capaz de mapear chaves numéricas ao endereço do nó responsável por essa chave. Um exemplo de uma rede destas é a fornecida por uma *hash table distribuída*, como o Chord [33] ou o Pastry [27].

Quando um ficheiro é ingerido, as chaves correspondentes às posições onde os fragmentos devem ser armazenados são calculadas de forma a que seja possível aceder-lhes conhecendo apenas a chave do ficheiro. Os fragmentos são armazenados nos nós de armazenamento responsáveis por essas chaves.

Como os nós podem ter baixa disponibilidade, as chaves pelas quais eles são responsáveis só são delegadas para outros nós após estes terem falhado durante um certo período de tempo. Ou seja, um nó pode ser considerado responsável por um conjunto de chaves mesmo não estando online. Aquando da ingestão de um ficheiro, caso o nó que devia ser responsável por um dos fragmentos não esteja online, esses fragmentos são descartados, já que podem ser recuperados mais tarde através de um sistema de recuperação. Se não existirem nós suficientes disponíveis, ou seja, menos que os necessários para armazenar fragmentos suficientes para recuperar o ficheiro, um dos vizinhos do nó armazena o fragmento e entrega-o mais tarde.

O mecanismo de recuperação permite aos nós recuperar fragmentos que têm em falta. A responsabilidade pelas chaves é atribuída de uma forma que permite que um nó saiba quais os restantes nós que irão armazenar fragmentos de ficheiros pelos quais ele também está responsável. Devido a isto, ele pode pedir a esses nós as listas dos ficheiros que eles armazenam, e assim saber que ficheiros têm fragmentos em falta nele próprio, podendo então recuperá-los.

2.2.4 LOCKSS

O LOCKSS [21] é um sistema em que diversos nós independentes colaboram uns com os outros para preservar conteúdos. As técnicas utilizadas para isto imitam as utilizadas por bibliotecários na vida real para documentos físicos.

O sistema LOCKSS existente é utilizado por bibliotecas para preservar acesso a revistas em formato elec-

trónico. As bibliotecas têm vários *web caches* de baixo custo e persistentes que obtêm conteúdos fazendo *crawling* de páginas *web*, distribuem-nos para os leitores locais e preservam-nos através da cooperação com outras caches que armazenem o mesmo conteúdo. A preservação é feita através de votações, em que vários nós votam sobre unidades de arquivo, que podem conter várias edições de uma revista. Se um nó perder uma votação, isto significa que a cópia do arquivo que ele armazena deve estar danificada. Quando isto ocorre, este nó pede que sejam feitas novas votações sobre partes da unidade de arquivo, para que seja localizada a parte danificada. Os outros nós podem cooperar com este nó fornecendo-lhe cópias dos dados danificados.

O desenho do protocolo de votação é crítico, já que deve prevenir perda de dados nos nós e fazer o sistema resistente a ataques. Devido a isto, este protocolo foi desenhado tendo em mente princípios muito conservadores e estritos.

2.2.5 Google File System

O Google File System (GFS) é um sistema de ficheiros distribuído utilizado pela Google [14]. Os clusters da Google são normalmente constituídos por hardware comum e barato. Por isso, o GFS é desenhado para suportar falhas frequentes. Os dados armazenados no GFS consistem normalmente em ficheiros grandes, que são divididos em partes de tamanho fixo. A maior parte das vezes, não há alterações aos ficheiros ou, se houver, são apenas adicionados dados ao final do ficheiro, sendo as escritas aleatórias extremamente raras. Isto é semelhante ao que acontece na preservação digital.

A arquitectura do GFS consiste num único servidor mestre e vários servidores de *chunks*. O servidor mestre guarda todos os metadados do sistema de ficheiros, que inclui o mapeamento dos ficheiros em *chunks* e que servidor guarda que *chunks*. O servidor mestre comunica com os servidores de *chunks* periodicamente para lhes dar instruções e pedir o seu estado, Para ler ou escrever um ficheiro, os clientes apenas contactam o mestre para saber com que servidor de *chunks* devem interagir. Todas as transferências de dados são feitas entre os clientes e os servidores de *chunks*.

Ter um único servidor mestre permite decisões sofisticadas sobre a redundância e colocação de *chunks*. De momento, o GFS utiliza apenas replicação, mas a utilização de *erasure coding* está a ser estudada. A colocação das réplicas é baseada em três critérios:

- As réplicas devem ser colocadas em servidores de *chunks* com utilização de disco abaixo da média
- O número de ficheiros criados recentemente num servidor de *chunks* deve ser limitada, já que a criação de ficheiros é normalmente seguida de um grande número de escritas
- As réplicas devem ser espalhadas por bastidores diferentes, já que algumas falhas, como falhas de electricidade, podem afectar um bastidor inteiro. Além disso, o débito da rede para cada bastidor é limitado, e espalhar os ficheiros pode permitir um balanceamento de carga melhor.

Os utilizadores podem definir o número de réplicas alvo para cada ficheiro. O número de réplicas pode estar abaixo do alvo caso haja uma falha ou o número alvo seja aumentado. Quando isto ocorre, devem ser criadas novas réplicas. A criação de réplicas é priorizada. Por exemplo, um ficheiro que esteja duas réplicas abaixo do alvo será normalmente replicado antes de um que esteja apenas uma. As réplicas são

também rebalanceadas periodicamente. Para criar réplicas, o servidor mestre diz ao servidor de *chunks* que irá armazenar a nova réplica para copiar o ficheiro de outro servidor de *chunks*. Novos servidores de *chunks* adicionados ao sistema serão enchidos quando novos ficheiros ou réplicas são criados ou quando houver rebalanceamento.

Tanto o servidor mestre como os de *chunks* estão desenhados para recuperar o seu estado e reiniciar o processo responsável pelo GFS rapidamente quando ocorre uma falha. Contudo, por vezes isto não é possível, como por exemplo quando a máquina falha completamente e não consegue voltar a corrê-lo. Devido a isto, o estado do servidor mestre é também replicado noutras máquinas. Um sistema externo ao GFS detecta a falha do mestre e lança um novo processo numa delas. A localização do mestre é descoberta utilizando DNS e a entrada correspondente é alterada para o endereço da nova máquina. Além disto, servidores chamados *shadow masters* fornecem acesso apenas de leitura aos ficheiros. Apesar destes servidores normalmente não estarem completamente sincronizados com o mestre, eles podem na mesma ser utilizados para encontrar servidores de *chunks* responsáveis por um certo ficheiro.

Os servidores de *chunks* utilizam *checksums* para verificar a integridade dos dados. Isto é feito quando há acessos aos ficheiros e também periodicamente, já que alguns ficheiros podem apenas ser acedidos raramente. O GFS utiliza logs para analisar o comportamento do sistema de forma a detectar bugs e aumentar a sua eficiência.

2.2.6 FarSite

O FarSite [9] é um sistema de ficheiros distribuído que utiliza recursos de armazenamento e de rede excedentários dos computadores desktop de uma organização. Este sistema cria um sistema de ficheiros fiável e seguro utilizando nós que, normalmente, não o são. O sistema visa obter uma disponibilidade de ficheiros alta, ou seja, tenta com que haja sempre pelo menos uma réplica de cada ficheiro disponível. Para isto, utiliza-se introspecção, monitorizando os nós do sistema e medindo a sua disponibilidade média.

Os algoritmos de replicação utilizados tentam distribuir as replicas de forma a que nenhum ficheiro tenha todas as replicas em nós com baixa disponibilidade [11]. Para isto, são feitas trocas de réplicas entre os nós á medida que a disponibilidade destes varia. Os algoritmos são executados de forma distribuida, ficando grupos pequenos e autónomos de nós responsáveis por conjuntos de ficheiros.

2.2.7 IrisStore

O sistema de armazenamento IrisStore [23] utiliza estratégias de redundância que têm em conta falhas correlacionadas. Para isto, é utilizado um modelo de falhas que prevê a frequência de falhas que afectam um determinado número de nós. Os parâmetros deste modelo são normalmente determinados através de medições prévias das falhas do sistema.

O IrisStore utiliza *erasure coding* e determina automaticamente quais os parâmetros de redundância mais adequados quando é configurado. Estes parâmetros incluem os valores m e n usados para o erasure coding. Para fazer esta configuração, define-se, além dos parâmetros do modelo de falhas, um requisito de disponibilidade dos ficheiros e uma função de custo. Os parâmetros de redundância são então escolhidos de

forma a minimizar este custo, cumprindo o requisito de disponibilidade. Esta função de custo recebe como entradas o overhead causado pela replicação e um nível de consistência dos dados.

Este sistema foi desenhado para ser utilizado em sistemas homogéneos, em que todos os nós têm configuração semelhantes. No futuro, está prevista a utilização de introspecção para ajustar o modelo de falhas automaticamente com base nas falhas observadas no sistema.

2.3 iRODS

As *grids* são sistemas de software que permitem a partilha de recursos distribuídos. As *data grids*, em particular, fazem-no para recursos de armazenamento [12].

O iRODS [1] é um sistema de software *data grid open source* desenvolvido pelo San Diego Supercomputer Center. O iRODS permite o acesso a recursos de armazenamento heterogéneos, como sistemas de ficheiros UNIX ou bases de dados, através de uma API uniforme, escondendo dos clientes a infraestrutura física da grid. Permite também que sejam efectuadas cópias de dados directamente de um recurso para outro, sem ser necessário estes passarem pelo cliente. Podem ser criadas federações iRODS, fazendo com que diferentes grids iRODS consigam interoperar entre elas. Com isto é possível, por exemplo, efectuar cópias de dados directamente entre grids diferentes.

Toda a informação necessária para o funcionamento dum sistema iRODS é guardada numa base de dados chamada iCAT. Uma instalação iRODS consiste num servidor iRODS que também corre esta base de dados, chamado *iCAT-enabled* e num número qualquer de outros servidores iRODS. Cada um destes servidores é responsável por um número qualquer de recursos de armazenamento. Quando um cliente pretende efectuar uma operação na grid, contacta um servidor iRODS qualquer que, de seguida, irá contactar o servidor iCAT-enabled para obter a informação necessária para a realização da operação, como qual é o servidor responsável pelo recurso onde um determinado ficheiro está armazenado. Quando existe transferência de ficheiros, como na inserção ou acesso a ficheiros, esta é feita directamente entre o cliente e o servidor onde estes estão ou serão armazenados.

O iRODS permite que se altere o seu comportamento em tempo de execução definindo regras. As regras são escritas numa linguagem específica do iRODS e cada uma define um conjunto de primitivas do sistema e quando é que estas devem ser executadas. Além disso, define também um outro conjunto de primitivas a ser executadas caso a execução do primeiro falhe. Estas primitivas são os microserviços. O iRODS traz já vários microserviços que permitem, por exemplo, fazer copias de ficheiros. Além disso, é possível definir novos microserviços usando a linguagem C. Actualmente, é necessário compilar os microserviços com o iRODS, mas no futuro estes poderão também ser adicionados em tempo de execução.

3 Descrição e análise do problema

A preservação digital consiste em assegurar que conteúdos em formato digital se mantêm acessíveis ao longo do tempo. Isto implica duas coisas: proteger os objectos digitais contra obsolescência tecnológica que impeça que os mesmos venham ser no futuro consumidos por humanos (por exemplo, como quando deixa de haver software capaz de interpretar um ficheiro com um certo formato) e armazenar estes objectos a longo-prazo, ou seja, assegurar que os seus ficheiros não se perdem ou são corrompidos.

A abordagem do projecto GRITO é criar um sistema de preservação que utilize não só recursos de *grids* dedicadas para preservação, mas também recursos excedentários de *grids* de dados cujo propósito principal é outro. A utilização de diferentes *grids* implicará, em muitos casos, um ambiente heterogéneo onde existirão, por exemplo, nós com diferentes configurações de software, localização geográfica ou administração. Isto pode ocorrer não só entre nós de *grids* diferentes, mas também dentro da mesma grid.

Num ambiente heterogéneo muitas das falhas a que o sistema está sujeito não afectam os nós aleatoriamente. Um exemplo disto são os desastres naturais, que mais provavelmente afectarão conjuntos de nós com posições geográficas semelhantes. Ou seja, parte das falhas que afectam um sistema heterogéneo são correlacionadas, já que podem ser causadas por um único evento. Além disso, num sistema que funcione durante um longo período de tempo, como os usados na preservação digital, torna-se provável que certos eventos raros ocorram durante o seu tempo de vida. Exemplos disto são desastres naturais ou a propagação em grande escala de *worms*. Estes eventos altamente destructivos, mesmo que raros, são os mais passíveis de causar perdas de ficheiros [22].

Uma estratégia de redundância define os mecanismos de redundância a utilizar e que acções tomar perante um determinado estado do sistema. Por exemplo, pode-se definir como estratégia de redundância utilizar replicação e, sempre que um ficheiro tiver menos que três réplicas no sistema, criar uma nova num recurso de armazenamento escolhido aleatoriamente, copiando o ficheiro de um outro que contenha uma das réplicas. Um sistema de redundância é responsável por implementar uma determinada estratégia de redundância em um ou mais sistemas de armazenamento. Num sistema de armazenamento para preservação digital os ficheiros são escritos apenas uma vez, não sendo nunca alterados. Isto elimina o problema de manter a coerência das réplicas.

O objectivo deste trabalho é estudar e implementar estratégias de redundância de dados num sistema GRITO. Assim, é necessário desenvolver um conjunto de estratégias, uma maneira de comparar o seu desempenho e um sistema de redundância para utilização no GRITO.

Na secção 2.1 foram já apresentados aspectos relevantes de um sistema de redundância para impedir que falhas causem perdas de dados. Devido a razões descritas na secção 3.4, neste trabalho apenas são abordadas estratégias cujo mecanismo de redundância é a replicação simples, não se considerando o *erasure coding*. Um outro aspecto importante, a auditoria, é abordada em [3]. Assim, este trabalho foca-se em dois dos outros aspectos: como tirar partido da diversidade e de mecanismos introspecção nas estratégias de redundância de um sistema GRITO. Estas estratégias devem ter em conta a diversidade do sistema e as falhas correlacionadas que resultam dela.

Resumindo, este trabalho aborda o problema da utilização da replicação de ficheiros num sistema com falhas correlacionadas para e pretende responder às seguintes questões:

- Que diferentes estratégias de redundância de ficheiros se deve considerar para utilizar num dado sistema de armazenamento com elevado número de falhas correlacionadas?
- Como prever os níveis funcionamento de um sistema destes a operar em certas condições se se utilizar uma determinada estratégia? Por exemplo, qual será a frequência de perdas de ficheiros, débito da rede utilizado e capacidade de armazenamento do sistema.
- Como implementar estas estratégias em sistemas de armazenamento baseados em *grids*.

Para endereçar estas questões, é necessário estabelecer primeiro um modelo de um sistema de armazenamento distribuído, que é descrito em 3.1. Em 3.2 são também identificados os tipos de falhas que podem afectar um sistema destes. Na secção 3.3 estão listadas diversas causas possíveis para essas falhas. Finalmente, na secção 3.4 explicam-se as razões para a limitação do problema a estratégias de redundância baseadas em replicação.

3.1 Modelo de sistema de preservação

Através da análise dos principais componentes de um sistema de armazenamento, foi criado um modelo que permite descrever um destes sistemas. Neste modelo, um sistema de armazenamento é constituído por:

- Ficheiros
- Recursos de armazenamento
- Rede de recursos

Um ficheiro corresponde ao que se quer armazenar, caso a estratégia de redundância utilizada seja a replicação, um ficheiro pode ter uma ou mais réplicas. Cada ficheiro tem um identificador único e um tamanho. Ao conjunto de todos os ficheiros a ser armazenados num sistema de replicação chama-se colecção.

Os recursos de armazenamento correspondem tipicamente a discos. Contudo, podem também representar um sistema RAID ou qualquer outro sistema que permita armazenar ficheiros. Cada recurso tem um identificador único e uma capacidade de armazenamento. Neste modelo é ignorado o tempo de leitura de ficheiros nestes recursos de armazenamento, focando-nos apenas no tempo que demora a cópia de ficheiros através da rede de um recurso para outro.

Os recursos de armazenamento são ligados por uma rede, que é utilizada para transferir ficheiros. Esta rede é representada como um grafo em que as arestas são ligações de rede e os vértices são recursos de armazenamento ou pontos de interligação. Estes pontos de ligação podem corresponder, por exemplo, a routers ou a redes que funcionem como “nuvens"(tal como a Internet). Cada ligação de rede une dois pontos e tem um determinado débito em cada sentido. Quando se pretende criar uma nova réplica, é necessário transferir um ficheiro de um recurso para outro através da rede. O tempo que esta transferência demora dependerá do tamanho do ficheiro e do débito da ligação entre esses dois recursos. Neste modelo, escolhemos ignorar a latência existente nas ligações, já que esta é pouco relevante para o tempo de transferência de ficheiros, desde que estes não sejam muito pequenos. Ignoramos também o funcionamento da stack de protocolos de rede a ser utilizada, devendo o débito da ligação corresponder ao débito útil ao nível de rede a que os ficheiros são transferidos.

O funcionamento de um sistema também é afectado pelo cenário de falhas a que ele está sujeito e pela estratégia de redundância utilizada. O cenário de falhas corresponde a todas as falhas que ocorrem no sistema, ou seja, nos seus recursos de armazenamento ou rede, durante o seu tempo de funcionamento. A estratégia de redundância à forma como o sistema faz a redundância de dados, ou seja, às operações que o sistema desencadeia e as condições em que o faz.

3.2 Tipos de falhas

Foram identificados dois tipos diferentes de falhas que podem afectar os recursos de armazenamento: indisponibilidade e perda de dados.

Quando há uma falha de indisponibilidade, o recurso de armazenamento deixa de estar contactável, permanentemente ou temporariamente. Isto pode ser causado, por exemplo, por uma falha de energia. Quando existe uma falha do tipo perda de dados, o recurso perde dados que fazem parte da estratégia de redundância a ser utilizada. Por exemplo, num sistema a utilizar replicação, uma perda de dados implica que uma réplica de um ficheiro seja perdida ou corrompida. Isto pode ser causado, por exemplo, pela destruição de um disco.

Estes dois tipos falhas podem ocorrer simultaneamente. Por exemplo, um servidor cujo disco seja substituído ficará indisponível durante algum tempo e perderá todos os dados lá guardados.

3.3 Ameaças

Os sistemas de armazenamento para preservação digital estão sujeitos, não só ás ameaças que afectam os sistemas convencionais, mas também a algumas que lhes são particulares. Aqui são descritos estes diferentes tipos de ameaças.

3.3.1 Erros em componentes

Podem haver diferentes tipos de problemas com os componentes de um sistema de armazenamento para preservação digital [5, 6, 26]. Os *componentes de armazenamento* podem falhar com o tempo, causando perda de dados total (um disco deixar de funcionar) ou parcial, devido a “bit rot”. Este último pode ser particularmente problemático, pois pode não ser detectado até que se tente aceder aos dados. *Outros componentes de hardware* como fontes de alimentação podem sofrer falhas tanto temporárias (falha de electricidade) como permanentes (fonte de alimentação queimar). Os *componentes de software*, como o firmware dos discos, podem ter bugs que causem falhas. A *rede de comunicação* pode também deixar de funcionar ou ter uma quebra significativa de performance. Os *serviços de rede* de que o sistema depende podem falhar. Por exemplo, serviços como o DNS podem deixar de funcionar ou mesmo ser descontinuados no futuro.

3.3.2 Obsolescência

A obsolescência pode afectar tanto hardware como software. Os *componentes de hardware* podem-se tornar obsoletos, o que pode implicar que se tornem incapazes de comunicar com outros componentes do sistema ou que seja impossível substitui-los em caso de falha. Em particular, suportes de armazenamento removíveis, como CDs ou disquetes, só sã úteis enquanto existir hardware capaz de os ler, tornando-se obsoletos caso contrário. *Obsolescência de software* manifesta-se muitas vezes como obsolescência de formatos, quando os bits de um ficheiro ainda podem ser lidos, mas não existe qualquer aplicação capaz de os descodificar e interpretar. Outro exemplo disto ocorre quando se armazena software para preservação mas deixa de haver hardware capaz de o executar.

3.3.3 Erros humanos

Os erros humanos são inevitáveis [25] e os humanos estão sempre envolvidos com os sistemas. Foi demonstrado que os operadores humanos podem ser a principal causa de falhas em diferentes tipos de sistemas [24]. Muitas vezes, as pessoas apagam dados que ainda são necessários e podem causar falhas em componentes tanto de hardware (desligando um cabo de alimentação acidentalmente) como software (desinstalando uma biblioteca de software necessária). A má configuração dos sistemas também pode ser uma fonte de erros.

3.3.4 Desastres naturais

Desastres naturais como terramotos, inundações ou fogos podem causar falhas em muitos componentes simultâneamente. Por exemplo, um terramoto pode causar a destruição de um centro de dados ou uma falha de electricidade em vários componentes.

3.3.5 Ataques

Ataques podem ter diversos objectivos, como destruição de dados, negação de serviço (DoS) e roubo ou alteração de dados. Os ataques podem ter motivações políticas, ideológicas ou financeiras. Podem ser legais ou ilegais e efectuados tanto por agentes externos como internos, que têm total acesso ao sistema. Podem ser rápidos e feitos a curto prazo ou podem ser duradouros e efectuados por pessoas ou organizações dedicadas.

Além disto, o facto de muitos sistemas de preservação estarem ligados á Internet e, por vezes, dependerem dela para funcionar, faz com que estejam vulneráveis a ataques automáticos de *worms* ou vírus.

3.3.6 Erros de gestão

A má gestão pode causar a falha de sistemas de preservação digital. A falta de dinheiro é o maior constrangimento ao sucesso da preservação digital [28], portanto existe um risco elevado de *falhas económicas*. Os custos com electricidade, ar condicionado e administração fazem com que a informação digital seja mais vulnerável a problemas económicos que a tradicional em papel. O orçamento para sistemas de preservação

digital pode variar muito ao longo do tempo ou mesmo ser completamente cortado em certas alturas. Existe também a possibilidade de *falhas organizacionais*, quando uma organização responsável por preservar conteúdos pode desaparecer ou deixar de ser responsável por eles. Neste caso, o sistema deve permitir que a sua administração seja transferida para outra organização ou que os seus dados sejam copiados para outro sistema.

3.3.7 Visibilidade e independência das falhas

Não se pode assumir que todas as falhas são imediatamente visíveis. Muitas vezes uma falha não é detectada durante muito tempo. Por exemplo, é possível que só se descubra que um formato de ficheiro está obsoleto quando é necessário aceder-lhe. Um ataque que corrompa dados também pode nunca ser detectado.

Não se pode também assumir que as falhas são normalmente independentes [6]. Uma falha de energia fará com que muitos componentes falhem ao mesmo tempo; um worm ou vírus pode apenas afectar computadores com uma determinada configuração de software e um desastre natural como um terramoto afectará vários computadores na mesma área geográfica. A probabilidade de ser afectado por falhas também pode depender das características de um componente. Por exemplo, um computador que utilize Windows pode estar mais vulnerável a vírus do que um que corra Unix. Esta probabilidade pode até variar com o tempo. Por exemplo, será mais provável que um componente na Flórida falhe durante a época de furacões do que durante o resto do ano. Ou seja, muitas das falhas que ocorrem num sistema são correlacionadas e vários nós são afectados ao mesmo tempo, o que as torna particularmente graves [7, 38].

A tabela 3.1 apresenta uma divisão em categorias das ameaças descritas anteriormente.

Erros em componentes	**Erros de gestão**
Erro em suporte de armazenamento	Falha organizacional
Erro de hardware	Falha económica
Erro de software	Obsolescência de hardware
Erro de comunicações	Obsolescência de software
Falha de serviços externos	
Desastres	**Ataques**
Desastre natural	Ataque interno
Erro humano	Ataque externo

Tabela 3.1: Ameaças aos sistemas de preservação digital

3.4 Replicação e “erasure coding”

Existem duas principais formas de fazer redundância de ficheiros: replicação e *erasure coding*. Tal como referido na secção 2.1.1.1, o *erasure coding* parecer ser uma boa opção para armazenamento de dados para preservação digital. Contudo, é bastante mais complexo que a replicação, e a maioria das tecnologias de *grid* existentes hoje em dia ainda não o suportam.

Assim, seria muito trabalhoso criar um sistema de armazenamento que suportasse *erasure coding* no tempo em que este trabalho foi efectuado. Devido a isto, e a ainda existir muito trabalho a ser feito no estudo da replicação simples num sistema de armazenamento para preservação digital, foi tomada a opção de neste trabalho apenas analisar estratégias de redundância baseadas em replicação.

4 Descrição da solução

Devido ao elevado número de falhas correlacionadas que pode existir num sistema de armazenamento distribuído heterogéneo, decidiu-se desenvolver estratégias de redundância que tenham isto em conta. Assim, desenvolveram-se algoritmos de replicação que tentam armazenar as réplicas de um ficheiro em recursos de armazenamento que sejam o mais independentes possível, de forma a minimizar a probabilidade de que todas sejam afectadas por uma série de falhas correlacionadas.

Para saber qual a eficiência de uma estratégia de redundância, é necessário saber qual será o comportamento do sistema que a utiliza. O funcionamento de um sistema de armazenamento para preservação digital é afectado muitos por parâmetros, como a quantidade de nós existentes, a maneira como estes estão ligados, as falhas que ocorrem e a correlação entre elas. Devido a isto, a análise do comportamento de um sistema destes é complexa demais para ser feita apenas analiticamente.

Outra forma de fazer esta análise seria implementar um sistema e observar o seu comportamento. Contudo, os sistemas de preservação digital têm um tempo de vida muito grande, que pode ser de décadas, e só podem ser realmente testados quando ocorrem as catástrofes a que eles devem sobreviver, algumas das quais podem ser muito raras. Devido a isto, esta hipótese também não é viável se quisermos analisar a fundo o comportamento de um sistema.

Portanto, a solução adoptada foi construir um simulador que permita simular o comportamento destes sistemas em vários cenários e recolher estatísticas que permitam tirar conclusões sobre os seus níveis de funcionamento.

Foi também desenvolvido um sistema de redundância que permita utilizar as estratégias de redundância referidas anteriormente em sistemas grid. Este sistema pode, para já, funcionar com *grids* iRODS, mas foi desenvolvido de forma a que seja fácil adicionar suporte para novos tipos de grids. As estratégias de redundância são definidas da mesma forma neste sistema e no simulador, o que simplifica a passagem das estratégias de um ambiente simulado para um real.

Para desenvolver estratégias que tenham em conta as correlações entre falhas e efectuar simulações que se aproximem da realidade, torna-se necessário desenvolver modelos que as permitam prever. Neste trabalho definimos dois tipos de modelos: modelos de falhas, que geram falhas para as simulações e modelos de previsão, que são utilizados pelos algoritmos de replicação para prever as falhas. Contudo, a criação destes modelos é difícil e não é abordada em muita profundidade neste trabalho, focando-se este na criação de ferramentas que permitam analisar a eficiência de diferentes algoritmos de replicação e pô-los em funcionamento em sistemas reais.

Além disso, são também propostos alguns algoritmos de replicação. Não existindo um estudo aprofundado sobre a modelação de falhas, estes algoritmos serão desenhados de forma a serem capazes de funcionar com modelos de previsão de falhas desenvolvidos posteriormente e utilizarão introspecção para melhorar o seu funcionamento. Estes algoritmos poderão também ser usados como base para o desenvolvimento de novos algoritmos.

Nas secções 4.1 e 4.2 são descritas algumas opções tomadas na arquitectura deste sistema. Os componentes da solução são descritos em capítulos distintos: o simulador no capítulo 4.3, os algoritmos desenvolvidos no capítulo 5.2 e o sistema de replicação no capítulo 4.4.

4.1 Decisões distribuídas ou centralizadas

Num sistema de redundância de ficheiros, uma das questões que é necessário abordar é se as decisões devem ser feitas de forma centralizada (por apenas um computador, como em [14]) ou distribuída (por mais do que um, como em [15]). A arquitectura distribuída tem a vantagem de permitir utilizar os recursos computacionais de vários nós, diminuindo assim o tempo de execução dos algoritmos necessários para a estratégia de redundância. Contudo, uma arquitectura centralizada elimina a necessidade de haver partilha de informação e sincronização entre diferentes nós e torna mais fácil a definição de estratégias de redundância.

O objectivo do nosso sistema de redundância é utilizar os recursos de armazenamento de sistemas *grid* existentes, portanto o nosso sistema deve ser o menos intrusivo possível. Ou seja, deve implicar o mínimo possível que seja necessário alterar a configuração de software destas grids, já que isto pode causar obstáculos à utilização destas, por exemplo, devido a políticas administrativas da grid. Caso se utilizasse um sistema distribuído e se quisesse que vários nós da *grid* fizessem parte dele, isto poderia implicar ter que alterar muito as configurações de software de todos esses nós. Por outro lado, com um sistema centralizado, só será necessário que esses nós estejam preparados para fornecer armazenamento de dados, o que muitas vezes pode ser feito apenas alterando parâmetros de configuração, já que os sistemas grid normalmente já têm primitivas que permitem guardar e eliminar dados de um nó.

Nos sistemas de preservação digital, não existe normalmente necessidade tomar decisões com grande rapidez, já que o principal objectivo não é a disponibilidade dos conteúdos instantaneamente, mas sim assegurar que eles não são perdidos. Portanto, a capacidade de utilizar o poder computacional de vários nós, que é a principal vantagem de um sistema distribuído, não tem uma importância tão grande como noutros sistemas de armazenamento.

Contudo, é importante que se consigam utilizar os recursos de armazenamento disponíveis de forma inteligente e eficiente, de forma a aumentar a quantidade de informação que se preserva no sistema e a prevenir que haja perda de dados mesmo quando ocorrem catástrofes. Devido a isto, as decisões referentes à estratégia de redundância devem ser feitas tendo em conta toda a informação disponível, o que é muito mais fácil fazer num sistema centralizado.

Devido a isto, o sistema que iremos desenhar irá tomar as decisões referentes á estratégia de redundância de forma centralizada, já que esta solução aparenta ter um maior número de vantagens.

4.2 Serviço externo ou interno

Muito do software de *grid* existente hoje em dia permite ser alterado ou adaptado. Um exemplo disto é o iRODS, cujo comportamento pode ser alterado utilizando microserviços e regras. Portanto, o sistema de redundância de ficheiros pode ser implementado nas próprias *grids* ou como um serviço externo, que interage com elas utilizando a sua API.

Será vantajoso que um sistema destes possa interagir com muitas tecnologias de grid diferentes, já que isto permitirá um número maior de *grids* potenciais onde utilizar o sistema. Portanto, deve ser fácil adicionar ao sistema a funcionalidade de interagir com uma nova tecnologia de grid. Contudo, alterar o funcionamento interno de cada tipo de *grid* é difícil, já que a forma de o fazer é diferente para cada um, tendo que se utilizar

diferentes linguagens e ferramentas. Sendo o serviço de redundância complexo, é mais fácil criar um serviço externo que possa interagir com todos os tipos diferentes de *grids* do que estar a reimplementá-lo em cada tecnologia. Assim, a única coisa que pode ser necessário implementar internamente em cada tecnologia de grid, caso ela não a suporte já, são as operações que o serviço externo requer para obter informação e tomar acções sobre a grid.

Além disso, como o sistema de armazenamento deve poder funcionar com *grids* que já estão em funcionamento, pode ser problemático ter que alterar o software dessa grid, já que isto pode implicar recompilar ou reiniciar o software da grid. Além disso, podem ser introduzidos bugs que comprometam o bom funcionamento da *grid* para o seu objectivo principal.

Devido a isto, a utilização de um serviço externo que interaja com a grid como um cliente dela parece ser menos intrusiva e mais vantajosa que alterar o software da *grid* em si. Os clientes interagem com o software de *grid* e ingerem ficheiros normalmente. O sistema de redundância monitoriza a *grid* para detectar a ingestão de ficheiros e falhas de recursos, criando e eliminando as réplicas necessárias automaticamente. Isto é feito interagindo com a *grid* como se fosse um simples cliente ou administrador, dependendo das permissões necessárias.

4.3 Serapeum

O Serapeum é o simulador desenvolvido neste trabalho. Foi implementado em Java e permite simular o comportamento de sistemas de armazenamento distribuídos com controlo central sobre as decisões de replicação.

Nesta secção é descrito o funcionamento deste simulador, os vários parâmetros que podem ser definidos para as simulações e como fazer recolha de estatísticas das simulações.

4.3.1 Funcionamento

O funcionamento do simulador passa por, a partir de uma descrição de um sistema, guardar um estado interno deste e um outro estado que é visível. Estes estados podem ser diferentes já que, por exemplo, quando uma falha é permanente, essa informação não está disponível no estado visível. O estado visível é então passado ao algoritmo de replicação, que vai definir um conjunto de operações que devem ser efectuadas sobre o sistema. Ao efectuar estas operações, o simulador irá alterar ambos os estados. Por exemplo, quando é iniciada uma nova operação de cópia, o débito a que as restantes cópias estão a ser feitas pode ser alterado. Os estados também podem ser alterados quando existem falhas (que são definidas como parâmetro da simulação).

Apesar de ser possível correr apenas uma simulação, normalmente é necessário correr várias, seja para testar diferentes parâmetros ou para obter um número estatisticamente relevante de resultados. Para isto, foram desenvolvidos scripts em Python, que permitem gerar e correr conjuntos de simulações mais facilmente.

4.3.2 Parâmetros de simulação

Há vários parâmetros que têm de ser definidos para uma simulação de um sistema de armazenamento para preservação digital. Estes parâmetros são uma consequência do modelo definido em 3.1:

1. Lista de ficheiros, incluindo em que recursos se localizam inicialmente as suas réplicas
2. Lista de componentes de armazenamento do sistema e o seu estado inicial (disponível ou indisponível)
3. Descrição da rede que interliga os componentes de armazenamento
4. O cenário de falhas, ou seja, descrição de todas as falhas que vão ocorrer durante o tempo de simulação
5. Lista ficheiros a ser adicionados ao sistema e tempo em que isso ocorre
6. Lista de componentes de armazenamento a ser adicionados ao sistema e tempo a que isso ocorre
7. Lista de alterações das características da rede e tempo em que ocorre (por exemplo, uma ligação na rede tornar-se mais lenta a partir de um determinado instante de tempo)
8. Algoritmo de replicação a ser utilizado e valores dos parâmetros do algoritmo
9. Tempo de simulação, ou seja, qual o período de tempo para o qual se vai fazer a simulação
10. Estatísticas a recolher. Os diferentes tipos de estatísticas que é possível recolher estão descritas em 4.3.4

No Serapeum, os itens 1, 2, 3 e 4 são definidos utilizando ficheiros XML e os itens 8, 9 e 10 como parâmetros ao correr o programa. Os itens 5, 6 e 7 não são para já suportados, pois considera-se que não são relevantes para a análise feita neste trabalho. Isto é, não é possível fazer simulações em que haja ingestão de novos ficheiros, adição de novos nós ou alterações ao estado da rede (além daquelas causadas por falhas) durante o tempo de vida do sistema. Contudo, poderá haver trabalho futuro para adicionar estas funcionalidades.

O cenário de falhas é representado por um ficheiro XML e existem duas formas diferentes de o fazer. Na forma “full”, o ficheiro XML contém a lista detalhada de todas as falhas que ocorrem, enquanto que na forma “seed”, o ficheiro XML apenas contém uma lista de modelos de falhas a utilizar e a “seed” a ser usada para inicializar o gerador de números aleatórios. Ambos permitem que simulações sejam repetidas mas, no último modo, os ficheiros ocupam muito menos espaço, o que pode ser importante quando se quer fazer um número elevado de simulações.

4.3.3 Definição da rede de recursos

Por razões de eficiência, é possível definir as redes de recursos de diferentes maneiras. A mais genérica é como um grafo, que permite definir uma rede com qualquer topologia, tal como a representada em 4.1. Contudo, isto faz com que seja necessário utilizar o algoritmo de Dijskstra para encontrar os caminhos entre os nós, o que baixa bastante a eficiência do simulador. O sistema permite escolher utilizar-se *caching* para resolver este problema, contudo esta solução só funciona em simulações de sistemas com poucos recursos de armazenamento, já que em sistemas maiores a quantidade de memória necessária é muito elevada. Por isso, também é possível definir uma topologia em que todos os recursos de armazenamento estejam ligados a um único ponto de interligação, tal como em 4.2.

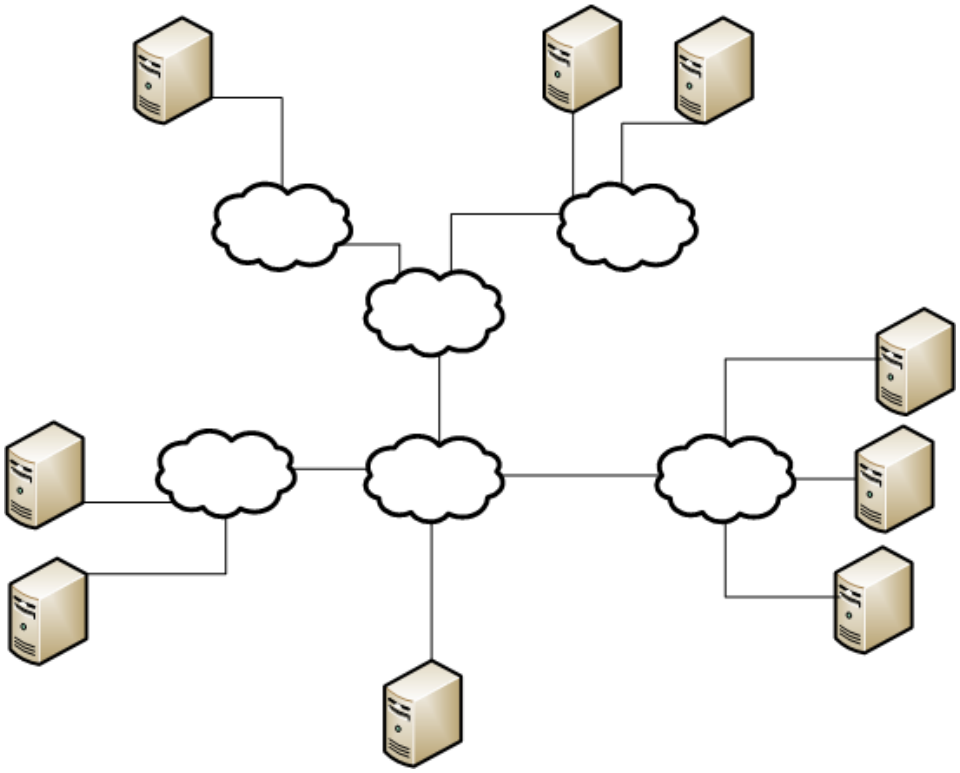

Figura 4.1: Rede definida utilizando um grafo

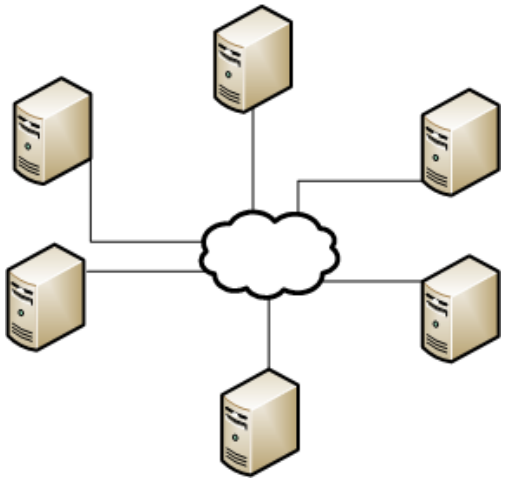

Figura 4.2: Rede com um único ponto de interligação

4.3.4 Estatísticas

As estatísticas recolhidas a partir duma simulação podem ser usadas para avaliar a eficiência de um sistema de armazenamento para preservação digital num determinado cenário. Foram identificadas diversas métricas relevantes para isto:

- Número de perdas de ficheiros que ocorrem
- Utilização da rede ao longo do seu tempo de funcionamento
- Disponibilidade dos ficheiros
- Número de réplicas dos ficheiros em cada instante
- Utilização da capacidade de armazenamento dos recursos em cada instante

O número de perdas de ficheiros permite saber que quantidade de perdas de dados se pode esperar de um sistema a operar em certas condições, o que será muito provavelmente o aspecto mais importante do seu funcionamento. A utilização total da rede dá uma ideia da quantidade de cópias que foi necessário efectuar para atingir o nível de perdas anteriormente referido, o que estará relacionado com a quantidade de réplicas criadas.

Tratando-se de sistemas para preservação digital, a disponibilidade dos ficheiros não será muito importante, já que a impossibilidade de aceder a um ficheiro temporariamente é muito menos relevante do que o ficheiro se perder para sempre.

As estatísticas do número de réplicas dos ficheiros e utilização dos recursos em cada instante podem ser úteis para analisar o comportamento do sistema e descobrir pontos em que a estratégia de replicação poderia ser melhorada.

No Serapeum, a recolha de estatísticas é implementada através de uma classe cujas funções são chamadas quando ocorre algum evento relevante, como a perda duma réplica dum ficheiro ou o inicio de uma cópia. Isto faz com que seja fácil implementar a funcionalidade de recolher novos tipos de estatísticas caso seja necessário.

4.4 Sistema de replicação

Nesta secção é descrito o sistema de replicação que foi implementado. Este sistema é um serviço em Java que obtém informação das grids, corre um algoritmo de replicação e, caso necessário, dá à *grid* as instrucções que o algoritmo sugere. Além disso, o sistema de replicação fornece informações sobre o estado do sistema ao utilizador.

O código dos algoritmos de replicação utilizados neste sistema é o mesmo que o do Serapeum. Isto faz com que não seja necessário implementar os algoritmos uma vez para a simulação e outra para funcionamento real, o que previne que sejam inseridos “bugs” durante a reimplementação.

A interacção com os sistemas *grid* é feita através de drivers específicos para cada sistema. Estes drivers devem ser implementados em Java e fornecer funções para obter informações sobre o sistema e para actuar sobre ele. As funções que devem existir nestes drivers estão listadas em 4.4.1. Neste trabalho foi implementado parte de um driver para o sistema iRODS, que está descrito em 4.4.2.

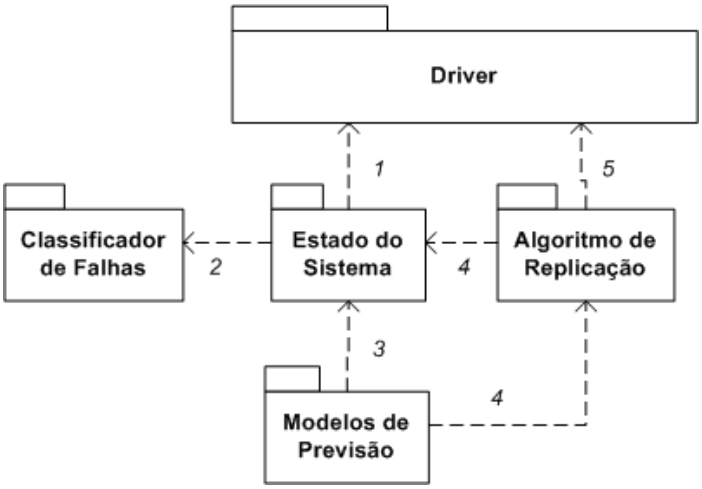

Figura 4.3: Funcionamento do sistema de replicação

Na figura 4.3 estão representados os principais blocos de software do sistema de replicação. A interacção entre eles é feita da seguinte forma: o driver detecta alterações no estado da grid e chama funções que actualizam o estado do sistema (1). É também guardada informação sobre estados passados para os mecanismos de introspecção. Periodicamente, um classificador de falhas lê o estado do sistema e altera-o, indicando as falhas que é previsto que tenham causado perdas de dados (2). De seguida, os modelos de previsão de falhas são ajustados conforme o estado presente e passado do sistema (3). O algoritmo de replicação pode agora ler o estado do sistema e, com base nos modelos de previsão actualizados, construir um conjunto de operações (4). Essas operações são então efectuadas na a grid através do driver (5) e o estado do sistema é alterado para que se guarde a informação de que essas operações foram ordenadas (4).

O sistema implementado neste trabalho é apenas um protótipo, já que não tem a capacidade de interagir com múltiplos sistemas *grid* diferentes ao mesmo tempo. Isto deve-se principalmente ao facto de não ser possível transferir ficheiros de um sistema *grid* para outro caso estes não sejam interoperáveis, o que torna muito complicado criar uma nova réplica de um ficheiro num sistema que não contenha já uma. Seria interessante no futuro estudar mecanismos que permitissem resolver este problema.

4.4.1 Funcionalidades de um driver

As funções que devem existir nos drivers para obter informações sobre o estado do sistema são:

- Listar os nomes dos ficheiros
- Listar os nomes dos nós
- Listar as réplicas de um ficheiro
- Obter o tamanho de um ficheiro
- Obter a capacidade de um nó
- Obter estado das cópias que estão a ser efectuadas
- Detectar falhas de nós
- Detectar perdas de réplicas

- Detectar inserção de novos ficheiros

Da mesma maneira, as funções que devem existir para actuar sobre o sistema são:

- Copiar um ficheiro de um nó para outro
- Apagar uma réplica de um nó
- Cancelar uma cópia

Nem todas estas funções têm que ser suportadas nativamente pelo sistema grid. Por exemplo, a função de detectar falhas de recursos pode ser implementada ao nível do driver pedindo periodicamente listas de todos os recursos e comparando-as, sendo assim apenas necessário que o sistema não retorne recursos que falharam nesta lista. Contudo, se o sistema de armazenamento já fornecer um mecanismo para as detectar, este poderá ser utilizado para obter maior eficiência.

4.4.2 Driver para iRODS

O driver parcialmente implementado para o sistema iRODS permite, apesar de não estar completo, demonstrar o funcionamento do sistema de replicação.

Este driver utiliza a API Java do iRODS (Jargon) para se ligar a um sistema destes como cliente. Após fazer isto, consegue obter listas de recursos de armazenamento, ficheiros e réplicas e dar instruções para que sejam criadas novas réplicas.

Para detectar falhas de recursos de armazenamento e réplicas, são comparadas diferentes versões destas listas, que são actualizadas graças ao trabalho desenvolvido em [3]. Apesar disto, o Jargon ainda tem algumas limitações, pelo que ainda não é possível obter a capacidade dos recursos de armazenamento nem apagar réplicas de ficheiros. Contudo, prevê-se que estas funcionalidades sejam suportadas em breve.

5 Estratégias

Neste capítulo são descritos os modelos e algoritmos de replicação desenvolvidos neste trabalho, bem como os mecanismos de introspecção implementados.

Os modelos estão descritos na secção 5.1 e são usados tanto para gerar falhas para as simulações como para fazer previsões sobre as falhas, que são usadas pelos algoritmos de replicação. Na secção 5.2 são descritos os algoritmos de replicação e as métricas por eles usados. Finalmente, na secção 5.3 são descritos os mecanismos de introspecção que permitem ajustar os modelos e os algoritmos ao estado observado do sistema.

5.1 Modelos

Existem dois tipos de modelos: os modelos de falhas e os modelos de previsão. Os primeiros são usados para definir as falhas que ocorrem durante uma simulação, enquanto que os últimos são usados pelos algoritmos de replicação para prever que falhas vão ocorrer e as correlações entre elas.

Apesar de ser possível definir individualmente o cenário de falhas de uma simulação, isto será normalmente feito utilizando modelos de falhas. Para isso, foi desenvolvida uma ferramenta em Java que gera cenários de falhas a partir da definição de um conjunto de modelos de falhas. Além disto, é possível definir também a duração de um "período de estabilização", um período de tempo no inicio da simulação durante o qual não existem falhas, para que o sistema possa estabilizar criando as réplicas que deveriam existir inicialmente.

Os modelos de falhas são implementados como classes Java que obedeçam a uma interface que contém um método que recebe como argumento o tempo actual e retorna uma lista com as próximas falhas que ocorrerão. Quando o simulador consumir essas falhas, o método voltará a ser chamado, gerando assim mais falhas. As falhas podem ser dos tipos referidos em 3.2.

Os modelos de previsão de falhas dão uma estimativa, para um conjunto de nós, da frequência de falhas simultâneas deles e o tempo médio até que o primeiro desses nós recupere. Apenas falhas que causem perdas de dados são analisadas por estes modelos. A informação fornecida pelos vários modelos de falhas é combinada para calcular a frequência total destas falhas e o tempo médio até à primeira recuperação.

Um modelo de previsão deve ser capaz de, para um certo conjunto de recursos, prever a frequência de falhas que causem perda de dados simultânea nesses recursos e também a frequência de falhas que causem perdas de dados simultânea nesses recursos exclusivamente (não afectando mais nenhum outro recurso). Além disso, devem conseguir prever qual o tempo esperado até que o primeiro recurso desse conjunto recupere da falha.

Estes modelos são implementados como classes Java que obedecem a uma interface que contém métodos para fornecer os dois tipos diferentes de previsões e o tempo até à primeira recuperação.

Foram usados três modelos diferentes, que podem ser usados tanto para gerar como prever falhas: falhas independentes, falhas baseadas em atributos e outras falhas correlacionadas.

5.1.1 Modelo de falhas independentes

Este modelo representa falhas independentes de diversos tipos para um único nó, em particular:

- Falha permanente do recurso de armazenamento
- Perda de dados no recurso de armazenamento
- Indisponibilidade temporária do recurso
- Indisponibilidade temporária e perda total de dados do recurso

O tempo entre falhas pode ser gerado utilizando uma distribuição exponencial ou de Weibull. Caso o tipo de falha gerado seja temporária, o tempo até o recurso voltar a estar disponível segue uma distribuição exponencial.

Este modelo de falhas é utilizado para modelar falhas de discos e sua substituição que, segundo [30], ocorrem como uma distribuição de Weibull. Estas falhas causam indisponibilidade temporária no recurso e perda total de dados.

5.1.1.1 Geração de falhas

O modelo de falhas independentes recebe os seguintes parâmetros:

- O recurso de armazenamento para o qual as falhas serão geradas
- Que tipo de falhas deve ser gerado, dos quatro apresentados anteriormente
- Se a frequência de falhas segue uma distribuição exponencial ou de Weibull
- O parâmetro λ da distribuição, que representa o tempo entre falhas
- Caso seja uma distribuição de Weibull, o parâmetro k desta distribuição
- Caso a falha seja temporária, qual o tempo médio até ao regresso do recurso afectado

Este modelo limita-se a então gerar falhas do tipo definido seguindo a distribuição definida.

5.1.1.2 Previsão de falhas

Sendo as falhas independentes e afectando apenas um recurso, quando o conjunto de recursos tem tamanho diferente de um, o tempo entre falhas é infinito. Caso contrário, é o valor esperado da distribuição usada.

As falhas podem seguir uma distribuição de Weibull, o que significa que quanto mais “antigos” forem os recursos, maior será a probabilidade deles falharem. Contudo, não é possível utilizar esta distribuição para fazer a previsão das falhas, já que as idades dos recursos não são conhecidas. Devido a isto, para efeitos de previsão, assume-se que os recursos falham segundo uma distribuição exponencial, cujo parâmetro λ é o valor esperado da distribuição de Weibull correspondente.

5.1.2 Modelo de falhas baseado em atributos

O modelo de falhas de Atributos é baseado no modelo de [16]. Neste modelo, podem-se definir vários atributos, como por exemplo “Localização Física”. Para cada atributo, podem-se depois definir os vários valores possíveis para esse atributo e os nós que assumem esse valor. Por exemplo, caso o atributo seja “Localização Física”, exemplos de valores possíveis podem ser “Lisboa” ou “Estocolmo”.

	Desastre Natural		Falha de Energia	
	Frequência	Efeitos	Frequência	Efeitos
Lisboa	100 anos	Tabela 5.2	6 meses	Tabela 5.3
Estocolmo	400 anos	Tabela 5.2	2 anos	Tabela 5.3

Tabela 5.1: Ameaças do atributo localização física'

Como se pode ver na tabela 5.1 para cada atributo pode-se definir um conjunto de ameaças que o podem afectar, neste caso, desastres naturais ou falhas de energia. As ameaças caracterizam-se pela frequência com que ocorre um evento devido a essa ameaça e os efeitos que este tem sobre os nós. Para cada ameaça, o tempo entre eventos causados por ela é uma variável aleatória que segue uma distribuição exponencial, sendo o a sua frequência a acima referida.

Prob	**Efeito**	**Recuperação**
5%	Falha permanente	
35%	Falha temporária com perda de dados, regresso não-simultâneo	4 meses
60%	Falha temporária sem perda de dados, regresso simultâneo	1 mês

Tabela 5.2: Efeitos possíveis para a ameaça de desastre natural

Prob	**Efeito**	**Recuperação**
98%	Falha temporária sem perda de dados, regresso simultâneo	6 horas
2%	Falha temporária com perda de dados, regresso não-simultâneo	4 meses

Tabela 5.3: Efeitos possíveis para a ameaça de falha de energia

Os efeitos de um evento podem ser falhas permanentes e/ou falhas temporárias de recursos de armazenamento com ou sem perda de dados. Tal como exemplificado nas tabelas 5.2 e 5.3, para cada ameaça, define-se qual a probabilidade de que um recurso seja permanentemente inutilizado caso exista um evento causado por esta, e também uma lista de falhas temporárias que podem ocorrer. Esta lista consiste na probabilidade de que um certo efeito ocorra num recurso, o tempo médio para recuperação dos recursos (sendo que este segue também uma distribuição exponencial) e ainda se todos os nós em que ocorrer este efeito recuperam simultaneamente ou não. Este último pode ser útil, por exemplo, se quisermos modelar uma falha de electricidade já que, em principio, todos os recursos voltarão a ter energia ao mesmo tempo.

É de notar que os efeitos de uma ameaça podem ser diferentes para valores diferentes do atributo. Por exemplo, o tempo médio que as falhas de electricidade duram pode ser diferente em Lisboa ou Estocolmo. Ou seja, podem existir tabelas 5.2 e 5.3 diferentes para cada valor de um atributo.

Finalmente, podem-se definir os valores para os atributos para cada recurso de armazenamento, tal como é ilustrado na figura 5.4.

	Localização Física	Configuração de software
Recurso 0	Lisboa	Windows
Recurso 1	Lisboa	Unix
Recurso 2	Lisboa	Unix
Recurso 3	Estocolmo	Windows

Tabela 5.4: Valores de cada recurso

5.1.2.1 Geração de falhas

Para gerar falhas segundo este modelo basta, para cada valor, gerar aleatoriamente o tempo entre falhas de acordo com a sua frequência. Para cada falha, escolhe-se então os efeitos que ocorrem sobre cada recurso segundo as suas probabilidades.

5.1.2.2 Previsão de falhas

O modelo de falhas por atributos encontra os atributos comuns ao conjunto de nós fornecido e calcula a soma das frequências das falhas que podem afectar estes nós, bem como o tempo médio até á recuperação do primeiro nó.

Para as falhas que afectem um conjunto de recursos mas que também admitam que falhem outros ao mesmo tempo, primeiro verificam-se quais os atributos que são comuns a todos os recursos do conjunto. Para cada um deles, obtêm-se as frequências médias dos eventos que o possam afectar e multiplica-se cada uma delas pela probabilidade desse evento causar perda de dados em todos os recursos do conjunto.

Por exemplo, considere-se um conjunto com 3 recursos que têm em comum a localização física “Lisboa". Os eventos que podem afectar esta localização são desastres naturais e falhas de energia. Os desastres naturais ocorrem com uma frequência de 0.01 por ano e têm uma probabilidade de causar perdas de dados num nó de 50%. As falhas de energia têm uma frequência de 5 por ano e uma probabilidade de 2% de causar perdas de dados. A probabilidade de um desastre natural causar perdas em todos os nós é de $0.5^3 = 0.125$, portanto, a frequência de desastres naturais que cause perdas de dados em todos os nós é de $0.125 \cdot 0.01 = 0.0125$. Da mesma maneira, a frequência de falhas de energia com este efeito é de $1 \cdot 0.02^3 = 0.000008$. Assim, a frequência total de eventos que causem perdas de dados simultâneas neste conjunto de nós é de 0.012508 por ano, ou seja, aproximadamente uma de 80 em 80 anos. Caso existissem outros atributos comuns a todos os recursos, teria que se calcular este valor também para esses e somá-lo.

Para as falhas que causem perdas de dados simultâneas exclusivamente no conjunto de recursos dado, é necessário fazer mais alguns cálculos. Além de se calcular a probabilidade de haver perda em todos os recursos do conjunto, tem que se verificar se existem mais recursos que tenham o mesmo atributo em comum. Caso existam, tem que se calcular a probabilidade de que nenhum desses recursos sofra perda de dados. Esse valor tem então que ser multiplicado pelo primeiro, obtendo-se assim a probabilidade da falha apenas causar perdas nos recursos do conjunto e em mais nenhuns outros.

Suponha-se que no exemplo anterior existiam mais 2 recursos com a localização física “Lisboa” que não faziam parte do conjunto de 3 dado. A probabilidade de um desastre natural não causar perda de dados em

nenhum deles será de $(1-0.5)^2 = 0.25$. Assim, a a frequência de desastres naturais que causem perdas exclusivamente no conjunto de 3 recursos dado é de $0.01 \cdot 0.125 \cdot 0.25 = 0.003125$. Da mesma maneira, este valor para as falhas de energia é $1 \cdot 0.02^3 \cdot (1-0.02)^2 = 0.0000076$. Portanto, a frequência total de falhas que afectem os 3 recursos e só eles é de 0.0031326 por ano, ou aproximadamente uma de 320 em 320 anos.

5.1.3 Outras falhas correlacionadas

Este modelo serve para adicionar falhas correlacionadas imprevistas ao sistema. O modelo tem dois parâmetros: a frequência de falhas λ e um nível de correlação $0 < c \leq 1$.

O tempo entre falhas segue uma distribuição exponencial com a frequência dada. Quando ocorre uma falha, o número de recursos afectado por ela segue uma distribuição geométrica de parâmetro c. Ou seja, a probabilidade de afectar apenas um recurso é c e decresce para números de recursos maiores.

Número	Probabilidade	Tempo (meses)
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

Tabela 5.5: Probabilidade de um número de recursos falhar

Na tabela 5.5 estão representadas as probabilidades de um determinado número de recursos falhar com o parâmetro $c = 0.6$ e havendo 8 recursos no sistema. A tabela mostra também o tempo médio entre falhas que afectem um número especifico de nós quando $\lambda = 1$ mês, ou seja, sendo o tempo médio entre falhas de 1 mês.

5.1.3.1 Geração de falhas

O tempo entre falhas é gerado segundo uma variável aleatória exponencial de parâmetro λ e, de seguida, gera-se o número de nós que segue uma distribuição geométrica. Os recursos afectados são depois escolhidos aleatoriament de todos os do sistema.

5.1.3.2 Previsão de falhas

O tempo médio de falhas de um conjunto de nós C (e apenas esses) pode ser calculada como:

$$\frac{t}{P(afectar\ todos\ recursos\ em\ C\ e\ so\ eles)} = \frac{t}{P(A=s) \cdot P(afectar\ todos\ os\ recursos\ em\ C | A=s)}$$

Sendo:

t - tempo médio entre falhas (parâmetro do modelo de falhas)

A - número de nós afectados por uma falha

s - número de nós em C

Sabendo que n é o número total de recursos no sistema e que quando existe uma falha que afecta A recursos estes são seleccionados aleatoriamente, pode-se calcular *P(afectar todos os recursos em C|A=s)* utilizando uma distribuição hipergeométrica, já que os recursos são escolhidos sem reposição.

Assim, se $H \sim$*Hipergeométrica(n,s,s)*, a expressão inicial pode-se escrever como:

$$\frac{t}{P(A = s) \cdot P(H = s)}$$

Para o calculo a frequência de falhas total deste conjunto incluindo falhas que afectem não só todos os recursos de C, mas também outros, pode-se usar a seguinte expressão:

$$\frac{t}{P(afectar\ todos\ os\ recursos\ em\ C)} = \frac{t}{\sum_{i=s}^{n} P(A = i) \cdot P(H_i = s)}$$

Sendo $H_i \sim$*Hipergeométrica(n,i,s)*. Ou seja, está-se a calcular a soma das probabilidades do número de nós afectados ser i e dos s nós de C fazerem parte desses i.

Número de nós	Admitindo outras falhas	Exclusivamente
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

Tabela 5.6: Exemplo de tempo médio (em meses) entre falhas de um conjunto de recursos

Na tabela 5.6 estão representados os valores calculados para o tempo médio entre falhas simultâneas de um conjunto de recursos de um determinado tamanho. Ambos os casos descritos anteriormente estão representados, ou seja, falhas exclusivas desses recursos e admitindo falhas de outros. Os valores estão em meses e são para um modelo com tempo médio entre falhas de 1 mês, um total de 8 recursos e $c = 0.6$. Assim, nestas condições, este modelo prevê que o tempo médio entre eventos que causem a falha simultânea de quaisquer 3 recursos específicos é de 83.94 meses, admitindo que outros recursos possam também ser afectados. Para eventos que afectem exclusivamente esses 3 recursos específicos e mais nenhum, o tempo médio é de 583.33 meses.

5.2 Algoritmos de replicação

Nesta secção são descritos diferentes algoritmos que foram implementados neste trabalho. Como referido anteriormente, os algoritmos de replicação devem receber o estado do sistema e retornar uma lista de operações a ser efectuadas sobre este. Estas operações podem ser cópias de ficheiros de um recurso para outro,

para criar uma nova réplica ou a eliminação de uma réplica num determinado recurso. Outra operação que poderia ser útil, mas não é para já suportada, é o cancelamento de uma cópia, o que poderá ser desenvolvido em trabalho futuro.

Uma componente muito importante para o funcionamento destes algoritmos são os modelos de previsão. Estes modelos, descritos em 5.1, permitem estimar qual a frequência de falhas simultâneas de um certo conjunto de recursos. Assim, podem ser usados para determinar conjuntos de recursos que tenham uma probabilidade baixa de falhar simultaneamente, ou seja, cujos recursos sejam mais independentes uns dos outros. Colocando as réplicas de um ficheiro nestes recursos, haverá uma probabilidade mais baixa de que se percam todas as suas réplicas do que se estas fossem colocadas aleatoriamente. A maneira como isto é feito é descrita em 5.2.1, onde se explicam as métricas que são calculadas a partir dos modelos de previsão.

Os algoritmos implementados tentam manter um nível adequado de réplicas para todos os ficheiros, criando-as e eliminando-as quando necessário. Estão divididos em dois tipos principais: os que mantêm um número de réplicas fixo e os que mantêm um número de réplicas variável.

Quando o nível de réplicas é alto ou baixo demais, é necessário eliminar ou criar novas réplicas, respectivamente. As decisões sobre que réplicas eliminar e onde criar as novas são feitas com base em heurísticas. Estas heurísticas utilizam métricas, que são descritas em 5.2.1. Também são usadas heurísticas nos algoritmos com número de réplicas variável para determinar se o nível de replicação de um ficheiro é adequado ou não.

Em 5.2.2 é descrita a implementação dos algoritmos e as classes de abstracção que foram definidas para o facilitar. Na secção 5.2.3 são explicados os algoritmos com número de réplicas fixo e na secção 5.2.4 os com número de réplicas variável.

5.2.1 Métricas

As heurísticas podem utilizar várias métricas, como por exemplo:

- Tempo médio entre falhas dos recursos que contêm as réplicas, descrito mais abaixo
- Utilização do espaço de armazenamento dos recursos
- Estado dos recursos que contêm as réplicas (disponíveis, temporariamente em baixo ou réplica em criação, quando a cópia ainda está a ser efectuada)
- Número de cópias de e para os recursos que contêm as réplicas
- Débito total e disponível das ligações dos recursos, caso essa informação seja fornecida

Contudo, as métricas mais importante são as que são informação sobre a probabilidade de perda dos ficheiros. Foram definidas duas métricas para estimar isto: o tempo médio entre falhas e o tempo médio entre “falhas duplas”. Estas métricas têm como base os modelos de previsão de falhas.

O tempo médio entre falhas das réplicas de um ficheiro é uma métrica baseada na utilizada no sistema Phoenix [16]. Utilizando diferentes modelos de previsão, calcula-se o tempo médio entre falhas que afectem simultâneamente o conjunto dos recursos que contêm as réplicas de um ficheiro. Quanto maior for este tempo, melhor será esse conjunto de recursos para armazenar as réplicas de um ficheiro. Por exemplo, caso seja utilizado um modelo baseado em atributos, este valor será melhor quando o conjunto de recursos tem poucos ou nenhuns atributos em comum.

Um problema desta métrica é que apenas tenta fazer com que não haja nenhum atributo em comum entre todos os nós, ignorando atributos que sejam partilhados apenas por um subconjunto deles. Por exemplo, caso existam recursos de armazenamento com os sistemas operativos “windows”, “linux” e “solaris” e seja necessário criar três réplicas de um ficheiro, esta métrica tem o mesmo valor caso os três recursos tenham todos sistemas operativos diferentes ou caso haja, por exemplo, duas réplicas em recursos com “windows” e uma num com “linux”, já que em nenhum dos casos existe nenhum atributo em comum. Isto pode causar problemas quando existem dois eventos seguidos que causem falhas neste atributo e o sistema não tem tempo para recuperar entre eles. Por exemplo, neste caso, o ficheiro pode ficar reduzido a apenas uma réplica após um primeiro evento que afecte os recursos com o sistema operativo “windows”, sendo esta destruída num segundo evento que afecte os que têm “linux”.

Devido a isto, foi definida uma nova métrica: o tempo médio entre “falhas duplas”. As “falhas duplas” são falhas consecutivas que afectam todas as réplicas de um ficheiro, não existindo tempo suficiente para o sistema recuperar da primeira falha antes de ocorrer a segunda. A frequência destas falhas é calculada da seguinte forma:

Num ficheiro com n réplicas, são sucessivamente calculadas todas as combinações de recursos que as armazenam. Seja R o conjunto de todos os recursos que armazenam réplicas de um ficheiro e C_x estes subconjuntos que são sucessivamente calculados. Para cada C_x podem ser obtidas as frequências de falhas simultâneas destes recursos exclusivamente. Esta frequência pode também ser obtida para o conjunto dos nós restantes $R \setminus C_x$. Além disso, pode-se ainda obter o tempo que é previsto que o sistema demore a recuperar parcialmente de cada uma das primeiras falhas, ou seja, até que uma das réplicas seja restabelecida.

Tendo esta informação, pode-se calcular a frequência de “falhas duplas” de um C_x da seguinte forma:

freqTotalDeFalhas$\leftarrow 0$

Para cada falha p *que possa afectar simultâneamente todos os recursos de* C_x*:*

freqSegundaFalha$\leftarrow$*média das frequências das falhas que podem afectar* $R \setminus C_x$

freqRecuperacao$\leftarrow$*tempo previsto para recuperacao parcial da falha* p

E~Exponencial(freqSegundaFalha)

freqTotalDeFalhas += frequência(p)·P(E<$\frac{1}{freqRecuperacao}$*)*

Ou seja, a frequência total de falhas de um C_x é a soma das frequências de todas as falhas que possam afectar simultâneamente todos os recursos que C_x contem, multiplicadas pela probabilidade de que exista uma segunda falha que afecte $R \setminus C_x$ antes que o sistema tenha tempo para recuperar da primeira, causando perda de todas as réplicas .

Somando as frequências totais de falhas dos vários C_x de R, obtém-se a métrica do tempo médio entre falhas duplas para um ficheiro cujas réplicas estejam nos recursos de R. Isto é, a frequência entre duas falhas consecutivas que afectem primeiro C_x e depois $R \setminus C_x$, num intervalo de tempo que não permita a recuperação de nenhuma réplica de C_x após a primeira falha.

Ainda assim, esta métrica não contempla algumas situações que podem ocorrer, como haver recuperação de um nó, mas este voltar a falhar na segunda falha. Por exemplo, se o conjunto de nós {A,B,C,D,E} contiver réplicas de um ficheiro e existir uma falha que afecte os nós {A,B,C}, mesmo que o nó A recupere antes da segunda falha, esta pode afectar os nós {A,D,E}, causando assim a perda de todas as réplicas do ficheiro.

Quanto ao tempo de recuperação, este é difícil de estimar. Apesar de os modelos de falhas fornecerem o tempo até que um nó volte a estar disponível, não se sabe quanto tempo demorará a cópia do ficheiro. Apesar do tamanho do ficheiro ser conhecido, o débito da rede pode não o ser. Além disso, este tempo também depende muito de quantos outros ficheiros é necessário copiar na altura e quais são os recursos de destino e fonte da cópia.

Devido a isto, apenas utilizamos o tempo até que o recurso volte a estar disponível como tempo de recuperação, o que é uma estimativa feita muito por defeito. Mesmo assim, serve para dar uma importância maior a falhas cuja recuperação demore mais tempo.

Como trabalho futuro tenciona-se usar introspecção para prever os tempos de recuperação, prevendo-os utilizando informação sobre recuperações passadas.

5.2.2 Implementação de algoritmos

Os algoritmos são implementados como classes Java. Estas classes devem ter um método que recebe o estado do sistema e retorna o conjunto de operações que devem ser efectuadas nele. Contudo, para facilitar a criação de novos algoritmos foram criadas algumas classes de abstracção.

Uma delas pode ser utilizada em algoritmos que mantenham um número de réplicas fixo. Esta classe permite que apenas tenham que se definir métodos que, para um determinado ficheiro e estado do sistema, retornem quais são os melhores recursos de armazenamento para ser utilizados como destino e fonte de uma nova réplica. Assim, quando um ficheiro fica com um número de réplicas abaixo do desejado, estes métodos são utilizados para determinar como deve ser feita a cópia para criar a nova réplica. Da mesma forma, deve ser também definido um método para determinar qual a réplica menos valiosa, que deve ser eliminada caso o número de réplicas esteja acima do desejado.

Outra destas classes pode ser utilizada para algoritmos com número de réplicas variável. Utilizando-a, apenas é necessário definir um método que dê uma medida de "segurança" para um determinado ficheiro quando este é armazenado num certo conjunto de nós. Quando o nível de segurança para um ficheiro está abaixo de um limiar definido anteriormente, vê-se qual seria o seu novo valor caso se criasse uma nova réplica em cada recurso e a nova réplica será criada no recurso que mais aumente este valor. Isto será repetido até que o nível de segurança esteja acima do limiar. Além disso, caso seja possível eliminar uma réplica sem que o valor de segurança desça abaixo do limiar, isto será feito.

Na verdade, é possível definir não um, mas dois limiares: um limiar mínimo e um máximo. Quando o nível de segurança de um ficheiro desce abaixo do limiar mínimo, serão criadas réplicas até que este esteja acima do limiar máximo. Ou seja, a descrição anterior corresponde ao caso em que o limiar mínimo é igual ao máximo. Usar valores diferentes para os dois limiares pode ajudar a prevenir que se criem réplicas desnecessariamente quando existem falhas temporárias, já que estas podem fazer que o nível de segurança desça abaixo do limiar máximo, mas não chegue a atingir o limiar mínimo.

5.2.3 Algoritmos com número de réplicas fixo

Os algoritmos que utilizam um número de réplicas fixo utilizam uma heurística que compara dois recursos de armazenamento e diz qual deles será melhor para criar uma nova réplica, tendo em conta as réplicas existentes actualmente. Assim, para cada ficheiro com um número de réplicas abaixo do desejado, o algoritmo obtém o conjunto de recursos que o pode armazenar e utiliza a heurística para escolher em qual deve criar uma nova réplica.

Para que o sistema não fique sobrecarregado, são impostas algumas limitações ás criações simultâneas de novas réplicas. Em primeiro lugar, se um recurso estiver a receber ou enviar um ficheiro que tem agora n réplicas, ele não poderá fazê-lo para nenhum ficheiro que tenha $n+1$ réplicas. Isto impede que a transmissão de um ficheiro seja atrasada devido a um que já tem mais réplicas que este e, portanto, não deve estar tanto em risco. Além disto, cada recurso só pode enviar um ficheiro para cada outro recurso de cada vez, de forma a que as réplicas sejam criadas o mais rapidamente possível.

Para que um recurso possa ser um candidato a armazenar uma nova réplica, é necessário que tenha espaço livre suficiente para armazenar o ficheiro e que não contenha já nenhuma réplica deste. Após estes recursos serem comparados e se identificarem quais os melhores candidatos, é verificado se existe algum outro recurso que esteja em condições de transmitir o ficheiro para um dos candidatos sem violar as restrições acima descritas. Caso exista, é adicionada uma operação de cópia ao plano que será retornado pelo algoritmo.

Da mesma forma, caso um ficheiro tenha mais réplicas que o número alvo, identifica-se qual o recurso cuja réplica é menos importante e essa é eliminada.

Para proteger mais rapidamente os ficheiros que estão mais em risco, eles são tratados pela ordem do número de réplicas que têm actualmente.

5.2.3.1 Heurísticas

Além de um algoritmo que cria réplicas aleatoriamente, foram definidas duas heurísticas para ser utilizadas com o algoritmo de número de réplicas fixo. Ambas funcionam de forma bastante parecida.

Na primeira, denominada “Simples", começa por comparar a métrica do tempo médio entre falhas definida em 5.2.1. Caso os dois recursos a ser comparados estejam empatados ou a diferença seja inferior a um limite mínimo que se define previamente, desempata-se utilizando a percentagem de ocupação de cada recurso. A segunda heurística denomina-se “Dupla” e utiliza a métrica do tempo médio entre “falhas duplas”, sendo de resto igual à primeira.

Assim, quando se utiliza este algoritmo, é necessário definir qual a heurística a utilizar e qual o limite mínimo a partir do qual se considera que um nó tem um tempo médio significativamente melhor que o outro.

5.2.4 Algoritmos com número de réplicas variável

As heurísticas dos algoritmos que utilizam um número de réplicas variáveis atribuem um valor a cada ficheiro que indica o risco de que ele seja perdido. Este valor depende normalmente dos recursos em que esse ficheiro tem réplicas. Quando este valor está abaixo de um certo limite, considera-se que o nível de replicação

não é adequado. Quando isto acontece, vê-se qual seria este valor caso existisse uma réplica do ficheiro em cada um dos candidatos, e a réplica será criada naquele que obtiver o melhor valor.

Mais uma vez, é imposta a restrição de cada recurso só poder enviar um ficheiro para cada outro recurso de cada vez. Os ficheiros que sejam considerados mais em risco serão tratados primeiro pelo algoritmo.

5.2.4.1 Heurísticas

As heurísticas utilizadas nos algoritmos com número variável de réplicas atribuem um valor numérico a um ficheiro que deverá ser tanto maior quanto mais seguro este estiver.

Foram definidas duas heurísticas, cuja diferença é, mais uma vez, serem baseadas na métrica do tempo médio entre falhas e no tempo médio entre "falhas duplas". O factor mais significativo das heurísticas é este tempo, sendo seguido da percentagem de réplicas que está disponível (podendo algumas réplicas estar temporariamente indisponíveis ou em criação) e, finalmente, o espaço livre médio dos recursos que armazenam as replicas. Mais especificamente, as heurísticas são calculadas da seguinte forma:

10000[número de meses entre falhas simples ou duplas] + 100*[% de recursos disponíveis] + [% de utilização dos recursos]*

5.3 Introspecção

Foram implementados dois mecanismos de introspecção que podem ser utilizados com os algoritmos descritos anteriormente para melhorar o seu desempenho. Estes mecanismos permitem observar o estado do sistema de armazenamento e alterar o comportamento do sistema de redundância conforme este.

Um dos mecanismos implementados permite ajustar os modelos de previsão utilizados pelos algoritmos de forma a que estes reflictam melhor a realidade. O outro mecanismo permite que o sistema de redundância utilize o Serapeum para escolher o melhor algoritmo e parâmetros a ser utilizados.

5.3.1 Ajustamento dos modelos de previsão

O comportamento dos algoritmos de replicação é altamente influenciado pelos modelos de previsão, já que estes são usados para decidir a colocação das réplicas e, nos algoritmos com número de réplicas variável, até para controlar o nível de replicação.

Este mecanismo de introspecção permite observar as falhas de recursos de armazenamento que ocorrem no sistema e, com base nestas, ajustar os parâmetros dos modelos de previsão que podem, à partida, estar errados.

Neste trabalho foi feita uma implementação *proof-of-concept* deste mecanismo, que detecta falhas individuais de recursos e permite ajustar modelos de previsão de falhas independentes. Para isto, é guardada informação sobre as falhas de recursos que ocorrem, como o tempo em que a falha ocorreu, o tempo em que o recurso retornou e se houve perda de dados ou não. Esta informação é depois passada aos modelos de previsão de falhas, que ajustam os seus parâmetros.

Nos modelos de previsão de falhas independentes, o parâmetro ajustado é a frequência média de falhas de cada recurso. Este parâmetro é definido com um valor inicial e, além disso, define-se também um valor

que representa a confiança que se tem nesta previsão inicial. Com base nas medições feitas, é calculada a frequência média de falhas observada de cada recurso e um valor de confiança nesta medição, que se baseia no tempo durante o qual esta foi feita e se calcula como $c = \frac{tempo\,observacao}{(tempo\,observacao + valor\,confianca)}$. Com estes valores, pode-se calcular o novo valor do parâmetro como: $frequencia\,observada \cdot c + valor\,inicial \cdot (1 - c)$. Ou seja, o valor medido tem maior peso quanto maior for o tempo de observação e quanto menor for o valor de confiança na previsão inicial.

Como trabalho futuro, este mecanismo poderá ser estendido para detectar também falhas correlacionadas de nós e ajustar outros modelos de previsão. Contudo, para isto é necessário resolver dois problemas que não são triviais. O primeiro é classificar as falhas que são observadas em correlacionadas ou não, já que a falha de dois recursos, mesmo num intervalo de tempo curto, pode dever-se a duas falhas independentes e não haver correlação entre elas. O segundo problema deve-se ao facto de um modelo de previsão poder ser usado para modelar diferentes causas de falhas e a se permitir que o sistema de redundância utilize vários modelos. Isto faz com que seja difícil descobrir qual a causa de uma falha que se observa e, por isso, saber qual dos modelos de previsão ajustar.

5.3.2 Simulações

Este mecanismo permite que o sistema de redundância utilize o Serapeum para simular o comportamento do sistema e alterar o algoritmo e parâmetros de replicação de forma a que este atinja o desempenho desejado. Podendo a modelação das falhas inicial estar errada, esta é alterada automaticamente através da observação do sistema.

Para isto, é definido inicialmente um conjunto de algoritmos e parâmetros possíveis, bem como um a ser utilizado inicialmente. Além disso, é definida uma métrica que será usada para comparar o desempenho dos diferentes algoritmos. Periodicamente, são corridas simulações em background com todos os algoritmos e parâmetros possíveis, o seu desempenho é comparado utilizando a métrica e o que tiver o melhor desempenho passará a ser utilizado.

Exemplos de métricas que podem ser utilizadas são o número de perdas, que escolhe o algoritmo que apresentar um número inferior de perdas nas simulações, ou o débito minímo total utilizado com limite de perdas, que selecciona o algoritmo que apresente um número de perdas inferior a um certo limite e que utilize menos débito no total.

O cenário de falhas utilizado nas simulações é definido utilizando o mecanismo descrito anteriormente que permite ajustar os modelos através da observação do sistema. Assim, caso os parâmetros utilizados inicialmente nos modelos estejam errados, estes serão ajustados à medida que se observam falhas no sistema e, através das simulações, será escolhido o algoritmo e parâmetros replicação que melhor se ajustam a este novo cenário de funcionamento.

Por exemplo, pode-se querer um nível mínimo de perdas de dois ficheiros por década. Com a modelação inicial das falhas, chega-se á conclusão de que apenas são necessários três réplicas para isto. Contudo, no sistema é observado um nível de falhas superior ao previsto. Com a utilização deste mecanismo, serão corridas simulações com os modelos ajustados automaticamente e, caso estas indiquem que o nível de replicação indicado é de quatro, este será ajustado automaticamente. O mesmo acontecerá se as simulações

indicarem que outro dos algoritmos testado tem um desempenho superior ao que está a ser utilizado.

6 Avaliação

Neste capítulo são algumas experiências com que se pretende analisar propriedades dos algoritmos apresentados anteriormente e também demonstrar as funcionalidades do Serapeum.

Nestas experiências é medido o número de ficheiros perdidos, ou seja, a quantidade de perda de dados que ocorre e o débito total utilizado na rede de recursos, que pretende ser uma medida da quantidade de cópias efectuada durante o tempo de vida do sistema. Normalmente, existe uma relação inversa entre estes valores. Ou seja, uma maior quantidade de cópias significa que existem mais réplicas dos ficheiros e, portanto, deverão existir menos perdas. Isto é amplificado pelo facto de que quando existe perda de ficheiros o sistema passar a armazenar menos ficheiros e, portanto, tem menos cópias para efectuar. Assim, um algoritmo será tanto mais eficiente quanto menos ficheiros perder, utilizando um determinado débito total na rede.

Em 6.1 são definidos os sistemas sobre os quais foram corridas as simulações. Em 6.2 é feita uma análise comparativa da eficiência dos vários algoritmos apresentados. Para isto, são feitas experiências em que os algoritmos utilizam um modelo de previsões equivalente ao modelo de falhas que foi utilizado nas simulações, ou seja, têm um conhecimento estatístico sobre as falhas que corresponde ao ambiente em que vão operar. Na experiência de 6.4 observa-se como é que o tempo que o sistema espera entre falhas e a criação de réplicas afecta a eficiência dos vários algoritmos e no funcionamento do sistema. Finalmente, em 6.5 e 6.6 pretende-se descobrir qual o efeito que modelos de previsão de falhas incorrectos podem ter na eficiência dos algoritmos.

Com estas experiências, pretende-se ficar com uma ideia de que algoritmo dos apresentados será o melhor para utilizar num sistema destes e qual a importância de ter parâmetros e modelos de previsão correctos.

6.1 Cenários de simulação

Foram definidos dois sistemas diferentes para as simulações.

No primeiro, denominado “Simples”, existem 14 recursos de armazenamento com capacidade de 500Gb. Estes recursos estão ligados a uma única rede com ligações bidireccionais de 100Mbit/s. A colecção a ser armazenada consiste em 208 ficheiros de 5Gb cada um, tendo assim um tamanho total de 1Tb. Foi escolhido este tamanho grande para os ficheiros pois permite que as simulações corram mais rapidamente e, portanto, possam ser efectuados mais ensaios para cada experiência.

Neste cenário, os recursos estão espalhados por três salas diferentes e têm um de três sistemas operativos possíveis: “Windows”, “Linux” e “Solaris”, sendo os dois primeiros mais comuns que o ultimo. Cada sistema operativo suporta três serviços, que cada recurso pode ou não correr, com uma probabilidade de 50%.

A sala, sistema operativo e serviços corridos são causas possíveis para falhas correlacionadas nos recursos. Existem outras que poderiam ser consideradas, mas nesta experiência apenas são usadas estas. Um modelo de ameaças baseado em atributos é utilizado para gerar estas falhas. Em 6.1.1 são descritas estas ameaças em maior detalhe.

Além disso, são geradas falhas utilizando o modelo de ameaças que representa a substituição de discos descrito em 5.1.1. Considera-se ainda que existem falhas independentes sem perda de dados com uma frequência média de 1 ano para cada nó, do qual este demora em média 12h a recuperar.

No segundo cenário, existem 16 recursos que estão espalhados por 3 edifícios em 2 cidades diferentes. Dentro do mesmo edifício os recursos continuam a ter ligações de 100Mbit/s, mas entre edifício estas são de apenas 10Mbit/s. Devido a isto, existem novos atributos "Edifício" e "Cidade", que são mais fontes possíveis de falhas correlacionadas. Estes são descritos à frente em mais detalhe.

Nestas simulações, é utilizado um tempo até cópia de 10 dias. Ou seja, quando um recurso de armazenamento falha, só após 10 dias é que se considera que ele perdeu os seus dados e, caso necessário, se inicia a criação de réplicas. Na experiência de 6.4 é analisado como é que a variação deste parâmetro afecta o funcionamento do sistema.

6.1.1 Atributos e ameaças

No primeiro cenário, existem três atributos que podem ser causa de falhas correlacionadas: a sala em que os recursos se encontram, o seu sistema operativo e os serviços de software que fornecem (por exemplo, um servidor ssh).

Os recursos estão espalhados por três salas diferentes que, em termos de ameaças, são semelhantes. Ou seja, não existem probabilidades de falha diferentes de sala para sala. Considera-se que existem quatro ameaças que podem causar falhas numa sala:

- Um desastre, tal como uma inundação ou desabamento do tecto da sala
- Falha de energia, devido a uma falha no fornecimento ou ao mau funcionamento de algum equipamento.
- Falha de comunicações, devido à falha de algum equipamento de rede
- Um ataque em que haja intrusão física na sala e o equipamento seja destruído ou roubado

A frequência de falhas causadas por estas ameaças estão descritas na tabela 6.1 e os seus efeitos e tempos de recuperação na tabela 6.3.

Quanto ao sistema operativo e aos serviços, considera-se que ambos podem ser atacados por virus ou *worms*, sendo mais provavel que o ponto de vulnerabilidade seja um serviço especifico que o sistema operativo em si. Esta ameaça está descrita na tabela 6.2 e os seus efeitos na tabela 6.3.

No segundo cenário, existem também os atributos "Edifício" e "Cidade". Considera-se que as ameaças a estes são semelhantes às de uma sala, mas numa escala diferente. Por exemplo, as falhas de electricidade tanto podem afectar um edifício ou cidade inteira como apenas uma sala. Assim, considera-se que tanto o "Edifício" como a "Cidade" também podem sofrer falhas de energia, desastres (incluindo desastres naturais), ataques (sendo comprometidos todos os centros de dados de um edifício, por exemplo) e falhas de comunicações (como quando há problemas na linha para edifício). Contudo, as frequências com que estas ocorrem e os tempos de recuperação são bastante diferentes dos considerados para as "Salas".

	Desastre	Energia	Comunicações	Ataque físico
Frequência	15 anos	5 anos	5 anos	15 anos

Tabela 6.1: Ameaças às salas

	Worm ou vírus
Windows	30 anos
Linux	90 anos
Solaris	250 anos
Serviço Windows X	15 anos
Serviço Linux X	50 anos
Serviço Solaris X	150 anos

Tabela 6.2: Frequências de ataques de software

Evento	**Prob**	**Efeito**		**Recuperação**
		Perda de dados	Regresso	
Desastre natural	60%	Sim	Não-simultâneo	6 meses
	40%	Não	Simultâneo	1 mês
Falha de energia	95%	Não	Simultâneo	4 horas
	2.5%	Sim	Não-simultâneo	6 meses
	2.5%	Não	Não-simultâneo	2 semanas
Falha de comunicações	100%	Não	Simultâneo	3 dias
Ataque físico	90%	Sim	Não-simultâneo	6 meses
Worm ou vírus	35%	Sim	Não-simultâneo	2 semanas
	50%	Não	Não-simultâneo	2 semanas

Tabela 6.3: Efeitos de eventos

6.2 Simulações com conhecimento perfeito

Nesta experiência, os algoritmos utilizaram um modelo de previsão baseado em atributos e um de falhas individuais de recurso. Estes modelos foram parametrizados da mesma forma que os utilizados para gerar as ameaças, pelo que os algoritmos têm um conhecimento estatístico "perfeito" sobre as falhas.

Foram feitas simulações para um período de 50 anos e medidas as perdas de ficheiros e débito total utilizado. Testaram-se algoritmos com número de réplicas fixo, tanto gerando réplicas aleatoriamente como utilizando as heurísticas "Simples" e "Dupla", e também algoritmos com número de réplicas variável, utilizando também estas heurísticas. Para ambos os tipos, foram testados diversos níveis de replicação. Para cada um dos casos, foram corridas 100 simulações.

Na figura 6.1 estão representados os resultados obtidos nas simulações com número de réplicas fixo. Pode-se ver que, como seria de esperar, quando o número de réplicas aumenta, a quantidade de perdas decresce e o débito total utilizado aumenta. Além disso, pode-se verificar que a heurística de falha dupla tem um comportamento melhor que a simples já que, para o mesmo número de réplicas, permite perdas menores gastando pouco mais recursos da rede.

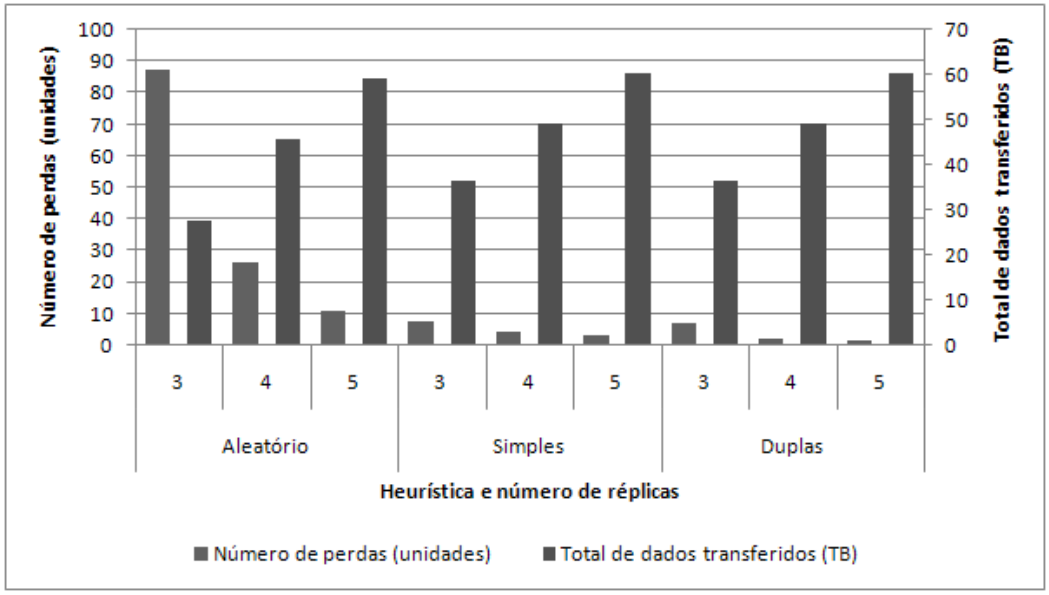

Figura 6.1: Número de réplicas fixo

As figuras 6.2 e 6.3 apresentam os resultados obtidos com os algoritmos com número de réplicas variável e as heurísticas "Simples" e "Dupla", respectivamente. Mais uma vez, quanto maior o factor de replicação, menor a quantidade de perdas que ocorre. Neste caso, o factor de replicação é representado pelo valor da heurística para os ficheiros, já que quando este é maior, têm que se criar mais réplicas para o satisfazer. É de notar que, apesar destes valores serem os mesmo para ambas as heurísticas, eles são calculados de forma diferente em cada uma, pelo que não deve ser utilizado para comparar os resultados. Em vez disso, deve-se utilizar a relação entre o débito total utilizado e a quantidade de perdas.

Pode-se ver que em 6.2, o débito total utilizado estabiliza. A explicação para isto, obtida após analisar outras estatísticas, é que todos os recursos de armazenamento do sistema estão a ser utilizados. Ou seja,

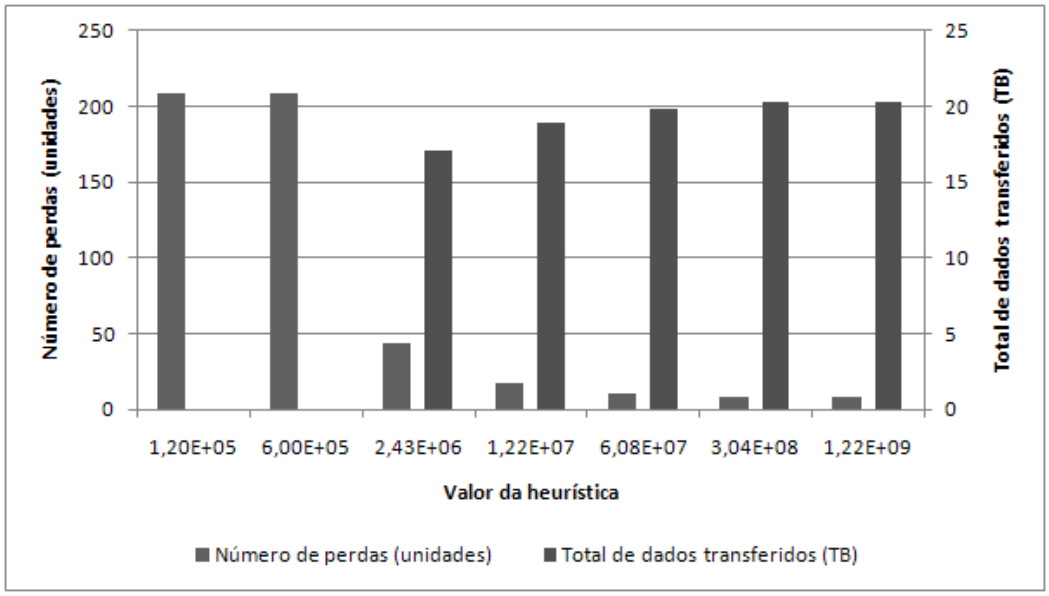

Figura 6.2: Número de réplicas variável, heurística “Simples”

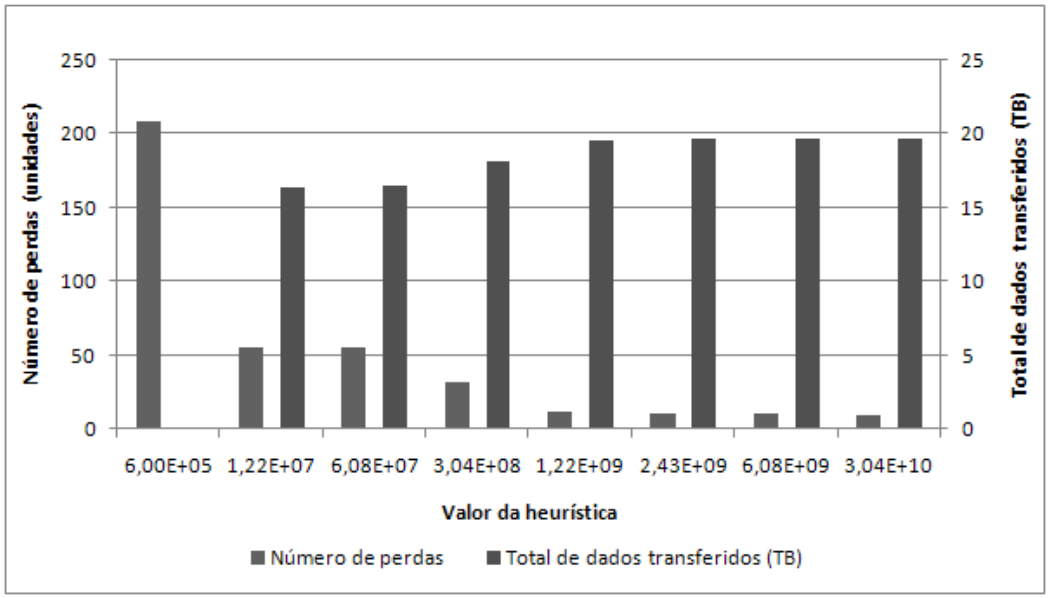

Figura 6.3: Número de réplicas variável, heurística “Dupla’

era impossível criar mais réplicas.

Apesar dos algoritmos com número de réplicas variável utilizarem muito menos débito, não são capazes de atingir o mesmo nível de perdas que os algoritmos com número de réplicas fixo. Isto acontece porque, com os modelos de previsão utilizados, estes algoritmos mantêm um número de réplicas que é sempre muito baixo, já que com as heurísticas utilizadas, isto lhes parece suficiente para evitar qualquer perda.

Para “forçar” estes algoritmos a criar um número maior de réplicas, foi adicionado um modelo de previsão de “outras falhas correlacionadas”, como o descrito em 5.1.3.2. Assim, o valor da heurística poderá sempre ser melhorado criando mais réplicas. As figuras 6.4 e 6.5 apresentam os resultados obtidos neste caso e mostram que assim se conseguem atingir níveis de perdas mais baixos.

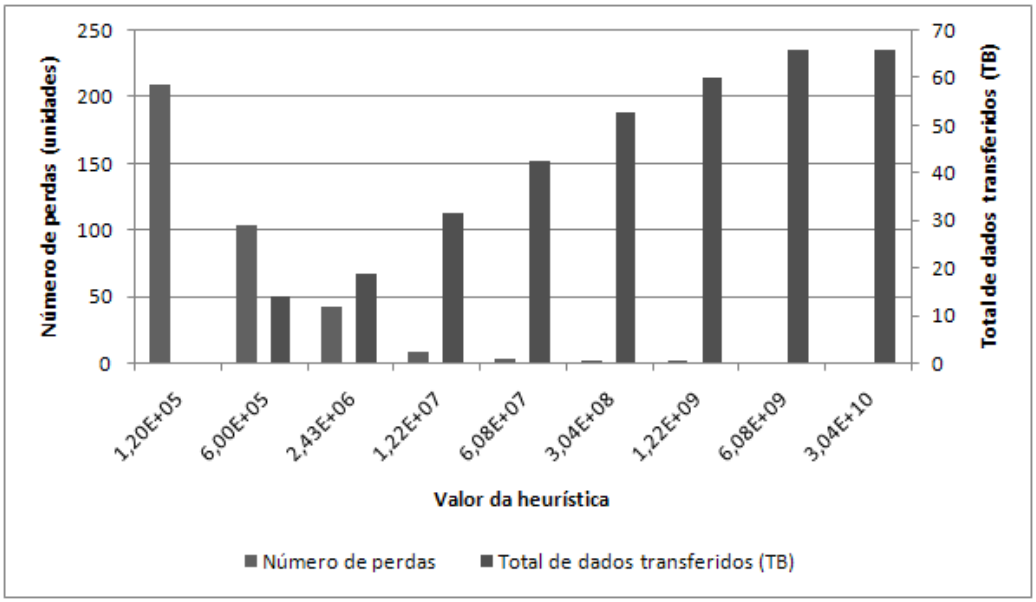

Figura 6.4: Número de réplicas variável, heurística “Simples”

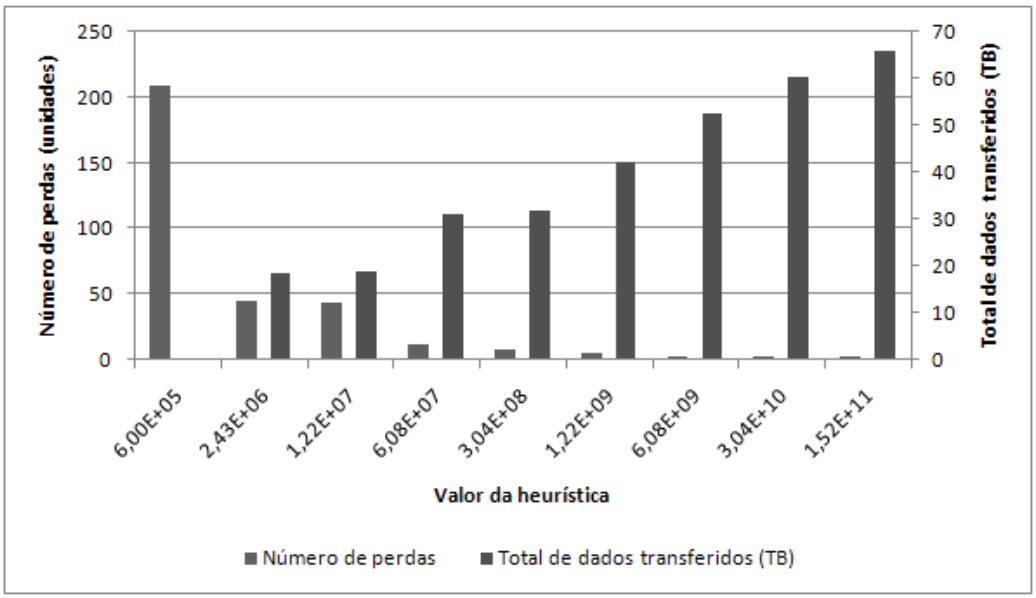

Figura 6.5: Número de réplicas variável, heurística “Dupla’

Os resultados dos vários algoritmos estão resumidos na figura 6.6. Aqui, pode-se ver que os algoritmos com número de réplicas variável têm performance semelhante ao algoritmo com número fixo utilizando a heurística “Dupla”. Contudo, os primeiros permitem maior flexibilidade, já que permitem definir mais “níveis” de replicação. Além disso, nos algoritmos com número de réplicas variável, a heurística “Simples” parece ter resultados ligeiramente melhores que a “Dupla”.

No segundo cenário, as simulações foram apenas feitas 50 vezes para cada caso, já que estas demoravam bastante mais tempo a correr que no primeiro. Os resultados obtidos estão resumidos na figura 6.7.

Como se pode ver, todos os algoritmos têm um número de falhas bastante elevado. Isto deve-se provavelmente ao menor débito da ligação entre cidades e à adição dos atributos “Cidade” e “Edifício”, que têm

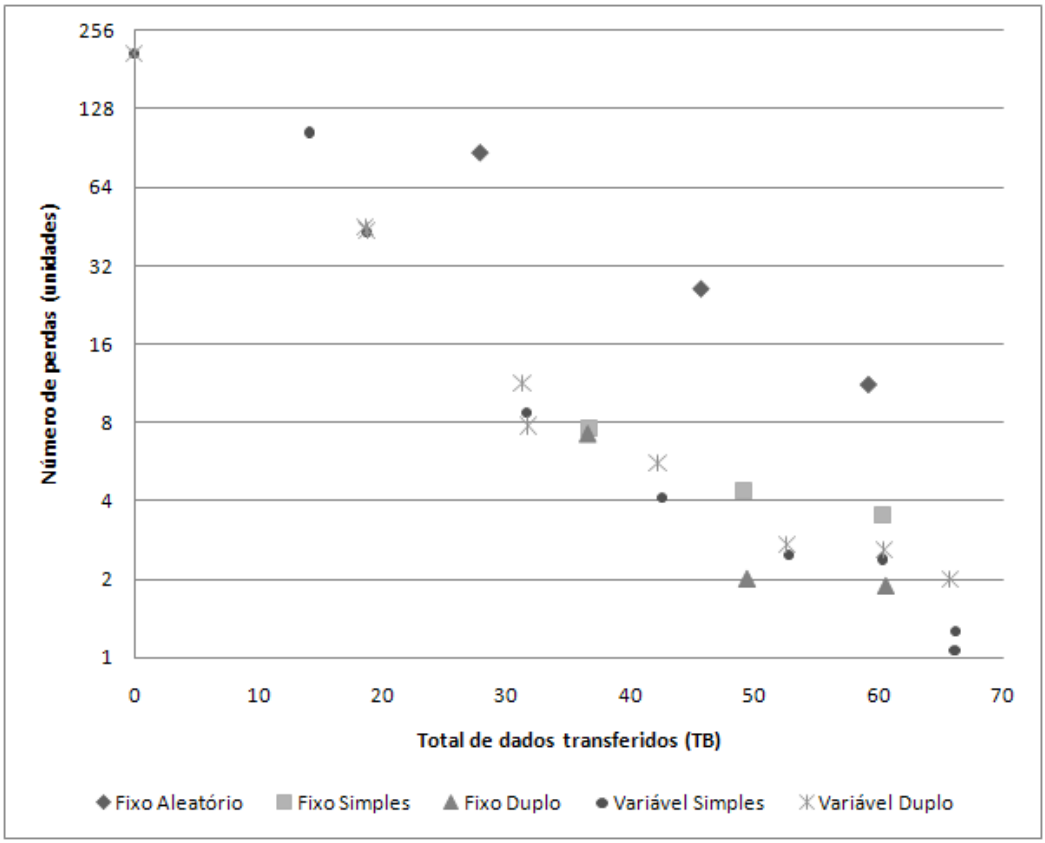

Figura 6.6: Débito utilizado vs perdas, primeiro cenário

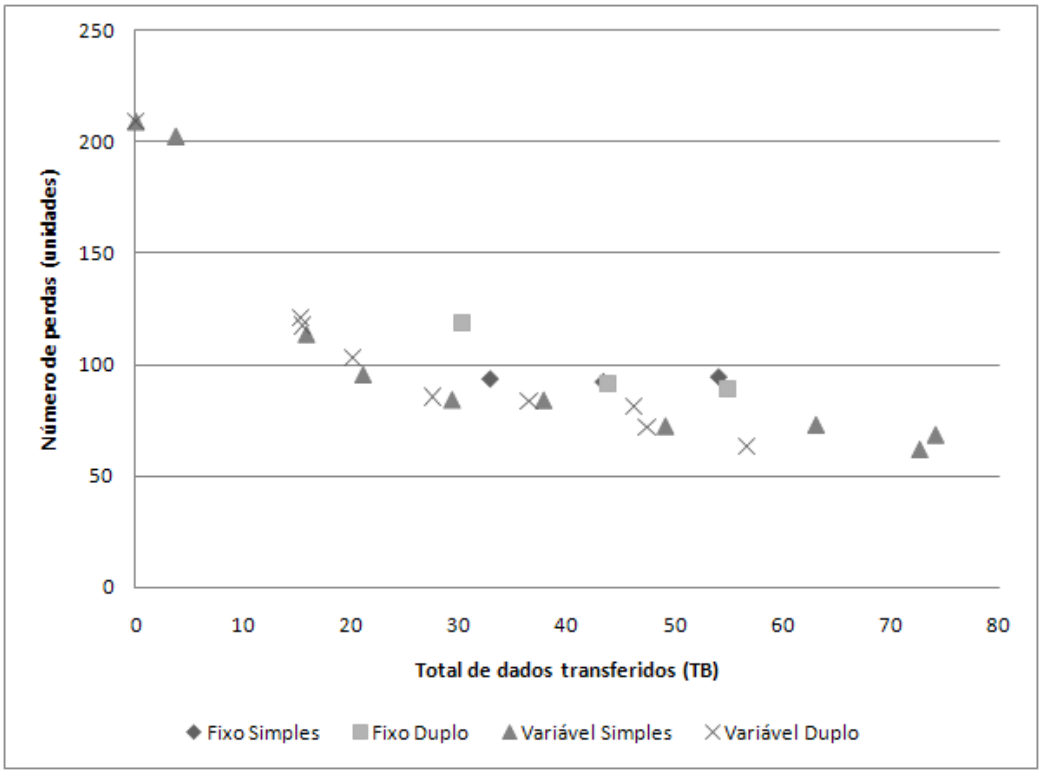

Figura 6.7: Débito utilizado vs perdas, segundo cenário

associadas mais causas possíveis para falhas. Os resultados obtidos para o algoritmo com número de réplicas fixo com a heurística “Simples” são um pouco estranhos, já que o número de perdas se mantém quase constante. Isto será provavelmente uma anomalia estatística que se deve ao facto de não ter sido corrido um número maior de simulações.

Estes resultados indicam que os recursos de armazenamento e os débitos das ligações não são suficientes para o sistema ter um nível aceitável de perdas com as ameaças que estão presentes. Isto é um exemplo de como o Serapeum pode ser utilizado para verificar se o sistema tem recursos suficientes para operar sob certas condições.

Além de aumentar a capacidade do sistema, outra coisa que poderia ajudar a reduzir o número de perdas aqui são algoritmos que tenham em conta os débitos das ligações entre recursos de armazenamento, já que estes podem variar bastante neste cenário. Contudo, isto implicaria fornecer de alguma forma informação sobre as ligações e os seus débitos aos algoritmos, o que seria um trabalho adicional para o administrador do sistema. Outra ideia seria obter informação sobre os débitos com introspecção, utilizando os tempos de transferências passadas.

6.3 Introspecção

Nesta secção é testado o comportamento dos mecanismos de introspecção descritos em 5.3. Para isto, foram feitas simulações com falhas independentes dos nós. Há 14 recursos, que têm tempos médios entre falhas diferentes, que estão apresentadas na tabela 6.4. Contudo, a modelação inicial das falhas prevê que esta seja igual para todos (neste caso 40 meses, que é a média). É utilizada introspecção para ajustar o valor da frequência de falhas para cada nó.

Nas figuras 6.8, 6.9 e 6.10 são apresentados os valores previstos para as frequências de falhas dos recursos ao longo de uma simulação. Cada uma das linhas é uma média das previsões para os nós que têm essa frequência. São utilizados valores diferentes para a confiança na previsão inicial, o que influencia a variação das previsões.

Como se pode ver, as previsões convergem para valores semelhantes à realidade. Devido à amostra de falhas maior, esta convergência é mais rápida quanto maior for a frequência de falhas.

Na figura 6.11 são apresentadas as perdas observadas ao longo de 100 simulações com e sem o mecanismo de introspecção. Foi utilizado um algoritmo com nível de replicação fixo em três replicas, a heurística de falha dupla e um nível de confiança na previsão inicial de 10 anos. Como se pode ver, a introspecção

Quantidade de recursos	Tempo médio entre falhas
2	6 meses
3	1 ano
4	2 anos
3	5 anos
2	10 anos

Tabela 6.4: Tempo médio entre falhas dos recursos

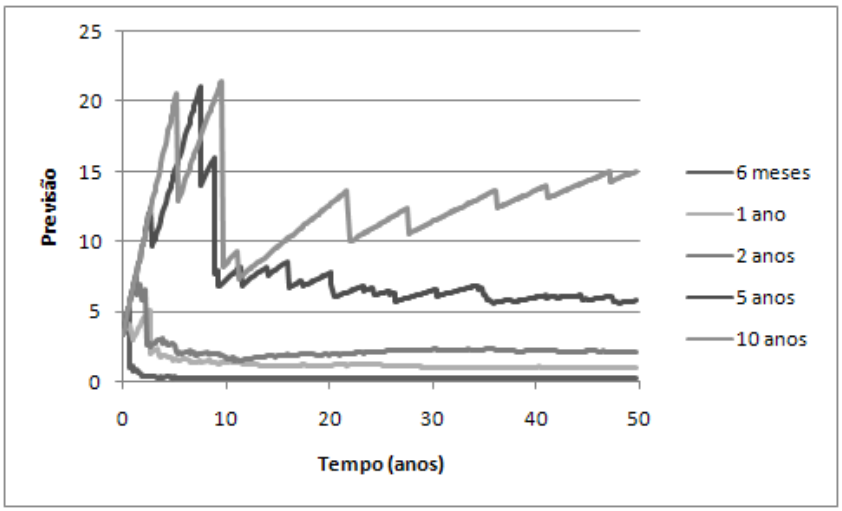

Figura 6.8: Previsões com valor de confiança de 1 ano

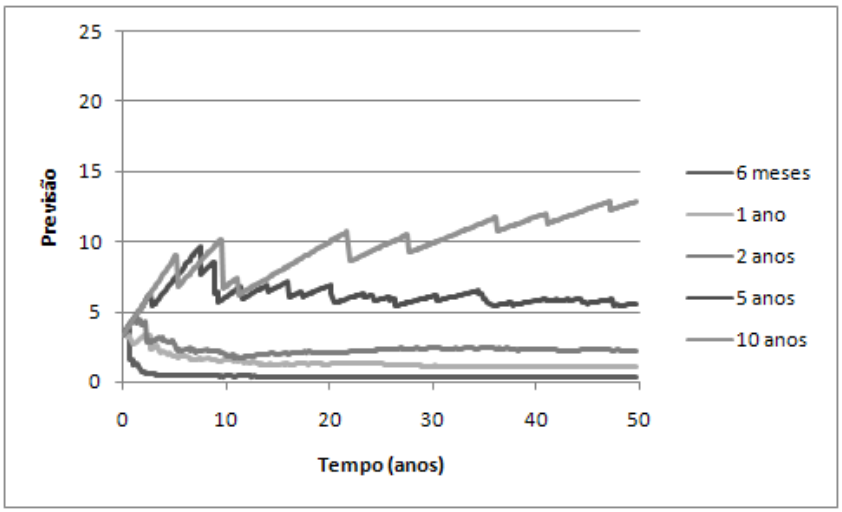

Figura 6.9: Previsões com valor de confiança de 3 anos

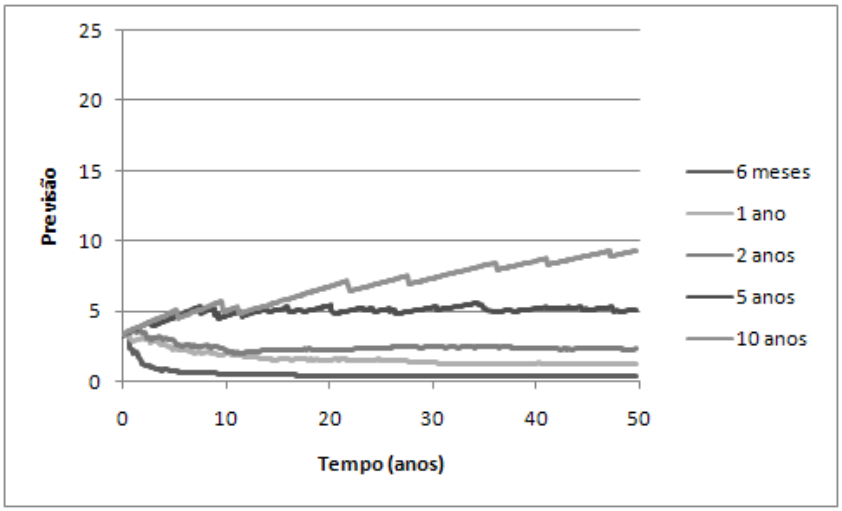

Figura 6.10: Previsões com valor de confiança de 10 anos

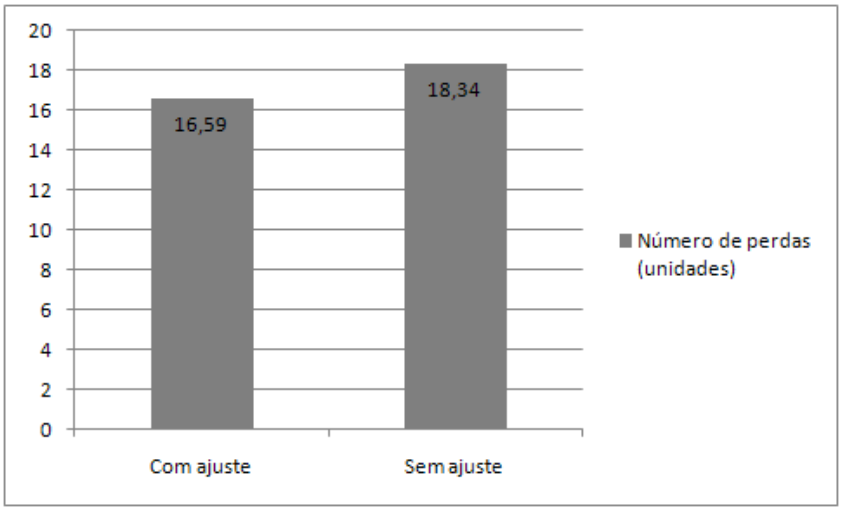

Figura 6.11: Falhas com e sem ajuste dos modelos de previsão de falhas

permite uma redução ligeira da quantidade de falhas, mesmo sendo este mecanismo muito simples, já que não permite a detecção de falhas correlacionadas entre nós.

Na próxima experiência, será testado o mecanismo que permite ao sistema de redundância correr simulações e ajustar o algoritmo e parâmetros a ser usados. Para ser possível fazer este teste em tempo útil, o sistema de replicação só corre as simulações de ano a ano. Aqui são também usadas falhas independentes com um tempo médio entre falhas de 1 ano. Contudo, a previsão inicial é de que este tempo é de três anos. Pretende-se ter um nível de perdas de ficheiros de dois por ano, o que, com a previsão inicial, poderá ser obtido com apenas três réplicas, como se pode ver na tabela 6.5. Esta tabela apresenta os valores obtidos para o número de perdas anuais utilizando três e quatro réplicas e com um tempo médio entre falhas de um e três anos.

Utilizando três réplicas gasta-se menos débito total do que com quatro, por isso, inicialmente opta-se pela primeira opção. Na figura 6.12 estão representados os valores médios para o tempo previsto entre falhas e o número de réplicas dos ficheiros do sistema. Como se pode ver, no ano 3 o número de réplicas passa de três para quatro. Isto ocorre porque nas simulações feitas com o valor previsto nessa altura para o tempo entre falhas (aproximadamente 1.8 anos), é observado que três réplicas já não são suficientes para manter um nível de perdas inferior a dois.

6.4 Tempo até cópia

Como referido anteriormente, nesta experiência avalia-se como é que uma variação no tempo até cópia afecta a quantidade de perdas e o débito utilizado para vários algoritmos.

	3 réplicas	4 réplicas
3 anos	0.45	0.17
1 ano	4.56	0.79

Tabela 6.5: Número médio de perdas por ano para um tempo médio entre falhas

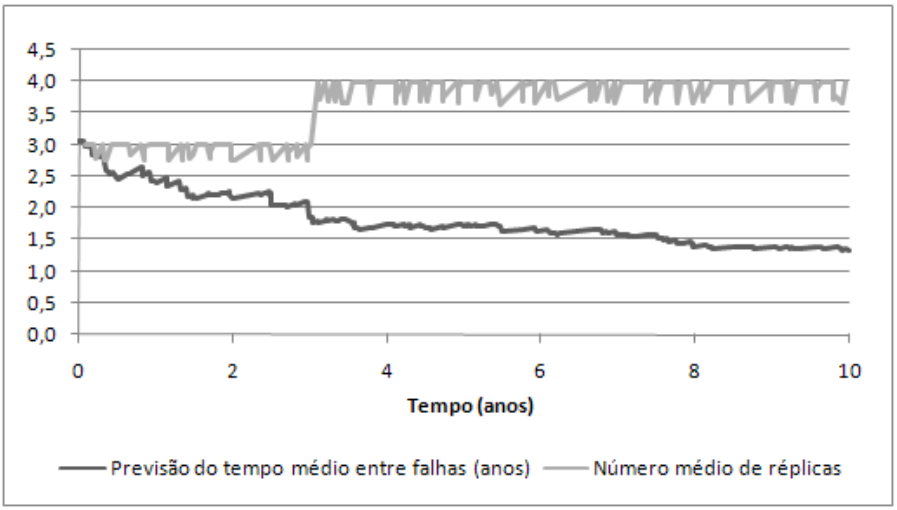

Figura 6.12: Previsão e nível de replicação ao longo do tempo

As simulações foram feitas no primeiro cenário descrito em 6.1. Nos algoritmos com número de réplicas fixo foram usadas 3 réplicas, enquanto que nos com número de réplicas variável, foi usado o valor 12160000 para a heurística Simples e 304160000 para a Dupla.

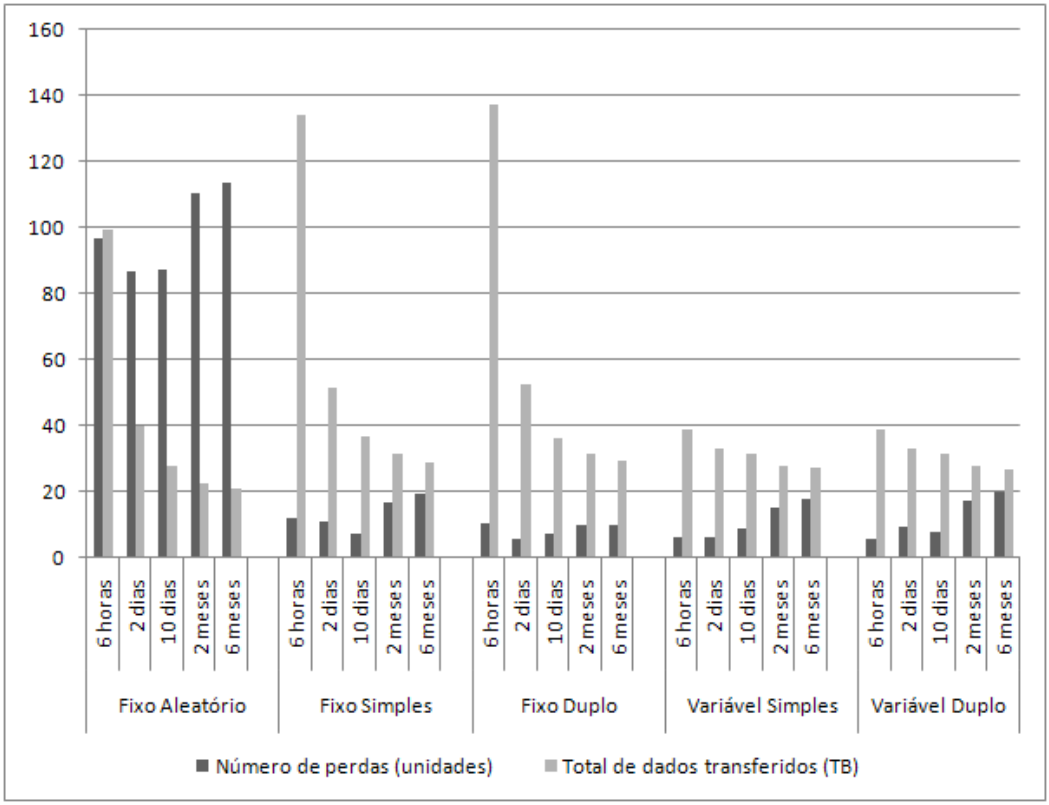

Figura 6.13: Perdas e débito utilizado

Como se pode ver na figura 6.13, o débito utilizado é maior quanto menor for o tempo até cópia. Isto seria de esperar, já que as cópias são efectuadas mais frequentemente neste caso. Contudo, o número de perdas não decresce da mesma forma, sendo maior para valores muito altos e muito baixos do tempo. A explicação

para isto é que com um tempo muito baixo, são criadas réplicas desnecessariamente muitas vezes, o que faz com que o sistema fique sobrecarregado e acabe por ter maiores perdas.

Como se pode ver, o valor do tempo até cópia pode ter bastante impacto no funcionamento do sistema, e o seu valor ideal pode ser difícil de descobrir. Apesar do Serapeum ajudar nesta tarefa, seria interessante no futuro estudar mecanismos que permitissem ajustar este valor automaticamente utilizando introspecção. Ou seja, que após ocorrer uma falha tentassem prever, utilizando conhecimento de falhas passadas, qual a probabilidade dos recursos de armazenamento falhados terem perdido os dados ou não.

6.5 Parâmetros incorrectos

Nesta experiência foi gerado um cenário de falhas diferente para cada ensaio, variando-se os parâmetros do modelo de falhas baseado em atributos descritos em 6.1, em particular, o tempo entre falhas e o tempo médio de recuperação. Para cada um desses modelos de falhas, foi corrida uma simulação em que o algoritmo utilizava um modelo de previsão “correcto” (em que os parâmetros correspondiam ao modelo de falhas gerado) e outra em que utilizava um “incorrecto"(com os parâmetros descritos em 6.1).

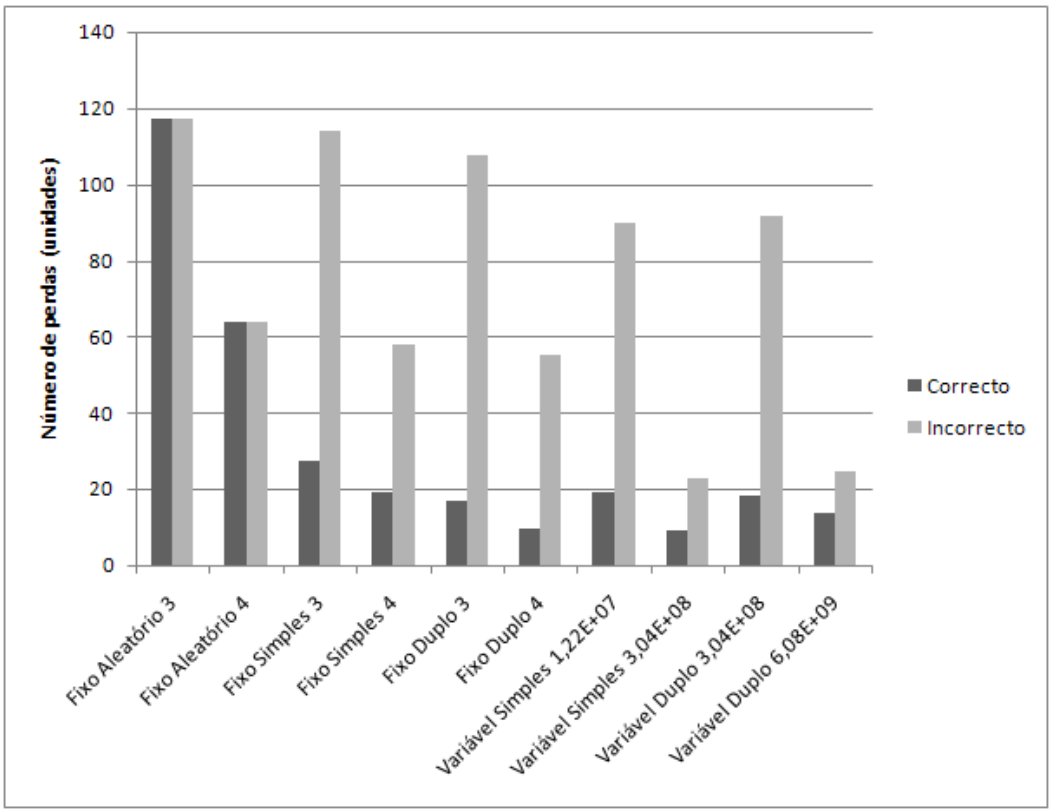

Figura 6.14: Perdas e débito total utilizado com parâmetros correctos e incorrectos

Na figura 6.14 pode-se observar o impacto de um modelo de falhas incorrecto no desempenho dos algoritmos. Obviamente, o algoritmo aleatório não é afectado, já que não utiliza qualquer modelo de previsão, mas em todos os outros os efeitos conseguem observar-se.

Como se pode ver, o desempenho dos algoritmos é mais afectado quando o número de réplicas utilizado é menor, já que isto faz com que se tornem muito mais vulneráveis a falhas “imprevistas”. Este efeito é tão

forte que os algoritmos com número de réplicas fixo chegam a ter um desempenho semelhante ao aleatório quando têm um modelo de previsão errado. Isto demonstra a importância de ter modelos de previsão que se aproximem o mais possível da realidade.

6.6 Falhas desconhecidas

Nesta experiência foram adicionadas algumas falhas aleatórias, utilizando o modelo descrito em 5.1.3, com um nível de correlação de 0.4 e frequências de falhas de 1 ano, 3 anos e 9 anos. Foram também efectuados ensaios sem falhas aleatórias. Foram testados vários algoritmos diferentes e, para cada caso, foram corridas 300 simulações.

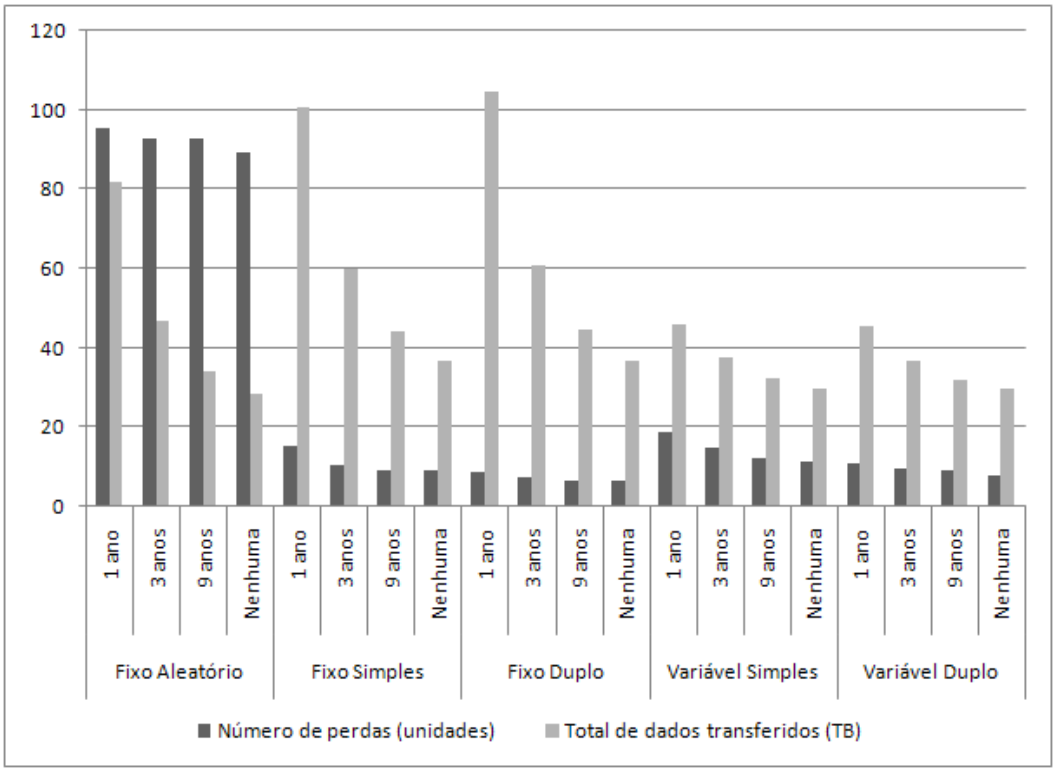

Figura 6.15: Perdas e débito utilizado com falhas desconhecidas

Na figura 6.15 estão representados os resultados obtidos. Como se pode ver, tanto o débito utilizado como a quantidade de falhas aumentam quando existem mais falhas. Contudo, ao contrário do que acontece na experiência anterior, a variação do número de perdas em relação à quantidade de falhas é pequena em todos os algoritmos, mesmo naqueles que utilizam um modelos de previsão para a replicação. A explicação para isto é que, mesmo que os seus modelos de previsão estejam agora incompletos, como as falhas afectam um conjunto aleatório de recursos de armazenamento, estes modelos continuam a ser válidos para prever os conjuntos de recursos em que é mais provável haver falhas correlacionadas.

7 Conclusões

A simulação pode ser usada para analisar o comportamento de sistemas de armazenamento e, assim, encontrar o melhor algoritmo e parâmetros a serem utilizados. Isto pode ser feito não só manualmente por um operador humano, mas também automaticamente, de forma a permitir ajustar os parâmetros do sistema durante a sua execução. Neste trabalho foi implementado um simulador que permite ambas estas coisas.

Através da implementação do protótipo de um sistema de redundância, pode-se concluir que é possível utilizar *grids* de dados para implementar sistemas de armazenamento para preservação digital.

Foram propostas estratégias de replicação que têm em conta as correlações entre as falhas dos recursos de armazenamento. Mostrou-se que estas permitem melhorar a eficiência de um sistema de armazenamento, potenciando menos perdas de dados. Contudo, para isto é necessário fazer uma boa modelação estatística das falhas a que o sistema está sujeito.

Foram implementados mecanismos de introspecção que podem ser usados para compensar modelos de falhas que não estejam correctos, ajustando-os durante o tempo de vida do sistema. Contudo, isto apenas pode ser usado para falhas que ocorrem frequentemente, já que é necessário uma amostra grande para fazer estes ajustes. Ou seja, a modelação de falhas raras, como desastres naturais, dificilmente poderá ser melhorada utilizando introspecção.

Contudo, foi também identificado muito trabalho a ser feito para a criação de um sistema de armazenamento fiável usando grids. Como estudar melhor como fazer a modelação das falhas que afectam um sistema destes, a utilização de erasure coding, mecanismos que possam ser utilizados para interacção entre diferentes tipos de grids e outras estratégias de redundância. Além disso, para a utilização de *grids* iRODS, é ainda necessário que este software evolua e suporte mais funcionalidades, tal como previsto. Estes problemas são descritos melhor na secção seguinte.

7.1 Trabalho Futuro

7.1.1 Estratégias de replicação

A utilização de introspecção pode permitir melhorar o desempenho das estratégias de replicação. Esta técnica pode ser utilizada para coisas como:

- Prever o tempo de recuperação de réplicas para as métricas descritas em 5.2.1
- Variar o nível de replicação conforme os recursos disponíveis do sistema
- Prever se uma falha que tenha causado indisponibilidade causou também perda de dados

As estratégias de replicação podem ter em conta os débitos da rede de recursos, por exemplo, para fazer as cópias de ficheiros entre recursos que tenham um débito mais elevado. Estes débitos poderão ser definidos manualmente pelo utilizador ou ser descobertos automaticamente utilizando introspecção.

Um factor limitativo da análise feita neste trabalho das diferentes estratégias de replicação foi o elevado tempo que as simulações demoram a correr. Assim, com uma capacidade de processamento maior, poderá ser feita uma análise mais profunda destas estratégias. Isto pode incluir simular sistemas maiores ou com um número maior de ficheiros.

O Serapeum pode ser utilizado para ajudar no desenvolvimento e estudo de novas estratégias de replicação. Estas estratégias podem, por exemplo, considerar utilizar níveis de replicação diferentes para ficheiros diferentes, conforme uma importância definida pelo utilizador. Pode-se também considerar variar este nível conforme a "raridade" de um ficheiro, replicando menos os ficheiros que também estão disponíveis noutros sistemas externos, por exemplo, num sistema LOCKSS. Isto implicaria desenvolver um mecanismo que permitisse interagir com esses sistemas.

A adição de uma operação de cancelar criação de réplica, além das existentes de criar e eliminar réplica a sua utilização em estratégias de redundância.

7.1.2 Sistema de replicação

O trabalho mais importante a realizar é, possivelmente, estudar mecanismos que permitam que várias *grids* interajam entre si, para ser possível copiar um ficheiro de uma *grid* para outra de forma a criar uma nova réplica. Isto poderá ser feito directamente caso as *grids* forneçam essa funcionalidade. No iRODS, está prevista a implementação da funcionalidade que permita a várias grids formarem uma federação (tal como no seu antecessor, o SRB), sendo assim possível copiar ficheiros directamente de uma para outra. Contudo, quando isto não é possível, pode ter que se usar outros mecanismos para fazer a transferência de ficheiros, como um intermediário que leia o ficheiro de um lado e o escreva no outro. Obviamente, isto terá impacto no desempenho de um sistema, o que deve também ser estudado.

A implementação de tolerância a faltas poderá ser feita utilizando servidores que monitorizem o estado do sistema de replicação e, caso este falhe, assumam eles próprios esse papel. Como as informações sobre o estado do sistema são obtidas da grid, não tem de haver sincronização entre os servidores, o que faz com que este problema não seja muito complexo.

O driver para interacção com o iRODS tem ainda que ser completado. Além disso, podem ser desenvolvidos drivers para outros sistemas grid, como o Globus.

Podem também ser estudadas formas de fazer migração de formatos no próprio sistema de armazenamento, permitindo que a migração seja feita pelos próprios nós da *grid* que armazenam os ficheiros e não sendo necessária qualquer transferência de informação para fora da grid.

Para facilitar a administração, pode-se desenvolver uma GUI que permita visualizar informação sobre o estado do sistema, como o nível de replicação dos ficheiros, a percentagem de utilização dos recursos de armazenamento e o seu estado.

7.1.3 Modelos de falhas

Apesar do levantamento feito sobre as causas de falhas em sistemas de armazenamento distribuídos, uma das maiores dificuldades encontradas neste trabalho foi a quantificação das ameaças, ou seja, definir valores para coisas como o efeito que um terramoto tem nos nós do sistema. Apesar de ser muito difícil, senão impossível, prever estes valores com precisão, deverá conseguir ter-se uma ideia melhor deles estudando as falhas de sistemas existentes. Um caso particular disto são as falhas de discos, sobre as quais já estão a ser feitos estudos, como o descrito em [31].

Outra coisa que pode facilitar a criação de modelos de falhas é automatizar partes deste processo. Por exemplo, a informação estatística sobre desastres naturais pode ser obtida a partir de sites disponíveis ao publico.

7.1.4 Simulador

Actualmente, a informação sobre o estado do sistema é guardada em estruturas de dados Java. Por razoes de eficiência, a mesma informação pode estar duplicada em várias estruturas, o que faz com que seja necessário manter a consistência da informação. A utilização de uma base de dados pode eliminar este problema, permitindo aos algoritmos obter qualquer tipo de informação de forma eficiente, sem que seja necessário estar a desenvolver novas estruturas.

Tal como descrito em 4.3.2, o simulador pode ser melhorado permitindo a adição de ficheiros, de recursos de armazenamento e alteração das caracteristicas da rede durante o tempo de vida simulado.

O simulador também poderia beneficiar com uma interface gráfica que proporcionasse uma utilização mais intuitiva, ou mesmo de uma interface que permitisse uma sessão interactiva, sendo possível parar a simulação e adicionar eventos em runtime.

Bibliografia

[1] R. Moore W. Schroeder A. Rajasekar, M. Wan. Prototype rule-based distributed data management system. In *HPDC Workshop on Next Generation Distributed Data Management*, May 2006.

[2] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger 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.

[3] Gonçalo José Branquinho Antunes. GRITO: Utilização de clusters GRID para um sistema de preservação digital. Master's thesis, Instituto Superior Técnico, Portugal, 2008.

[4] Andrea Arpaci-dusseau, Remzi Arpaci-dusseau, John Bent, Brian Forney, Sambavi Muthukrishnan, Florentina Popovici, and Omer 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.

[5] 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.

[6] Mary Baker, Mehul Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, TJ Giuli, and Prashanth 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.

[7] Mehmet Bakkaloglu, Jay J. Wylie, Chenxi Wang, and Gregory R. Ganger. On correlated failures in survivable storage systems. Technical report, 2002.

[8] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey 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.

[9] William J. Bolosky, John R. Douceur, and Jon Howell. The farsite project: a retrospective. *SIGOPS Oper. Syst. Rev.*, 41(2):17–26, 2007.

[10] Peter M Chen, Edward K Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. Raid: High-performance, reliable secondary storage, 1993.

[11] John R. Douceur and Roger Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 48–62, London, UK, 2001. Springer-Verlag.

[12] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

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

[14] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak 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.

[15] Andreas Haeberlen, Alan Mislove, and Peter 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.

[16] Flavio Junqueira, Ranjita Bhagwan, Alejandro Hevia, Keith Marzullo, and Geoffrey 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.

[17] Kimberly Keeton and John 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.

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

[19] Ramakrishna Kotla, Lorenzo Alvisi, and Michael Dahlin. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142. USENIX, 2007.

[20] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*. ACM, 2000.

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

[22] Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 17–17, Berkeley, CA, USA, 2006. USENIX Association.

[23] Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 17–17, Berkeley, CA, USA, 2006. USENIX Association.

[24] University 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.

[25] James Reason. *Human error*. Cambridge University Press, 1990.

[26] David S. H. Rosenthal, Thomas S. Robertson, Tom Lipkis, Vicky Reich, and Seth Morabito. Requirements for digital preservation systems: A bottom-up approach. *D-Lib Magazine 11*, 2005.

[27] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

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

[29] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[30] Bianca Schroeder and Garth 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.

[31] Thomas Schwarz, Mary Baker, Steven Bassi, Bruce Baumgart, Catherine Van Ingen, Kobus Joste, Mark Manasse, and Mehul Shah. Disk failure investigations at the internet archive. In *In Work-in-Progess session, NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST2006*, 2006.

[32] Roy Sterritt and Dave Bustard. Autonomic computing - a means of achieving dependability? *Engineering of Computer-Based Systems, IEEE International Conference on the*, 2003.

[33] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[34] Hakim Weatherspoon and John 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.

[35] Hakim Weatherspoon, Tal Moscovitz, and John Kubiatowicz. Introspective failure analysis: Avoiding correlated failures in peer-to-peer systems. In *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, page 362, Washington, DC, USA, 2002. IEEE Computer Society.

[36] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. In Hai Jin, Toni Cortes, and Rajkumar 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.

[37] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliççöte, and Pradeep K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, 2000.

[38] Praveen Yalag, Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan. Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems. In *In Proc. of USENIX Workshop on Real, Large Distributed Systems (WORLDS*, 2004.