UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

OurBackup: Uma Solução P2P de Backup Baseada em Redes Sociais

Marcelo Iury de Sousa Oliveira

(Mestrando)

Walfredo Cirne

(Orientador)

Campina Grande – PB
Maio - 2007

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

OurBackup: Uma Solução P2P de Backup Baseada em Redes Sociais

Marcelo Iury de Sousa Oliveira

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Orientador: Walfredo Cirne

Campina Grande – PB
Maio - 2007

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

O48o
2007 Oliveira, Marcelo Iury de Sousa
OurBackup: uma Solução P2P de Backup Baseada em Redes Sociais / v
Marcelo Iury de Sousa Oiliveira . — Campina Grande , 2007.
134f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia elétrica e Informatica.
Referências.
Orientador: Dr. Walfredo Cirne.

1. Sistemas Distribuídos. 2. Sistemas Peer-to-peer. 3. Backup. 4. Redes Sociais. I. Título.

CDU- 621.324(043)

"*OURBACKUP:* UMA SOLUÇÃO P2P DE BACKUP BASEADA EM REDES SOCIAIS"

MARCELO IURY DE SOUSA OLIVEIRA

DISSERTAÇÃO APROVADA EM 07.05.2007

PROF. WALFREDO DA COSTA CIRNE FILHO, Ph.D
Orientador

PROF. DALTON DARIO SERBY GUERRERO, D.Sc
Examinador

PROF. LUCIANO PASCHOAL GASPARY, Dr.
Examinador

CAMPINA GRANDE – PB

"Desconectar o cabo de rede é uma ótima opção quando estamos escrevendo a dissertação."

(Ayla Débora Dantas Rebouças)

Agradecimentos

Antes de tudo, agradeço a Deus.

Aos meus pais, Dona Neuma e Dr. Ilom, pela preocupação e por ter contribuído em toda minha formação.

Ao amor da minha vida, Vanessa, por seu carinho, ajuda e compreensão durante a realização deste trabalho.

Ao meu Orientador e Guru Walfredo, pela amizade, pela orientação científica e filosófica e por ter acompanhado de perto meu trabalho tendo competência em guiá-lo até sua conclusão.

A Izabelle, Tamires e Izadora que sempre me recorreram nos momentos de saudade. (Adoro muito vcs!)

Aos professores Fubica e Dalton, por toda a ajuda dispensada em diversas etapas deste trabalho e pelas contribuições fundamentais para a conclusão do mesmo.

Aos meus companheiros, amigos e irmãos, Alexandro, Eduardo, Paolo e Milena que me ajudaram a desenvolver o OurBackup, ensinaram muitas coisas e ajudaram no decorrer deste trabalho.

Aos meus amigos do Mestrado e do Laboratório de Sistemas Distribuídos, que contribuíram para que o meu ambiente de trabalho fosse sempre agradável. Em especial a Erica Gallindo, Ayla, Guiga, Flavio Barata, Elizeu, Daniel Fireman, Mirna, Marcell, Nazareno, Lívia, Raquel, Aliandro, Alexandre, Eric Moreno, Nigini, Thiago e Moises, Amanda, Lile, Celso e Degas e a todos os outros que me esqueci agora.

Ao povo que entrou no LSD no mesmo período que eu e que estimo muito, em especial a Ana Cristina, Pergentino, Marcelo Mercadinho, Bárbara, William e Matheuz B2 pelos momentos felizes que passamos juntos. Agradeço também a Zane, Keka e Marcelo Meira pelo apoio e amizade.

Ao pessoal do Apt 103B do Caio Lela, Alexandre e Rodrigo, que me suportaram neste período de mestrado. E aos amigos que fiz durante o tempo de mestrado que me ajudaram a me adaptar a cidade, em especial a Larissa, Yuri (Primo), Caio, Moema, Eveline, Zé de Guga, Harison e Jailson Capitão.

Para o Fulano que descobriu o café. Sem esta bebida não teria conseguido trabalhar com afinco durante as madrugadas solitárias.

Resumo

Os computadores estão cada vez mais presentes nas nossas vidas, facilitando a execução de nossas atividades e permitindo armazenar informações digitalmente. Todavia, os computadores também tornaram mais fácil a perda acidental de dados. Uma solução para evitar ou amenizar perdas de dados é a realização de backup (*i.e.* cópias de segurança). Visto que boa parte dos computadores não utiliza totalmente a sua capacidade de armazenamento, podemos utilizar esse espaço de armazenamento ocioso para a realização de backup. Há diversas propostas de sistemas *peer-to-peer* (P2P) que exploram capacidade de armazenamento ocioso para realizar backups. Naturalmente, cada vez que um *peer* deixar permanentemente o sistema, os dados que ele armazenava precisam ser copiados para outros *peers*, preservando, assim, a redundância de dados do backup. O problema é que a taxa de abandono dos sistemas P2P existentes sugere que a quantidade de banda necessária para manter a redundância do backup seja maior que a banda hoje existente. Este trabalho apresenta uma solução para este problema. A idéia básica que exploramos é que um sistema P2P baseado em rede social, por sua natureza, terá uma taxa de abandono muito mais baixa de que os sistemas P2P atuais (onde os *peers* são anônimos), permitindo a praticabilidade de sistemas P2P de backup.

Abstract

Computers help our lives and allow us to store information digitally. With this greater digitization of our lives comes the greater possibility of a catastrophic loss. A solution to prevent data loss is to backup data. Since the capacities of modern hard disks have outgrown the needs of many users, leaving them with much idle storage space, we could use this space of idle storage for the accomplishment of backups. There are several proposals of peer-to-peer (P2P) systems that exploit unused storage space to backup data. Naturally, every time a peer leaves permanently the system, all data stored by it need to be transferred to another peer, preserving, thus, the redundancy of data of backup. The problem is that the rate of permanent departures of existing P2P systems suggests that the amount of bandwidth necessary to keep the redundancy of backup is greater than the current existing bandwidth. This work presents a solution for this problem. The basic idea that we explore is that social network-based P2P system will, by its nature, have rate of permanent departures so much lower of that current P2P backup systems (which peers are anonymous), allowing the feasibility of P2P backup systems.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

Os computadores estão cada vez mais presentes nas nossas vidas, facilitando a execução de nossas atividades e permitindo armazenar informações digitalmente. Fotos, músicas, documentos, informações financeiras, entre outros, são todos agora armazenados regularmente em nossos computadores. Todavia, os computadores também tornaram mais fácil a perda acidental de dados. Vírus de computador, falhas de hardware ou erros de usuário podem corromper os dados armazenados em nossos computadores. Uma solução para evitar ou amenizar perdas de dados é a realização de backup (*i.e.* cópias de segurança).

Existem várias técnicas de backup, sendo a mais simples a cópia dos dados para mídias removíveis (*e.g.* CD-R e DVD-R) ou para outros dispositivos de armazenamento (*e.g.* HDs). Apesar de ser bastante simples, esta solução requer que o usuário realize o processo de cópia dos dados manualmente. Além disso, a ocorrência de falhas catastróficas (*e.g.* roubo, incêndio) pode comprometer permanentemente todos os backups. Outra alternativa é a utilização de serviços Web de backup. O usuário utiliza uma ferramenta para escalonar os backups e enviar pela Internet os dados para um local seguro. Os dados são copiados para locais geograficamente diferentes, diminuindo, assim, o risco de todas as cópias serem afetadas por uma catástrofe. Estes serviços não são gratuitos e, além disso, consomem um tempo considerável para realizar a transferência dos dados. O uso de uma solução simples que não requeira nenhum custo adicional para o usuário, em termos de *hardware* e *software*, e que realize o processo de backup de forma automática pode aumentar a prática de backup pelos usuários domésticos.

Visto que boa parte dos computadores não utiliza totalmente a sua capacidade de armazenamento [DB99], poderíamos utilizar esse espaço de armazenamento ocioso para a realização de backup. Naturalmente, falhas de *hardware* tornam inapropriado realizar backup de dados de um HD no espaço ocioso dele mesmo. Entretanto, uma coleção de PCs pode ser conectada de forma colaborativa utilizando o espaço de disco livre de cada computador para a realização de backups mútuos.

1.1 Backup Peer-to-Peer

Existem vários sistemas *peer-to-peer* (P2P) [MMGC02, KBC+00, RD01a] que exploram espaço de armazenamento ocioso para construir um sistema cooperativo de armazenamento. Sistemas distribuídos de armazenamento possuem como requisito fundamental garantir alta disponibilidade dos dados. Definimos *disponibilidade* como a probabilidade de um dado poder ser acessado em um dado instante de tempo *t*.

Um sistema para prover disponibilidade necessita se preocupar com todas as falhas que podem impossibilitar permanentemente ou temporariamente o acesso aos dados. Desta forma, sistemas de armazenamento se utilizam de redundância de dados para tolerar a ocorrência de falhas. Quando o fator de redundância cai devido à ocorrência de falhas, novas cópias precisam ser criadas para substituir aquelas que falharam.

Blake e Rodrigues [BR03] mostraram que não é possível garantir alta disponibilidade, armazenamento escalável e suporte à participação intermitente dos *peers* sem extrapolar o montante de largura de banda hoje disponível. O problema é que, em sistemas P2P atuais, os *peers* tendem a sair e retornar freqüentemente à rede, gerando uma alta taxa de falhas. Assim, cada vez que um *peer* deixa o sistema, a redundância deve ser reparada, armazenando novas cópias dos dados em outros *peers*. Criar tais cópias, contudo, consome grandes quantidades de largura de banda, tornando impraticável a manutenção de disponibilidade. Considerando que é improvável que a razão entre os requisitos de espaço de armazenamento e da capacidade dos canais de comunicação em instalações domésticas venha a mudar significativamente nos próximos anos [BR03], podemos concluir que é necessário reduzir a taxa de falha dos *peers* para melhorar a capacidade de armazenamento de sistemas P2P.

Há diversas propostas de sistemas P2P [LZT04, CMN02, BBST02, LEB+03] que exploram capacidade de armazenamento ocioso para realizar backups. Embora os usuários de sistemas de backup possam tolerar a indisponibilidade ocasional dos seus dados, eles requerem garantias fortes de que seus dados nunca serão perdidos. Assim, diferentemente de armazenamento, os sistemas de backup não requerem uma disponibilidade elevada, sendo o grande objetivo de um sistema de backup garantir a confiabilidade dos dados que estão sendo

suportados. Confiabilidade é a habilidade de recuperar o backup em um intervalo finito de tempo. Ela é geralmente medida pela probabilidade de um dado não ser corrompido permanentemente.

A definição de confiabilidade motiva distinguir falhas permanentes (o dado é definitivamente perdido) de transientes (o dado retorna intacto depois de algum tempo). Em sistemas P2P, as falhas transientes e permanentes ocorrem freqüentemente devido às saídas temporárias e permanentes do sistema. As saídas são temporárias quando o *peer* torna a se conectar ao sistema após certo tempo, e são permanentes quando o *peer* nunca retorna, caracterizando, assim, o abandono do sistema.

Um sistema P2P de backup, para prover confiabilidade, necessita somente se preocupar com as falhas permanentes, podendo, assim, ignorar falhas transientes, desde que consiga diferenciá-las. Dado que a maioria das falhas é transiente em um sistema P2P, poderíamos imaginar que a necessidade de redundância em sistemas de backup seria reduzida.

Entretanto, os sistemas P2P existentes supõem *peers* anônimos, tornando difícil, senão impossível, diferenciar saídas permanentes e temporárias. Assim, um sistema de backup tem que considerar todas as saídas como falhas permanentes, tendo como resultado uma taxa de falhas ainda elevada. Conseqüentemente, a manutenção da alta confiabilidade torna-se tão cara quanto a da alta disponibilidade. Assim como armazenamento P2P, os sistemas P2P anônimos de backup estão fadados a não funcionarem.

Se um *peer* interagisse somente com *peers* de amigos conhecidos, os *peers* raramente deixariam o sistema de maneira não anunciada, permitindo que se diferenciassem as saídas temporárias das permanentes [MGGM04a, YCZ+04, PCT04, PGB+06]. Ao invés de apresentar um esquema de redundância de dados para endereçar a participação intermitente de *peers*, um sistema P2P baseado em rede social, por sua natureza, terá uma taxa mais baixa de saídas permanentes, reduzindo a taxa de falhas a ser tratada. Li e Dabek [LD06] apresentaram um modelo analítico para sistemas P2P no qual os *peers* compartilham recursos somente entre amigos. Eles avaliaram este modelo e mostraram que esta suposição reduz drasticamente o consumo de largura de banda.

1.2 Redes Sociais

Redes Sociais são representações dos relacionamentos existentes entre pessoas ou organizações [Bar72]. As pessoas relacionam-se em níveis qualitativa e culturalmente diferenciados. Esses relacionamentos possuem regras sociais associadas e caracterizam como o ser humano está inserido dentro de uma comunidade. As comunidades também possuem um conjunto de regras que são determinadas colaborativamente pelas pessoas ou organizações que as compõem. Comunidades também são representadas por redes sociais e podem possuir conexões entre si. Assim, as redes sociais são sistemas organizacionais capazes de reunir indivíduos e instituições de forma descentralizada e participativa.

Com o advento da Internet, emergiram ferramentas que oferecem novas formas de relação, comunicação e organização, contribuindo assim, para o desenvolvimento das redes sociais. Sistemas de mensagens instantâneas (*e.g.* Yahoo, MSN, ICQ) monitoram os usuários e informam se pessoas conhecidas no sistema estão conectadas. Sites de relacionamento, como Orkut e Friendster, permitem a construção de novos relacionamentos e comunidades. Estas ferramentas permitiram que relacionamentos no mundo real possuíssem representações no mundo virtual submetidas às mesmas regras sociais.

Redes sociais são valiosas no desenvolvimento de sistemas computacionais porque capturam relações de confiança entre entidades do mundo real. Estas relações podem ser utilizadas para criação de mecanismos mais eficazes de reputação de entidades computacionais [SS02, HA04], soluções de roteamento e localização de dados mais confiáveis [MGGM04b, Upa02] e ferramentas de compartilhamento de dados mais eficientes [SDM+05].

No OurBackup, a rede social de um usuário é composta pelo conjunto de amigos cadastrados que ele possui. Cada usuário é responsável por construir sua própria rede social. Os participantes de uma mesma rede social são chamados de *amigos*. Esta relação de "amizade" é bidirecional, se um usuário Alfa possui como amigo o usuário Beta, então o usuário Beta possui como amigo o usuário Alfa. Esse tipo de rede social é similar às redes presentes nas ferramentas de mensagens instantâneas. Os usuários do OurBackup utilizarão a

rede social para efetuar o backup de seus dados, utilizando as máquinas dos amigos para manter as suas cópias de segurança.

A probabilidade de um amigo estar conectado ao sistema pode ser influenciada pela rede social. Caso um usuário precise recuperar um backup que está em um computador de um amigo desconectado, pode-se utilizar outro meio de comunicação (*e.g.* telefone, e-mail) para pedir ao dono do computador que se conecte ao sistema. Em outro caso, o usuário pode solicitar aos amigos que permaneçam conectados ao sistema por um determinado período de tempo. Conseqüentemente, diminui-se o tempo necessário para realizar ou recuperar um backup. A influência que um amigo tem em relação aos participantes de sua rede social também pode ser usada para criar uma saída mais suave do sistema, pois esta poderá ser sinalizada.

No OurBackup, as regras sociais e as relações de confiança presentes nas redes sociais dos usuários também serão usadas para amenizar comportamentos maliciosos dos participantes (*e.g. free riding*). Um usuário só poderá se utilizar do espaço de armazenamento de um computador se o dono permitir. Esta permissão é influenciada pelo grau de confiança, no mundo real, depositado na pessoa que está requisitando espaço para backup.

1.3 Estrutura da dissertação

O restante desta dissertação está estruturado como segue: O capítulo 2 realiza um estudo sobre a viabilidade do sistema. O capítulo 3 descreve a arquitetura e o projeto do OurBackup, descrevendo as funcionalidades que ele fornece e os procedimentos que ocorrem durante operações de backup e recuperação. O capítulo 4 apresenta detalhes sobre a implementação do OurBackup. O capítulo 5 revisa trabalhos relacionados, realizando um estudo sobre sistemas P2P de armazenamento e backup. Finalmente, o capítulo 6 apresenta as conclusões.

Capítulo 2 Avaliação de viabilidade

Sistemas de backup tolerantes a falhas tipicamente possuem como requisito de projeto a garantia da confiabilidade e da recuperabilidade do backup. A *confiabilidade* é a propriedade que assegura que o backup pode ser recuperado, sendo aceitável algum atraso se o backup não estiver imediatamente acessível. É, intrinsecamente, a propriedade mais importante de um sistema de backup. Enquanto a confiabilidade é uma propriedade que determina se um backup é recuperável, a *recuperabilidade* determina o quão rapidamente um backup pode ser recuperado. A recuperabilidade mede a velocidade de o sistema recuperar um backup. São conceitos distintos, mas interdependentes. Se a confiabilidade do backup for nula, não é possível recuperar um backup, e, conseqüentemente, a recuperabilidade do backup também será nula.

Cumprir requisitos de confiabilidade e recuperabilidade em sistemas P2P de backup é um desafio devido ao comportamento autônomo e imprevisível dos *peers*. As redes P2P são caracterizadas por alta dinamicidade dos nós [SR06]. Os *peers* podem se conectar ou sair da rede a qualquer momento. Tais saídas podem ser transientes ou permanentes. São transientes quando o *peer* torna a se conectar ao sistema, e são permanentes quando o *peer* nunca retorna, caracterizando, assim, um abandono. Os *peers* também estão sujeitos a desconexões temporárias devido a problemas no canal de comunicação. Desconexões temporárias e saídas transientes dos *peers* afetam a recuperabilidade dos backups. Por sua vez, as saídas permanentes afetam a confiabilidade do backup.

Outro problema enfrentado por sistemas P2P de backup é a existência de *free riders*. Os *peers* possuem total controle sobre o espaço de armazenamento que compartilham para backups. Um *free rider* pode se aproveitar desta liberdade, descartando silenciosamente backups de outros *peers* que estejam armazenados em seu computador.

Apesar da existência de tais comportamentos autônomos e imprevisíveis, técnicas de redundância de dados permitem que sistemas P2P de backup atendam aos requisitos de confiabilidade e recuperabilidade, mesmo se os *peers* participantes não forem confiáveis.

Duas técnicas de redundância de dados bastante aplicadas em sistemas de armazenamento são: replicação e *erasure codes*.

Replicação é a técnica pela qual k cópias de um dado são disseminadas entre diferentes nós ou dispositivos de armazenamento, sendo k o fator de replicação. Replicação é a técnica de redundância usada pelos sistemas P2P de armazenamento PAST [RD01a] e Freenet [CMN02].

Erasure code é uma técnica de redundância mais sofisticada que a replicação. Em tal esquema, cada arquivo é fragmentado em b blocos e então recodificado em $k \cdot b$ blocos, que são disseminados entre diferentes nós ou dispositivos de armazenamento, sendo k o fator de expansão usado na codificação. A principal propriedade do *erasure code* é que o arquivo original pode ser reconstruído com quaisquer b blocos dos $k \cdot b$ blocos codificados. O *erasure code* é a técnica de redundância usada pelo sistema P2P de armazenamento OceanStore [KBC+00].

Como as duas técnicas apresentam vantagens e desvantagens, nas próximas seções, analisaremos as duas quanto à garantia dos requisitos de confiabilidade e de recuperabilidade para um sistema de backup P2P. Analisaremos, também, o uso de redes sociais e *peers* anônimos na garantia destes mesmos requisitos.

2.1 Confiabilidade

A meta fundamental de sistemas de backup é garantir a confiabilidade dos dados de modo que os usuários possam, ocasionalmente, recuperar seus backups. Como já foi mencionado, Confiabilidade (*C*) é a habilidade de o backup ter sua persistência assegurada por um dado intervalo de tempo. Quanto maior a confiabilidade de um backup, maior a chance de recuperar os dados em um tempo futuro.

A confiabilidade dos backups é uma função da confiabilidade dos componentes envolvidos no armazenamento dos backups. Para isso, estes sistemas devem garantir a alta confiabilidade dos backups, independente de qual forma de armazenamento esteja sendo utilizada. No caso de sistemas P2P de backup, os componentes envolvidos são, tipicamente, não confiáveis, tornando o serviço de armazenamento provido por eles não confiável.

Inevitavelmente, cópias de um backup poderão ser comprometidas por falhas permanentes sofridas por estes componentes. Assim, sistemas P2P de backup, para garantir alta confiabilidade, devem tolerar a ampla gama de falhas que pode comprometer, permanentemente, um backup. Sempre que as cópias de backup forem comprometidas pela ocorrência de falhas permanentes, deverão ser substituídas por novas cópias.

Uma ampla gama de falhas pode atingir sistemas P2P de backup, quer sejam falhas nos dispositivos de armazenamento, nos dispositivos de controle ou nos computadores que gerenciam estes componentes. A tolerância a falhas em sistemas P2P de backup é complexa, devido à congregação de diferentes tipos de falha e à proliferação de técnicas de proteção dos dados. Sem tentar uma enumeração exaustiva, algumas das razões mais comuns de perda de dados são:

- Desastre natural: Normalmente, os componentes dos computadores (e os dados armazenados) não apresentam robustez em relação a ocorrência de eventos naturais, como fogo e inundações, ou mesmo surtos de energia. Desastres naturais de larga escala devem ser tomados em consideração, podendo ser correlacionados com falhas de comunicação e *hardware*.
- Vírus e *Worms*: Todos os sistemas conectados através da Internet são vulneráveis a ataques de vírus e *worms*. Calcula-se que o vírus de e-mail MyDoom de 2004 tenha custado às empresas mais de 250 milhões de dólares em perdas relacionadas à produtividade e em gastos relacionados ao suporte técnico – com as pequenas e médias empresas sendo as mais afetadas [wor07]. Muitos anos atrás, o vírus Melissa espalhou-se rapidamente no mundo todo, causando estragos de $80 milhões de dólares [vir07].
- Erro humano: Erros humanos são possíveis e de fato acontecem. Além disso, é possível que um colega de trabalho, marido ou esposa, amigo ou filho modifique ou apague por acidente dados importantes.
- *Free ridding*: O comportamento de *free riding* também pode ser classificado como falha, e pode, permanentemente, corromper um backup. Uma falha do tipo "*free*

riding" ocorre quando o *peer* descarta, silenciosamente, backups já armazenados. Este tipo de falha pode ocorrer devido a razões econômicas ou emocionais [KCK+].

- Falhas de disco: Embora seja, de modo geral, confiável o disco rígido pode experimentar alguma falha mecânica (*e.g.* braço mecânico), apresentar setores ruins ou ter algum problema com um componente elétrico.

Assim, a confiabilidade de um backup é determinada pela combinação de todas as probabilidades de ocorrência de eventos danosos ao armazenamento de um dado. Entretanto, é difícil, senão impossível, modelar cada possível falha de disco, erro de *software* ou ações acidentais/propositais comandadas pelo administrador do computador que possam corromper permanentemente um backup. Analisaremos a confiabilidade do backup através de modelos analíticos. Embora os modelos analíticos não possam refletir a confiabilidade dos sistemas de backup de forma perfeita, fornecem uma recursos para uma avaliação sobre o grau de confiabilidade do sistema.

2.1.1 Avaliação analítica

O custo geral de um sistema P2P de backup é determinado pela sobrecarga de armazenamento (k) e pela taxa de falhas (f). Para replicação, k é simplesmente o número de cópias. Para *erasure code*, k é o fator de expansão (*i.e.* quantidade de redundância adicionada ao arquivo) usado na codificação dos blocos. *Erasure codes* também definem b como a quantidade de blocos gerados na fragmentação do arquivo. Destes, k e b, dependem do limiar de confiabilidade definido para o sistema e da taxa de falhas permanentes ao qual o sistema deve suportar. Seus valores são escolhidos *a priori*, isto é, antes que as falhas tenham ocorrido. Como não temos como saber quais cópias serão comprometidas devido a falhas, tudo o que nós podemos fazer é supor que as cópias e os blocos falham permanentemente com probabilidade, máxima, f.

Supomos uma coleção grande, dinâmica e geograficamente distribuída de *peers* que armazenam os backups cooperativamente. Assumimos, também, que as cópias e os blocos têm a mesma probabilidade, f, de serem destruídas e falham independentemente. Novamente, esta probabilidade de falha incorpora a probabilidade de falha de múltiplos fatores, tais como

falhas de disco e ataques de vírus. A Tabela I sumariza os parâmetros usados nas análises de confiabilidade.

TABELA I - Sumário dos parâmetros usados na análise de confiabilidade

Parâmetro	**Descrição**
f	Probabilidade de falha do *peer*
k	Sobrecarga de armazenamento
b	Fragmentação usada pelo *Erasure Code*

2.1.2 Confiabilidade usando Replicação

Quando usamos replicação como técnica de redundância, a confiabilidade do backup é definida pela probabilidade de pelo menos uma cópia não ter falhado. Sendo os eventos de sobrevivência e de falha das cópias independentes, a probabilidade de pelo menos uma cópia não ter falhado é calculada pelo somatório das probabilidades de uma única cópia ter sobrevivido, de duas cópias terem sobrevivido, de três cópias terem sobrevivido, ..., de todas as cópias terem sobrevivido (ver Equação 1). Ou, alternativamente, a confiabilidade pode ser calculada através da probabilidade de todas as cópias terem falhado (ver Equação 2). Assim, a confiabilidade (*C*) resultante do backup é dada por [LCL04]:

$$C = \binom{k}{1}(1-f)^1 (f)^{k-1} + \binom{k}{2}(1-f)^2 (f)^{k-2} + \ldots + \binom{k}{k}(1-f)^k (f)^{k-k}$$

$$C = \sum_{i=1}^{k} \binom{k}{i}(1-f)^i (f)^{k-i} \qquad (1)$$

falhas de disco e ataques de vírus. A Tabela I sumariza os parâmetros usados nas análises de confiabilidade.

TABELA I - Sumário dos parâmetros usados na análise de confiabilidade

Parâmetro	**Descrição**
f	Probabilidade de falha do *peer*
k	Sobrecarga de armazenamento
b	Fragmentação usada pelo *Erasure Code*

2.1.2 Confiabilidade usando Replicação

Quando usamos replicação como técnica de redundância, a confiabilidade do backup é definida pela probabilidade de pelo menos uma cópia não ter falhado. Sendo os eventos de sobrevivência e de falha das cópias independentes, a probabilidade de pelo menos uma cópia não ter falhado é calculada pelo somatório das probabilidades de uma única cópia ter sobrevivido, de duas cópias terem sobrevivido, de três cópias terem sobrevivido, ..., de todas as cópias terem sobrevivido (ver Equação 1). Ou, alternativamente, a confiabilidade pode ser calculada através da probabilidade de todas as cópias terem falhado (ver Equação 2). Assim, a confiabilidade (C) resultante do backup é dada por [LCL04]:

$$C = \binom{k}{1}(1-f)^1(f)^{k-1} + \binom{k}{2}(1-f)^2(f)^{k-2} + \ldots + \binom{k}{k}(1-f)^k(f)^{k-k}$$

$$C = \sum_{i=1}^{k}\binom{k}{i}(1-f)^i(f)^{k-i} \qquad (1)$$

$$C = 1 - P(\textit{todas as } k \textit{ cópias falharem})$$

$$C = 1 - P(\textit{uma cópia falhar})^k$$

$$C = 1 - P(f)^k \tag{2}$$

O parâmetro k deve ser escolhido de tal forma que C obtenha o nível desejado de confiabilidade. A Figura 1 e a Figura 2 apresentam os resultados obtidos quando usamos a Equação 1 para diferentes valores de k e f. O eixo x apresenta a variação da sobrecarga de armazenamento k, o eixo y apresenta a confiabilidade C obtida, e cada curva representa diferentes valores para f. Analisando-as, podemos perceber que aumentar o valor de k tem um benefício direto, que é a melhoria da confiabilidade dos backups. Porém, a curva de crescimento não é constante, havendo um ponto de saturação a medida que C se aproxima de 1. Dentro dos cenários apresentados, foi possível obter cinco noves de confiabilidade (C = 0,99999) com uma probabilidade de falha igual a 0,1, 0,2 e 0,3 usando $k = 5$, $k = 8$ *e* $k = 10$ para os respectivos casos. A Figura 2 apresenta uma filtragem nos resultados, visualizando mais facilmente quais taxas f de falhas alcançaram uma confiabilidade de cinco noves para $k > 2$ e $k <= 10$. Taxas de falhas superiores a 0,3 requisitam valores maiores para k. Por exemplo, se calcularmos a sobrecarga de armazenamento necessária para se obter cinco noves de confiabilidade, usando a Equação 1, verificaríamos que é preciso $k = 18$ diante de uma taxa de falhas $f = 0,6$. Naturalmente, se o valor escolhido para f for menor do que o observado na prática, o backup possuirá uma confiabilidade real menor do que a esperada. Analogamente, se o valor escolhido para f for mais alto que na prática, o backup possuirá confiabilidade real maior, aumentando, contudo, a sobrecarga de armazenamento.

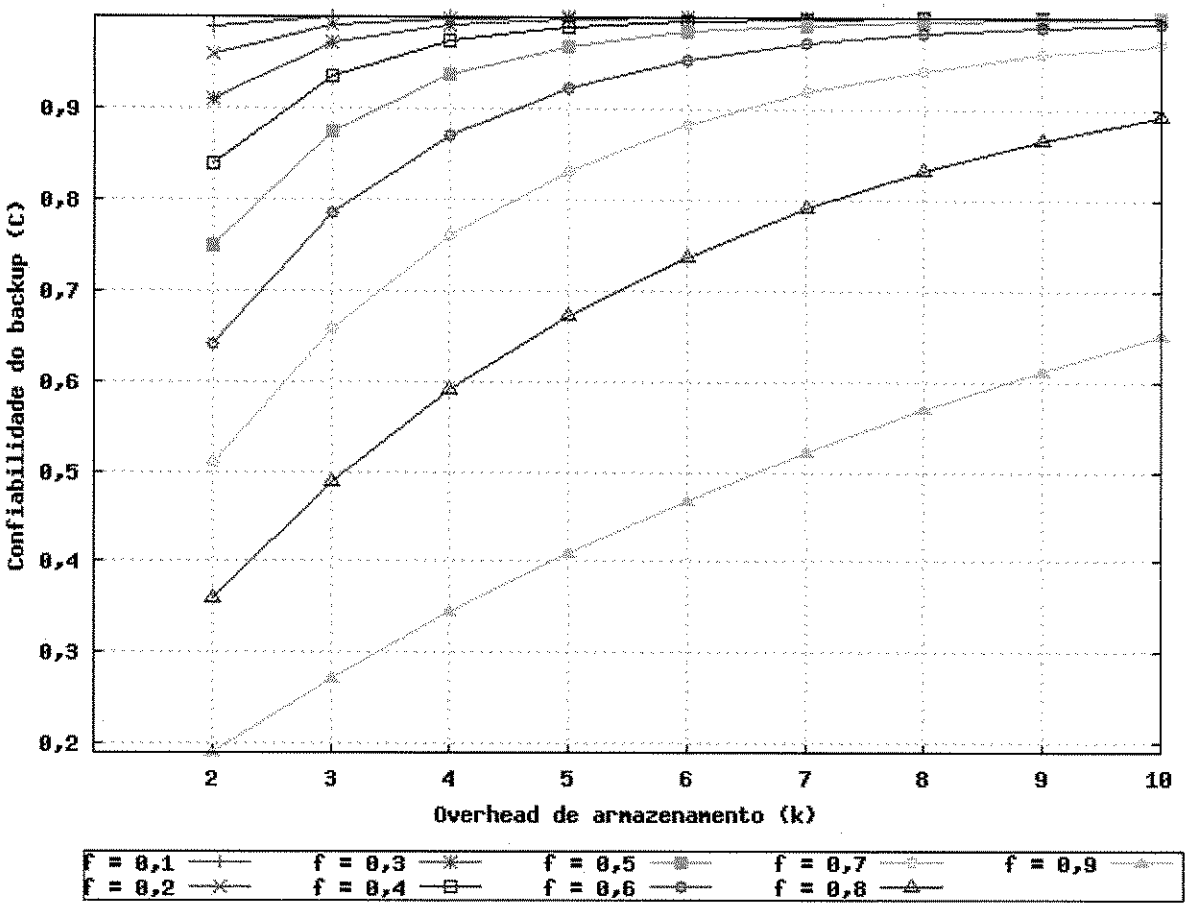


Figura 1 – Confiabilidade do backup

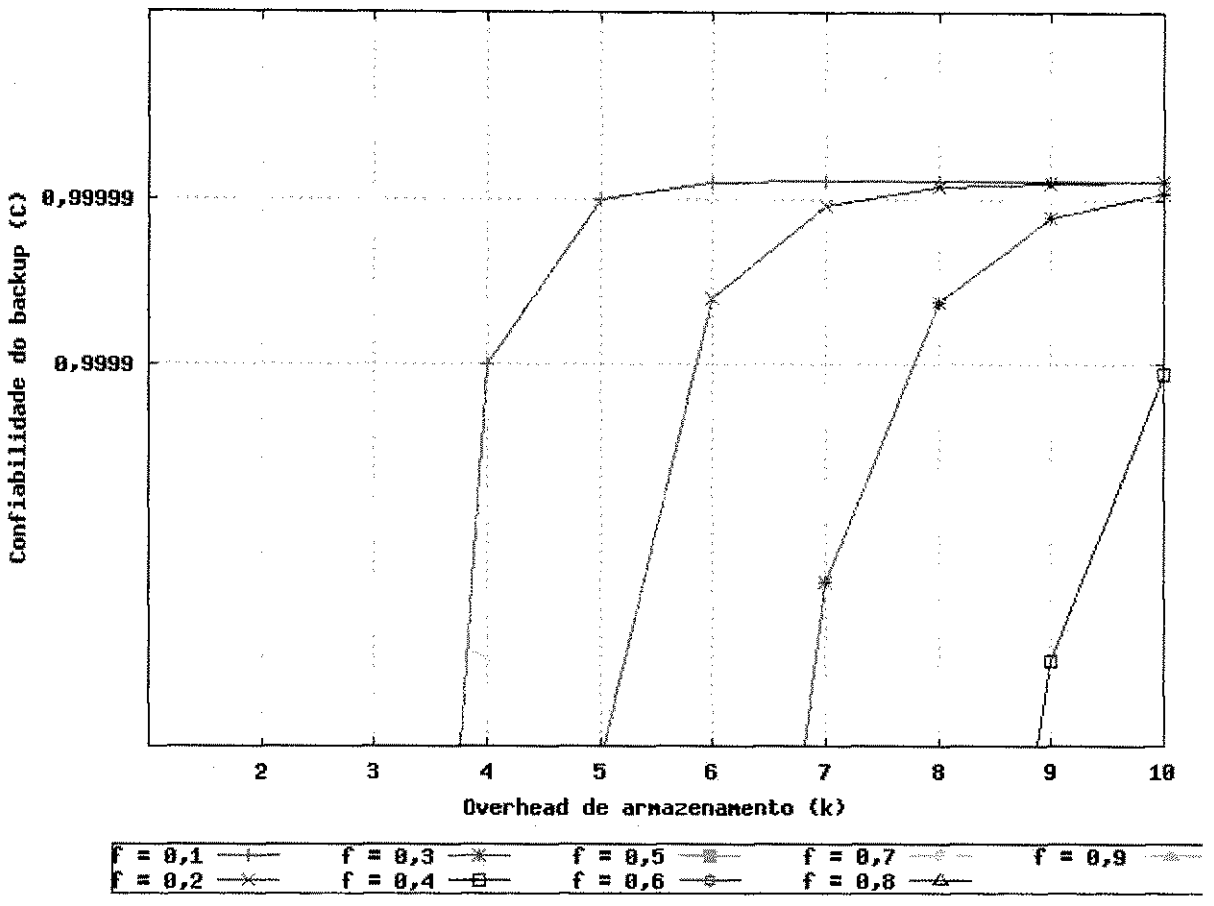


Figura 2 – Cinco noves de confiabilidade (C = 0,99999)

Quanto maior o valor de k, maior será o custo de armazenamento de um backup. Os custos são importantes, pois existem diversos casos onde o desejo inicial de "proteger tudo" foi abandonado devido a custos financeiros [MP07]. Podemos reduzir o custo total diminuindo o limiar de confiabilidade desejado ou diminuindo a probabilidade de ocorrência de falhas. Em particular, podemos tentar melhor entender as falhas possíveis, com especial atenção para as falhas que mais freqüentemente corrompem dados. Segundo estudos da *Ontrack Data Recovery* [dat07], empresa norte-americana especializada em proteção de dados, as principais causas da perda de dados são: falhas de disco (56%), erro humano (26%), falhas de *software* (9%), vírus (4%) e desastres naturais (2%).

Em sistemas de backup P2P, acreditamos que *free riding* venha a ser uma causa de perdas de dados bastante comum. Os sistemas P2P são muito vulneráveis à participação de

free riders, em especial se não houverem mecanismos de incentivos usados no sistema [ADS03, DMS03]. A habilidade de abandonar identidades e de criar outras com baixíssimo custo torna praticamente impossível manter o histórico de ações maliciosas dos usuários. Em decorrência disso, os sistemas P2P de backup anônimos assumem que os *peers* não são confiáveis e aumentam a sobrecarga de armazenamento com intuito de tolerar a ocorrência de *free riding*.

Utilizando a nossa rede social para armazenar nossos backups, em vez de interagirmos com *peers* não confiáveis dentro de uma rede P2P, podemos assumir que nossos amigos não apagarão nossos backups propositadamente. Evitando, assim, a interação com *peers* possivelmente maliciosos, e, conseqüentemente reduzindo a probabilidade de ocorrência de *free riding*. Assumimos, portanto, que em sistemas P2P de backups baseados em redes sociais, a taxa de falhas *f* seja dominada pela ocorrência de falhas do disco.

Em suma, acreditamos que falhas em um sistema de backup P2P anônimo sejam dominadas por *free riding*, enquanto que em um sistema de backup P2P baseado em rede social, esperamos que falhas sejam dominadas por falhas de disco. A probabilidade destes eventos se realizarem é determinada por:

- Falha de disco: Um dos modelos mais populares para representar o tempo para falhar é a distribuição exponencial [Kra88]. Esta distribuição implica que a taxa de falha é constante sobre a faixa de prognóstico, que para certas situações de falha ou trecho da vida do produto pode ser bastante apropriado [Kap88]. A função que descreve a probabilidade de ocorrência de uma falha de disco em um determinado intervalo de tempo, exponencialmente distribuída, é dada por [WV07]:

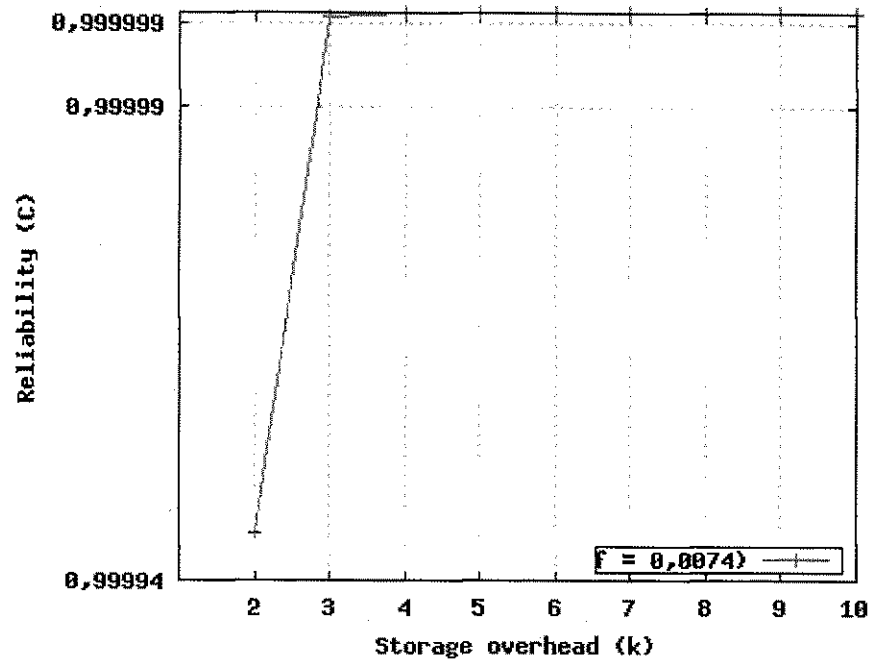
$$f(\Delta t) = 1 - e^{-(\Delta t / MTBF)} \qquad (3)$$

  Em que, Δt indica o tempo de interesse no disco (*i.e.* o tempo necessário para recuperação do backup), e MTBF (*Mean Time Between Failures*) representa a quantidade de tempo médio entre falhas. Os discos rígidos atuais possuem geralmente MTBF avaliado entre 300.000 e 1.200.000 horas [mtb07]. Consideramos, nas nossas análises, um MTBF de 300.000 horas, que corresponde a

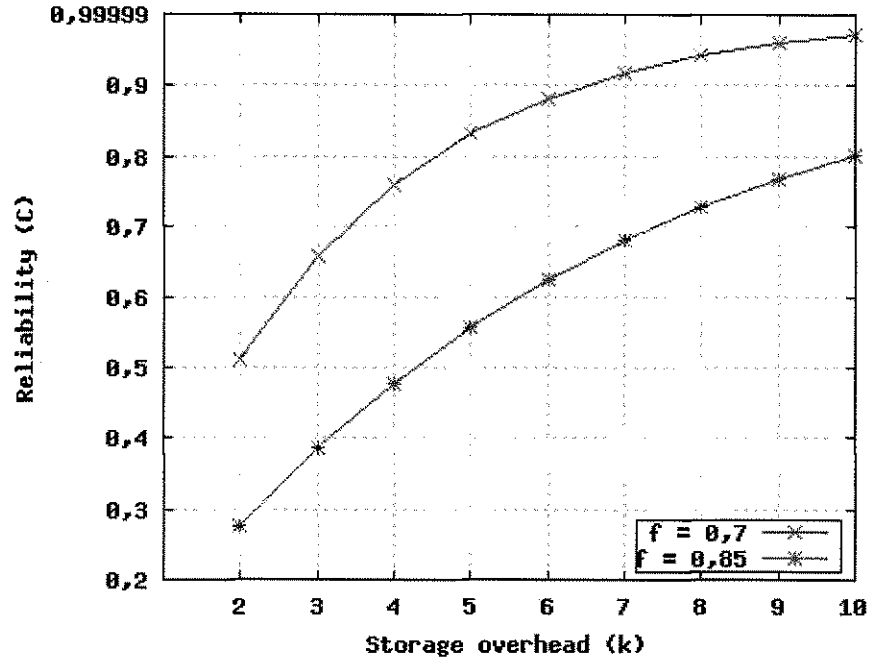
hipótese mais pessimista de tempo médio entre falhas em discos, e um Δt de 3 meses (2.232 horas) que corresponde a tempo mais que suficiente para recuperar um backup de 100 GB a uma taxa de transferência de 300 Kbps.

- *Free riding*: Consideraremos a participação de *free riders* obtida em estudos sobre aplicações P2P de compartilhamento de arquivos como a probabilidade de ocorrência de uma falha *free riding* para um *peer*. Em 2000, *Adar* e *Huberman* [AH00] realizaram um estudo sobre o Gnutella concluindo que, aproximadamente, 70% dos *peers* são *free riders*. O mesmo padrão de comportamento foi observado em estudos subseqüentes sobre Napster e ainda sobre Gnutella [DB99]. Em 2005, *Hughes et al.* [HCW05] observaram que a quantidade de *free riders* no Gnutella aumentou para 85%.

A Figura 3 apresenta os resultados obtidos quando usamos a Equação 1 com valores para f referentes à probabilidade de ocorrência de falhas disco ($f = 0{,}0074$) e *free riding* ($f = 0{,}70$ e $f = 0{,}85$). Novamente, o eixo x apresenta a variação da sobrecarga de armazenamento k, o eixo y apresenta a confiabilidade C obtida, e cada curva representa diferentes valores para f. Os resultados obtidos são similares aos apresentados pelas Figuras 1 e 2. Os eventos *free riding*, por possuírem alta probabilidade de ocorrência, requisitam uma sobrecarga de armazenamento muito alta para alcançar um limiar de cinco noves de confiabilidade. Mais precisamente, calculamos a sobrecarga de armazenamento necessária usando a Equação 1, resultando em $k = 33$ e $k = 71$ para se obter cinco noves de confiabilidade diante das taxas de falhas $f = 0{,}7$ e $f = 0{,}85$, respectivamente. Por sua vez, as falhas de disco, por possuírem uma baixa probabilidade de ocorrência ($f = 0{,}0074$), requisitam uma sobrecarga de armazenamento ($k = 3$) muito menor que o *free riding* para atender o mesmo requisito de confiabilidade.



(a) Falha de disco



(b) *Free riding*

Figura 3 – Impacto de free riding e falhas de disco na confiabilidade de um backup, quando usado replicação como técnica de redundância de dados

2.1.3 Confiabilidade de backup aplicando *erasure codes*

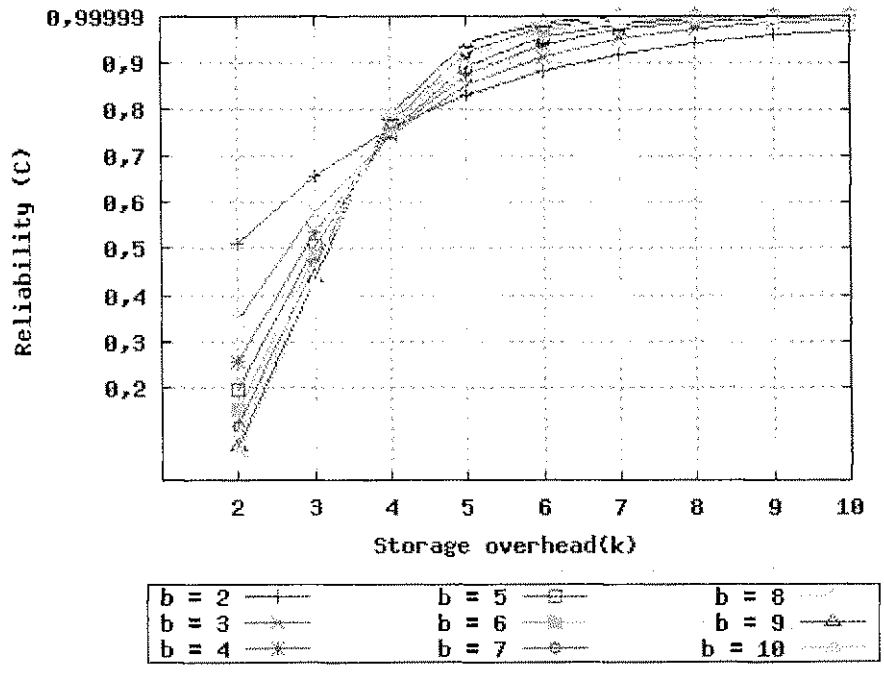
Lembrando que, quando usamos *erasure code* como técnica de redundância de dados, com quaisquer b blocos dos k·b *peers* é possível recuperar o dado original, a confiabilidade do backup usando *erasure codes* é definida pela probabilidade de pelo menos b blocos não terem falhado. Sendo os eventos sobrevivência e falha do bloco independentes, a probabilidade de pelo menos b blocos não terem falhado é calculada pelo somatório das probabilidades de b blocos terem sobrevivido, de $b + 1$ blocos terem sobrevivido, de $b + 2$ blocos terem sobrevivido, ..., de k·b blocos terem sobrevivido. Assim, a confiabilidade (C) resultante do backup é dada por [SS02]:

$$C = \binom{kb}{1}(1-f)^1(f)^{kb-1} + \binom{kb}{2}(1-f)^2(f)^{kb-2} + \ldots + \binom{kb}{kb}(1-f)^{kb}(f)^{kb-kb}$$

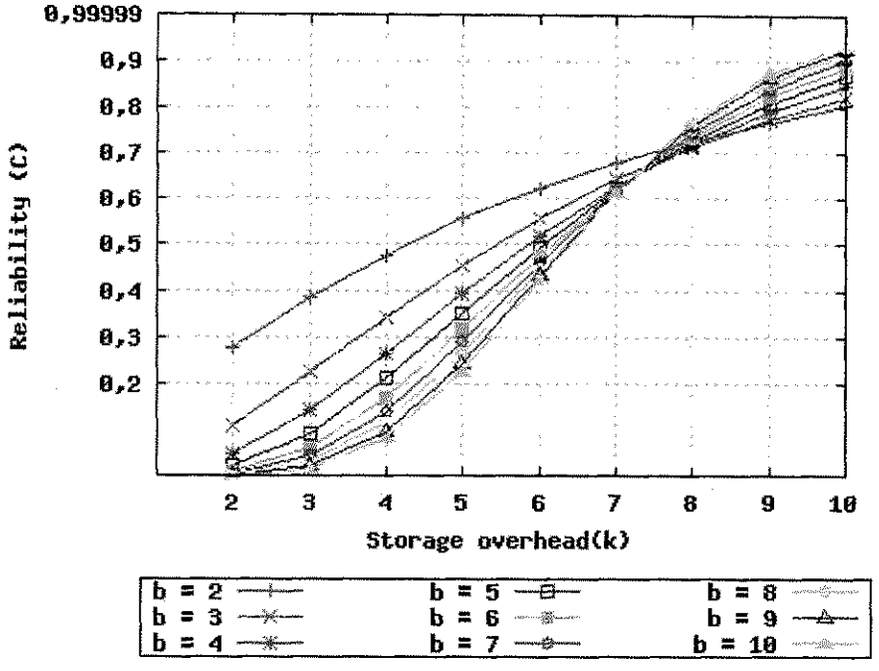
$$C = \sum_{i=1}^{kb}\binom{kb}{i}(1-f)^i(f)^{kb-i} \tag{4}$$

Os parâmetros k e b devem ser escolhidos de tal forma que C obtenha o nível desejado da confiabilidade. A Figura 4 apresenta os resultados obtidos quando usamos a Equação 4 com valores para f referentes à probabilidade de ocorrência de falhas de disco (f = 0,0074) e *free riding* (f = 0,70 e f = 0.85). O eixo x apresenta a variação da sobrecarga de armazenamento k, o eixo y apresenta a confiabilidade obtida, e cada curva representa diferentes valores para b. Analisando-as, podemos perceber que aumentar o valor de k tem um benefício direto, que é a melhoria da confiabilidade dos backups. Porém, a curva de crescimento não é constante. Como esperado, há um ponto de saturação à medida que C se aproxima de 1, onde o crescimento passa a ser mais suave. Entretanto, aumentar o valor de b

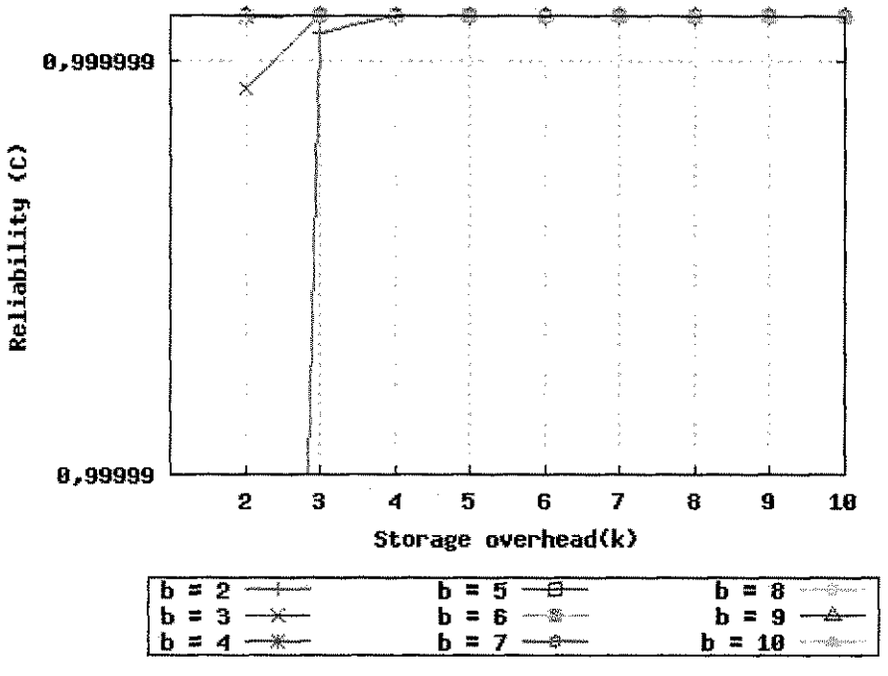
não resulta, necessariamente, em melhorias na confiabilidade. Quando a probabilidade de ocorrência de falhas é alta (*i.e.* $f > 0{,}6$) e a sobrecarga de armazenamento é baixa (*i.e.* $k < 3$), aumentando o valor de b teremos como resultado uma confiabilidade progressivamente baixa. Por outro lado, à medida que aumentamos o valor de k, valores altos para b conduzem ao formato mais íngreme da curva e, como conseqüência, existe um ponto de convergência no qual valores altos para b resultam em melhor confiabilidade que os baixos.



(a) *Free riding* $f = 0{,}7$



(b) *Free riding* $f = 0{,}85$



(c) falha de disco $f = 0{,}0074$

Figura 4 – Impacto de free riding e falhas de disco na confiabilidade de um backup, usando erasure codes como técnica de redundância de dados

2.1.4 Replicação x *Erasure Codes*

Como já mencionado, sistemas P2P de backup trabalham mantendo pelo menos um limiar de redundância de dados com o objetivo de assegurar os requisitos de confiabilidade. Quando o fator de redundância cai abaixo do limiar definido, novas cópias precisam ser criadas para substituir aquelas que falharam, armazenando-as em outros *peers*. A alta incidência de *free riders* resulta em uma elevada taxa de criação de novas cópias que consome grandes quantidades de banda passante, tornando impraticáveis sistemas de backup P2P anônimos [BR03].

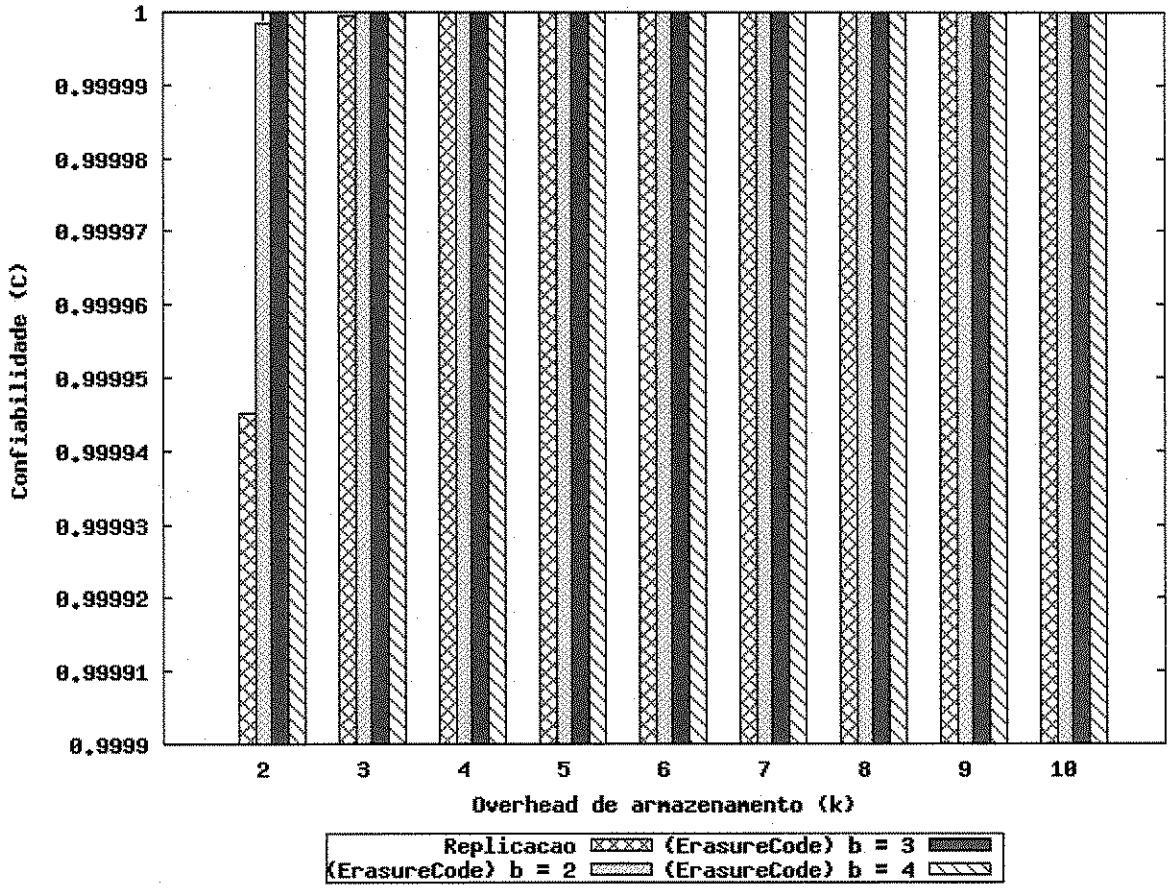


Figura 5 – Comparação entre a confiabilidade obtida usando Replicação e Erasure codes diante de ocorrência de falhas de disco (f =0,0074)

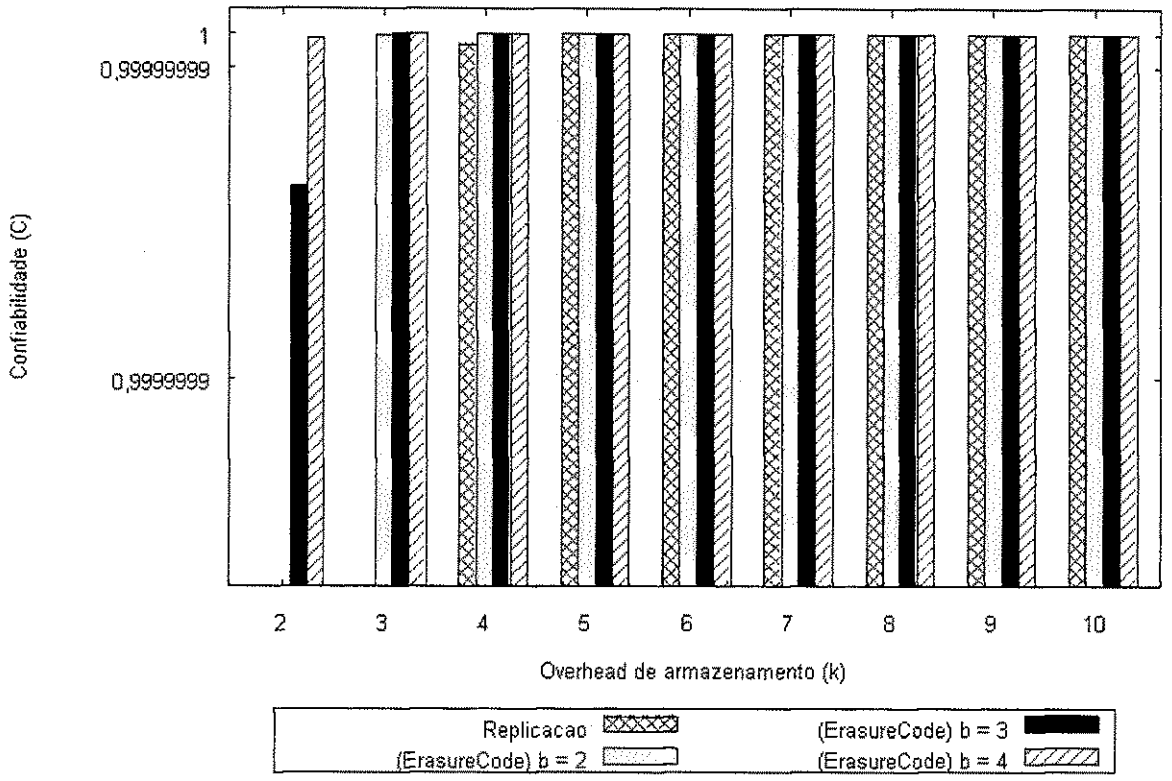


Figura 6 – Comparação entre a confiabilidade C > 0,99999 usando Replicação e Erasure codes diante de ocorrência de falhas de disco (f =0,0074)

Considerando que a taxa de falhas é dominada por falhas de disco, precisamos saber qual esquema de redundância de dados assegura os requisitos de confiabilidade mais eficientemente. A Figura 5 apresenta os resultados obtidos usando a Equação 1 e a Equação 4 com valores para f referentes à probabilidade de ocorrência de falhas de disco. O eixo x apresenta a variação da sobrecarga de armazenamento k e o eixo y apresenta a confiabilidade obtida C. A Figura 6 apresenta os mesmos resultados da Figura 5, realizando um zoom para os resultados que obtiveram $C > 0{,}99999$. Analisando estes resultados, podemos perceber que as duas técnicas - *erasure codes* e replicação – resultam em uma confiabilidade muito próxima para $k > 2$. Quando a taxa de falhas é muito baixa, aumentar a sobrecarga de armazenamento e a fragmentação não resulta em ganhos significativos.

2.2 Recuperabilidade

Recuperabilidade (*R*) refere-se à velocidade com que se consegue restaurar dados após uma falha, ou seja, atributos de software que evidenciam a sua capacidade de recuperar dados diretamente afetados em caso de falha e o tempo de esforço para tal.

Segundo *Kraus* [Kra88], o parâmetro de recuperabilidade mais comumente utilizado é o tempo médio para reparo, ou MTTR (*Mean Time to Repair*). O MTTR é medido como o tempo transcorrido para se efetuar uma operação de reparo, e é utilizado para se estimar o tempo em que o sistema não está operacional e também a sua disponibilidade. No caso específico de um sistema P2P de backup, o MTTR equivale ao tempo de recuperação do backup (*TR*), que é medido como o tempo gasto para se efetuar o download dos backups, sendo o TR ótimo (*i.e.* o melhor tempo de recuperação possível) definido por S/d, em que d é a capacidade de *download* do *peer* que está recuperando o backup e S o tamanho do backup a ser recuperado.

Contudo, não podemos utilizar somente o *TR* como parâmetro de recuperabilidade de backup. O problema é que *TR* varia dependendo de vários outros fatores, tais como qualidade da conexão, quantidade de *peers on-line*, capacidade de *upload* dos *peers*, tráfego da rede, compartilhamento da conexão de internet, além de S e d. Sempre que um desses fatores muda, o *TR* é alterado. Isto torna inadequado usar simplesmente *TR* para comparar a recuperabilidade em cenários distintos de recuperação, pois não necessariamente um backup ter sido recuperado em menos tempo que outro backup quer dizer que o primeiro possua uma recuperabilidade melhor que o último. Por exemplo, um sistema A consome dois dias para recuperar um backup de 1GB e o sistema B consome quatro dias para recuperar um backup de 100GB. Apesar de o sistema A oferecer um *TR* menor que o sistema B, é notório que o sistema B ofereceu melhor recuperabilidade que o sistema A, visto que o sistema B atendeu a um backup de duas ordens de grandeza maior que o sistema A usando apenas o dobro do tempo.

Desta forma, o cálculo da recuperabilidade precisa ser independente dos fatores que influenciam o TR, senão sempre que alterarmos um dos fatores, a recuperabilidade do backup

também será alterada independente de quão eficiente seja o sistema de backup. Assim, definiremos a recuperabilidade do backup como a relação entre o tempo real gasto recuperando um backup (*TR*) e o tempo ótimo de recuperação (S/*d*):

$$R = \frac{\left(\frac{S}{d}\right)}{TR} = \frac{S}{(TR)\,(d)}$$

Assim, *R* expressa o quanto um determinado sistema está próximo ou distante do tempo ótimo de recuperação de backup. Quanto mais próximo for o *TR* em relação à S/d, maior será recuperabilidade do backup. Assim, $R = 1$ denota um resultado ótimo (o sistema de backup não poderia ter recuperado o backup mais rápido). À medida que *R* diminui, temos a indicação de uma recuperação cada vez mais distante da ótima.

Um sistema P2P de backup, para cumprir requisitos de recuperabilidade, necessita se preocupar com os fatores que influenciam o tempo de recuperação de backup. Entre a ampla gama de fatores existentes, a disponibilidade dos *peers* e a ocorrência de falhas permanentes são as principais causas de aumento do *TR*. Sempre que um *peer* fica *off-line*, os dados armazenados por ele tornam-se inacessíveis, impossibilitando temporariamente a recuperação dos backups, e, conseqüentemente, aumentando o *TR*. A recuperabilidade do backup também pode ser afetada pela ampla gama de falhas que pode comprometer, permanentemente, um backup. Se todas as cópias de um backup forem comprometidas permanentemente, a recuperabilidade do backup será nula.

Assim, a recuperabilidade de um backup é afetada pela combinação de fatores danosos ao tempo de recuperação de backup. Entretanto, é difícil, senão impossível, modelar cada possível evento que possa prolongar o processo de download de um backup. Estudos realizados por *Saroiu et al.* indicaram que o conjunto de *peers* participantes dos sistemas Napster e Gnutella são heterogêneos com respeito a muitas características: capacidade de banda passante, latências, tempo de vida, dados compartilhados [SGG02]. Na realidade, o valor destas características varia entre três e cinco ordens de magnitude entre *peers*.

Outra dificuldade se deve à análise da recuperabilidade de um backup quando usamos *erasure codes*. Neste caso específico, um backup somente está recuperado quando *b* blocos forem inteiramente recuperados dos *peers*. Em particular, ter meio bloco do peer A e meio bloco do peer B não resulta em ter um bloco útil, uma vez que os blocos de dados possuem diferentes conteúdos entre si.

A simulação permite que se modele um sistema muito mais próximo do real, aumentando a confiabilidade nos resultados e nas decisões. Cenários de simulação, que representem diferentes características dos *peers* (*e.g.* capacidade de *upload* e *download*) e do backup a ser recuperado (*e.g.* tamanho dos arquivos), nos permitem verificar a influência de cada parâmetro na recuperabilidade de backups. Então, optamos por usar um simulador alimentado por *traces* de sistemas P2P em funcionamento para avaliar a eficácia dos esquemas de redundância de dados quanto à garantia dos requisitos de recuperabilidade. O simulador e o modelos utilizado na simulação são apresentados logo a seguir.

2.2.1 Modelo da simulação

No modelo utilizado neste estudo, o ambiente de simulação é composto por um conjunto de *peers* que interagem entre si através do envio ou recebimento de backup. Cada *peer* tem uma capacidade de transferência de dados limitada. O número de *peers* e a capacidade de transferência de dados podem variar. O comportamento dos *peers* é determinado pelos estados que podem assumir: UP (*i.e.* o *peer* está *on-line*), DOWN (*i.e.* o *peer* está *off-line*) e FAILED (*i.e.* o *peer* falhou permanentemente). A mudança de estados é realizada através dos eventos *Join* (*i.e.* o *peer* se conectou ao sistema), *Leave* (*i.e.* o *peer* deixou o sistema) e *Fail* (*i.e.* o *peer* falhou permanentemente), como pode ser visualizado na Figura 7.

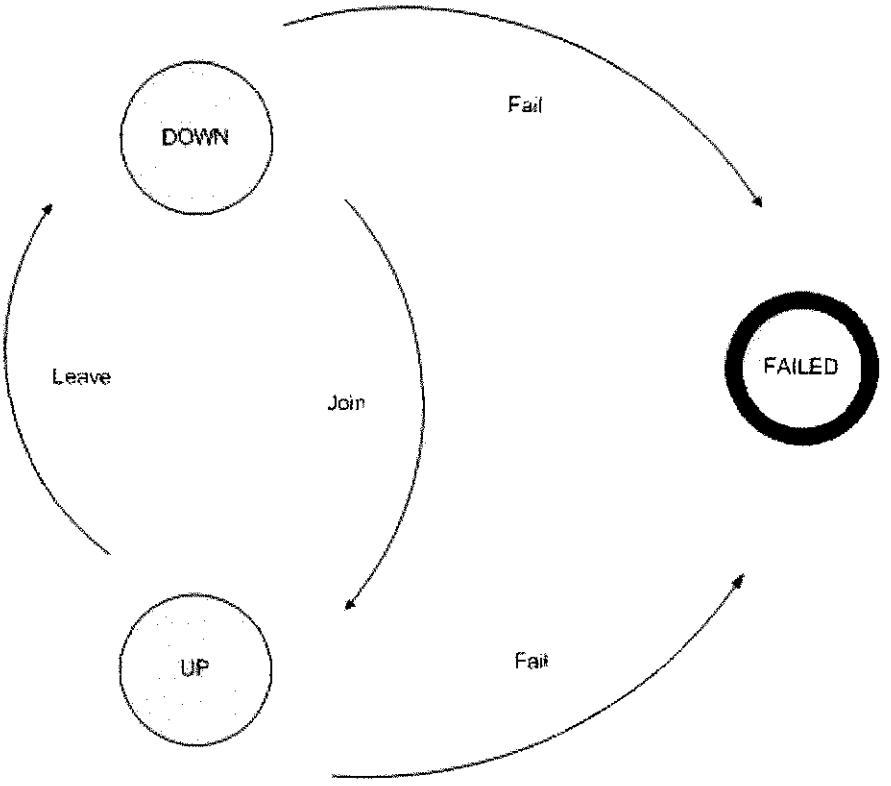


Figura 7 – Estados dos peers

Os *peers* são classificados como consumidores ou provedores. O *peer* atua como consumidor quando submete requisições de recuperação de backup para outros *peers*. O *peer* atua como provedor quando atende às requisições de backup do consumidor. O *peer* consumidor, devido à meta de recuperação do backup, permanece no estado UP durante todo o processo de simulação. Os *peers* provedores podem variar o seu estado no decorrer do processo de simulação.

Nossa simulação avalia a recuperabilidade de um backup. Assim, consideramos que existe um único consumidor durante o processo de simulação e que os provedores estão armazenando coletivamente um mesmo backup. O *peer* consumidor pode recuperar seus backups através de requisições de *download* para os *peers* que armazenam o conjunto de dados em que está interessado. Entretanto, estas requisições somente podem ser dirigidas a *peers* que estejam no estado UP (ver Figura 8). Quando os *peers* estão no estado DOWN, comunicação está temporariamente interrompida, impossibilitando a requisição de recuperação de backup. Caso existam mais de um provedor no estado UP, o *peer* consumidor

assume uma abordagem gulosa, requisitando transferências de dados a todos provedores *on-line*. Uma vez possuindo os dados requisitados e estando UP, os *peers* disponibilizam todos os recursos de banda passante para executar as transferências de dados solicitadas.

Quando a soma da capacidade de *upload* dos provedores for superior à capacidade de *download* do consumidor, é aplicado um procedimento de particionamento da banda passante. O particionamento pode ocorrer de forma igualitária, quando todos os provedores recebem porções iguais de banda, ou de forma proporcional, quando os provedores recebem porções de banda proporcionais a algum atributo ou característica (*e.g.* confiabilidade, disponibilidade). Avaliaremos as duas abordagens no nosso modelo de simulação.

Consideramos que o backup armazenado pelos provedores foi transferido em um momento anterior à necessidade de recuperação do mesmo pelo consumidor e que todos os *peers* possuem capacidade ilimitada de armazenamento. Todos os *peers* provedores armazenam a mesma quantidade de dados, entretanto, o conteúdo armazenado pode variar dependendo da técnica de redundância de dados aplicada. Quando é usada replicação como abordagem de redundância de dados, todos os *peers* possuem o mesmo conteúdo armazenado. Por outro lado, quando é usada *erasure code* como abordagem de redundância de dados, os *peers* possuem conjuntos de dados diferentes, pois o processo de codificação aplicado resulta na criação de blocos de conteúdos distintos. O conteúdo do backup consiste de um conjunto arquivos, cujos tamanhos variam de acordo com distribuição estatística. Consideramos o conceito de arquivo no simulador para verificar o impacto da diferença de conteúdo dos blocos quando usamos *erasure codes*.

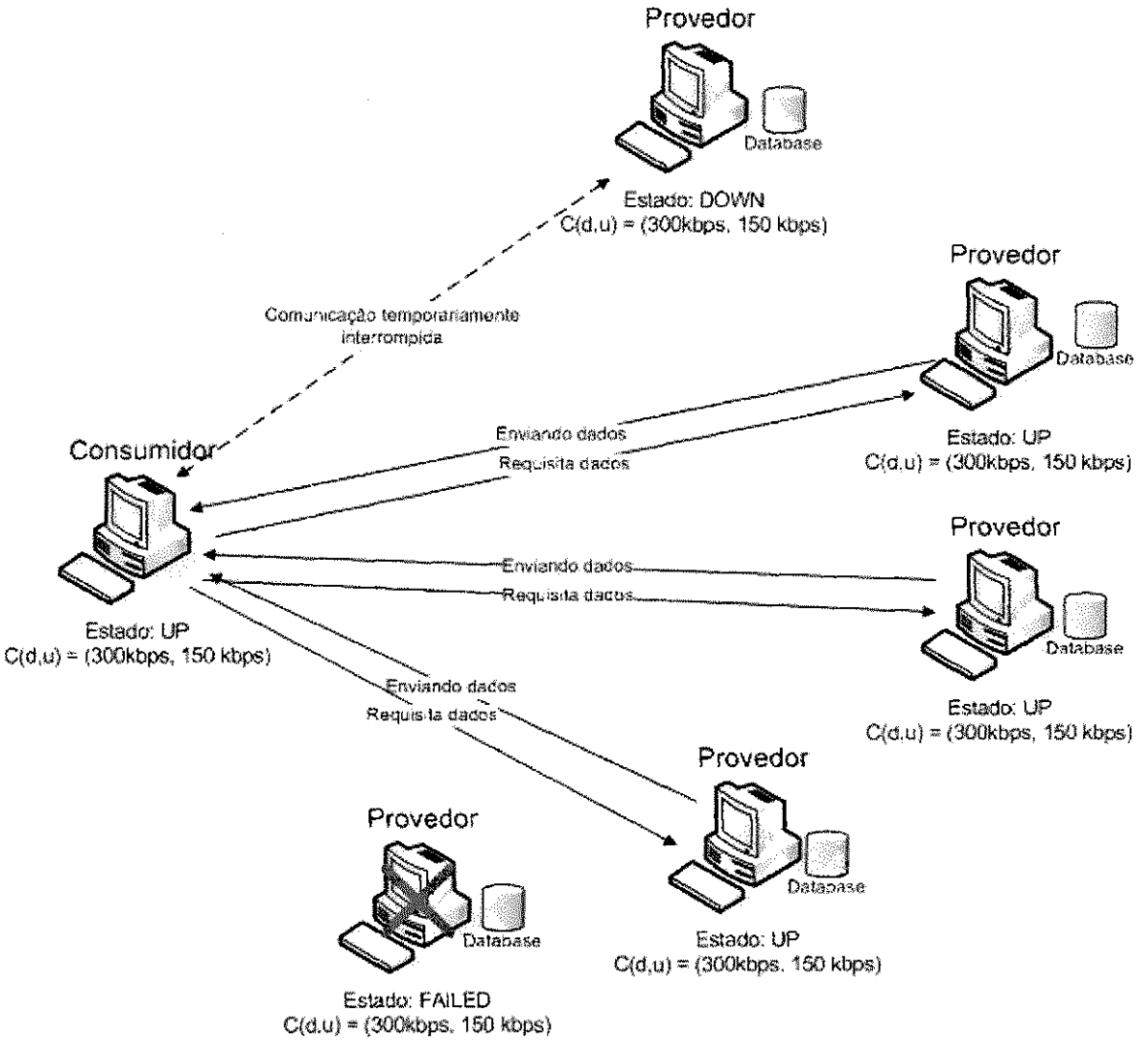


Figura 8 – Modelo de simulação

O consumidor aplica a transferência particionada de dados com o objetivo de alcançar uma maior eficiência na recuperação dos backups. Na transferência particionada, um conjunto de informações disponíveis em um conjunto de provedores é enviado para o destino em vários canais. Essa técnica visa fazer um uso otimizado da rede, onde, mesmo que uma máquina não consiga utilizar toda a capacidade da rede, um conjunto de máquinas faz o uso compartilhado resultando em um alto desempenho na operação como um todo.

Consideramos, também, que todos os trajetos da rede são independentes, de modo que não haja nenhum gargalo no processo de recuperação de backup. Supomos também que se um *peer* estiver UP, é alcançável pelo *peer* consumidor.

Qualquer processo de simulação necessita de um evento de 'fim simulação', que define a duração do processo de simulação. Este evento pode ocorrer em duas situações: 1) quando um limiar de *peers* no estado FAILED for atingido ou 2) quando todos os dados forem recuperados. Para replicação, o backup está recuperado quando S bytes forem recebidos de quaisquer provedores, e para *erasure code*, o backup está recuperado quando recebe b blocos de quaisquer provedores, de tal forma que receber 1 bloco significa receber S/b bytes de um dado provedor.

2.2.2 Simulador

Como forma de avaliar os esquemas de redundância de dados em uma gama de cenários distintos, nós desenvolvemos um simulador para o nosso modelo. Este simulador é baseado em eventos, no qual a representação do tempo é discretizada. Cada evento é definido pelo instante em que deve ocorrer e por um valor que indica a sua duração. Um escalonador mantém a ordem cronológica em que os eventos devem ser disparados e possui um mecanismo de avanço de tempo – sempre que o evento na cabeça da fila é disparado, o relógio global é atualizado para o instante em que o próximo evento na fila deverá ocorrer. Cada evento é "executado" por uma rotina que atualiza as variáveis que definem o estado do sistema e, possivelmente, insere outros eventos na fila do escalonador.

As simulações são executadas da seguinte maneira: os provedores se conectam (*i.e.* ocorrência de um evento *Join*), desconectam (*i.e.* ocorrência de um evento *Leave*) do sistema e falham (*i.e.* ocorrência de um evento *Fail*), baseados em amostras de distribuições estatísticas, até que o sistema alcance um estado estacionário, isto é o consumidor tenha recuperado todo o backup ou seja impossível prosseguir devido a um numero de falhas que não se consegue tolerar. A geração dos eventos JOIN e LEAVE é baseada em observações empíricas realizadas por *Stutzbach* [SR06]. Suas análises sugerem que a distribuição Weibull (*shape* β, *scale* α) fornece um modelo apropriado para tempo de sessão e tempo de

desconexão dos *peers*, representando um meio termo entre as distribuições Exponencial e Pareto, observadas em estudos anteriores [LNBK02, SW04]. Ainda baseados no mesmo trabalho, geramos amostras de valores usando como parâmetros *shape* = 0,59 e *scale* = 40.

Devido ao alto custo para a tolerância de falhas *free riding* (ver Seção 2.1.2), consideraremos apenas a utilização de redes sociais para a realização de backups. Como estamos usando um modelo de backup baseado em redes sociais, consideramos apenas a ocorrência de falhas de disco para os eventos FAIL. Apesar de simples, mas apropriada, nós utilizamos, novamente, a Equação 3 para estimar a probabilidade de falha de discos, usando um MTBF de 300.000 horas e um Δt de 3 meses (2.232 horas).

O tamanho dos arquivos utilizados na simulação é baseado em observações empíricas realizadas por *Crovella et al.* [CTB98]. Suas análises sugerem que a distribuição dos tamanhos dos arquivos do UNIX seguem uma distribuição Pareto com parâmetros *shape* = 1,05 e *scale* = 3800. Outro trabalho [DB99] demonstra que a distribuição dos tamanhos dos arquivos do Windows poderia ser modelada com a ajuda de uma distribuição LogNormal e uma cauda (*tail*) seguindo uma distribuição LogNormal Two-Step. Como as duas distribuições tratam-se de uma distribuição *long tail*, escolhemos a distribuição de Pareto para modelar a distribuição de tamanho dos arquivos do usuário por ser mais simples de implementar e, mesmo assim, apropriada por representar a distribuição do tamanho dos arquivos em um sistema operacional amplamente usado.

Nesse simulador, alguns parâmetros que podem ser variados para a execução das simulações são:

- Banda passante: A banda passante é a capacidade de comunicação ou da taxa de transmissão de dados de um canal de comunicação. Para melhorar a qualidade de transmissão surgiram novas técnicas que dividem a linha digital em dois canais para tráfego de *upstream* e *downstream*. Diferenciaremos, então, a capacidade de *download* (d) e a capacidade de *upload* (u) da banda passante dos *peers*. A variação deste parâmetro foi realizada para verificar a sua influência na recuperabilidade de um backup. Baseamos a capacidade de banda passante dos *peers* no serviço de banda larga provido pela empresa

Velox [vel07]. O Velox utiliza a tecnologia ADSL (*Asymmetric Digital Subscriber Line*) que possibilita a transmissão simultânea de voz e dados. Atualmente o serviço está disponível nas velocidades de 300 Kbps, 600 Kbps e 1 Mbps para *downloads* e até 150 kbps, 150 Kbps e 300 Kbps para *uploads*.

- Tamanho do backup (S): Nosso objetivo é verificar se o aumento da quantidade de dados a ser recuperada conduz ao crescimento linear do tempo necessário para recuperar o backup. Uma vez definido S, geramos arquivos, usando a distribuição Pareto acima mencionada, até obter S bytes.
- Sobrecarga de armazenamento (k): A variação deste parâmetro foi realizada para verificar a sua influência na recuperabilidade de um backup.
- Fragmentação de um arquivo (b): A variação deste parâmetro foi realizada para verificar a sua influência na recuperabilidade de um backup.
- Número de *peers*: Nós variamos a quantidade de *peers* envolvidos no backup. O número de *peers* foi determinado pelos valores dos parâmetros k e b.

Variando estes parâmetros, observamos a recuperabilidade do backup (R), que é a métrica de desempenho por nós utilizada.

2.2.3 Cenários de simulação

Estudamos cenários de simulação que representassem mudanças na capacidade de banda passante dos *peers* e no tamanho do backup a ser recuperado. Um cenário de simulação compreende a carga de trabalho, um conjunto de *peers* que armazenam os backups e um *peer* interessado na recuperação do backup. A carga de trabalho varia de acordo com três parâmetros: tamanho do *dataset* a ser recuperado S, sobrecarga de armazenamento k e fragmentação do arquivo b. A quantidade de *peers* que armazenam backups é definida por k e $k \cdot b$ para as técnicas de replicação e *erasure codes*, respectivamente. A variação dos parâmetros entre os cenários partiu da carga de trabalho de pequena escala a uma carga de trabalho de maior escala. O objetivo dessa variação é verificar a influência de cada parâmetro na recuperabilidade de backups.

A variação de todos os parâmetros (*i.e. S, d, u, k* e *b*) resulta em 900 cenários de simulação. A escolha destes valores teve como objetivo aproximar o ambiente simulado de cenários reais de recuperação de backup voltado para usuários domésticos. A Figura 9 sumariza todas as variações de parâmetros usadas.

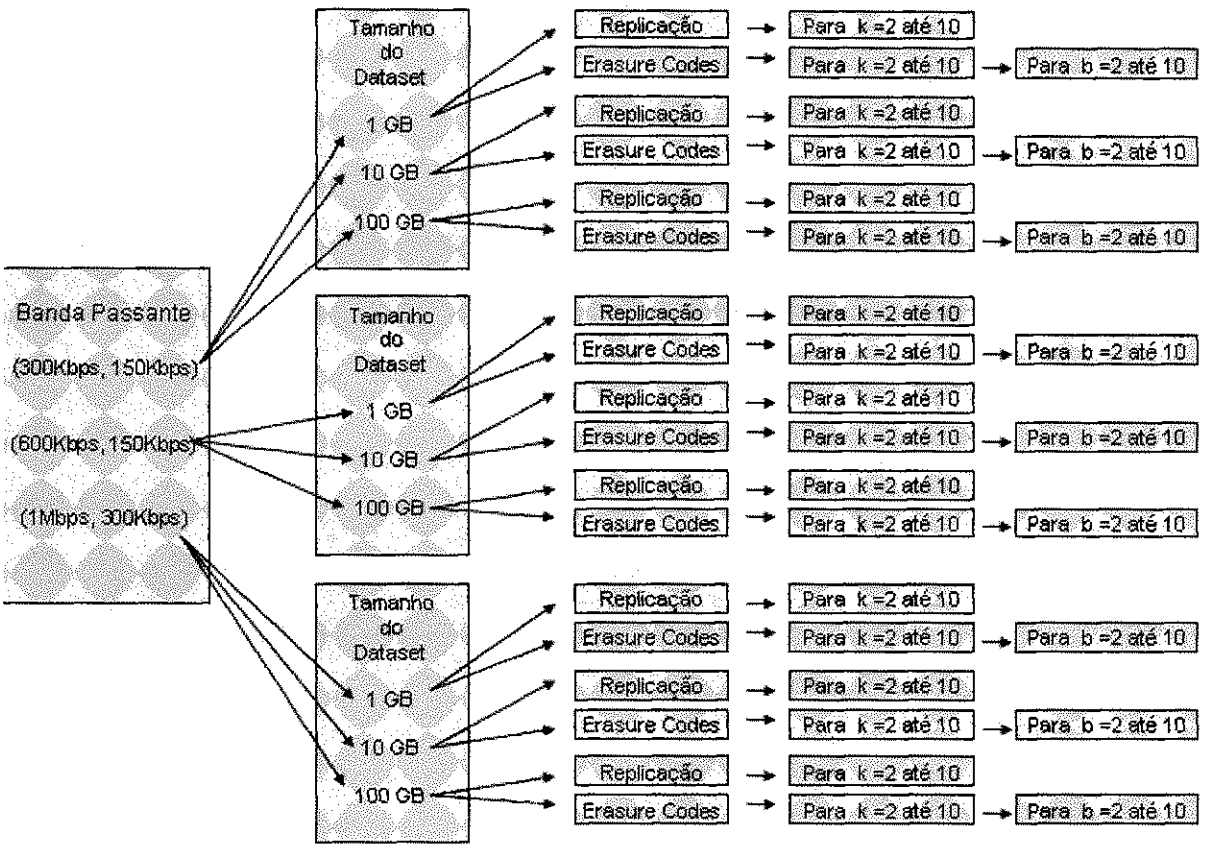


Figura 9 – Cenários de simulação.

2.2.4 Intervalo de confiança

Para assegurar que as amostras analisadas representavam, de maneira correta, os cenários, definimos o intervalo de confiança que as amostras deveriam atingir para que pudessem ser analisadas. O intervalo de confiança escolhido foi de 95%, por nos permitir uma confiabilidade satisfatória dos resultados em um número de execuções factíveis.

Executamos um número reduzido de simulações (20 simulações) para auxiliar no cálculo do número de amostras necessário para atingir o intervalo de confiança. Para calcular o número de amostras *N*, utilizamos a seguinte fórmula [FFC91]:

$$N = \left(\frac{2 z_{\alpha/2} s}{w} \right)^2$$

Em que *s* é o desvio padrão do espaço amostral e *w* é o intervalo de confiança das amostras válidas. Para uma distribuição normal e nível de confiança de 95%, o valor de $z_{\alpha/2}$ é de 1,96.

Como não conhecíamos o desvio padrão e o intervalo de confiança da amostra, realizamos um experimento com 20 execuções de cada cenário com o objetivo de calcularmos o desvio padrão e o número de simulações necessárias para obter o intervalo de confiança. Os experimentos preliminares apontaram que, para grande maioria dos cenários, as 20 execuções preliminares eram mais do que suficiente para se obter uma estimativa suficientemente confiável do intervalo. Alguns poucos cenários relacionados com S = 1 GB, necessitaram uma amostra maior de simulações, exigindo que executássemos novamente os experimentos. Os resultados deste experimento são apresentados no Apêndice A.

2.2.5 Execução dos cenários de simulação

Após o cálculo do número de amostras necessárias, geramos as instâncias de cada cenário. Para executar as mais de 36.000 (1.800 cenários X 20 amostras) simulações com resultados analisados nesta dissertação, utilizamos a grade computacional OurGrid. As simulações executadas durante a realização deste trabalho pertencem à classe de aplicações *bag-of-tasks* por não precisarem de comunicação durante a execução. A comunidade OurGrid nos forneceu em média 30 máquinas, fazendo com que a execução das simulações presentes nesta dissertação fosse realizada em por volta de 1 dia. Note também que nesta análise não está incluído o tempo gasto nas simulações que foram desconsideradas, por apresentarem erros ou por não serem significativas para as conclusões contidas nesta dissertação.

2.2.6 Análise dos resultados

A Figura 10 mostra os resultados obtidos através da simulação de seis cenários distintos usando replicação como a técnica da redundância de dados para diferentes valores de *k*. Estes cenários podem ser classificados quanto à capacidade de banda passante usada pelos *peers* e quanto ao tamanho do backup *S* a ser recuperado. O eixo *x* apresenta a variação da sobrecarga de armazenamento *k*, o eixo *y* apresenta a recuperabilidade *R* obtida, e cada curva representa diferentes valores para *S*, *d* e *u*. O tamanho do backup usado variou entre os valores 1GB, 10GB e 100GB. A capacidade de banda passante usada pelos *peers* foi $d = 300$ Kbps e $u = 150$ Kbps e $d = 1$ Mbps e $u = 300$ Kbps. Analisando estes resultados, podemos observar que os cenários que usaram $d = 300$ Kbps e $u = 150$ Kbps apresentaram melhor recuperabilidade que os cenários que usaram $d = 1$ Mbps e $u = 300$ Kbps. Este comportamento acontece porque nem sempre o *peer* que está recuperando o backup consegue utilizar a sua capacidade de *download d* por completo. Nem sempre existem *peers* suficientes para alocar todo o canal de download do *peer* consumidor. Isso faz com que o aumento da banda não apresente uma melhora linear no tempo de recuperação, pois o *TR* também depende de *u* dos *peers* provedores.

Podemos perceber, ainda analisando a Figura 10, que aumentar o valor de *k* tem um benefício direto, que é a melhoria da recuperabilidade dos backups. Devido ao fato de todos os provedores possuírem o mesmo conjunto de dados, o consumidor pode requisitar diferentes porções de dados a diferentes provedores, efetuando, assim, uma transferência particionada, que utiliza toda capacidade de *download* do consumidor. Contudo, a melhora da recuperabilidade não é linear, havendo um ponto de saturação no qual o aumento da sobrecarga de armazenamento não oferece melhoras significativas. Isto acontece quando a soma da capacidade de *upload* dos provedores é superior à capacidade de *download* do consumidor, impondo, assim, um limite superior para a otimização obtida através de transferências em paralelo. Dentre os cenários apresentados, podemos observar que a recuperabilidade *R* obtida se aproximou de 1 para $k = 10$, significando que, mesmo em um ambiente altamente dinâmico, o sistema de backup pode chegar próximo do tempo ótimo de recuperação, o que todavia requer um elevado grau de replicação.

Entretanto, devemos lembrar que uma recuperabilidade melhor não implica, necessariamente, em um TR menor. A Figura 11 mostra o *TR* obtido através da simulação dos mesmos cenários apresentados pela Figura 10, usando replicação como a técnica da redundância de dados para diferentes valores de *k*. O eixo *x* apresenta a variação da sobrecarga de armazenamento *k*, o eixo *y* apresenta a tempo de recuperação *TR* obtido, e cada curva representa diferentes valores para *S*, *d* e *u*. Analisando estes resultados, podemos observar dois comportamentos óbvios. Primeiramente, podemos perceber que o *TR* melhora quando a banda passante dos *peers* aumenta, isto acontece porque a quantidade de dados que os *peers* podem transferir cresce. Finalmente, também podemos perceber que o *TR* aumenta linearmente ao tamanho backup.

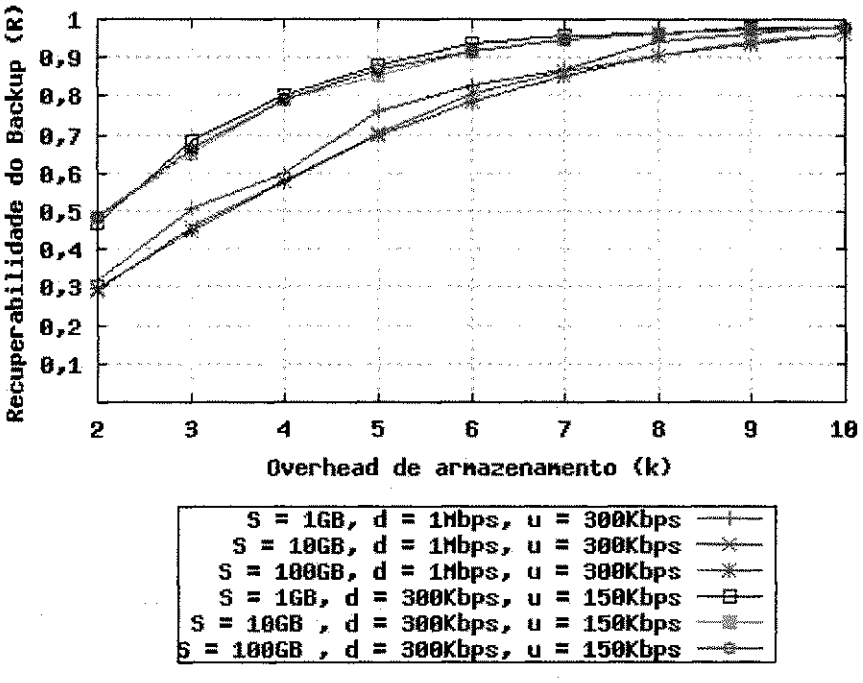


Figura 10 – Recuperabilidade usando replicação

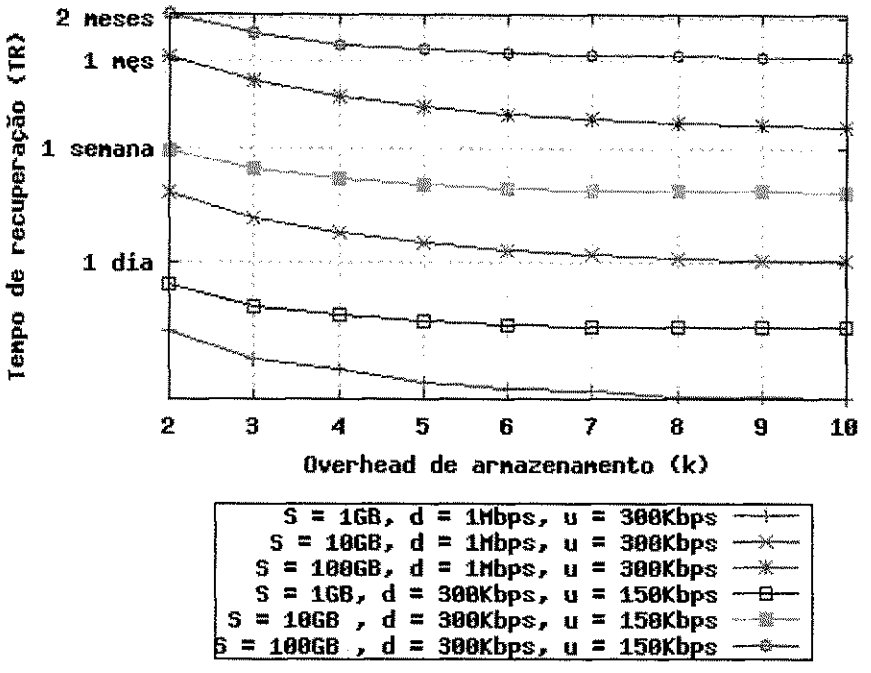


Figura 11 – Tempo de recuperação usando replicação

Em relação a *erasure codes*, aumentar os valores de k e b não implica, necessariamente, em uma melhoria da recuperabilidade. Diferentemente da replicação, quando usamos *erasure codes*, os provedores armazenam diferentes blocos de dados. Para recuperar um arquivo, é necessário recuperar pelo menos b blocos completos referentes a este arquivo. Como cada *peer* armazena diferente conteúdo de dados, a execução de transferência de dados particionada fica impossibilitada. E ao aumentarmos a quantidade de provedores que podem participar do processo de recuperação de *download*, elevamos a probabilidade que a soma da capacidade de *upload* dos provedores seja superior à capacidade de *download* do consumidor, conseqüentemente, aumentando o tempo necessário para transferir um bloco.

A Figura 12 mostra os resultados obtidos através da simulação de dez cenários distintos usando *erasure codes* como a técnica da redundância de dados para diferentes valores de k e b. O eixo x apresenta a variação da sobrecarga de armazenamento k e fragmentação de arquivo b nas figuras 12a e 12b respectivamente. O eixo y apresenta a recuperabilidade obtida, e cada curva representa diferentes valores para k ou b. O tamanho do backup a ser recuperado foi 10GB. A capacidade de banda passante usada pelos *peers* foi d =

300 Kbps e u = 150 Kbps, sendo aplicado um particionamento igualitário de banda passante por parte do *peer* consumidor.

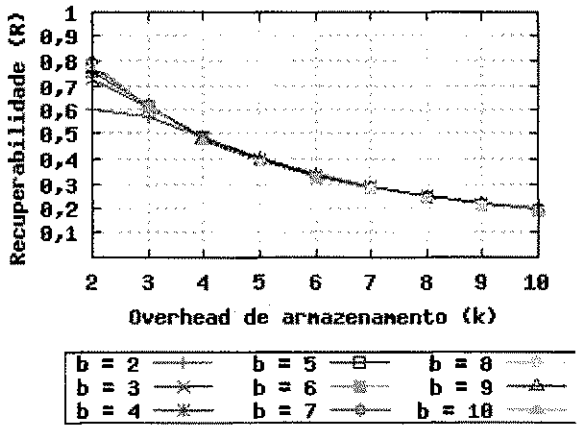
Analisando estes resultados, podemos observar que a recuperabilidade do backup deteriora à medida que elevamos o valor de k. Isto acontece porque quando aumentamos o valor de k, aumentamos a quantidade de provedores e, conseqüentemente, elevamos a probabilidade da soma da capacidade de *upload* dos provedores ser superior à capacidade de *download* do consumidor. Desta forma, ao aumentar o valor de k seriam alocadas porções de capacidade de *download* cada vez menores para os provedores, enquanto a quantidade de dados que precisa ser recuperada se mantém constante. Assim, quanto menor porção de download, maior o tempo necessário para recuperar um bloco, e conseqüentemente, maior a probabilidade de um provedor ficar DOWN, interrompendo processos de transferências correntes. Como os blocos possuem conteúdos distintos, não é possível restabelecer processos de recuperação interrompidos a partir de outro provedor, fazendo com que a recuperação efetiva de um arquivo aconteça somente quando b blocos sejam inteiramente recuperados. De tal forma que uma vez que b blocos tenham sido recuperados, todos os outros processos de recuperação relativos a este arquivo que não tenham terminados sejam descartados diminuindo a taxa de transferência agregada do *peer* consumidor.

Entretanto, ao aumentarmos o valor de b, a recuperabilidade se manteve relativamente estável para a maioria dos casos. Assim como k, quando aumentamos o valor de b, aumentamos a quantidade de provedores e, conseqüentemente, elevamos a probabilidade da soma da capacidade de *upload* dos provedores ser superior à capacidade de *download* do consumidor. Porém, a quantidade de dados que precisa ser recuperada de cada provedor para termos um bloco completo (S/b) diminui, pois quando aumentamos a fragmentação de arquivos, diminuímos o tamanho dos blocos a serem codificados.

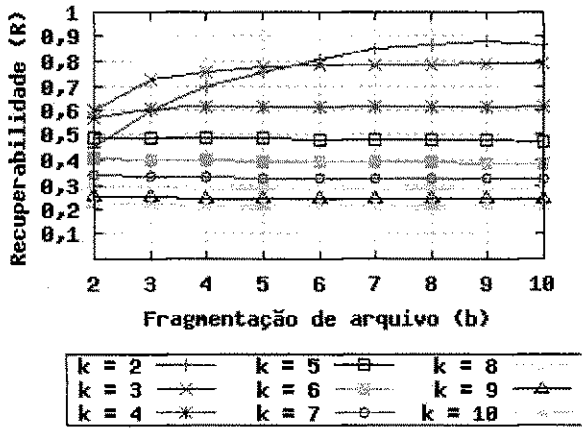
Assim como a técnica de replicação, o aumento da banda não apresenta uma melhora linear no tempo de recuperação, pois o TR também depende de u dos *peers*. Este comportamento acontece porque nem sempre o *peer* que está recuperando o backup consegue utilizar a sua capacidade de download d por completo. Nem sempre existem *peers* suficientes para alocar todo o canal de download do *peer* consumidor.

Podemos observar este comportamento através da Figura 13 que apresenta os resultados obtidos através da simulação de recuperação de backup usando *erasure codes* como a técnica da redundância de dados para diferentes valores de k, d e u. O eixo x apresenta a variação da sobrecarga de armazenamento k, o eixo y apresenta a recuperabilidade obtida R, e cada curva representa diferentes valores para d e u. O tamanho do backup a ser recuperado foi 10GB. A capacidade de banda passante usada pelos *peers* foi d = 300 Kbps e u = 150 Kbps, d = 600 Kbps e u = 150 Kbps e d = 1 Mbps e u = 300 Kbps, sendo aplicado um particionamento igualitário de banda passante por parte do *peer* consumidor.

Segundo dados do Comitê Gestor da Internet no Brasil (CETIC), a capacidade de banda passante d = 300 Kbps e u = 150 Kbps está presente na faixa de velocidade de banda larga mais usada nos domicílios do Brasil [cet07]. Como o aumento da banda passante não apresenta melhoras significativas na recuperabilidade do backup, analisaremos a recuperabilidade considerando d = 300 Kbps e u = 150 Kbps como capacidade de banda passante.

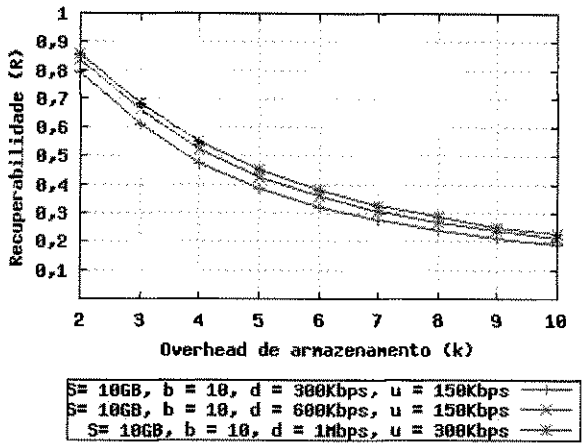


(a) S = 10GB, d = 300Kbps e u = 150Kbps para diferentes valores de b

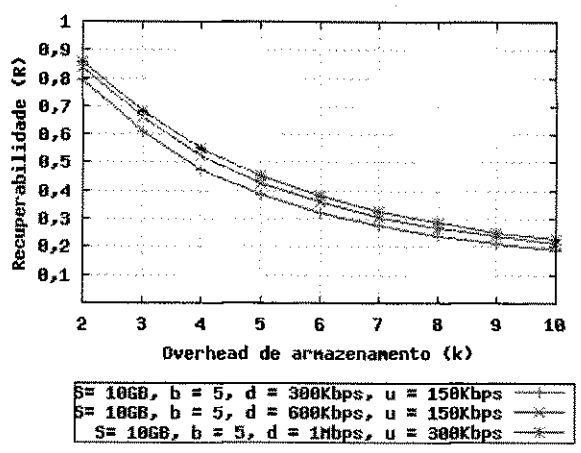


(b) S = 10GB, d = 300Kbps e u = 150Kbps para diferentes valores de k

Figura 12 – Recuperabilidade usando erasure code



(a) Particionamento igualitário de banda e $b = 5$



(b) Particionamento igualitário de banda e $b = 10$

Figura 13 – Impacto da banda passante na Recuperabilidade

Um particionamento de banda passante mais inteligente que o igualitário pode melhorar a recuperabilidade do backup. Uma alternativa é atribuirmos porções de banda passante proporcionais à disponibilidade média dos *peers*. Esta informação poderia ser obtida acompanhando o histórico de entrada e saída dos *peers*. No caso especifico do simulador, assumimos um oráculo que conhece todo o sistema e ele nos fornece a disponibilidade média dos *peers*. Podemos analisar este esquema de particionamento através da Figura 14, que apresenta os resultados obtidos através da simulação de recuperação de backup usando *erasure codes* como a técnica da redundância de dados para diferentes valores de k e b. O eixo x apresenta a variação da sobrecarga de armazenamento, o eixo y apresenta a recuperabilidade obtida, e cada reta representa diferentes valores para b. O tamanho do backup a ser recuperado foi 10GB. A capacidade de banda passante usada pelos *peers* foi d = 300Kbps e u = 150Kbps.

Analisando estes resultados, podemos observar que um particionamento mais inteligente da banda passante resulta em uma melhora na recuperabilidade dos backups. A recuperabilidade, quando usamos $b = 2$ e $k < 4$, é ligeiramente pior que a recuperabilidade obtida para $b > 2$. Isso acontece porque a quantidade de dados (*i.e.* tamanho dos blocos) que precisa ser recuperada de cada provedor é maior quando $b = 2$, e quanto menor o valor de k

menor a probabilidade de termos *b peers on-line* para recuperar o backup, aumentado, assim, o tempo necessário para a recuperação do backup.

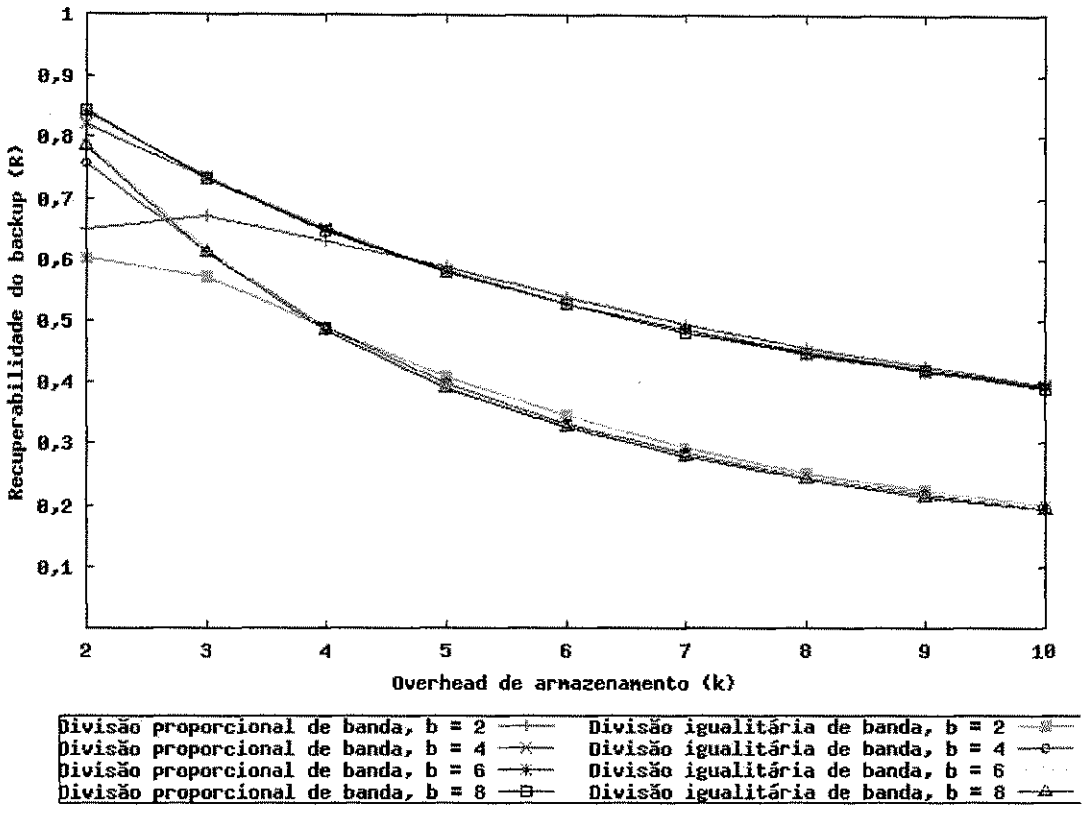


Figura 14 – Particionamento de banda passante igualitário versus proporcional

Enfim, analisando a Figura 15, podemos comparar as técnicas de replicação e *erasure codes* quanto à recuperabilidade. O eixo *x* apresenta a variação da sobrecarga de armazenamento, o eixo *y* apresenta a recuperabilidade obtida. O tamanho do backup a ser recuperado foi 10GB. A capacidade de banda passante usada pelos *peers* foi $d = 300$Kbps e $u = 150$Kbps. Percebemos que para $k < 5$, *erasure code* oferece uma melhor recuperabilidade que replicação. Isso acontece porque quanto menor o valor de k, menor a probabilidade de termos *peers on-line* para recuperar o backup, aumentado, assim, o tempo necessário para a recuperação do backup. Porém, para $k > 5$, a técnica de replicação supera a de *erasure code*.

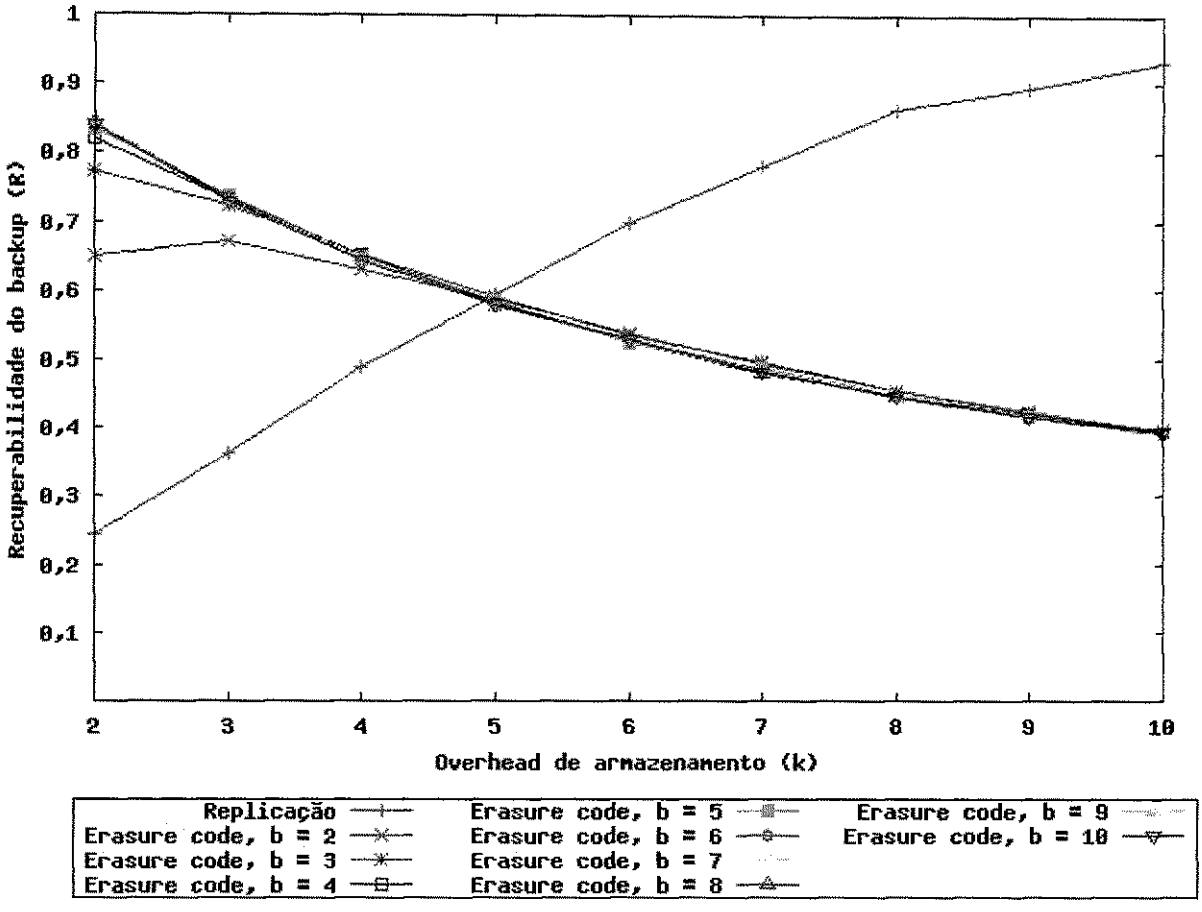


Figura 15 – Replicação versus erasure codes

2.3 Conclusões

Nesta seção, realizamos uma série de análises sobre a utilização das técnicas de redundância de dados replicação e *erasure codes*, e a utilização de redes sociais e *peer* não confiáveis sob diferentes cenários. Procuramos, por estas análises, nos ater à viabilidade da utilização de tais técnicas.

A interação com *peers* não confiáveis torna os sistemas de backup P2P anônimos fadados a não funcionarem devido ao alto custo para a tolerância de falhas *free riding*. A criação de novas cópias em substituição àquelas que falharam consome grandes quantidades de banda passante, tornando o sistema impraticável. Acreditamos que a utilização de redes sociais para a realização de backup P2P permite que a taxa de falhas seja reduzida à

ocorrência de falhas de disco, diminuindo, assim, a taxa de falhas a ser tolerada, e, conseqüentemente, permitindo a praticabilidade de sistemas P2P de backup.

Uma vez definida a utilização de redes sociais na construção de sistemas P2P de backup, precisamos definir qual técnica de redundância de dados a ser utilizada. Baseados nos resultados das nossas análises, a técnica replicação consome mais espaço de armazenamento que os *erasure codes* para garantia de um mesmo limiar de alta confiabilidade de backup perante a ocorrência falhas de disco. E oferece, ainda, menor recuperabilidade de backup que os *erasure code* para $k < 5$.

Entretanto, as economias da redundância de usar *erasure codes* no lugar de replicação têm um preço. A tarefa de recuperar apenas um subconjunto de dados de um arquivo, quando usado *erasure codes*, é complicada devido à fragmentação e codificação aplicada nos arquivos. Diferentemente de *erasure codes*, a técnica de replicação permite um acesso aleatório eficiente a um subconjunto de dados, pois os arquivos são armazenados em seus formatos originais, não sendo aplicada nenhuma transformação, preservando, assim, a localidade dos dados. Os *erasure codes* introduzem gastos de processamento adicionais ao sistema referentes à codificação e decodificação dos dados [LCL04]. Especialmente para arquivos grandes na ordem de centenas de megabytes, os algoritmos de codificação e decodificação poderiam introduzir gargalos do desempenho significativos no sistema. Outra desvantagem em relação ao uso de *erasure codes* é a latência de recuperação de um backup em ambientes onde a latência dos nós é muito heterogênea. Ao usarmos replicação, um backup pode ser recuperado a partir da cópia que estiver mais próxima do consumidor, enquanto que, ao usamos *erasure code*, a latência do backup está limitada a latência dos b blocos mais próximos do consumidor [mtb07].

Os *erasure codes* não introduzem apenas complexidade devido aos processos de codificação e decodificação, mas o projeto inteiro do sistema torna-se mais complexo. É necessária uma quantidade maior de metadados para controlar o armazenamento dos blocos, e o processo de atualização dos arquivos requer uma atenção especial, pois uma pequena atualização em um arquivo resulta na mudança de todos os blocos que pertencem ao arquivo. Como um princípio geral, acreditamos que inserir complexidade no projeto do sistema deve

ser evitado, a menos que o beneficio advindo desta introdução seja claramente recompensador.

Enfim, existe um tradeoff entre complexidade e economias de armazenamento entre essas duas técnicas de redundância de dados. Devido estas questões, decidimos por projetar o OurBackup oferecendo suporte para tanto replicação e *erasure codes*. Devido a sua maior simplicidade, implementamos primeiramente a realização de backup através de replicação, entretanto, futuramente, também avaliaremos a introdução de *erasure codes*.

Capítulo 3 OurBackup

Este capítulo descreve a arquitetura e o projeto de um novo sistema P2P de backup, chamado OurBackup, uma solução simples, com baixo custo administrativo, que se utiliza da rede social dos usuários para a construção da relação entre os *peers*. Examinaremos o OurBackup sob uma perspectiva de alto nível, dando uma visão geral das funcionalidades que o sistema oferece ao usuário, argumentando sobre algumas decisões de projeto que foram tomadas. Este capítulo também apresenta OurBackup sob uma perspectiva de mais baixo nível, explicando a arquitetura do sistema e descrevendo os módulos que compõem o sistema.

O OurBackup possui um conjunto de funcionalidades que permite aos usuários efetuarem backup na máquina de amigos, explorando espaço de armazenamento ocioso nessas máquinas. Cada usuário é responsável por construir sua própria rede social. A criação de novas amizades é feita através do cadastro do identificador de usuários na lista pessoal de amigos. O OurBackup realiza backups através da replicação integral dos arquivos.

A linguagem Java foi usada na implementação do OurBackup, tendo sido escolhida por apresentar facilidades para a implementação do sistema. Entre as facilidades providas, pode-se destacar: alta portabilidade do código-fonte, existência de suporte para comunicação através da rede, bibliotecas para acesso a banco de dados e suporte a múltiplas *threads* de execução.

Ao longo do restante deste documento, a menos que indicado de outra maneira, o termo *amigo* se refere a um computador de algum usuário que participa do OurBackup. Um *agente* do OurBackup é a ferramenta que os usuários usam para interagir com o sistema. Os arquivos são classificados no sistema como *originais* e *cópias*. Arquivos originais são aqueles selecionados pelo usuário para backup. Cópias são as réplicas dos arquivos originais armazenadas nas máquinas de amigos. A quantidade de cópias de um arquivo original é determinada pelo grau desejado de confiabilidade do backup.

3.1 Requisitos

O OurBackup possui como requisitos básicos de projeto garantir alta confiabilidade e recuperabilidade dos backups, bem como a confidencialidade e integridade. A confiabilidade é indicada pela probabilidade de um backup poder ser recuperada. A recuperabilidade indica o quão rápido um backup poder ser recuperado. Garantia de confidencialidade de dados é a capacidade de permitir que apenas usuários autorizados consigam ler o conteúdo dos backups. Garantir integridade é ser capaz de proteger os backups contra modificações acidentais ou mal-intencionadas.

Backups podem incluir informações sigilosas. Como os backups são armazenados em computadores que não estão sob controle administrativo do dono dos arquivos, é de extrema importância que se mantenha a confidencialidade dos dados. Muito provavelmente nenhum usuário gostaria que seus dados sigilosos, como senhas de bancos, estivessem disponíveis para outras pessoas.

Os mecanismos de integridade são responsáveis por detectar mudanças acidentais ou mal-intencionadas nos backups e, quando possível, impedi-las. A integridade de um backup poderá ser afetada quando o processo de transferência de dados não for concluído com êxito. Um amigo pode se desconectar do sistema durante a transferência, deixando o estado do backup corrompido. Assim, é imprescindível para o sistema identificar e corrigir essas falhas de integridade.

3.2 Design do Sistema

A macro-arquitetura do OurBackup é composta por um servidor centralizado e agentes, como pode ser visualizado na Figura 16. O servidor é responsável pelo gerenciamento dos usuários, das redes sociais e dos metadados de onde cada cópia está armazenada. Os agentes são responsáveis pela execução dos backups, realizando a negociação de armazenamento dos backups diretamente entre si, não envolvendo o servidor durante o processo de transferência dos arquivos. Contudo, os agentes não são capazes de descobrir a localização (*i.e.* endereço de rede) de outros agentes desejados de forma distribuída, os

mesmos dependem de um servidor central. Além disso, os agentes notificam o servidor depois de transferências bem sucedidas.

Quando um agente deseja se conectar ao sistema, deve conectar-se ao servidor central e então informar seu atual endereço de rede. Uma vez estabelecida a conexão, o agente consulta o servidor sobre a localização de todos os amigos que fazem parte da rede social do usuário. Ocasionalmente, o agente pode desejar estabelecer um canal de comunicação para transferências de backup. Os backups são transferidos utilizando o protocolo HTTP diretamente entre dois agentes, sem intervenção do servidor.

Pode-se notar facilmente que esta arquitetura não é inteiramente do tipo P2P. Esta arquitetura combina vantagens do modelo Cliente/Servidor e do modelo P2P. A transferência dos arquivos ocorre de forma distribuída, não envolvendo o servidor e, conseqüentemente, aumentando a escalabilidade do sistema. O uso do servidor centralizado fornece suporte à cooperação entre *peers*, pois facilita a implementação de técnicas de contabilidade de compartilhamento de recursos e autenticação entre os *peers* [MK06]. Assim, a arquitetura híbrida representa o melhor dos dois mundos, resolvendo via servidor centralizado o que é complicado fazer via P2P em virtude da necessidade de confiança e resolve via P2P o que é dispendioso fazer via servidor centralizado devido à quantidade de recursos que seriam necessários.

O uso de um servidor centralizado apresenta, ainda, vários benefícios devido ao fato de poder ser usado para controlar ou gerenciar alguns aspectos do sistema, facilitando, assim, o suporte à segurança, manutenibilidade e gerenciabilidade. A manutenibilidade representa a facilidade na qual o sistema pode ser atualizado ou reparado, já estando em funcionamento. A gerenciabilidade representa a facilidade na qual o sistema como um todo pode ser gerenciado.

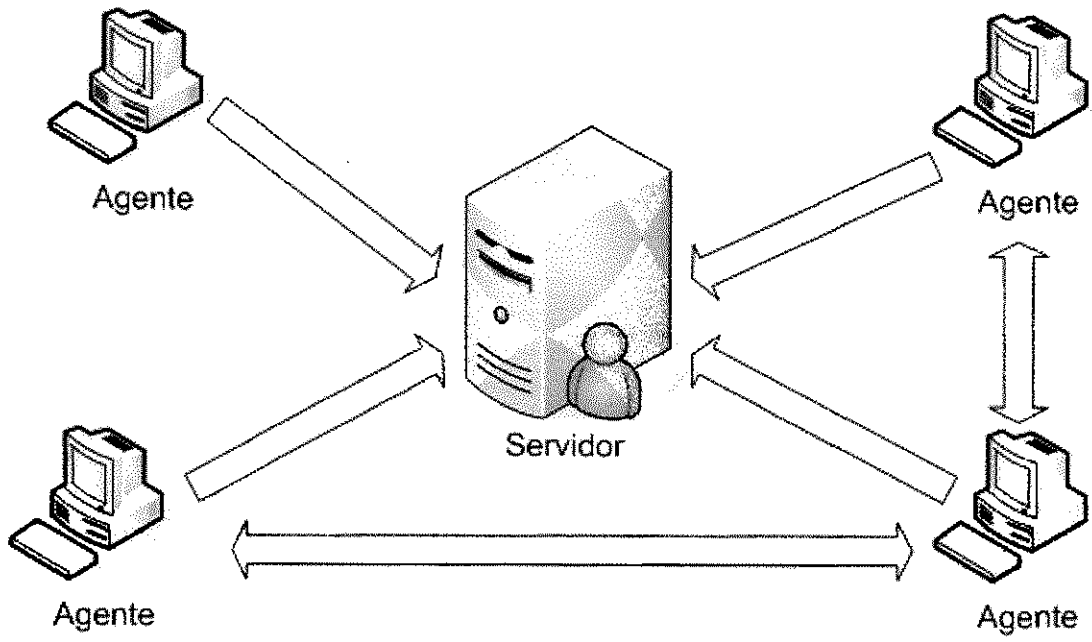


Figura 16 – Macro-Arquitetura do OurBackup

3.2.1 Servidor

Uma das maiores dificuldades na administração de um sistema diz respeito ao gerenciamento dos usuários. A melhor estratégia para esta tarefa é utilizar uma base única, o que evita o retrabalho no cadastro e na alteração de metadados destes usuários. O servidor também precisa manter cópias de segurança dos metadados de backup com intuito de garantir a confiabilidade, pois quando o usuário precisa de backup, há uma grande chance de sua máquina ter falhado, deixando seu agente inoperante.

O servidor fornece uma interface de acesso a sua base de metadados que possui atrelada mecanismos de autorização. Um usuário tem acesso apenas à porção de metadados para a qual possui permissão. O mecanismo de controle de acesso possibilita a manipulação dos metadados estritamente de acordo com a política de autorização especificada para o sistema. São pré-condições para o seu adequado funcionamento que o usuário que solicita acesso tenha sido corretamente autenticado.

Esta interface de acesso permite a execução de procedimentos de criação, atualização, remoção e consultas de usuários, requisição e atualização de relações de

amizades, e ainda o registro, atualização, remoção e consultas a metadados de backup (*i.e.* onde cada arquivo original está replicado). Cada agente deve periodicamente atualizar o servidor sobre o estado do usuário, a rede social e metadados de backup. Quando o usuário deseja obter alguma informação (*e.g.* atual endereço de rede dos amigos), deve consultar o servidor para obter as informações mais recentes.

O armazenamento centralizado de informações sobre todos os usuários do OurBackup pode gerar limites de escalabilidade ao modelo, uma vez que requer servidores maiores quando o número de requisições aumenta, e mais espaço para armazenamento à medida que a quantidade de usuários cresce. Mas pode-se também usar um *cluster* de servidores para se obter maior escalabilidade. E, na prática, a experiência do Napster mostrou que, exceto por questões legais, esse modelo pode ser bastante robusto e eficiente [SKS+05].

3.2.2 Agentes

Os agentes possuem autonomia parcial em relação ao servidor. Cada agente possui um catálogo local com os metadados de seu backup. Os agentes operam localmente com os metadados de backup do repositório local e periodicamente atualizam essas informações no servidor. A indisponibilidade do servidor impossibilita o funcionamento de todo o sistema, pois o agente não pode atualizar os metadados de backup no servidor e consultá-lo sobre a presença de amigos no sistema.

O usuário pode utilizar o OurBackup a partir de um agente instalado em qualquer máquina que possua acesso à Internet. Quando o usuário não estiver usando sua máquina de origem, o agente realiza consultas de metadados apenas no servidor.

3.3 Modelo de metadados

O OurBackup necessita manter diversos tipos de metadados. Informações sobre os arquivos para os quais um usuário quer manter backup, configurações e ajustes utilizados no processo de backup, associações entre os arquivos originais e suas cópias e a localização das cópias, os amigos que fazem parte da rede social do usuário, são todos metadados a que um agente do OurBackup necessita ter acesso.

Existem duas formas de manipulação de metadados no OurBackup: via acesso direto ao Servidor, ou via *caching*. O primeiro é utilizado para o controle de dados de pouca volatilidade, como: constantes do sistema controladas pelo administrador e dados de configuração de usuários; o segundo é utilizado no controle de metadados de backup. As formas de manipulação podem ser visualizadas na Figura 17.

- A manipulação via *caching*, manipula os dados, primeiramente, em um repositório local e somente após a consolidação dos mesmos é realizada a atualização do repositório central. O *caching* de dados é utilizado com o propósito de não inserir inconsistências de dados no servidor e também de diminuir a carga de requisições de atualização ou recuperação ao servidor.
- No acesso direto, não existe a utilização de repositórios intermediários de dados.

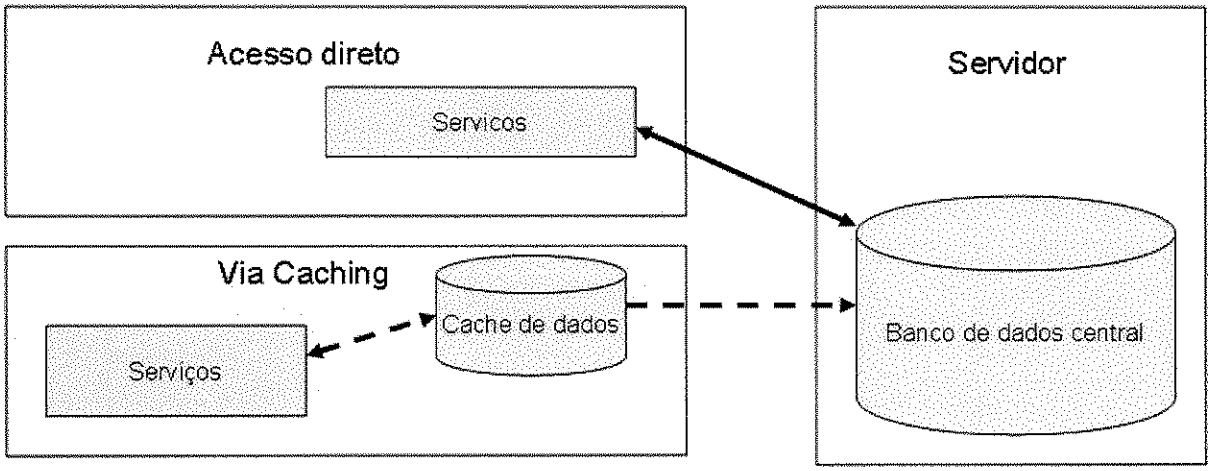


Figura 17 – Modelo de acesso a metadados

3.3.1 Organização dos metadados

Os metadados são representados por entidades de dados que possuem relacionamentos e atributos. Cada entidade representa um conjunto de pessoas ou conceitos sobre as quais estarão submetidas as regras de negócio do OurBackup. Cada relacionamento representa a associação entre duas entidades. Cada atributo representa uma característica ou parte da informação de uma entidade. As entidades de dados presentes no OurBackup são: *user*, *file*, *replica* e *friendship*.

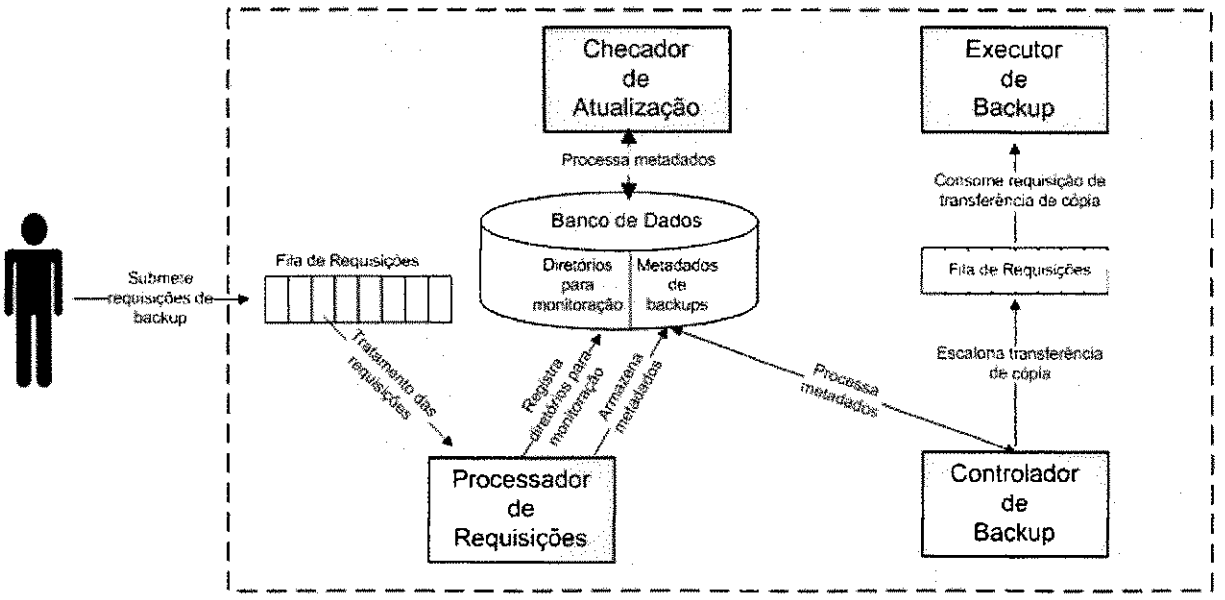


Figura 18 – Processo execução de backup

O tratamento de requisições de backup é procedido pelo módulo Processador de Requisições, realizando a extração dos metadados relativos ao conjunto de arquivos selecionados e armazenamento dos mesmos para processamento *a posteriori*. Uma requisição de backup pode compreender arquivos específicos ou diretórios inteiros a partir de uma árvore de diretórios. Em ambos os casos, o sistema extrai os metadados relativos ao conjunto de arquivos que se deseja efetuar backup. E, no caso específico de diretórios, o tratamento da requisição possui, ainda, uma atividade adicional que é o registro destes diretórios para monitoração sobre a criação de novos arquivos. Sempre que são criados novos arquivos dentro de um diretório selecionado para backup, o sistema efetua, automaticamente, o backup deles.

O processamento dos metadados de backup é sucedido pelo módulo Controlador de Backup. Este módulo realiza uma varredura no catálogo de metadados com o objetivo de localizar backups que ainda requerem a criação de cópias. Esta varredura é realizada somente quando novos arquivos foram marcados para backup ou quando backups precisam ser atualizados. Uma vez encontrado um conjunto de arquivos que necessitem ser tratados, o módulo Controlador de Backup inicia o processo de criação de cópias, efetuando a alocação de espaço de armazenamento para as cópias.

A fase de alocação determina quais são os amigos que irão armazenar as cópias. Esta escolha é determinada pela disponibilidade de espaço de armazenamento suficiente para receber as novas cópias. No sistema, quando um usuário desejar realizar backup ele deve angariar espaço nos computadores dos amigos. A alocação de espaço de armazenamento para um amigo é um contrato de backup. O valor deste contrato é a quantidade de espaço que foi alocada. Cada usuário mantém todos os contratos de backup dos quais ele participa e somente requisita novos contratos para backups quando nenhum dos firmados anteriormente puder ser utilizado para armazenar as cópias. Os contratos de backups são definidos manualmente. Os usuários definem, através do agente, a quantidade de espaço que querem doar para cada amigo.

O processo de transferência de novas cópias também deve ser realizado de forma assíncrona com tratamento das requisições, pois enquanto um módulo do agente está tratando as requisições do usuário, outro módulo está efetuando a transferência das cópias. Assim, o Controlador de Backup não deve remover ou transferir diretamente as cópias, encaminhado esta responsabilidade para outro módulo independente chamado Executor de Backup.

O Controlador de Backup requisita ao Executor de Backup a manipulação das cópias através de uma fila compartilhada, podendo inserir requisições na fila, enquanto o Executor de Backup consome requisições. Esta fila de requisição possui uma capacidade de *slots* limitada pela quantidade de requisições que o Executor de Backup consegue atender por vez. Quando o Controlador de Backup desejar adicionar um elemento à fila, e esta já estiver totalmente preenchida, ele fica bloqueado aguardando espaço disponível na fila. Similarmente, se o Executor de Backup quiser retirar um item da fila e a encontrar vazia, deve aguardar até que o Controlador de Backup enfileire alguma nova requisição.

O módulo Executor de Backup é responsável pelas fases de preparação e transferência das cópias. A fase de preparação é responsável apenas por cifrar e compactar as cópias que deverão ser enviadas aos amigos já escolhidos. Ao final desta, são gerados os metadados relativos às cópias, sendo usados para permitir decriptação e descompactação das cópias mesmo quando o computador de origem do usuário tenha sofrido uma falha catastrófica.

A fase de transferência de backup é responsável por realizar a transferência das cópias para as máquinas dos amigos selecionados. As operações de uploads das cópias são executadas assincronamente em relação à execução das outras fases, permitindo, assim, que o Executor de Backup possa escalonar outras operações de *upload*.

Arquivos originais podem, porventura, ser atualizados pelo usuário. Quaisquer modificações feitas nos arquivos originais devem ser propagadas para todas as suas cópias. O sistema também procede a atualização de backups de forma automática e transparente para o usuário. A checagem de atualizações dos arquivos é realizada pelo módulo Checador de Atualização. Este módulo realiza uma varredura no sistema de arquivo com o objetivo de verificar se os arquivos marcados para backup foram alterados. Sempre que um arquivo original for atualizado, ele será representado no sistema como uma nova versão. Devido à participação dinâmica dos amigos, não é possível atualizar as cópias de forma atômica, pois nem sempre temos todos os amigos presentes para receberem as últimas atualizações.

A detecção de alterações nos arquivos originais é realizada periodicamente pelo Checador de Atualização, sendo o intervalo de tempo entre as checagens configurado pelo usuário. O sistema varre toda coleção de arquivos originais, verificando se a *data de modificação* foi atualizada. Se foi, o sistema ainda realiza comparação dos *checksums* do arquivo atual e da ultima versão do backup deste arquivo. Caso os *checksums* sejam diferentes, o sistema cria uma nova versão para este arquivo original, que será tratada pelo módulo Controlador de Backup. No OurBackup, versão é um inteiro que é incrementado monotonicamente.

Quando é criada uma nova versão de um arquivo, o OurBackup interrompe o processo de criação de cópias para as versões antigas e a medida que uma cópia para versão mais recente é criada, uma cópia antiga é escalonada para remoção. Todo este processo é escalonado automáticamente através do módulo Controlador de Backup, que repassa as requisições de remoção para o Executor de Backup da mesma forma que uma requisição de transferência. Quando uma versão antiga não possuir mais nenhuma cópia no sistema, esta será removida do catálogo de metadados.

3.4.2 Recuperação de Backup

O agente do OurBackup provê interfaces visuais que possibilitam a um usuário selecionar o conjunto de arquivos originais a serem recuperados. Esta interface permite a visualização das configurações e do estado (*e.g.* versão, compactação, criptografia) backups já efetuados. O usuário necessitará apenas selecionar o conjunto de arquivos que deseja recuperar e o agente realizará o escalonamento de cada requisição.

Uma vez que o usuário tenha selecionado os arquivos originais que deseja recuperar, o sistema inicia o processo de recuperação. Assim como o processo de execução de backup, o sistema não deve bloquear durante o processo de recuperação de backup, funcionando assincronamente de tal forma que as requisições sejam aceitas independentemente de o agente estar ou não pronto para recebê-las. O funcionamento assíncrono é necessário pois o tempo de recuperação pode ser muito longo. Após a requisição da recuperação de backup, o usuário pode continuar operando normalmente o agente. Independentemente de o agente estar, ou não, preparado, as requisições de recuperação são inseridas em uma fila onde o agente possa tratá-las posteriormente (Ver Figura 19).

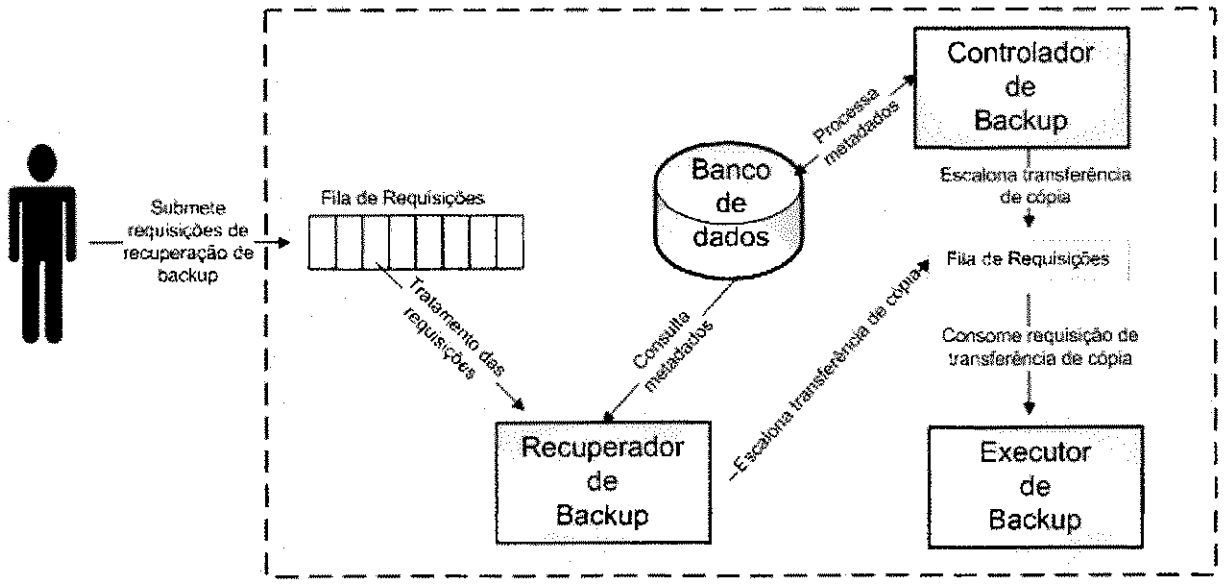


Figura 19 – Processo de recuperação de backup

Uma requisição de recuperação pode compreender arquivos específicos ou diretórios inteiros a partir de uma árvore de diretórios. Em ambos os casos, o sistema busca os metadados relativos ao conjunto de arquivos que se deseja recuperar. Subseqüentemente, é consultado o estado de disponibilidade (*i.e. on-line* ou *off-line*) dos amigos que armazenam este backup. Uma vez que a localização, o estado e a disponibilidade das cópias são determinados, o sistema inicia o processo de escalonamento das operações de *download*. Este escalonamento prioriza a recuperação das cópias mais recentes que estejam prontamente acessíveis, i.e, prioriza as cópias mais recentes que estejam armazenadas nos amigos *on-line*.

O processo de recuperação das cópias também deve ser realizado de forma assíncrona com tratamento das requisições. Pois, enquanto um módulo do agente está tratando as requisições do usuário, outro módulo está recuperando as cópias. Assim, o Recuperador de Backup não deve transferir diretamente as cópias, encaminhado esta responsabilidade para o Executor de Backup, que acumula a responsabilidade de realizar o *download* das cópias. O Recuperador de Backup requisita ao Executor de Backup a recuperação das cópias através da mesma fila compartilhada usada pelo Controlador de Backup.

O processo de recuperação de um backup pode ser realizado a partir de qualquer máquina que possua um agente instalado. Arquivos que demandem muito tempo de transferência pela Internet podem ser recuperados diretamente do computador do amigo que o possui armazenado. Os dados recuperados estarão disponíveis localmente no computador do amigo, podendo ser transferidos para uma mídia portável tais como DVD, *pendrives*.

O uso de redes sociais no OurBackup permite influenciar na disponibilidade dos agentes. Um usuário pode se utilizar das regras sociais e relações de confiança presentes na sua rede social para solicitar para amigos deixarem suas máquinas conectadas por um determinado período, facilitando processos de recuperação de backups.

3.4.3 Inspeção dos Backups

Dados que estão armazenados nos amigos não possuem garantia de persistência eterna. Amigos que estão com espaço de armazenamento escasso podem apagar os arquivos

do OurBackup. Os computadores de amigos também estão sujeitos a falhas. No pior caso, mesmo sujeito às regras da rede social, um amigo pode agir maliciosamente.

O OurBackup possui mecanismos de auditoria que permitem detectar casos de ruptura de contratos de backup. Periodicamente, o agente requisita aos amigos o *hash* de determinados blocos de dados aleatórios. Uma série desses desafios pode, probabilisticamente, identificar comportamento malicioso dos amigos. O sistema então notifica o usuário sobre este tipo de comportamento.

Se um amigo não respondeu corretamente ao *hash* requisitado, o agente o cadastrará em uma "lista negra" para que o usuário tome atitude apropriada, removendo ou reinviando os backups para este amigo. Caso um amigo, também, não tenha respondido em um intervalo de tempo signicativo, o agente também o cadastrará em na "lista negra" para que o usuário tome as atitudes apropriadas.

3.4.4 Manipulação da rede social

No OurBackup, o processo de backup pressupõe a existência de uma rede social. O sistema provê mecanismos que permitem ao usuário construir sua própria rede social, sendo as operações básicas de manipulação: a adição, a remoção e a consulta de amigos. Um usuário recém cadastrado possui uma rede social vazia.

Todos os metadados relativos à rede social do usuário são recuperados através do servidor. Estes metadados armazenam as relações de amizades estabelecidas, pendentes e removidas. São armazenadas também notificações sobre o estado de uma amizade e a quantidade de espaço doada para cada amigo.

3.4.4.1 Inserção de amigos

O processo de inserção de novos amigos em uma rede social será realizado de acordo com o seguinte protocolo:

1. O usuário cadastra o identificador do amigo a ser inserido;

2. O sistema verifica a validade do identificador. Em caso de invalidade sintática ou inexistência de um usuário cadastrado no sistema com esse identificador, o usuário é notificado sobre a inconsistência, e o processo de inserção é abortado. No caso de se tratar de um identificador válido, o processo continua normalmente.
3. É enviada para o usuário selecionado uma requisição de autorização para estabelecer a relação de "amizade". Vale ressaltar que pode existir um atraso na confirmação do pedido de amizade, pois o futuro amigo pode não estar conectado ao sistema.
4. Caso a autorização seja concedida, é estabelecida uma relação de amizade entre os dois usuários. Caso contrário, o usuário é notificado sobre a negação e nenhuma relação é estabelecida.
5. Depois de confirmada a amizade, o usuário pode estabelecer um contrato de backup, definindo, manualmente, quanto de espaço de armazenamento quer doar para o seu amigo.

3.4.4.2 Remoção de amigos

Quanto à remoção de relações de amizade, existirão duas abordagens a serem escolhidas pelo usuário.

Numa abordagem abrupta, uma vez requisitada a remoção, todas as informações relacionadas a esta "ex-amizade" são removidas do sistema, inclusive backups efetuados entre os dois usuários. O amigo que foi removido é notificado sobre a quebra da relação de amizade, mas não possui nenhum mecanismo de proteção/garantia dos backups que porventura tenham sido efetuados através desta amizade. Neste caso, o sistema escalonará novas cópias para serem criadas através da atualização do metadados de backup. O processo de criação de novas cópias é comandado pelo Controlador de Backup (Ver seção 3.4.1).

A segunda abordagem de remoção de amizades de maneira mais suave será provida pelo sistema no futuro. O amigo continuará sendo notificado da quebra da relação de amizade,

e seus backups serão excluídos à medida que forem reconduzidos para outro amigo. Neste caso, a relação de amizade será rotulada com o status de DELETION_PENDING.

3.4.4.3 Consulta de amigos

O uso de identificador torna a descoberta de usuários uma tarefa não-trivial. Um identificador não é, obrigatoriamente, a opção mais amigável de identificação de uma pessoa. O sistema fornece mecanismos de busca de usuários que permitem encontrar amigos sem a necessidade dos identificadores como chave das consultas. O mecanismo de consulta pode ser usado para localizar ou convidar amigos através de consultas por nome, sobrenome, país e estado.

3.4.5 Mecanismo de presença

Um mecanismo da presença fornece meios para notificar e ser notificado sobre mudanças de presença dos usuários. Presença significa a aplicação estar consciente da disponibilidade (*i.e. on-line* e *off-line*) e localização (*i.e.* endereço de rede) dos usuários. Tal mecanismo notifica eventos de entrada e saída de amigos no sistema. Após o usuário se autenticar no sistema, o seu agente notifica sobre a sua presença a todos os amigos já conectados. Esta notificação contém o endereço de rede sob o qual é possível realizar uma conexão com este usuário. O processo de saída acontece de maneira análoga, o usuário ao solicitar sua desconexão do sistema, o seu agente notifica os amigos de sua saída. Estas informações serão utilizadas pelos mecanismos de backup para enviar, remover, e recuperar cópias.

3.4.6 Segurança

A segurança de daos é um dos principais requisitos exigidos de sistemas computacionais que lidam com informações importantes. Nesse contexto, um sistema seguro é aquele que garante integridade e confidencialidade dos dados, e que os acessos às

informações sejam feitos apenas por pessoas autorizadas, permitindo protegê-las contra acessos indesejáveis.

3.4.6.1 Autenticação e Autorização

O serviço de autenticação é responsável por validar a identidade de um usuário e verificar suas permissões, a fim de autorizar a execução de tarefas críticas (*e.g.* excluir arquivos armazenados). Autorização e autenticação são dois conceitos fundamentais na área de controle de acesso. São conceitos distintos, mas interdependentes, de forma que a autorização de acesso a um determinado recurso é, na verdade, dependente da autenticação. Enquanto a autenticação é o processo de determinar quem somos para o sistema, a autorização determina o que podemos a fazer. Autorização refere-se a decisão se um usuário tem o acesso garantido a um recurso do sistema.

A autorização necessariamente depende da realização da autenticação. Se o sistema não for capaz de ter certeza a respeito da identidade de determinado usuário, não haverá uma maneira correta de determinar se o usuário deve ou não acessar o recurso. O mecanismo de autenticação mais popular e usado nos sistemas de computação é a autenticação através de senhas.

O OurBackup autentica os usuários através da comparação de senhas. Entretanto, por questões de segurança, as senhas não são armazenadas em texto claro no servidor. Se um arquivo de senhas for roubado ou um banco de dados com registros de senhas for violado, o estrago pode ser enorme. Como um *hash* não é reversível e, para serem usadas, as senhas precisam ser conferidas, é muito mais prudente armazenar os resultados *hash* das senhas do que as próprias senhas. Assim, o mecanismo de armazenamento de senha aplicado no OurBackup é aquele em que a senha é, primeiramente, cifrada, usando uma técnica de *message digest* ou *hash* criptográfico como SHA-1 ou MD5, para posteriormente ser armazenada.

O uso de uma senha pressupõe que um usuário a digite. Tendo a senha como entrada, é fácil e rápido calcular o resultado *hash* da senha fornecida e compará-lo com o valor arquivado. Se forem idênticos, a senha confere, mostrando que o usuário conhecia uma senha

válida. Este procedimento reduz sensivelmente os riscos porque o único momento em que a senha pode ser roubada é enquanto está sendo digitada e antes de ser transformada em *hash*.

Entretanto, este tipo de autenticação é considerado vulnerável, pois está sujeito a vários incidentes que podem comprometer a segurança do método, tais como senha "fraca", esquecimento da senha, furto de senha. Uma senha fraca não fornece uma proteção efetiva contra acesso não autorizado a um recurso, pois não são resistentes a ataques de força bruta e dicionário. Uma senha fraca é aquela na qual se utilizam textos de fácil associação com o dono da senha, ou que seja muito simples ou pequenas, tais como: datas de nascimento, o próprio nome, o nome de familiares, seqüências numéricas simples, palavras com significado. Contudo, o uso de uma senha forte, que possui pelo menos seis caracteres, não contém parte do nome do usuário e da conta do usuário e contem pelo menos três das quatro categorias seguintes: maiúsculas, minúsculas, números e caracteres especiais (*e.g.* #, @), pode aumentar a resistência a ataques de forca bruta e dicionários.

O esquecimento da senha impossibilita o usuário de acessar o sistema, pois não é possível recuperá-la a partir do servidor devido ao seu armazenamento criptografado. Porém, o usuário poderá cadastrar um lembrete que poderá ajudá-lo a recordar a sua senha em caso de esquecimento. Isto é uma limitação importantate do sistema! Pretendemos implementar um mecanismo mais robusto de autenticação no futuro.

3.4.6.2 Confidencialidade e Integridade

Um esquema de identificador de *login* e senha pode fornecer um serviço básico de autenticação, mas uma vez o usuário autenticado, não garante por si só a confidencialidade e integridade dos dados. As comunicações entre agente e servidor podem ser interceptadas e os dados copiados ou alterados. Quando o meio utilizado para transportar informação é um meio compartilhado como a Internet, mensagens trocadas devem ser encriptadas e assinadas digitalmente para assegurar a confidencialidade e integridade dos dados. Grande parte destes requisitos de segurança pode ser implementada a partir do protocolo SSL [FKK], cujo objetivo é prover um canal seguro, isto é, com garantia de privacidade, de autenticidade dos pares e de integridade da mensagem.

Ao estabelecer a conexão, o SSL estabelece um identificador de sessão e um conjunto de algoritmos criptográficos. Os algoritmos são usados para:

- Troca de chaves entre as partes envolvidas;
- Cifragem de dados;
- Inserção de redundância nas mensagens.

O algoritmo para troca de chaves será um algoritmo de criptografia de chave pública que será utilizado para enviar uma chave privada do algoritmo de cifragem de dados. Assim, o SSL utiliza-se de um algoritmo assimétrico apenas para criar um canal seguro para enviar uma chave secreta, a ser criada de forma aleatória e que será utilizada para cifrar os dados utilizando-se de um algoritmo simétrico. O algoritmo simétrico é utilizado para efetivamente cifrar os dados da camada de aplicação. Por fim, o algoritmo de inserção de redundância é utilizado para garantir a integridade da mensagem.

Assim, cada usuário possui um par de chaves, pública e privada utilizado no funcionamento do SSL. A chave pública é disponibilizada a qualquer pessoa disposta a corresponder-se com o proprietário do par de chaves. A chave pública pode ser usada para ler uma mensagem assinada com a chave privada ou para criptografar mensagens que só podem ser decriptografadas com a chave privada. A segurança das mensagens criptografadas assimetricamente depende da segurança da chave privada, que deve estar protegida contra uso não autorizado.

O sistema mantém a confidencialidade e a integridade dos backups perante amigos maliciosos. Todas as cópias de usuário são armazenadas nos amigos somente após passarem por um processo de cifragem de dados. Tanto o conteúdo quanto o nome (*i.e.* caminho absoluto) do arquivo são cifrados de modo que não seja possível obter alguma informação útil sobre os dados, garantindo, assim, a confidencialidade. Antes da transferência das cópias para os amigos, o sistema calcula o *checksum* do conteúdo do arquivo original e da cópia cifrada. Estas informações são utilizadas para verificar, eventualmente, se a cópia foi modificada, assegurando, assim, a integridade do backup.

O processo de criptografia usado na cifragem dos dados é simétrico. Por realizar operações matemáticas relativamente simples, o tempo computacional gasto pelos algoritmos de chave simétrica é menor que o tempo gasto por algoritmos de criptografia assimétricos [MVO96]. O processo de criptografia simétrica fundamenta-se na utilização de uma chave secreta pré-definida que serve tanto para cifrar como para decifrar um dado. A qualidade da encriptação está diretamente ligada ao tamanho da chave usada, porque quanto maior o tamanho da chave, maior o número de chaves possíveis, o que dificulta a quebra da criptografia por força bruta. Como a chave privada usada no SSL é grande o suficiente para dificultar o ataque por força bruta e é mantida em segredo, optamos por usá-la como chave secreta no processo de encriptação dos dados.

Os backups devem ser passíveis de recuperação pelo seu dono, inclusive em casos de *crash* da máquina de origem. Desta forma, os segredos usados nos mecanismos de confidencialidade são armazenados remotamente no Servidor com o objetivo de garantir a recuperabilidade do backup.

Mesmo sendo o mantenedor das informações dos usuários, o servidor não deve ser capaz de ler o conteúdo dos backups. Assim como a senha, o armazenamento do par de chaves de um usuário requer um mínimo de segurança criptográfica, para que não haja risco de serem lidas ou obtidas diretamente do local onde são armazenadas. Assim como a senha do usuário, a chave privada deve ser primeiramente cifrada antes de ser armazenada no servidor.

Como a chave privada é utilizada como segredo no processo de cifragem dos backups, ela não pode ser criptografada utilizando um algoritmo irreversível de cifragem (*e.g. hash*), pois deve ser passível de recuperação pelo usuário para permitir a recuperação de backups em caso de comprometimento do computador do usuário. Devendo, assim, ser criptografada simetricamente.

Naturalmente, o segredo usado no processo de cifragem da chave privada deve ser conhecido somente pelo usuário de forma a garantir que até o administrador do OurBackup seja incapaz de quebrar a confidencialidade dos backups. Como apenas o usuário possui conhecimento da sua senha, optamos, então, por usá-la como chave secreta na encriptação da sua chave privada.

A Tabela II sumariza os elementos usados na garantia da confidencialidade dos backups e visualiza onde são armazenados.

TABELA II – Armazenamento das chaves, do identificador de *login* e da senha de um usuário.

	Memória do Usuário	**Agente**	**Servidor**
Identificador de *login*	X	X	X
Senha	X		
Hash da senha			X
Chave publica		X	X
Chave privada		X	
Chave privada cifrada simetricamente usando a senha		X	X

Os metadados armazenados pelo servidor também devem ter sua confidencialidade e integridade asseguradas perante a participação de usuários maliciosos. Para isso, o sistema restringe o acesso às informações sejam feitos somente por pessoas autorizadas, permitindo protegê-las contra acessos indesejáveis. Uma autorização estabelece as permissões (direitos ou privilégios) de acesso que um usuário pode exercer. O controle de acesso limita as ações ou operações que um usuário legítimo pode realizar, com base nas autorizações aplicáveis ao mesmo no momento do acesso.

3.5 Arquitetura do OurBackup

As metas principais que guiaram o processo de desenvolvimento do OurBackup foram a reusabilidade e manutenibilidade do código. As idéias e os algoritmos nos quais o OurBackup está fundamentado foram implementadas de tal forma que houvesse um consumo eficiente de recursos de processamento e de memória. Além disso, o OurBackup, como todo sistema de *software*, necessita ser manutenível de modo que o processo de adicionar novas funcionalidade, assim como o processo de melhorar as funcionalidades nele existentes, pode ser efetuado facilmente. E finalmente, o OurBackup foi desenvolvido de modo que tornasse mais fácil o reuso de alguns dos componentes por outras aplicações.

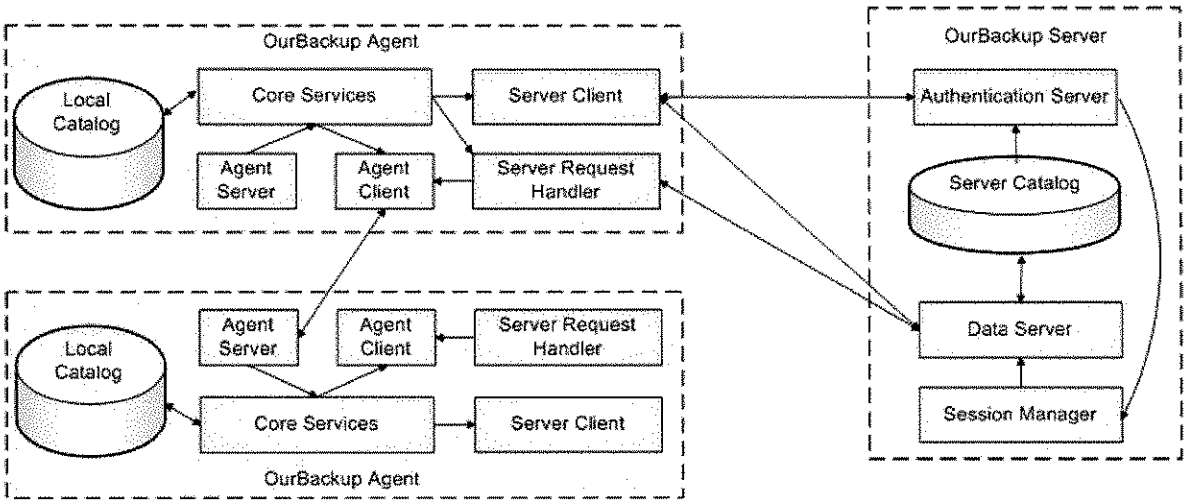


Figura 20 – Os principais componentes do OurBackup e suas interações

De acordo com as metas de reusabilidade e manutenibilidade, encapsulamos as funcionalidades e regras de negócios do sistema em módulos com responsabilidades bem definidas. Estes módulos são usados para compor os elementos da arquitetura OurBackup (ver Figura 20) e são organizados em camadas de *software* distintas. A escolha da divisão em camadas propicia ao sistema modularidade, permitindo fazer alterações em uma camada sem afetar as demais. São identificadas quatro camadas independentes, interligadas por interfaces bem definidas:

- Camada de apresentação;
- Camada de negócios;
- Camada de comunicação;
- Camada de metadados.

A camada de apresentação é composta por uma interface gráfica, promove a interação do usuário com o sistema. A interface de entrada e saída é implementada através de uma GUI (*Graphical User Interface*), na qual o usuário se comunica com o sistema através de uma interface gráfica. Toda a interação do usuário com GUI é traduzida em chamadas

apropriadas na camada de negócios, que provê todos os serviços capazes de tratar e executar todas as requisições submetidas de acordo com o desejo do usuário via GUI.

A camada de negócios é responsável pela implementação de todos os serviços e da lógica de negócio, permitindo a independência entre o sistema e a camada de metadados. Qualquer alteração sobre essa camada, não influi sobre as demais camadas do sistema, fornecendo uma abstração sobre como a camada de dados está sendo implementada. No agente, esta camada é composta pelos módulos pertencentes ao *CoreServices*. Por sua vez, no servidor, esta camada é composta pelo módulo *AuthenticationServer*.

A camada de comunicação utiliza os recursos de rede, promovendo a interação entre Agentes e a interação entre Agente e Servidor através de protocolos de comunicação. Os módulos que pertencem a esta camada são *AgentServer, AgentClient e ServerClient*.

A camada de metadados permite o armazenamento e a recuperação dos dados, sendo responsável pela lógica de metadados do sistema. É utilizado um sistema de catálogo local de dados, no qual a manipulação sobre os metadados é realizada através do uso de pacotes pré-definidos da linguagem Java. O módulo pertencente a esta camada é o *MetaDataServer*. Todo o acesso a dados será realizado através do padrão de projeto *Data Access Object* (DAO) [JH04]. O DAO é responsável por separar a camada de persistência de dados da camada de negócios. O DAO permite que o sistema possa trabalhar com diferentes fontes de dados. Para isso, basta que exista uma implementação específica para cada uma delas.

3.5.1 GUI

O objetivo da GUI é expor a lógica de negócios ao usuário e possibilitar a interação do usuário com a aplicação. A GUI possui mecanismos de interação entre usuário e o OurBackup baseados em controles visuais, como ícones, menus e janelas. Ao invés de executar ações através de linha de comando, o usuário desenvolve as tarefas desejadas usando-se de um mouse para escolher entre um conjunto de opções apresentadas na tela. A GUI é organizada em Abas (*i.e. Friends, Files, Backups* e *Status*), cada uma com um conjunto de funcionalidades específica.

A aba *Friends* (ver Figura 21) permite ao usuário construir sua rede social, adicionando, removendo e procurando amigos. Quando o usuário se autentica no sistema, o agente se conecta no servidor, registra o usuário como on-line, e visualiza na tela a lista de amigos do usuário. O agente periodicamente atualiza a lista de amigos consultando o servidor sobre novos dados. Existem cinco possíveis estados de disponibilidade para as pessoas que estão na lista de amigos:

- *Online*: O amigo está conectado no sistema;
- *Offline*: O amigo não está conectado no sistema;
- *Pending*: O usuário enviou um pedido de amizade para as pessoas pertencentes a este grupo e elas ainda não aceitaram, nem rejeitaram o pedido. O usuário está aguardando autorização da pessoa para tê-la na sua lista de amigos, enquanto ela não é concedida, não é possível estabelecer contratos de backup;
- *Rejected*: As pessoas pertencentes a este grupo rejeitaram uma relação de amizade com o usuário. Uma rejeição de amizade pode ocorrer devido à negação de um pedido de amizade, à remoção da relação de amizade ou abandono do sistema por parte do amigo;
- *Incoming*: O usuário recebeu um pedido de amizade para as pessoas pertencentes a este grupo. Cabe a ele aceitar ou rejeitar os pedidos de amizade. Caso aceite o pedido de amizade, os amigos serão classificados como *online* ou *offline* se estiverem ou não conectados no sistema.

Figura 21 – Aba Friends

Quando o usuário iniciar o OurBackup pela primeira vez, sua lista de amigos estará vazia. Para procurar amigos para que sejam adicionados à sua lista de amigos, o usuário deve selecionar a opção "*Find Friends*" no menu *Friends*. Uma nova janela de pesquisa será aberta e você poderá procurar amigos pelo nome ou qualquer outra informação que tenham incluído em seus perfis pessoais. Para adicionar um amigo, basta clicar com o botão direito do mouse no nome que aparecer nos resultados da pesquisa e clicar no botão "*Add as friend*".

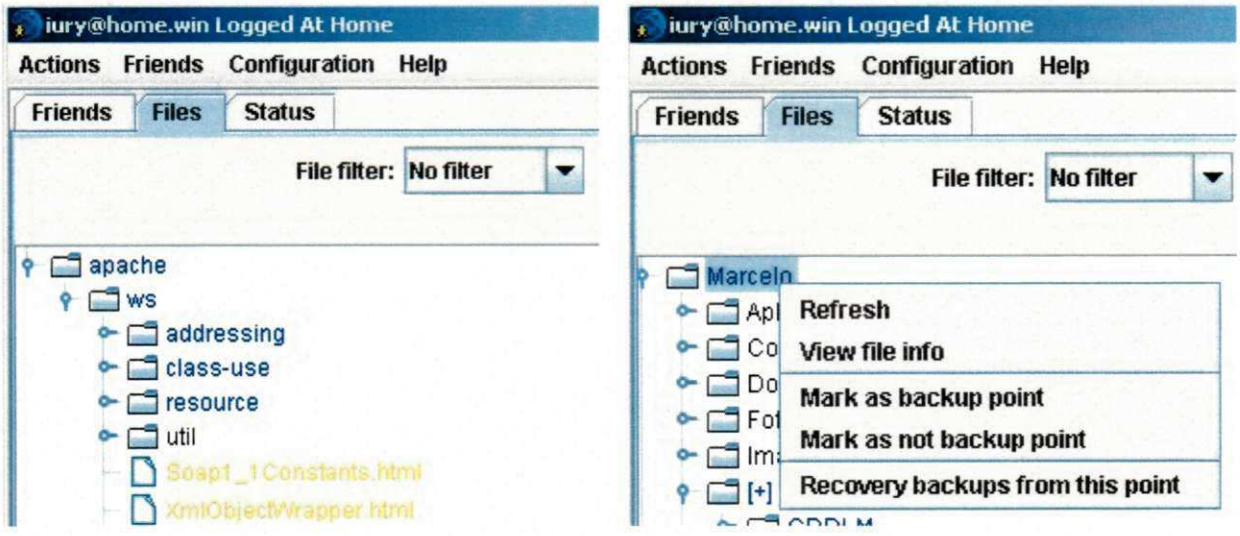


Figura 22 – Aba Files

A aba *Files* (ver Figura 22) permite ao usuário efetuar e recuperar seus backups, sendo somente acessível quando o usuário está utilizando o OurBackup na sua *home machine*.

Esta aba usa uma interface similar ao Windows Explorer que permite que usuário faça uma varredura no sistema de arquivos do computador e selecione os dados que deseja recuperar ou efetuar backup.

O usuário pode fazer backup de arquivos específicos, de diretórios inteiros ou de sistemas de arquivos inteiros a partir da árvore de diretórios, clicando com o botão direito do *mouse* no arquivo ou diretório de que deseja fazer o backup e, em seguida, clicando na opção "*Mark as backup point*" no menu. Quando o usuário marca um diretório para backup, ele está especificando um ponto de inclusão de backup que indica que todos os arquivos (atuais e futuros) abaixo do ponto de inclusão devem ser mantidos pelo sistema.

Pode haver arquivos abaixo de algum ponto de inclusão que não se deseja fazer backup. Esses arquivos podem ser arquivos temporários, *caches* locais dos sistemas de arquivo da rede, arquivos de aplicativos ou sistemas operacionais, que podem ser facilmente substituídos pela reinstalação do programa, ou qualquer outro arquivo que poderia ser reconstruído. O usuário pode especificar arquivos/diretórios para serem excluídos do processo de backup, clicando com o botão direito do *mouse* no arquivo ou diretório de que deseja fazer o backup e, em seguida, clicando na opção "*Mark as not backup point*" no menu. Quando o usuário exclui um arquivo/diretório do processo backup, ele está especificando um ponto de exclusão que indica que não deve ser mantido backup para os arquivos abaixo deste ponto. Assim, um ponto de exclusão só é útil quando se trata de um subdiretório de um ponto de inclusão.

O usuário pode recuperar backups, clicando com o botão direito do *mouse* no arquivo ou diretório de que deseja recuperar e, em seguida, clicando na opção "*Recovery backup from this point*" no menu. Quando o usuário seleciona um diretório, ele está especificando que todo o conteúdo do diretório (*i.e.* arquivos e subdiretórios) deve ser recuperado.

O status do processo de backup pode ser visualizado através de um esquema de coloração que indica quais arquivos foram marcados para backup e quais arquivos ainda necessitam da criação de cópias. Este esquema é aplicado na estrutura de árvore que visualiza o sistema de arquivo, de tal forma, que cada cor representa um status de backup diferente. Sendo elas:

- Preto: O arquivo não está marcado para backup;
- Azul: O arquivo está marcado pra backup e já possui cópias criadas;
- Laranja: O arquivo está marcado pra backup, mas nenhuma cópia foi criada;
- Vermelho: O arquivo não existe mais no sistema de arquivos, mas foi marcado para backup em momento anterior e ainda possui cópias mantidas pelo sistema.

O usuário ainda possui a opção de solicitar um relatório sobre o status de backup de um arquivo ou diretório especifico, clicando com o botão direito do *mouse* no item de interesse e, em seguida, clicando na opção "*View file info*" no menu. Caso se peça o status de um arquivo, mostra-se os detalhes deste arquivo. Caso se peça o status de um diretório, mostra-se o sumário do status sobre o diretório, *i.e.*, total de arquivos existentes, quantos destes arquivos estão marcados pra backup e quantos arquivos já possuem cópias.

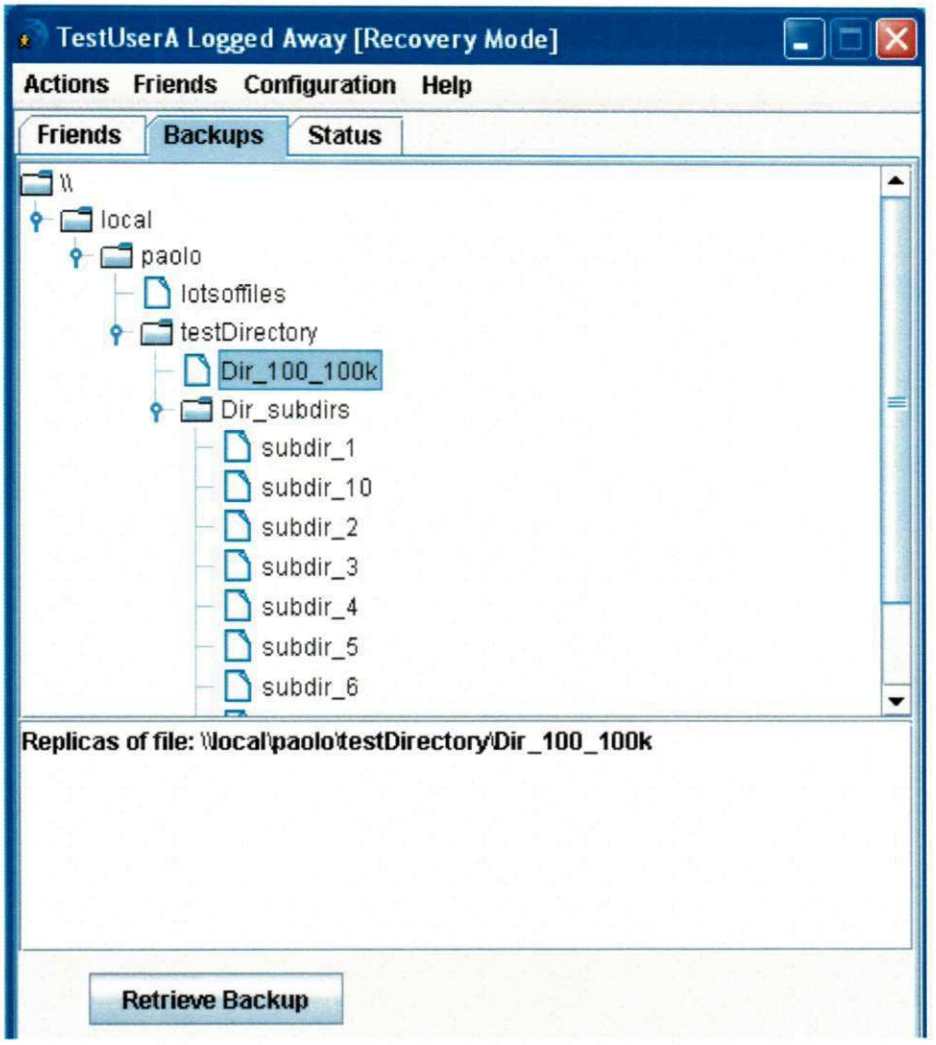


Figura 23 – Aba Backups

A aba *Backups* (ver Figura 23) permite ao usuário recuperar seus backups quando estiver utilizando o OurBackup fora de sua *home machine*. Esta aba usa uma interface similar ao Windows Explorer que permite que usuário faça uma varredura na árvore de arquivos que marcou para backup e selecione aqueles que deseja recuperar.

O usuário pode recuperar backups de arquivos específicos, de diretórios inteiros ou de sistemas de arquivos inteiros a partir da árvore de diretórios, selecionado com o botão esquerdo do *mouse* no arquivo ou diretório de que deseja recuperar o backup e, em seguida, clicando na botão "*Retrieve Backup*" no menu. Quando o usuário marca um diretório para

recuperação, ele está especificando que todos os arquivos abaixo do ponto especificados devem ser recuperados pelo sistema.

O usuário pode, ainda, visualizar um relatório sobre o status de backup de um arquivo especifico, clicando com o botão esquerdo do *mouse* no item de interesse e, em seguida, visualizando o relatório no painel abaixo da árvore de backup.

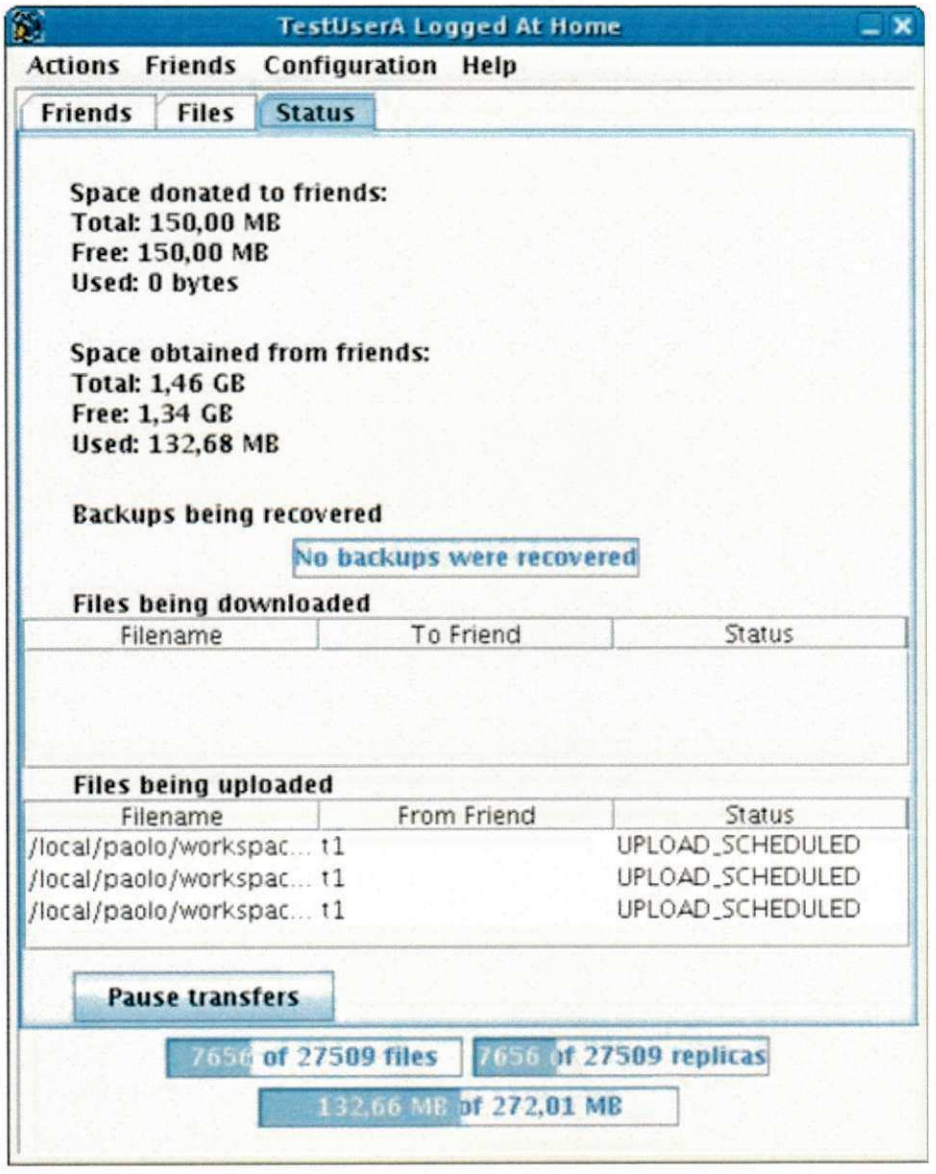


Figura 24 – Aba Status

A aba Status (ver Figura 23) permite ao usuário verificar o status do sistema através de controles visuais. As barras de progresso, na parte inferior da aba, visualizam como o processo de backup está a evoluir, em termos de percentagem de backup já tratada: a barra *files* indica quantos arquivos ainda necessitam de cópias, a barra *replicas* indica quantas cópias ainda precisam ser criadas e a barra de espaço indica a quantidade de dados que ainda precisa ser transferida. A barra *replicas* apresentará uma contagem diferente da barra *files,* quando o usuário especificar um limiar de cópias maior que 1 para todos os arquivos, sendo esta configuração de backup é acessível através do menu *Preferences* (ver Figuras 25 e 26).

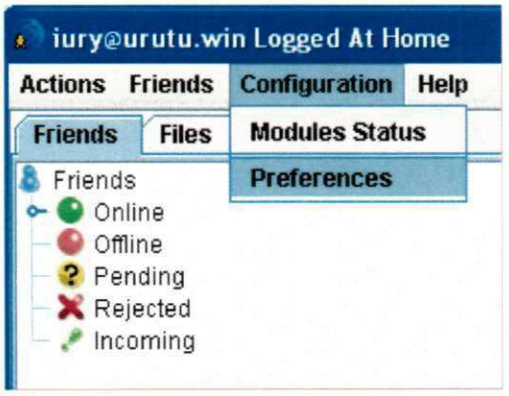


Figura 25 – Menu Preferences

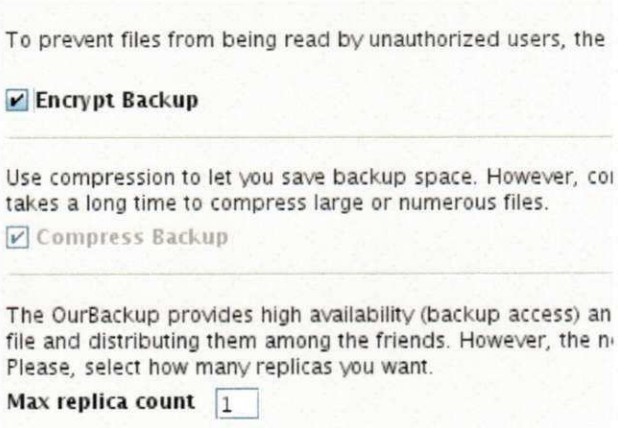


Figura 26 – Configurações de backup

Esta aba ainda apresenta a quantidade de espaço de armazenamento obtida e doada para backup e os arquivos que estão escalonados para transferência e os que já estão sendo transferidos.

3.5.1.1 Design da GUI

Usamos o padrão de arquitetura *Model View Controller* (MVC) [JH04] na implementação da GUI. O MVC divide os elementos da camada de apresentação em três tipos: o *controller* (*OurBackupGUIController*), encarregado de receber a entrada do usuário, acessar a camada de negócios para manipular o modelo e selecionar a visão; o *model* (*OurBackupGUIModel*), um objeto representando uma parte do domínio e que provê os dados que a visão vai exibir. É o contrato entre o *controller* e a *view*; a (*OurBackupGUI*), encarregada de exibir os dados do modelo para o usuário.

A *view* mapeia requisições em objetos que implementam o padrão de projeto comando (*Command*) [JH04]. Os comandos são usados para submeter requisições ao *controller*. Ao receber uma requisição, o *controller* assume o controle da aplicação, cabendo a ele extrair os parâmetros da requisição, que são do tipo String, convertê-los para tipos mais específicos da aplicação, que podem ser tipos primitivos mais restritos como inteiros e booleanos ou objetos usados pelo *model*, e validá-los. Em caso de erro de validação, o *controller* notifica a *view* sobre o erro. Quando não ocorre erro de validação, o *controller* encaminha a requisição para o *model*. Terminada a operação, o comando sinaliza para o controlador, através do valor de retorno, por exemplo, qual visão deve ser exibida e este finaliza a requisição mostrando esta visão.

3.5.2 CoreServices

CoreServices, como o próprio nome diz, são os módulos que formam o núcleo do OurBackup (ver Figura 27). Estes módulos ainda podem ser classificados quanto a sua responsabilidade. Cada evento disparado pela GUI, será tratado pelo módulo *OurBackupGUIModel*, que é uma fachada para os serviços providos pelo Agente.

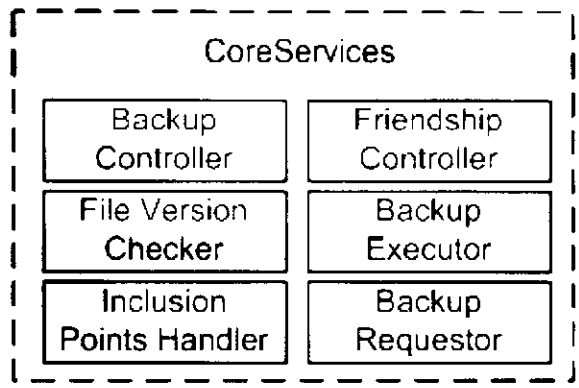


Figura 27 – Módulos que compõem o CoreServices

Os módulos de Controle de Backup são responsáveis pelo provimento das funcionalidades de manutenção de backup (ver seção 3.4). As operações de criação, remoção e recuperação de backups são acionadas de acordo com o desejo do usuário via GUI. Já a atualização de backups é realizada de forma completamente automática pelo sistema. As interações de cada um destes módulos entre si podem ser visualizados na Figura 28.

O controle de backup ocorre através de uma relação produtor-consumidor entre os módulos através do repositório local de dados. Os módulos produtores (*i.e. FileVersionChecker* e *InclusionPointsHandler*) devem inserir modificações no catálogo local e o módulo consumidor (*i.e. BackupController*) deve verificar estas modificações procurando inconsistências a serem tratadas. Cada um destes módulos será detalhado nas seções seguintes deste capitulo.

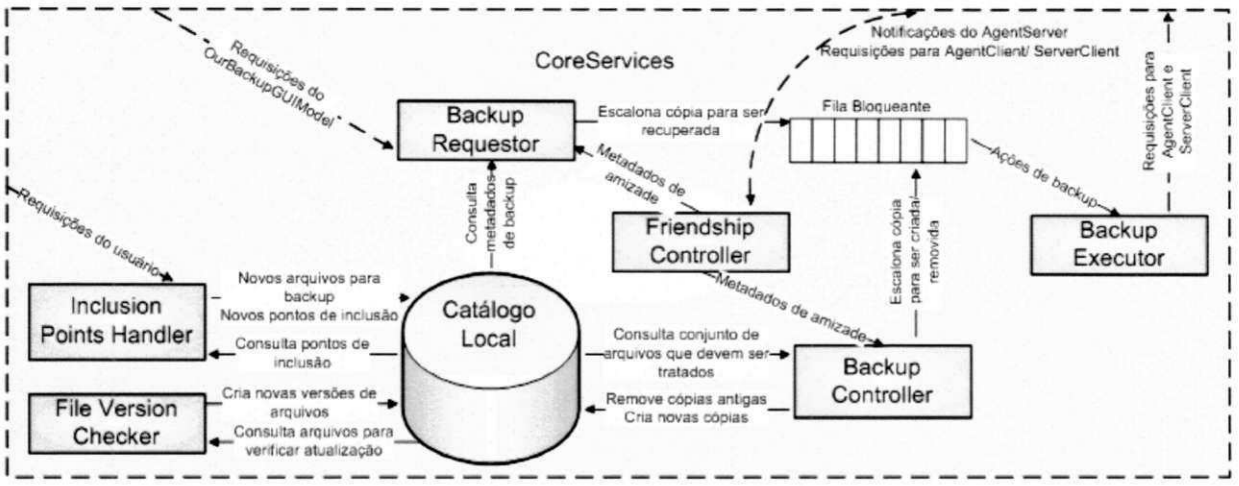


Figura 28 – Interações entre os módulos CoreServices

3.5.2.1 InclusionPointsHandler

Responsável por tratar os pontos de inclusão e exclusão de backup. Um ponto de inclusão é um diretório ou arquivo que se deseja fazer backup. Um ponto de exclusão é um diretório ou arquivo que não se deseja fazer backup. Pode haver pontos de inclusão dentro de pontos de exclusão, bem como pontos de exclusão dentro de pontos de inclusão, em qualquer nível de profundidade.

O algoritmo de tratamento de pontos de inclusão e exclusão recebe como entrada dois conjuntos, *E* e *I*, onde *E* é um conjunto de pontos de exclusão de diretórios para backup e *I* é um conjunto de pontos de inclusão de diretórios para backup. Os dois conjuntos são ordenados lexicograficamente pelo caminhos absolutos de diretórios. O algoritmo de tratamento dos pontos de inclusão e exclusão procede como abaixo:

```
tratarPontosDeInclusão( I, E )
    Para cada ponto de inclusão i em I
        Remova i de I
        Se i está contido em E
            Remova i de E
            Adicione todos os filhos de i em E
```

```
            Marque os filhos de i para exclusão
        Caso contrário
            Se i é um arquivo
                Se já existe uma versão do arquivo
                    Se a versão está marcada para exclusão
                        Retire a ponto de exclusão
                Case contrário
                    Adicione o arquivo no conjunto de arquivos para backup
            Se i é um diretório
                Adicione todos os filhos de i em I
```

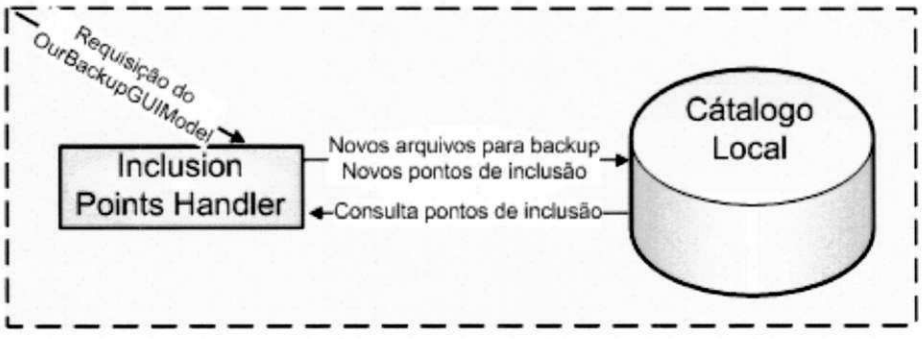


Figura 29 – Interações do InclusionPointsHandler

O *InclusionPointsHandler* recebe requisições de inserção de pontos de inclusão e exclusão do *OurBackupGUIModel*, como pode ser visualizado através da Figura 29. Sendo os pontos de inclusão e exclusão definidos pelo usuário através da aba Files na GUI (ver seção 3.6). É responsabilidade do *InclusionPointsHandler* salvar os pontos de inclusão/exclusão no catálogo local, bem como inserir os metadados dos arquivos que devem ser efetuado backup (*i.e.* marcar arquivos para backup).

3.5.2.2 FileVersionChecker

Este módulo é responsável por verificar se ocorreu alguma modificação em algum arquivo que já foi marcado e efetuado backup. Quando o *FileVersionChecker* percebe que um

arquivo foi modificado, ele insere uma nova versão dos metadados do arquivo no catálogo local, e marca a versão anterior como *deprecated*, como pode ser visualizado pela Figura 30. O processo de criação das cópias referentes às novas versões de arquivos é tratada pelo módulo *BackupController*.

O módulo *FileVersionChecker* ainda é responsável por detectar se um arquivo original deixou de existir fisicamente no sistema de arquivos. Como arquivos podem eventualmente serem excluídos por acidente, não ocorre a remoção imediata do backup. Para estes casos, é concedido um "período de graça" durante o qual as cópias do arquivo original "fantasma" ainda são mantidas pelo sistema. Quando expirado o período de graça o *FileVersionChecker* marca os arquivos originais para remoção.

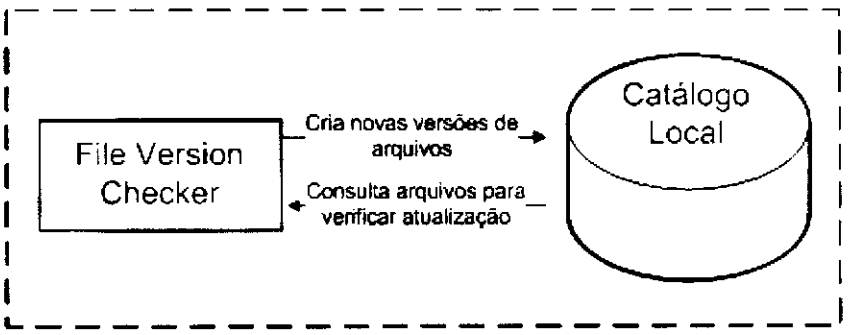


Figura 30 – Interações do FileVersionChecker

3.5.2.3 BackupController

Este é o principal módulo de controle de backups, responsável pelo escalonamento de novas cópias e remoção de cópias antigas, como pode ser visualizado pela Figura 31. Sua tarefa é resolver inconsistências do backup, isto é, verificar no catálogo local quais arquivos não estão com backup adequado e atuar quando necessário. Estas inconsistências são geradas pelo *FileVersionChecker*, ao inserir dados de uma nova versão de arquivo ou marcar um arquivo com *purged*, e pelo *InclusionPointsHandler*, ao marcar um novo arquivo para backup. As inconsistências de backup são percebidas exclusivamente através do catálogo local, contudo, quando o *InclusionPointsHandler* e o *FileVersionChecker* modificam o catálogo local *BackupController* é notificado.

As inconsistências refletem em cópias a serem transferidas e/ou cópias a serem removidas. O primeiro caso ocorre quando uma versão mais atualizada do arquivo original necessita de cópias para atingir o nível de confiabilidade configurado pelo usuário. O segundo caso ocorre ou quando um backup for encerrado pelo usuário ou quando uma cópia de uma versão antiga for substituída por uma mais recente. Quando for necessário criar novas cópias para atender a uma nova versão de um arquivo original, o *BackupController* deve selecionar quais amigos serão hospedeiros destas novas cópias. Esta seleção é determinada pelo estado de presença do amigo (*i.e.* o amigo deve está conectado no sistema) e pela disponibilidade de espaço de armazenamento suficiente para receber as novas cópias.

O *BackupController* não remove ou transfere diretamente as cópias. Ele requisita ao *BackupExecutor* a manipulação destas cópias através de uma fila compartilhada. Assim o *BackupController* pode inserir requisições na fila, enquanto o *BackupExecutor* consome requisições.

Esta fila de requisição é bloqueante e possui uma capacidade de *slots* limitada pela quantidade de requisições que o *BackupExecutor* consegue atender por vez. Quando o *BackupController* desejar adicionar um elemento à fila, e esta já estiver totalmente preenchida, ele fica bloqueado aguardando espaço disponível na fila. Similarmente, se o *BackupExecutor* quiser retirar um item da fila e a encontrar vazia, deve aguardar até que o *BackupController* enfileire alguma nova requisição.

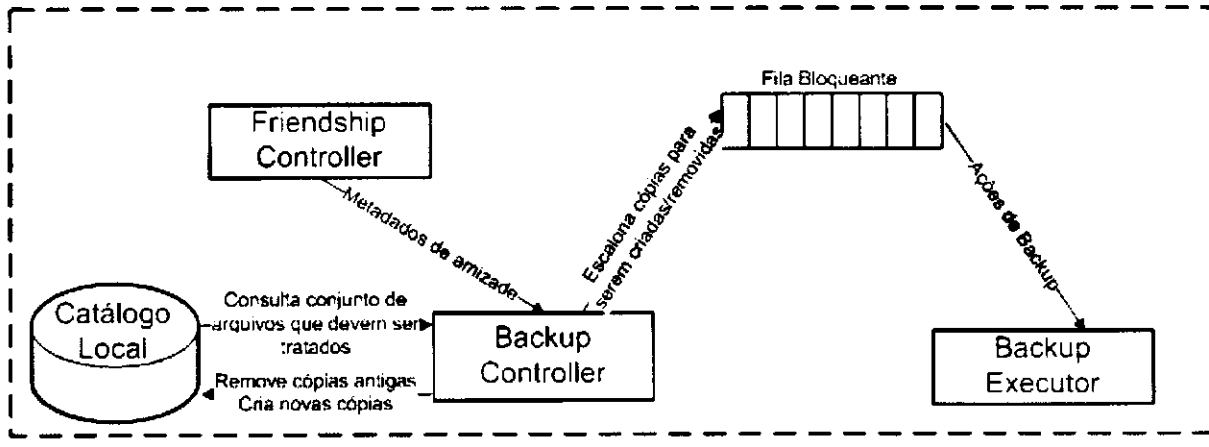


Figura 31 – Interações do BackupController

3.5.2.4 BackupRequestor

Responsável pela recuperação de backups. Obtém os metadados dos arquivos a serem recuperados do catálogo local. As informações sobres os amigos que possuem a cópia é obtida através do *FriendshipController*. As requisições de recuperação são recebidas do *OurbackupGUIModel*, tratados e enfileiradas em uma fila produtor-consumidor bloqueante, para posterior execução por parte do *BackupExecutor*. Caso um arquivo original possua mais de uma cópia disponível:

- O *BackupRequestor* escolhe sempre a cópia mais recente para ser recuperada;
- Caso existam mais de uma cópia recente, o *BackupRequestor* escolhe aleatoriamente a cópia que está armazenada no amigo;
- Caso a cópia mais recente não esteja imediatamente disponível, ele escolherá a segunda cópia mais recente que esteja disponível;
- Caso não tenha nenhuma versão das cópias imediatamente disponíveis, o BackupRequestor registra este arquivo original para tentar ser recuperado em outro momento.

A interação do *BackupRequestor* com a fila bloqueante e o catálogo local pode ser visualizada através da Figura 32.

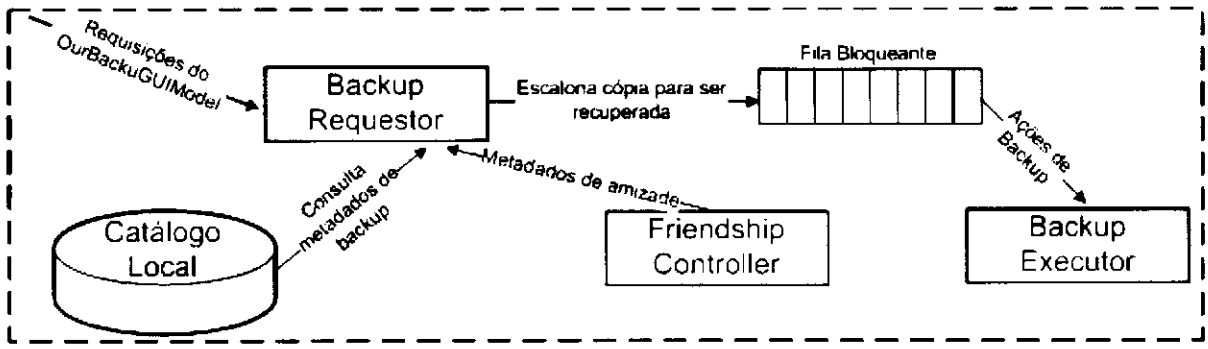


Figura 32 – Interações do BackupRequestor

3.5.2.5 BackupExecutor

Responsável pela execução de todas as requisições enfileiradas pelo *BackupController* e *BackupRequestor*, ou seja, é o responsável pela recuperação, envio e remoção efetiva de réplicas. O *BackupExecutor* também é responsável por registrar os metadados de backup no catálogo de metadados local e servidor. As requisições são consumidas de uma fila e consistem, basicamente, de operações de realização, remoção ou atualização de cópias. Esta interação do *BackupExecutor* com a fila bloqueante e o catálogo local pode ser visualizada através da Figura 33.

O *BackupExecutor* possui um *pool* de *threads* que consome da fila as operações a serem realizadas e as executa. O *pool* gerencia o ciclo de vida dessas entidades, mantendo o controle do número (configurável) de operações que podem ser executadas simultaneamente.

O consumo da fila, a conseqüente transferência de arquivos, quando necessária, e a atualização dos metadados do catálogo local e do servidor compõem a última fase do processo de backup. Uma vez que o processo de transferência de arquivos é demorado, considerando grandes volumes de dados, é possível que uma requisição permaneça na fila sem ser atendida por um período de tempo suficiente para que o arquivo referenciado seja modificado ou excluído do sistema de arquivos, de forma que não é garantida a existência do arquivo referenciados pela requisição no momento da execução da transferência. Caso os arquivos não sejam encontrados, a operação é "silenciosamente" cancelada e a notificação de erro é registrada no *log* do sistema. Se os arquivos constarem no sistema de arquivos, o *BackupExecutor* efetua a transferência e, após a obtenção de êxito na conclusão do processo, atualiza a os metadados do catálogo do servidor e, logo em seguida, os metadados do catálogo local. A ordem de atualização dos metadados é importante, uma vez que dois catálogos distintos devem ser atualizados e o processo de atualização não é atômico. Atribuir prioridade de atualização ao servidor central garante a persistência da informação referente à conclusão da operação requisitada, na possibilidade de ocorrência de falha após uma das atualizações.

É possível que falhas ocorram durante o processo de transferência. Nesse caso, o *BackupExecutor* realiza um número de tentativas de repetição do processo. Se em algumas das tentativas a transferência completar, o processo continua normalmente.

No caso específico de o servidor falhar (*i.e.* não ser possível contactar o servidor) durante um processo de transferência de arquivo, o *BackupExecutor* cancela todas os processos de transferência que estejam em execução, notifica o *OurBackupGUIModel* sobre a falha do servidor e entra em modo de espera.

Uma vez que o *OurBackupGUIModel* tenha sido notificado sobre uma falha de comunicação com o servidor, ele inicia o processo de restauração de conexão, primeiramente, migrando o agente para um estado de espera. Este processo de migração consiste, na verdade, da interrupção temporária do funcionamento de todos os módulos do sistema.

Quando todos os módulos estiverem em estado de espera, o *OurBackupGUIModel* inicia um ciclo de tentativas, que periodicamente tenta restaurar a conexão com servidor. Uma vez uma destas tentativas tenha êxito, o *OurBackupGUIModel* reinicia os módulos do sistema para que procedam ao seu funcionamento normal.

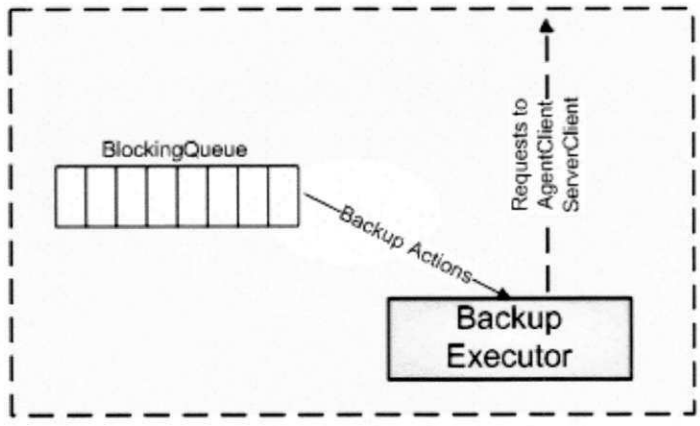


Figura 33 – Interações do BackupExecutor

3.5.2.6 FriendshipController

O *FriendshipController* mantém um micro-catálogo que armazena as informações da rede social do usuário. Estas informações consistem do conjunto de relações de amizades já estabelecidas, relações de amizades rejeitadas e pedidos de amizades requisitados e recebidos pelo usuário, sendo obtidas diretamente do servidor e atualizadas através de notificações enviadas pelos amigos. Este micro-catálogo é *stateless*, no sentido de que não há conservação das informações entre sessões de autenticação do usuário, sendo sempre necessário obtê-las novamente do servidor.

O *FriendshipController* é responsável por prover mecanismos de manipulação da rede social do usuário, suportando operações de consulta de amigos e de requisição, confirmação, negação e remoção de amizades. Todas as consultas relativas a amizades são atendidas usando as informações mantidas pelo micro-catálogo. Diferentemente das consultas, as operações que alteram a rede social do usuário são propagadas todas para o servidor através do módulo *ServerClient*.

O *FriendshipController* ainda é responsável pelo mecanismo da presença (ver seção 3.4.5), fornecendo meios para notificar e ser notificado sobre mudanças de presença dos amigos. As notificações dos amigos são recebidas através do *AgentServer* e são transmitidas através do *AgentClient*. As informações de presença dos amigos também são mantidas no micro-catálogo, permitindo a outros módulos consultar quais amigos estão conectados ou não no sistema.

O FriendshipController ainda interage com os módulos *BackupRequestor* e *BackupController*, fornecendo para eles informações sobre a rede social do usuário, como pode ser visualizado na Figura 34.

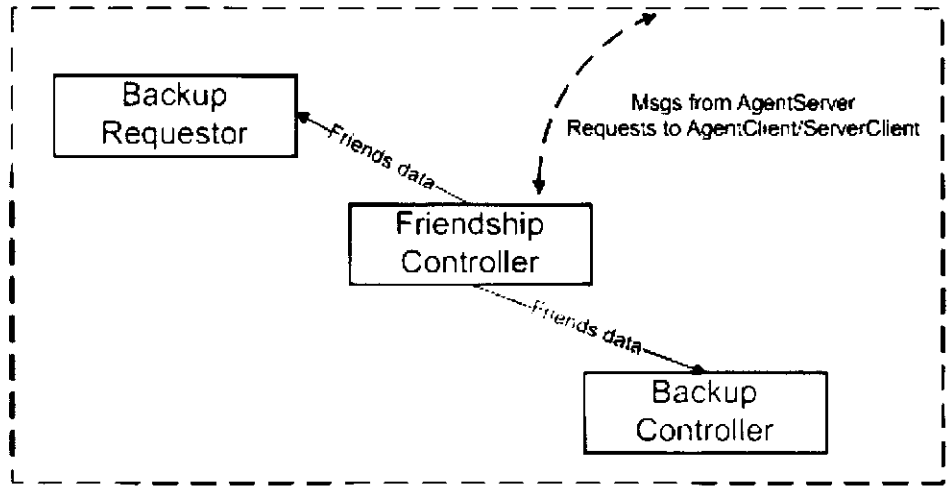


Figura 34 – Interações do FriendshipController

3.5.3 Módulos de comunicação

Os módulos de comunicação interagem através de uma relação Cliente-Servidor. A entidade cliente (*i.e. AgentClient* e *ServerClient*) realiza chamadas à entidade servidor (*i.e. AgentServer*, *MetaDataServer* e *AuthenticationServer*). Estas chamadas são, na realidade, direcionadas para um objeto *proxy* local, que tem a mesma interface do servidor original. O *proxy* identifica qual forma de comunicação deve ser usada e cuida de empacotar as informações necessárias para que a requisição seja enviada ao servidor. Em seguida o *proxy* encaminha estas informações para o servidor através de uma interface padronizada, que serializa e transmite as mesmas. Na estação remota, o servidor recupera as informações necessárias e processa a requisição do cliente. O retorno de resultados e as exceções que possam surgir são transferidos de forma semelhante.

A camada de comunicação depende dos mecanismos de comunicação adotado, sendo as tecnologias de comunicação adotadas pelo sistema: *sockets* e invocação de métodos remotos (RMI).

O uso de *sockets* é a maneira mais primitiva de se implementar a comunicação entre aplicações utilizando o protocolo TCP/IP (*Transmission Control Protocol / Internet Protocol*). Um *socket* consiste num canal bidirecional dedicado, identificado através dos endereços IP e dos números das portas dos interlocutores. O RMI (*Remote Method*

Invocation) oferece uma arquitetura com objetivos similares, mas está restrito a objetos distribuídos, no nosso caso, escritos na linguagem Java. Esta tecnologia torna a tarefa de desenvolver aplicações distribuídas mais simples do que utilizando *sockets*, eliminando a necessidade de projetar um protocolo específico de colaboração entre as entidades.

Optamos por usar *sockets* na implementação das interações Agente-Agente devido a dois motivos: (1) são naturalmente apropriados para a implementação de protocolos que envolvem múltiplas trocas de mensagens que é o caso dos processos de transferência de arquivos entre agentes e (2) são mais facilmente configuráveis que RMI, facilitando ajustes com o objetivo de otimizar a transferência de dados.

Por sua vez, optamos usar RMI na implementação das interações Agente-Servidor devido ao fato de o mecanismo de serialização de objetos em Java permitir que objetos completos sejam passados como parâmetros de invocações remotas, simplificando a implementação da fachada de acesso a dados do servidor. Além disso, facilita a programação de uma aplicação distribuída na medida em que esconde do programador a maioria dos detalhes relacionados com a comunicação em ambiente distribuído.

3.5.3.1 AgentClient e AgentServer

O *AgentClient* é responsável pelo envio de mensagens e requisições a agentes remotos. A comunicação é estabelecida com o *AgentServer* do agente remoto, que é responsável pelo tratamento da mensagem enviada, como visualizado na Figura 35.

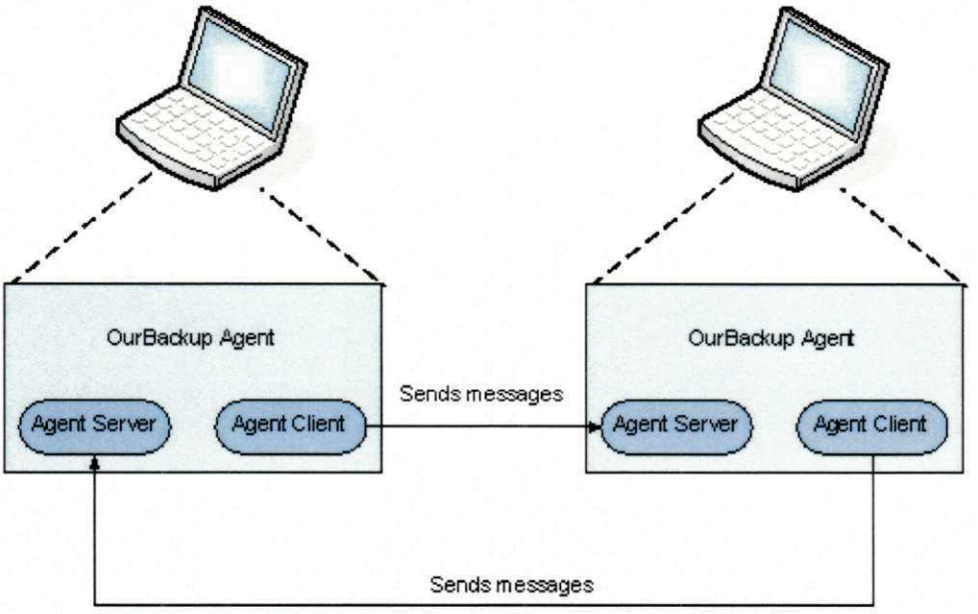


Figura 35 – Interação AgentClient X AgentServer

As mensagens trocadas entre os agentes podem ser de dois tipos:

Transferência de arquivos: O backup é realizado através da transferência (*uploading*) de um arquivo ou conjunto de arquivos para uma máquina de um amigo, enquanto a recuperação de um backup será realizada de maneira inversa (*downloading*). Ambos os processos de transferência deverão ser realizados diretamente entre os dois agentes envolvidos.

Mensagens de notificação: O agente notifica aos agentes de seus amigos sobre os eventos de entrada e saída na rede, de solicitação, aceite ou de recusa de amizade e de abandono do sistema.

O *AgentServer* é o componente responsável pelo recebimento e tratamento de mensagens provindas de outros agentes. Este é composto por um *OBAgentServer* e múltiplos *RequestProcessors*, como visualizado na Figura 36.

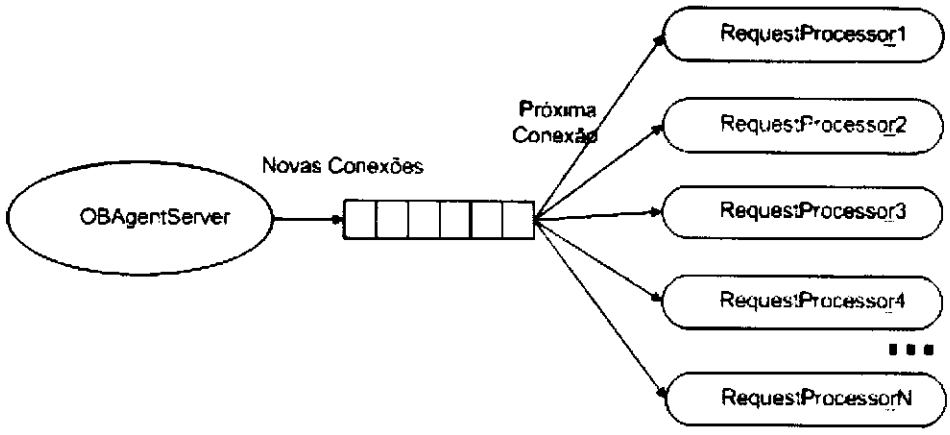


Figura 36 – Componentes do AgentServer

OBAgentServer: Consiste de uma única *Thread* que recebe as conexões e as coloca em uma fila bloqueante com política *first-come first-served.*

RequestProcessor: Consiste de uma *Thread* em que as conexões colocadas na fila pelo *OBAgentServer* serão tratadas e servidas. Algumas mensagens poderão ser resolvidas diretamente pelo *RequestProcessor*, sem que nenhum outro componente seja notificado pela mensagem, enquanto outras necessitarão que eventos sejam para propagados para outros componentes (*e.g.* notificações de aceite e de recusa de amizades devem ser propagadas ao *FriendshipController*). Um *AgentServer* possuirá múltiplos *RequestProcessors*, sendo capaz de atender a vários agentes remotos simultaneamente.

3.5.3.2 ServerClient

O *ServerClient* tem como responsabilidade ser um Proxy do *AuthenticationServer* e do *MetaDataServer*, escondendo (quando possível) o acesso a objetos remotos.

3.5.4 AuthenticationServer

O *AuthenticationServer* é um servidor central dedicado aos processos de autenticação e estabelecimento de sessão do usuário. No processo de autenticação, o usuário submete o identificador e a senha de acesso para serem processados pelo cliente. O processamento consiste na aplicação de um algoritmo de *hash* à senha e envio do resultado ao servidor de autenticação para validação, efetuada com sucesso na ocasião em que o *hash*

enviado e o *hash* armazenado no servidor central, referente ao identificador fornecido, coincidem.

Após a autenticação do usuário, o *AuthenticationServer* delega a um servidor de metadados a responsabilidade de atender às requisições do usuário, esse servidor é denominado *MetaDataServer* e é provido ao servidor de autenticação pelo *MetaDataServerProvider*. O estabelecimento da sessão é consolidado com o fornecimento de uma instância da entidade *UserSession* ao cliente, contendo todas as informações pessoais e referentes à rede social do usuário, além de uma referência para o servidor de metadados responsável por atender às requisições.

A confidencialidade da interação cliente/servidor é provida com a aplicação de criptografia ao canal de comunicação através de SSL, assegurando confidencialidade e integridade de dados. O uso de certificados SSL assegura a identidade do servidor central, garantindo ao cliente que o servidor ao qual está conectado é confiável, sendo a chave pública do servidor disponível *hardwired* no código do agente

3.5.5 MetaDataServer

Responsável pelas operações de criação, atualização, remoção e recuperação dos metadados de recuperação de backup e da rede social dos usuários. O acesso ao *MetaDataServer* só é possível após o estabelecimento de uma sessão do usuário. Um usuário tem acesso apenas à porção de dados que possui permissão.

3.6 Ferramental de teste

O processo de qualificação do desenvolvimento do OurBackup envolveu a execução de uma bateria de testes para avaliar o comportamento do sistema e cumprimento dos seus requisitos. Durante este período, os defeitos detectados foram repassados à equipe de desenvolvimento que forneceram uma versão corrigida do sistema, num processo que somente era finalizado quando o *software* estivesse totalmente estabilizado.

O OurBackup utilizou de teste de unidade, integração e aceitação na composição da bateria de testes. O teste de unidade visa verificar um componente que possa ser logicamente

tratado como uma unidade de implementação. O teste de integração visa encontrar falhas provenientes da integração dos componentes do sistema. Por sua vez, o propósito do teste de aceitação é evitar surpresas desagradáveis depois que um sistema entra em funcionamento, determinando se a implementação está atendendo os requisitos do sistema ou não.

Adotamos o JUnit [jun07] na composição dos testes de unidade e integridade. O JUnit nos ajudou a organizar o desenvolvimento de testes automáticos e nos proporcionou um método sólido e consistente para atestar a capacidade das classes e métodos resolverem os problemas a que se propõem resolver.

Adotamos o EasyAccept [eas07, SNC06] na composição dos testes de aceitação. O EasyAccept é uma ferramenta que ajuda o testador a criar e executar testes de aceitação de modo fácil, rápido, e de forma limpa, construindo uma ponte de comunicação entre clientes e desenvolvedores. O usuário fornece à ferramenta o script de testes em forma de arquivos texto e uma fachada para acessar a lógica de negócio do sistema. O EasyAccept instancia a fachada, e todos os métodos públicos contidos na fachada são chamados de acordo com o script de teste.

Durante as primeiras etapas do desenvolvimento do OurBackup, o EasyAccept exerceu importante papel, direcionando e delimitando os temas tratados nas reuniões de análise e projeto. Entretanto, com a evolução do *software* para um sistema distribuído, o EasyAccept foi aos poucos oferecendo dificuldades e obstáculos em virtude da sua inadequação suportar teste que envolviam componentes distribuídos.

Um caso de teste de sistemas distribuídos consiste em dois ou mais componentes que residem computadores diferentes. Estes componentes interagem entre si durante a execução do teste. Devido à impossibilidade de obter determinismo perfeito em um sistema distribuído, escrever o código de teste e realizar testes distribuídos é difícil. O problema fundamental consiste em distinguir se uma falha foi realmente causada por um *bug* no software, ou por alguma alteração no ambiente.

No intuito de eliminar essa barreira, estendemos o EasyAccept para possibilitar a execução de testes de aceitação que envolvesse a utilização de componentes distribuídos,

desenvolvendo assim o DistributedEasyAccept que suporta de forma automática testes distribuídos, provendo as funcionalidades de geração automática de fachadas distribuídas e implantação do ambiente de teste.

O DistributedEasyAccept recebe como entrada um arquivo de mapeamento que especifica o conjunto de máquinas usado para compor o ambiente distribuído de testes, e, assim como o EasyAccept, um script de testes em forma de arquivos texto e uma fachada para acessar a lógica de negócio dos componentes distribuídos. Um pequeno exemplo de teste do DistributedEasyAccept segue abaixo:

```
expect "ok" createUser userID=alpha  userPassword=Swordfish switchID=alpha
expect "ok" createUser userID=beta  userPassword=Swordfish switchID=beta
login userName=alpha userPassword=Swordfish switchID=alpha
login userName=beta userPassword=Swordfish switchID=beta
expect "[]" getSocialNetwork switchID=alpha
expect "ok" addFriend friendID=beta switchID=alpha
expect "[beta]" getPendingFriendshipRequests switchID=alpha
expect "[]" getSocialNetwork switchID=beta
expect "[alpha]" getIncomingFriendshipRequests switchID=beta
expect "[]" getSocialNetwork switchID=beta
expect "ok" acceptFriendship friendID=alpha switchID=beta
expect "[alpha]" getSocialNetwork switchID=beta
```

Como pode ser visto na Figura 37, o DistributedEasyAccept estende a fachada dos componentes distribuídos, adicionando suporte a invocação remota de métodos através de RMI. A fachada RMI (*i.e. RMIFacade*) de testes é usada para permitir a execução de testes em máquinas remotas. O DistributedEasyAccept cria, ainda, uma macro fachada (*i.e. SwitchFacade*) responsável pela distribuição da invocação de métodos entre os componentes distribuídos. A *SwitchFacade* implementa a mesma interface da *RMIFacade*, porém todos os métodos públicos possuem um parâmetro (*i.e. switchID*) a mais que os pertencentes a fachada

original. Este parâmetro é usado para indicar qual dos componentes distribuídos deverá receber o estimulo descrito no script de testes.

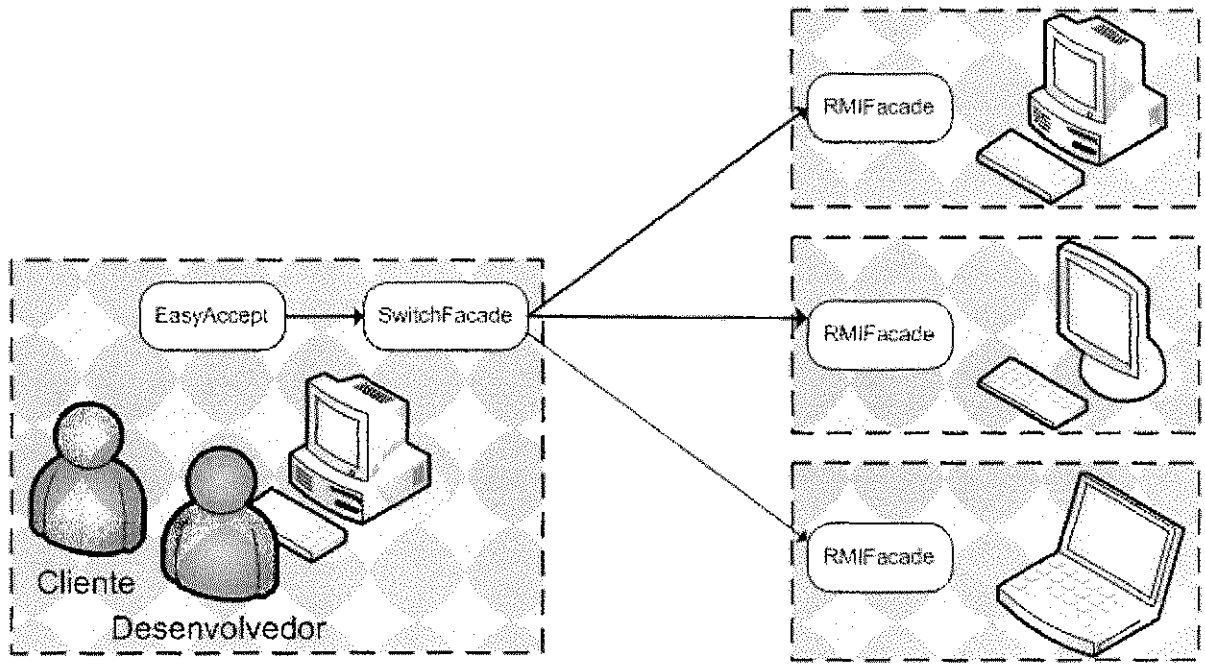


Figura 37 – EasyAccept distribuído

Uma vez gerado as fachadas *SwitchFacade* e *RMIFacade*, o DistributedEasyAccept prepara o ambiente de testes, realizando implantação dos componentes distribuídos segundo o arquivo de mapeamento. O *deployment* dos componentes distribuídos é realizado através de SSH para as máquinas escolhidas.

O DistributedEasyAccept executa, então, os testes de aceitação fornecendo ao EasyAccept o script de testes fornecido pelo usuário e a *SwitchFacade* recém gerada, como pode ser visualizado na Figura 36. O EasyAccept instância a *SwitchFacade*, e todas as invocações de métodos recebidas pela *SwitchFacade* são redirecionadas para o componente distribuído especificado pelo script de testes através do parâmetro *switchID*.

Capítulo 4 Trabalhos Relacionados

A computação P2P tem se tornado um paradigma muito usado para trocas de dados. A possibilidade de colaboração e comunicação direta entre os usuários e seus recursos está promovendo uma grande modificação nos padrões de uso da Internet. Um dispositivo numa rede P2P pode permitir o acesso a qualquer tipo de recurso que possui ao seu dispor, sejam documentos, capacidade de armazenamento ou capacidade de processamento.

O uso de redes P2P tem crescido substancialmente nos últimos anos e cada vez mais aplicações são construídas sobre essas estruturas, sendo os sistemas P2P de backup e de armazenamento duas destas aplicações. A seção 4.1 explica as diferenças entre estes dois tipos de aplicações. As seções 4.2 e 4.3 dão uma visão geral de algumas das propostas de sistemas P2P de armazenamento e backup mais referenciadas na literatura. A seção 4.4 realiza uma comparação entre o OurBackup e outros sistemas P2P de backup e armazenamento, enquanto a seção 4.5 discute a incipiente área de sistemas de backup P2P baseados em redes sociais.

4.1 Sistemas P2P de backup versus Sistemas P2P de armazenamento

Sistemas P2P de backup são similares aos sistemas P2P de armazenamento, no sentido que ambos devem lidar com armazenamento de dados. Ambas as aplicações são responsáveis por armazenar remotamente dados de um usuário e prover mecanismos de recuperação para estes dados. Entretanto, há diferenças significativas entre elas.

Uma diferença fundamental entre estas duas classes de aplicações é a freqüência no qual os dados são acessados. Não é esperado, por definição, que um backup seja acessado muito freqüentemente, devendo ser requisitado somente no caso excepcional em que os dados originais se tornarem inacessíveis. Em contraposição, é esperado que dados de um sistema de armazenamento seja acessado muito freqüentemente, exigindo, então, requisitos de disponibilidade mais estritos.

Outra diferença fundamental entre estas duas classes de aplicações diz respeito ao tipo de acesso aos dados. O backup é um processo que envolve o armazenamento de dados com o intuito de protegê-los em caso de perdas catastróficas, de modo que possam ser recuperados quando não puderem ser acessados a partir do seu local de origem. Não sendo, assim, necessário o suporte ao acesso simultâneo e concorrente dos dados.

Este não é o caso de sistemas P2P de armazenamento, que tem como responsabilidade fornecer um sistema de arquivos comum a um grupo dos usuários de modo que eles possam trabalhar simultaneamente, sendo portanto necessário o suporte a operações concorrentes de escrita e leitura. Isto faz com que esta classe de aplicação P2P seja mais complexa de ser desenvolvida, em comparação com as aplicações P2P de backup.

O fato de um sistema de armazenamento distribuído ser usado por muitos usuários impõe requisitos de segurança que adicionam ainda mais complexidade. Os usuários podem ter diferentes permissões de acesso no que diz respeito às operações de leitura, escrita, e criação de arquivos, podendo haver usuários que não possuem nenhuma permissão de acesso a arquivos específicos. Todas estas questões de segurança em relação ao compartilhamento de dados entre os usuários necessitam ser asseguradas pelo sistema.

4.2 Sistemas P2P de backup

As subseções seguintes discutem as propostas de soluções de sistemas P2P de backup: pStore [BBST02], Pastiche [CMN02] e *Cooperative Internet Backup Scheme* [LEB+03].

4.2.1 pStore

O pStore [BBST02] é um sistema de backup incremental. Ele divide cada arquivo em blocos e os armazena em uma *Distributed Hash Table* (DHT). Os blocos são indexados pelo seu conteúdo e podem ser compartilhados por outros *peers* que possuam o mesmo dado. O pStore também suporta o controle de versão para os arquivos armazenados.

Os três objetivos preliminares do projeto do pStore são: confiabilidade, segurança e consumo eficiente de recursos. A confiabilidade é assegurada através da replicação dos dados,

disseminando cópias por diversos nós em diferentes localidades, no caso de alguns destes nós tornarem-se indisponíveis ou serem maliciosos. A segurança é garantida através da cifragem dos dados. Os dados confidenciais são, conseqüentemente, passíveis de leitura apenas pelo seu proprietário, e que é também única pessoa que pode removê-los do sistema. Além disso, o proprietário pode facilmente detectar se seus dados foram alterados. Como o processo de backup pode ser freqüente e relativamente longo (*i.e.* backup de grandes quantidades dados), o objetivo do pStore é reduzir o consumo de recursos, armazenando novos dados somente quando necessário.

Entretanto, o pStore não apresenta nenhum mecanismo explícito de proteção contra usuários maliciosos que por ventura possam apagar os dados que estão no seu computador. O pStore simplesmente replica os dados dentro de DHT Chord [SMK+01], confiando na capacidade da DHT de restaurar os nós que ocasionalmente podem falhar.

O processo de backup no pStore trabalha da seguinte maneira: quando um usuário deseja armazenar um arquivo, o pStore calcula um identificador que seja específico para este arquivo e que não tenha conflitos com outros arquivos que pertençam a este mesmo usuário ou a outros usuários. O arquivo é, então, cifrado e dividido em blocos que são assinados digitalmente. Os metadados usados para remontar os blocos são, também, assinados. Os metadados e os blocos são introduzidos em uma rede DHT.

Para recuperar um backup, um usuário especifica qual arquivo e qual versão deste deseja, podendo, também, especificar uma hierarquia de diretório para ser recuperada. Uma vez que os metadados tenham sido recuperados através da DHT, todos os blocos que pertençam ao arquivo podem ser recuperados e remontados. Suas assinaturas são, então, usadas para verificar a integridade dos mesmos.

No pStore, um arquivo é representado por uma lista de blocos (*file block list* ou FBL) e pelos blocos de dados em si (*file blocks* ou FB). Cada FB contém uma parcela dos dados do arquivo. O FBL contém uma lista ordenada de todos FBs de um arquivo, mantendo as informações relativas à lista de FBs que compreendem um arquivo. O FBL armazena quatro informações para cada FB: um identificador do bloco, o *checksum* do conteúdo não

encriptado do FB, o tamanho em bytes do FB, e o deslocamento dos dados do FB em relação ao arquivo original.

As diferentes versões de um arquivo são mantidas em FBLs diferentes. FBs inalterados entre diferentes versões são simplesmente referenciadas com o intuito de preservar a capacidade de espaço de armazenamento e a capacidade de banda passante. As partes novas ou atualizadas do arquivo são novamente inseridas na rede DHT. Para determinar quais FBs mudaram entre as versões é usada uma adaptação do algoritmo do *rsync* [TM98].

FBLs e FBs são cifrados simetricamente com o objetivo de assegurar a confidencialidade. FBs são cifrados usando uma encriptação convergente na qual o *hash* do conteúdo não encriptado do FB é usado como chave simétrica para ao cifrar o FB. O *checksum* do conteúdo encriptado do FB é usado como chave no processo de inserção do FB na DHT, permitindo, portanto, o compartilhamento de blocos entre usuários diferentes.

Para eliminar colisões não desejadas do identificador do FBL entre diferente usuários, cada usuário do pStore têm um *namespace* privado em que todas os arquivos são introduzidos e recuperados. O *namespace* privado é criado através do par de chaves público/privado pertencente ao usuário.

O pStore realiza a replicação integral dos FBs para aumentar a confiabilidade do backup. Os FBs são armazenados em diversos *peers*, sendo distribuídos aleatoriamente na DHT.

4.2.2 Pastiche

O Pastiche [CMN02] foi projetado com objetivo de economizar o espaço de armazenamento total do sistema. Cada nó procura *peers* com os quais possui maior similaridade de dados armazenada. A busca por dados iguais é possível através da indexação por conteúdo, sendo realizadas através da DHT Pastry [RD01b].

No Pastiche, cada nó que quer efetuar um backup de seus dados deve, primeiramente, encontrar um conjunto de nós, denominados *buddies*, que compartilham com

ele uma quantidade significativa de dados. Em geral, cada nó do Pastiche deve manter em torno de cinco *buddies*, podendo restaurar seus dados a partir qualquer um dos seus *buddies*.

O Pastiche explora a similaridade de conteúdo entre versões do arquivo original para realizar um processo de backup incremental. Dessa forma, o consumo de banda nas transferências é reduzido. O reconhecimento de similaridade entre versões de um mesmo arquivo será possível através de uma abordagem baseada em divisão de conteúdo e de indexação, no qual o *Fingerprints Rabin* [Rab81] são usados para identificar regiões de dados e cada região é dividida em blocos através de âncoras [Man94]. Estes esquemas permitem o Pastiche identificar blocos de dados dentro de arquivos. Os blocos são rotulados com o resultado de uma função *hash* do seu conteúdo. Se dois blocos possuírem um mesmo rótulo, necessariamente eles possuirão o mesmo conteúdo.

Os metadados de um arquivo contêm uma lista dos blocos que compõem um arquivo e informação sobre a posse, as permissões, as datas de criação e modificação e entre outros atributos.

O Pastry [RD01b] é o mecanismo usado para encontrar os *buddies* de um nó, sendo as noções de assinatura e sumários essenciais para este procedimento. A assinatura é definida como a lista do *hash* dos blocos que compõem o sistema de arquivos de um nó. Os nós podem emitir sua assinatura para outros nós com o intuito de encontrar *buddies*. Entretanto, as assinaturas podem tornar-se significativamente grandes, tornando seu uso ineficiente. Para resolver este problema um sumário é emitido preferivelmente. O sumário de um nó é um subconjunto aleatório de sua assinatura.

Um nó que deseja encontrar um conjunto de *buddies* deve distribuir seu sumário a outro nó aleatório. Cada nó encontrado nesta rota computará a sua taxa de cobertura (*i.e.* a fração dos blocos do sumário que este nó armazena localmente) e retorná-la. Se esta sondagem inicial não produzir um conjunto suficiente de *buddies*, deve ser repetida até que atinja o limiar necessário de *buddies*. Uma vez que o nó tenha recebido a taxa de cobertura, ele seleciona os cinco nós que obtiveram maior taxa de cobertura para compor o conjunto de *buddies*.

4.2.3 Cooperative Internet Backup Scheme

O objetivo do *Cooperative Internet Backup Scheme* [LEB+03] é garantir a simetria entre doação e consumo de espaço compartilhado para backup. As relações simétricas são orquestradas através de um servidor centralizado. Para efetivar a simetria das relações, os nós que participam do sistema estabelecem trocas iguais de capacidade de espaço de armazenamento. Os *peers* envolvidos em uma negociação de armazenamento são chamados de parceiros, sendo, preferencialmente, situados em posições geográficas diferentes (a informação de localidade é obtida através da faixa de IP usada pelos *peers*). Como um backup pode mudar ao longo do tempo (*e.g.* quantidade de arquivos), parcerias podem ser criadas ou desfeitas. Um parceiro pode ser substituído se sua disponibilidade média for muito baixa ou se um novo computador necessitar de parceiros. Os parceiros concordam quanto a uma quantidade de espaço de armazenamento a ser trocada e um limiar de disponibilidade média a ser cumprido. Cada nó é responsável por manter a lista dos seus parceiros.

Um servidor centralizado mantém informações sobre os participantes do sistema, podendo ser usado como mecanismo de consulta para encontrar parceiros. Os nós informam ao servidor sobre os parceiros que possui e quais parcerias está querendo firmar. Quando um nó necessita de um novo parceiro, ele consulta o servidor sobre quais nós são compatíveis com as suas necessidades de armazenamento e disponibilidade, contactando, então, os candidatos para se certificar se ainda estão interessados em uma nova parceria. Quando não há nenhum outro nó a procura de parceiros, o nó A deve interceder entre dois parceiros B e C existentes que têm um acordo similar ao que ele deseja. O nó A assume a parceria de B e C para ele, de tal forma que os nós B e C não são mais parceiros, sendo sua antiga parceria assumida, agora, pelo nó A.

A confiabilidade do backup é assegurada usando a técnica de *erasure code* Reed-Solomon [Pla96].

4.3 Sistemas P2P de armazenamento

As subseções seguintes discutem as soluções de sistemas P2P de armazenamento: OceanStore [KBC+00] e PAST [RD01a].

4.3.1 OceanStore

O OceanStore [KBC+00] tem como objetivo prover um sistema global de armazenamento de dados. Foi construído para suportar bilhões de usuários e prover consistência, alta-disponibilidade e capacidade de armazenar informação por um longo período de tempo.

As unidades fundamentais no OceanStore são os objetos persistentes que são identificados por identificadores globalmente únicos. Os objetos são replicados e disseminados entre múltiplos servidores espalhados pelo mundo.

OceanStore foi projetado para fornecer o serviço de armazenamento em uma escala global de aproximadamente 10^{10} usuários, e suporte à 10^{14} arquivos. Isto requer o sistema seja altamente escalável. Em um sistema de grandes proporções como OceanStore, a localidade de informações deve ser algo importante, principalmente para a garantia de desempenho. Entretanto, o objetivo do sistema é garantir que dados podem ser armazenados em *caches* localizados em qualquer parte do sistema. Para alcançar este objetivo, o OceanStore busca melhorar a localidade dos dados através de *caching promiscuous* que armazena os dados em qualquer lugar sempre que possível. Assim, o armazenamento do objeto persistente não é limitado a uma máquina específica, o objeto pode mover-se livremente de uma máquina para outra. Este conceito do OceanStore é denominado de dados nômades.

Oceanstore fornece o *deep archival storage* que assegura que os objetos sobrevivem no sistema apesar da ocorrência de múltiplas falhas, sendo esta funcionalidade alcançada através do uso de *erasure codes*. Os fragmentos gerados através da aplicação da técnica de *erasure codes* são distribuídos entre múltiplos usuários assegurando que um arquivo possa ser reconstruído na presença de catástrofes localizadas.

A respeito da segurança, supõe-se que a infra-estrutura do OceanStore é fundamentalmente não confiável, e, conseqüentemente, todos os dados persistidos no sistema devem ser cifrados. Além disso, OceanStore tolera falhas bizantinas elegendo um pequeno conjunto de réplicas para comandar as atualizações de um arquivo. As réplicas primárias executam um protocolo de consenso com tolerância a falhas bizantinas e distribuem os

resultados a todas as réplicas restantes no sistema. Os escritores não autorizados são impedidos de forma similar.

OceanStore localiza os objetos de duas maneiras. Primeiramente, um usuário tenta encontrar um objeto usando um algoritmo probabilístico baseado em Filtros de Bloom [Blo70]. O Filtro de Bloom é uma estrutura de dados usada para representar de forma compacta um conjunto de elementos, sendo constituído por um vetor de bits e por funções de mapeamento *hash* independentes.

Se o algoritmo probabilístico falhar, então o nó utiliza uma estrutura de dados distribuída do tipo Plaxton [PRR97] para localizar o objeto. Esta estrutura aplica um algoritmo determinístico mais lento que o algoritmo probabilístico. A DHT Plaxton é fundamentada na busca por objetos distribuídos numa rede que, quando satisfazem uma requisição de consulta, cria uma cópia dos mesmos nas proximidades do nó que o requisitou.

4.3.2 PAST

O PAST [RD01a] é um sistema de armazenamento P2P que objetiva fornecer alta confiabilidade e alta disponibilidade dos dados, escalabilidade e segurança. O sistema é composto de nós que são capazes de iniciar e distribuir requisições de armazenamento ou recuperação de arquivos. Os nós podem opcionalmente contribuir o armazenamento ao sistema. Os arquivos armazenados no sistema são replicados em múltiplos nós para assegurar a confiabilidade e a disponibilidade dos mesmos.

A distribuição das réplicas pelos nós do sistema é feita de uma forma aleatória, de modo a garantir a diversidade do conjunto de nós que armazenam cada réplica. Pretende-se, deste modo, uma maior confiabilidade dos dados em face de desastres não globais, assim como o balanceamento da carga dos servidores de réplicas.

O PAST utiliza a DHT Pastry [RD01b] para identificar quais nós devem receber os dados, e então enviar os dados diretamente para este nó. O Pastry é um algoritmo para encaminhamento de mensagens e localização de objetos em grandes redes cujos nós estejam conectados através da Internet. Cada nó na rede PAST possui um identificador único. Dado

um envio de uma mensagem e uma chave, o nó receptor encaminha-a para um nó cujo identificador seja mais próximo do valor da chave.

Identificadores únicos distribuídos uniformemente são atribuídos aos arquivos controlados pelo PAST. O identificador de um arquivo é gerado como resultado da aplicação de uma função de *hash* do conteúdo, do nome textual e do identificador do dono de cada arquivo, sendo cada identificador associado a uma versão imutável de um arquivo.

Um arquivo armazenado no PAST pode ser compartilhado pelo proprietário distribuindo o identificador do arquivo para outras pessoas. Entretanto, somente o proprietário do arquivo pode solicitar a sua remoção do sistema, não sendo permitidas, assim, operações de remoção por parte de outros nós.

O PAST aplica um esquema de armazenamento baseado em *smart cards* para assegurar-se de que os clientes não possam usar mais espaço de armazenamento remoto do que eles estão fornecendo localmente. Os *smart cards* são mantidos por cada usuário PAST e certificados por uma entidade terceira. Eles suportam um sistema de quota fixas que realiza o balanceamento entre oferta e consumo do espaço de armazenamento no sistema, fazendo com que os usuários somente utilizem um espaço de armazenamento simétrico em relação com a quantidade de armazenamento que contribuem.

4.4 Comparação entre os sistemas P2P de backup/armazenamento e o OurBackup

As seções anteriores forneceram uma visão geral de alguns sistemas P2P de armazenamento e backup mais conhecidos na literatura. A seção 4.1 apresentou, brevemente, similaridades e diferenças entre as duas classes das aplicações. Nesta seção nós compararemos os sistemas apresentados com o OurBackup, tendo como pontos de referência algumas questões importantes que foram consideradas no projeto do OurBackup.

4.4.1 Armazenamento dos dados e metadados

Os sistemas pStore, Pastiche, OceanStore e Past se utilizam de *distributed hash table* (DHT) no design do sistema, embora cada deles a esteja usando de maneira diferente. A DHT

é uma estrutura que permite a realização das funções de uma tabela de *hash* normal. Nela, pode-se armazenar um par (chave, valor) e procurar por esse valor utilizando a chave. As DHTs possuem a particularidade de obter e alocar informações de uma maneira descentralizada, escalável e balanceada, permitindo a localização eficiente de dados armazenados, desde que um identificador seja escolhido para estes dados.

O pStore e o OceanStore usam a DHT tanto para armazenar os metadados quanto os dados em si. Neste sistema, os blocos de dados e os metadados, que devem ser introduzidos na DHT, recebem um identificador derivado do *hashing* de seus dados. Este identificador é usado como chave tanto na busca quanto no armazenamento. O Pastiche e o PAST não utilizam as DHTs para armazenar seus dados ou metadados. O Pastiche usa propriedades do roteamento do Pastry a fim estabelecer o conjunto de *buddies* que utilizará para efetuar o backup. O PAST, por sua vez, usa o Pastry para determinar aonde os dados devem ser armazenados.

Diferentemente dos outros sistemas apresentados, o *Cooperative Internet Backup Schema* utiliza um servidor centralizado para manter as informações sobre os participantes do sistema. Entretanto, o *Cooperative Internet Backup Schema* não possui nenhum mecanismo de armazenamento de metadados, ficando a cargo do usuário manter os metadados e lista dos seus parceiros.

O OurBackup armazena os backups nas máquinas dos amigos e utiliza um servidor centralizado para armazenar os metadados de backup. Isso permite que, em caso de *crash* da máquina local, o usuário possa recuperar todos os seus backups, utilizando os metadados armazenados pelo servidor.

4.4.2 Alocação de espaço de armazenamento

Como já foi mencionado, quando um usuário desejar realizar backup ele deverá angariar espaço nos computadores dos amigos. A alocação de espaço de armazenamento para um amigo é um contrato de backup. Entre os sistemas apresentados, podemos identificar duas abordagens distintas de alocação de espaço de armazenamento:

- O armazenamento pode ser alocado a um conjunto especifico de nós;
- O armazenamento pode ser alocado a todos os participantes através de uma DHT;

No primeiro caso, os relacionamentos entre os nós são relativamente simples: cada nó participante escolhe um conjunto de parceiros. Então, para cada backup, emite diretamente aos parceiros os blocos de dados que devem ser mantidos. No Pastiche, cada participante escolhe o conjunto de parceiros, que se manterá praticamente estático no decorrer do funcionamento do sistema. O *Cooperative Internet Backup Scheme* apresenta mecanismos explícitos de negociação de espaço de armazenamento. Quando um nó necessita de um novo parceiro, ele consulta o servidor sobre quais nós são compatíveis com as suas necessidades de armazenamento e disponibilidade, contactando, então, os candidatos, para se certificar se ainda estão interessados em uma nova parceria.

A segunda abordagem é baseada em uma técnica chamada redes *overlay* que é fundamental para sistemas de compartilhamento de arquivos [LCP+05]. As redes *overlay* ou redes virtuais criam uma arquitetura com nível mais alto de abstração, de modo a poder solucionar vários problemas que, em geral, são difíceis de serem tratados ao nível dos roteadores da rede subjacente. Estas redes usam a noção de DHT para alocar blocos dos dados. Cada nó da rede é responsável pelo armazenamento dos blocos cujos identificadores são próximos (numericamente) ao seu próprio identificador. A vantagem de usar um DHT é que os blocos estão distribuídos uniformemente sobre a rede. Os sistemas pStore, OceanStore e PAST utilizam este esquema para armazenar os dados.

Assim como *Cooperative Internet Backup Schema*, o OurBackup apresenta mecanismos explícitos de negociação de espaço de armazenamento. Cada usuário mantém todos os contratos de backup dos quais ele participa e somente requisita novos contratos para backups quando nenhum dos firmados anteriormente puder ser utilizado para armazenar as cópias. Contudo, o OurBackup ainda não fornece um mecanismo automático de negociação de espaço de armazenamento. Todos os contratos de backup são definidos manualmente pelo usuário, e sempre que necessitarem de mais espaço para backup devem utilizar um canal de

comunicação alternativo ao sistema (*e.g.* e-mail ou telefone) para solicitarem mais espaço de armazenamento.

4.4.3 Inspeção dos backups

Os sistemas que nós descrevemos fornecem diversas técnicas para assegurar que os nós estejam realmente armazenando os dados que foram alocados para armazenar. O Pastiche e o *Cooperative Internet Backup Schema* usam desafios para assegurar a guarda dos dados. Um nó desafia aleatoriamente os nós que mantêm os backups de seus dados. Se não responderem corretamente ao desafio, o nó dono dos dados pode punir o nó desafiado, descartando alguns de seus dados, e no caso específico de um nó não responder a um desafio durante um longo período de tempo, deve, então, ser substituído por outro nó. O OceanStore assegura a guarda dos dados através de um modelo econômico. Se um nó não está armazenando os dados que deveria manter, então o fornecedor deste nó é responsável por compensar esta perda.

O OurBackup possui mecanismos de auditoria que permitem detectar casos de ruptura de contratos de backup. Periodicamente, o agente requisita aos amigos o *hash* de determinados blocos de dados aleatórios. Em caso de quebra de contratos por um amigo, o usuário é informado sobre o impasse, ficando ao seu critério decidir se rejeita ou não esta amizade. O usuário pode, também, utilizar um canal de comunicação alternativo ao sistema (*e.g.* e-mail ou telefone) para consultar o amigo sobre a quebra de contrato.

4.4.4 Confiabilidade dos dados

Replicação e *erasure codes* são as técnicas de redundância de dados mais extensamente usadas para assegurar a confiabilidade dos dados. O OurBackup, o pStore, o Pastiche e o PAST replicam os dados entre os nós da rede P2P. Com esta técnica, se um nó que armazena os dados falhar permanentemente, os dados podem ser recuperados a partir de outro nó que esteja armazenando réplicas dos dados desejados. Por outro lado, o *Cooperative Internet Backup Schema* aplica os *erasure codes* Reed-Solomon [Pla96] para assegurar a confiabilidade dos dados.

O OceanStore emprega tanto o *erasure codes* quanto a replicação. A replicação é aplicada nos objetos que possuem a versão mais recente dos dados. Para as versões anteriores, são usados *erasure codes*.

4.5 Sistemas P2P Backups baseados em redes sociais

Li e Dabek [LD06] demonstraram a praticabilidade de redes sociais serem usadas em sistemas P2P de backups. Entretanto, o trabalho deles não endereça algumas das questões desafiadoras relacionadas ao projeto destes sistemas. Em detalhe, não discutem como criar, evoluir e manter redes sociais; como o espaço de backup é negociado entre amigos; (como) garantir recuperabilidade do backup e da cooperação entre os *peers*.

A versão atual do OurBackup encontra-se plenamente funcional. Permite registrar amigos e executar, bem como recuperar, backups em/de suas máquinas. A privacidade é garantida pelo uso de técnicas de criptografia. O sistema usa um servidor centralizado para rastrear informações de usuários e de localização de seus backups. Pelo que nos consta, o OurBackup é o primeiro sistema de backup P2P, baseado em redes sociais, desenvolvido.

Capítulo 5 Conclusões

O OurBackup é um sistema P2P de backup baseado em redes sociais que tem como objetivo ser uma solução de backup simples e de baixo custo. Acreditamos que a utilização de redes sociais para a realização de backup P2P permite que a taxa de falhas seja dominada por falhas de disco e, conseqüentemente, permitindo a praticabilidade de sistemas P2P de backup. Verificamos que a técnica de replicação consome mais espaço de armazenamento que os *erasure codes* para garantia de alta confiabilidade de backup perante a ocorrência falhas de disco. E oferece, ainda, menor recuperabilidade de backup que os *erasure code* para $k < 5$. Entretanto, os *erasure codes* tornam o projeto inteiro do sistema mais complexo e não suporta atualizações incrementais tão bem como replicação. Como o *erasure* não oferece economias significativas de espaço de armazenamento, optamos por usar replicação como técnica de redundância de dados no desenvolvimento do OurBackup.

O OurBackup possui um conjunto de funcionalidades que permite aos usuários efetuarem backup na máquina de amigos, explorando espaço de armazenamento ocioso nessas máquinas. Cada usuário é responsável por construir sua própria rede social.

O OurBackup mantém a confidencialidade dos backups perante amigos maliciosos e perante um servidor malicioso. Os amigos que armazenam os backups não são capazes de ler o conteúdo dos backups. O servidor também não é capaz de ler o conteúdo dos backups, mesmo sendo o mantenedor das informações dos usuários. Os backups são passíveis de recuperação pelo seu dono, inclusive em casos de *crash* da máquina de origem. A confidencialidade dos dados é garantida através de criptografia. Cada usuário possui um par de chaves, pública e privada, usado para criptografar simetricamente os backups e são armazenadas no servidor a fim de garantir a recuperabilidade dos backups. O armazenamento das chaves no servidor não permite que o administrador do OurBackup seja capaz de decifrar o conteúdo dos backups, quebrando o requisito de confidencialidade dos dados. A alternativa seguida pelo OurBackup é armazenar a chave privada critografada simetricamente usando o *password* do usuário. Assim, o usuário ainda é capaz de recuperar o seu backup em caso de

comprometimento do seu computador e nem mesmo o administrador do OurBackup é capaz de quebrar a confidencialidade.

5.1 Trabalhos Futuros

Mesmo sendo a versão atual do OurBackup inteiramente funcional, esta ainda apresenta algumas limitações. Melhorias gerais na interação com o usuário ainda são necessárias. Em particular, os usuários atribuem manualmente quanto espaço de armazenamento deve ser alocado para cada amigo. Nós esperamos automatizar este processo de maneira que o *fairness* seja assegurado, assim liberando o usuário desta tarefa e fazendo OurBackup mais autonômico e mais simples de usar.

A maioria de sistemas P2P de backup oferece quase nenhum incentivo para que os *peers* permaneçam conectados ao sistema por períodos mais longos. Por causa disso, o sistema necessita aumentar a redundância para melhorar a disponibilidade de dados. Nós pretendemos investigar mecanismos de incentivo para aumentar o tempo de permanência dos *peers on-line*.

Outra questão é a intrusividade dos sistemas de backup P2P e de outros sistemas de backup baseados em redes. Nestes sistemas, transferências podem consumir toda a largura de banda de uma máquina doméstica por prolongados períodos de tempo. Os sistemas P2P de backup existentes não apresentam nenhum mecanismo para controlar a largura de banda consumida. O que sistemas P2P de backup fazem? Conseqüentemente, uma máquina pode tornar-se muito lenta para atividades de Internet. Nós pretendemos investigar como fazer o uso mais oportunistico da largura de banda, assim fazendo OurBackup uma solução pouco intrusiva.

A recuperação de grandes quantidades de dados de um sistema P2P de backup pode ser problemática e demorada, mesmo na presença de *peers* altamente disponíveis. Esta situação é especialmente inconveniente no caso de falhas catastróficas que levam à perda de grandes quantidades de dados. Contudo, em qualquer momento do tempo, um usuário requer apenas uma fração pequena de seus dados para estar apto a fazer trabalho útil. Se o sistema de

backup for capaz de, rapidamente, recuperar tais arquivos, então o tempo de indisponibilidade percebido pelo usuário, *i.e.* o tempo no qual o usuário encontra-se impossibilitado de trabalhar, pode ser drasticamente reduzido, mesmo que uma grande porção dos dados perdidos ainda esteja sendo recuperada. Também investigaremos como definir um modelo do conjunto de trabalho do sistema de arquivos do usuário que, com alta probabilidade, identificará o conjunto de arquivos que um usuário está efetivamente usando em um dado ponto no tempo. Explorando tal modelo, pretendemos reduzir o tempo de indisponibilidade percebido pelo usuário.

Referências

[ADS03] Alessandro Acquisti, Roger Dingledine, e Paul Syverson. On the Economics of Anonymity. In Rebecca N. Wright, editor, *Proceedings of Financial Cryptography (FC '03)*. Springer-Verlag, LNCS 2742, Janeiro 2003.

[AH00] Eytan Adar e Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[Bar72] John A. Barnes. *Social Networks*. Addison-Wesley Module in Anthropology, 1972.

[BBST02] Christopher Batten, Kenneth Barr, Arvind Saraf, e StanleyTrepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, Outubro 2002.

[BR03] Charles Blake e Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 1–6, Lihue, Hawaii, Maio 2003.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. In *Communications of the ACM*, 13(7):422–426, 1970.

[cet07] Velocidade da conexão à internet utilizada no domicílio. Web Page, Accessed in: Janeiro 2007.

[CMN02] Landon P. Cox, Christopher D. Murray e Brian D. Noble. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36(SI):285–298, 2002.

[CSWH01] Ian Clarke, Oskar Sandberg, BrandonWiley,and TheodoreW. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[CTB98] Mark E. Crovella, Murad S. Taqqu e Azer Bestavros. Heavy-tailed probability distributions in the world wide web. In *A Practical Guide to Heavy Tails, edited by R. J. Adler, R. E. Feldman, e M. S. Taqqu*, pp. 3—26. London: Chapman and Hall, 1998. páginas 3–25, 1998.

[dat07] Understanding data loss. Web Page found at http://www.ontrackdatarecovery.com.au/understandingdataloss/ , Accessed in: Janeiro, 2007.

[DB99] John Douceur e William Bolovsky. A large scale study of file-system contents. In *Proceedings of the Conference on Measurement and Modeling of Computer Systems*, Atlanta, Georgia, Maio 1999.

[DMS03] Roger Dingledine, Nick Mathewson, e Paul Syverson. Reputation in P2P Anonymity Systems. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, Junho 2003.

[eas07] Easyaccept. Web Page found at https://sourceforge.net/projects/easyaccept/, Accessed in: Janeiro 2007.

[FFC91] Soares F., Farias A., Cesar C., Introdução à Estatística. Livros Técnicos e Científicos Editora S.A., Rio de Janeiro, 1991.

[FKK] Alan O. Freier, Philip Karlton, e Paul C. Kocher. The ssl protocol. Technical report, Netscape.

[HA04] Tad Hogg e Lada Adamic. Enhancing reputation mechanisms via online social networks. In *EC'04: Proceedings of the 5th ACM conference on Electronic commerce*, páginas 236–237, New York, NY, USA, 2004. ACM Press.

[HCW05] Daniel Hughes, GeoffCoulson e James Walkerdine. Free riding on gnutella revisited: The bell tolls? *IEEE Distributed Systems Online*,6(6):1, 2005.

[JH04] Rod Johnson e Juergen Hoeller. *Expert One-on-One J2EE Development without EJB*. John Wiley & Sons, 2004.

[jun07] Junit. Web Page found at htttp://www.junit.org, Accessed in: Janeiro 2007.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells e Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. *In Proceedings of ACM ASPLOS*. ACM, Novembro 2000.

[KCK+] Michelle Keeney, Dawn Cappelli, Eileen Kowalski, Andrew Moore, Timothy Shimeall e Sephanie Rogers. Insider threat study: Computer system sabotage in critical infrastructure sectors. Technical report, Joint SEI and U.S. Secret Service Report, Maior 2005.

[Kap88] K. C. Kapur. Mathematical and statistical methods and model in reliability and life studies. In: *IRESON, W. G.; COOMBS JR. Handbook of reliability engineering and management.* New York: McGraw-Hill, 1988. cap. 19, p.19.1-19.48.

[Kra88] J. W. Kraus. Maintainability and reliability. In: *IRESON, W. G.; COOMBS JR. Handbook of reliability engineering and management*. New York: McGraw-Hill, 1988. cap. 15, p.15.1-15.38.

[Kap88] K. C. Kapur. Mathematical and statistical methods and model in reliability and life studies. In: *IRESON, W. G.; COOMBS JR. Handbook of reliability engineering and management*. New York: McGraw-Hill, 1988. cap. 19, p.19.1-19.48.

[LCL04] W. K. Lin, D. M. Chiu e Y. B. Lee. Erasure code replication revisited. In *P2P'04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, páginas 90–97, Washington, DC, USA, 2004. IEEE Computer Society.

[LCP+05] Keong Lua, J. Crowcroft, M. Pias, R. Sharma e S. Lim. A survey and comparison of peer-to-peer overlay network schemes. In *Communications Surveys & Tutorials*, IEEE, páginas 72–93, 2005.

[LD06] Jinyang Li e Frank Dabek. F2F: reliable storage in open networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, Fevereiro 2006.

[LEB+03] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Michael Burrows, e Michael Isard. A cooperative internet backup scheme. In *USENIX AnnualTech-nical Conference, GeneralTrack*, páginas 29–41, 2003.

[LNBK02] David Liben-Nowell, Hari Balakrishnan e David Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, páginas 233–242, NewYork, NY, USA, 2002.ACM Press.

[LZT04] Martin Landers, Han Zhang e Kian-Lee Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *P2P'04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, páginas 72–79,Washington, DC, USA, 2004. IEEE Computer Society.

[Man94] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA, 17–21 1994.

[MGGM04a] Sergio Marti, Prasanna Ganesan e Hector Garcia-Molina. DHT routing using social links. In Geoffrey M. Voelkerand Scott Shenker, editors, *IPTPS*, volume 3279 of *Lecture Notesin Computer Science*, páginas 100–111. Springer, 2004.

[MGGM04b] Sergio Marti, Prasanna Ganesan e Hector Garcia-Molina. Sprout: P2P routing with social networks. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and AthenaVakali, editors, *EDBTWorkshops*, volume 3268 of *Lecture Notes in Computer Science*, páginas 425–435. Springer, 2004.

[MK06] Anirban Mondal, Masaru Kitsuregawa: Privacy, Security and Trust in P2P environments: A Perspective. In *DEXA Workshops* 2006: 682-686

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil e Benjie Chen. Ivy:A read/write peer-to-peer file system. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[MP07] Ivan Luizio Magalhães e Walfrido Brito Pinheiro. *Gerenciamento de Serviços de TI na Práica -Uma abordagem com base na ITIL*. Novatec, 2007.

[mtb07] Maxtor diamondmax16 datasheets.WebPage, Accessed in: January, 2007.

[MVO96] Alfred J. Menezes, Scott A. Vanstone e Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[PCT04] Bogdan C. Popescu, Bruno Crispo, e Andrew S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. In *Proc. 12th Cambridge International Workshop on Security Protocols*. Springer-Verlag, Abril 2004.

[PGB+06] J.A. Pouwelse, P. Garbacki, J. Wangand A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M.R. van Steen, e H.J. Sips. Tribler: A social-based based peer to peer system. In *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Fevereiro 2006.

[Pla96] James S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Technical report, Knoxville, TN, USA, 1996.

[PRR97] C. Greg Plaxton, Rajmohan Rajaraman, e Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, páginas 311–320, New York, NY, USA, 1997. ACM Press.

[Rab81] Michael O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard Aiken Computation Laboratory, 1981.

[RD01a] A. Rowstron e P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, In *Proc. ACM SOSP'01*, Banff, Canadá, Outubro 2001

[RD01b] Antony I. T. Rowstron e 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*, páginas 329–350, London, UK, 2001. Springer-Verlag.

[SDM+05] Steffen Staab, Pedro Domingos, Peter Mika, Jennifer Golbeck, Li Ding, Tim Finin, Anupam Joshi, Andrzej Nowak e Robin R. Vallacher. Social networks applied. *IEEE Intelligent Systems*, 20(1):80–93, 2005.

[SGG02] Stefan Saroiu, P. Krishna Gummadi e Steven Gribble. A measurement study of peer-to-peer file sharing systems. In *SPIE Multimedia Computing and Networking (MMCN2002)*, 2002.

[SKS+05] Djamel Sadok, Carlos Kamienski, Eduardo Souto, Joé Rocha, Marco Domingues, and Arthur Callado. Colaboração na internet e a tecnologia peer-to-peer. In *XXV Congresso da Sociedade Brasileira de Computação (SBC 2005)*, páginas 1407–1454, Julho 2005.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek e Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. *In Proceedings of the 2001 ACM SIGCOMM Conference*, páginas 149–160, 2001.

[SNC06] Jacques Philippe Sauvé, Osório Lopes Abath Neto e Walfredo Cirne. Easyaccept: a tool to easily create, run and drive development with automated acceptance tests. In *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, pages 111–117, New York, NY, USA, 2006. ACM Press.

[SR06] Daniel Stutzbach e Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, páginas 189–202, New York, NY, USA, 2006. ACM Press.

[SS02] Jordi Sabater and Carles Sierra. Reputation and social network analysis in multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 475– 482, New York, NY, USA, 2002. ACM Press.

[SW04] Subhabrata Sen e Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw.*, 12(2):219–232, 2004.

[TM98] A. Trigdell e P. Mackerras. The rsync algorithm. Technical report, Australian National University, 1998. http://rsync.samba.org.

[Upa02] Y. Upadrashta. Emerging Social Networks in Peer-to-Peer Systems. In 2002-2003 Grad Symposium, CS Dept, University of Saskatchewan

[vel07] Velox. Web Page found at www.velox.com.br, Accessed in: Janeiro 2007.

[vir07] Criador de vírus condenado a 20 meses de prisão. Web Page found at http://www.bbc.co.uk/portuguese/ciencia/020502_virusmtc.shtml, Accessed in: Janeiro 2007.

[wor07] Custo dos estragos do mydoom: US$ 250 milhões. Web Page found at http://www.homenews.com.br/article.php?sid=1906 , Accessed in: Janeiro 2007.

[WV07] Torell W. e Avelar V. Mean time between failure: Explanation and standards. Web Page found at http://whitepapers.silicon.com/0,39024759,60137969p-39000667q,00.htm, Accessed in: Janeiro 2007

[YCZ+04] M. Yang, H. Chen, B. Y. Zhao, Y. Dai, e Z. Zhang. Deployment of a large-scale *peer*-to-*peer* social network. In *USENIX WORLDS*, 2004.

[ZHYD06] Yang Zhao, Xiao Hou, MaoYang, e Yafei Dai. Measurement study and application of social network in the maze P2P file-sharing system. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, página 57, NewYork, NY, USA, 2006.ACM Press.

Apêndice A

Este apêndice apresenta os resultados obtidos nos 20 experimentos preliminares de Recuperabilidade com o objetivo de calcular o desvio padrão e o número de simulações (*N*) necessárias para obter o intervalo de confiança.

1 Replicação

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =100GB

k	N
2	2
3	2
4	2
5	2
6	1
7	1
8	1
9	1
10	1

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =10GB

k	N
2	21
3	18
4	21
5	17
6	5
7	7
8	5
9	2
10	2

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =1GB

k	N

2	222
3	138
4	123
5	96
6	39
7	37
8	20
9	5
10	2

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k	N
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k	N
2	9
3	7
4	5
5	4
6	1
7	1
8	1
9	1
10	1

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k	N
2	179
3	63
4	53
5	16
6	5
7	4
8	2
9	1

10	2

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k	N
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k	N
2	8
3	8
4	9
5	14
6	10
7	6
8	5
9	3
10	2

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k	N
2	99
3	80
4	98
5	68
6	82
7	59
8	18
9	15
10	8

2 Erasure Codes

a. Particionamento de Banda Igualitário

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =100GB

k\b	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =10GB

k\b	2	3	4	5	6	7	8	9	10
2	11	8	4	2	1	2	1	1	1
3	3	2	4	3	3	3	3	2	2
4	3	5	4	7	3	3	2	2	2
5	3	5	3	3	2	2	2	2	1
6	5	5	4	3	2	2	2	2	1
7	3	3	2	1	2	1	1	1	1
8	4	3	2	2	1	1	1	1	1
9	3	2	2	1	1	1	1	1	1
10	3	2	1	1	1	1	1	1	1

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =1GB

k\b	2	3	4	5	6	7	8	9	10
2	107	39	17	27	14	11	9	6	6
3	29	17	11	13	11	12	12	8	8
4	9	15	15	16	9	12	9	7	11
5	17	16	23	16	10	11	12	6	4
6	24	20	10	11	10	7	9	7	5
7	24	15	10	10	9	8	5	5	5
8	19	13	8	12	11	7	5	5	4
9	19	15	7	10	10	5	5	5	4
10	15	10	9	6	6	5	4	4	4

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k\b	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k\b	2	3	4	5	6	7	8	9	10
2	3	2	1	1	1	1	1	1	1
3	2	1	1	1	1	1	1	1	1
4	2	1	2	1	1	1	1	1	1
5	2	2	1	1	1	1	1	1	1
6	2	1	1	1	1	1	1	1	1
7	2	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k\b	2	3	4	5	6	7	8	9	10
2	51	12	6	5	4	3	3	3	4
3	16	10	7	7	6	7	7	6	4
4	14	12	8	11	8	10	7	5	6
5	11	13	11	7	7	6	4	4	3
6	13	12	8	7	5	4	3	3	4
7	9	6	7	4	4	3	2	3	3
8	12	7	6	3	2	3	3	2	2
9	5	5	5	4	2	3	4	1	2
10	9	6	3	3	2	3	2	2	2

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k\b	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1

5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k\b	2	3	4	5	6	7	8	9	10
2	5	3	2	2	1	1	1	1	1
3	3	2	2	2	2	2	2	2	2
4	2	2	2	2	2	1	1	2	1
5	2	3	2	2	1	1	1	1	1
6	2	2	2	1	1	1	1	1	1
7	2	2	2	1	1	1	1	1	1
8	2	2	1	1	1	1	1	1	1
9	2	1	1	1	1	1	1	1	1
10	2	1	1	1	1	1	1	1	1

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k\b	2	3	4	5	6	7	8	9	10
2	83	32	18	23	8	8	6	5	4
3	32	8	12	11	17	13	11	8	6
4	4	10	11	14	13	9	8	13	7
5	11	14	15	13	10	8	6	5	6
6	10	20	15	7	9	6	6	5	4
7	11	13	7	7	8	4	5	5	4
8	15	9	12	7	5	5	6	3	3
9	14	10	8	7	5	3	4	4	3
10	18	6	6	5	3	4	2	3	3

b. Particionamento de Banda Proporcional

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =100GB

k\b	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1

4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =10GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	16	12	5	3	1	1	1	1	1
3	6	4	2	1	2	2	1	1	1
4	3	2	2	2	2	2	2	1	2
5	5	3	3	3	1	3	2	2	2
6	6	6	4	2	3	3	3	2	1
7	5	4	3	3	3	2	3	2	1
8	6	6	4	3	3	3	3	2	2
9	7	7	4	3	3	2	3	2	2
10	7	5	4	4	3	2	3	2	2

Banda Passante - d = 1 Mbps e u = 300 Kbps
Tamanho do backup - S =1GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	68	90	70	45	16	21	25	7	5
3	26	12	5	6	6	5	4	3	3
4	10	7	14	6	7	8	6	4	5
5	16	11	15	8	11	7	6	7	4
6	24	12	12	11	8	8	7	8	5
7	13	13	17	6	4	7	7	5	4
8	20	14	13	10	8	8	7	5	4
9	17	15	13	7	6	7	6	4	4
10	30	11	13	6	10	6	4	6	5

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1

9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	3	2	1	1	1	1	1	1	1
3	3	2	2	1	1	1	1	1	1
4	2	3	3	2	1	1	1	1	1
5	3	3	2	1	2	2	1	1	1
6	5	3	3	2	2	2	2	1	1
7	4	4	3	2	2	2	1	1	1
8	6	4	2	3	2	2	2	1	1
9	6	5	3	2	3	2	2	1	1
10	7	6	4	3	3	2	2	1	2

Banda Passante - d = 300 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	45	24	14	6	2	3	3	2	1
3	13	3	4	3	3	3	3	2	2
4	9	4	6	6	5	3	4	3	3
5	11	13	7	6	4	4	3	5	3
6	9	7	8	6	4	4	4	5	3
7	11	11	8	5	6	4	3	3	4
8	16	10	6	6	5	5	3	3	3
9	10	6	7	7	7	7	5	4	3
10	16	9	8	5	4	4	5	3	3

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =100GB

k\b	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =10GB

k\b	2	3	4	5	6	7	8	9	10
2	6	6	4	3	2	1	1	1	1
3	5	4	2	2	1	1	1	1	1
4	4	2	2	2	2	2	1	1	1
5	4	2	3	2	2	2	2	2	1
6	6	4	2	2	2	2	2	1	1
7	7	3	4	2	2	2	2	2	2
8	5	5	3	2	4	2	2	2	1
9	7	5	3	3	3	2	2	2	1
10	8	3	3	3	3	3	2	2	2

Banda Passante - d = 600 Kbps e u = 150 Kbps
Tamanho do backup - S =1GB

k\b	2	3	4	5	6	7	8	9	10
2	76	42	30	23	9	5	7	4	4
3	24	11	5	3	3	4	4	3	3
4	13	7	6	6	6	3	4	3	3
5	12	10	7	10	4	5	5	3	4
6	10	12	9	8	6	6	6	4	5
7	13	9	9	8	6	5	5	4	3
8	13	10	9	4	5	7	6	4	3
9	17	13	10	10	5	5	5	5	4
10	19	15	8	7	8	3	5	6	3