UNIVERSIDADE FEDERAL DA BAHIA

INSTITUTO DE MATEMÁTICA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TÁSSIA CAMÕES ARAÚJO

ANÁLISE COMPARATIVA ENTRE O BRF E MÉTODOS TRADICIONAIS PARA GERENCIAMENTO DE BACKUPS

Salvador

2006

TÁSSIA CAMÕES ARAÚJO

ANÁLISE COMPARATIVA ENTRE O BRF E MÉTODOS TRADICIONAIS PARA GERENCIAMENTO DE BACKUPS

Monografia apresentada ao Curso de graduação em Ciência da Computação, Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof^{o}. Imre Simon
Co-Orientador: Antônio de Azevedo Terceiro

Agradeço especialmente ao Prof^{o}. Imre Simon pela oportunidade de trabalharmos juntos. Tivemos praticamente dois meses para realizar algo que parecia bem distante. Ao final, conseguimos obter ótimos resultados e mais vontade de continuar o trabalho. A dedicação, orientação e paciência desprendidas por ele nestes últimos meses certamente ficarão guardadas na minha memória. Terceiro também teve papel importante nesta história, me dando força nos momentos de aperto, estando presente também como amigo e colega.

Agradeço também a todas as pessoas que estiveram ao meu lado na fase de elaboração deste trabalho (ainda que virtualmente) e me deram força e estímulo pra que eu continuasse seguindo em frente e não deixasse a peteca cair. A Tiago, que esteve sempre comigo no dia-a-dia e presenciou momentos bons e ruins mais de perto, sempre me dando a maior força como companheiro, amigo e salva-vidas de plantão. À minha família querida que me encheu de carinho e incentivo, em mais uma jornada. Alguns de perto, outros de longe, mas sempre com muito amor e repeito. E aos amigos, que estão cada vez mais espalhados pelo mundo, mas não perdem o cantinho reservado no lado esquerdo do meio peito.

RESUMO

O termo *backup* no contexto deste trabalho refere-se à cópia de segurança de um conjunto de dados que pode ser usada em caso de perda ou corrupção dos dados originais. Na definição da política de *backups* de uma organização, uma das questões que merecem atenção é o gerenciamento do espaço utilizado para o armazenamento de backups. Este trabalho apresenta detalhadamente a proposta do *BRF, Backup, Remember, and Forget*, produzindo como resultado este documento, que é atualmente a fonte de informações mais completa sobre o seu funcionamento. Num segundo momento é realizada uma análise comparativa entre as principais políticas que podem ser adotadas com o BRF e outros métodos clássicos, concluindo que existe superioridade do BRF em alguns aspectos e demonstrando sua viabilidade técnica para implementação.

Palavras-chave: *Backup, BRF, Bacula, Software Livre, Segurança, Python*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em computação, o termo *backup* pode ser utilizado em diversas situações, geralmente com o sentido de redundância. Designa-se ao termo *backup de dados* uma cópia de segurança de um conjunto de dados, que pode ser usada em caso de perda ou corrupção dos dados originais [Backup 2006].

Alguns cenários comuns para utilização de *backup* são:

- **Comprometimento lógico de um sistema**

  Quando ocorre uma invasão, geralmente não se sabe o grau de comprometimento do sistema invadido. Na maioria dos casos, a solução mais rápida e segura para que o sistema volte à ativa é a sua restauração para um estado que certamente não esteja comprometido. Ainda que se realize em paralelo a análise forense do sistema comprometido, o que é recomendável, o *backup* se configura como um ”salva-vidas”, podendo assumir o papel do sistema original imediatamente.

- **Ocorrência de um desastre (comprometimento físico de um sistema)**

  Um incêndio ou inundação, por exemplo, pode resultar na perda total de um sistema, sendo necessária sua reconstituição por completo, com as configurações originais e os dados armazenados nele.

- **Perda de arquivos isolados**

  Existem casos onde os próprios usuários apagam ou modificam arquivos acidentalmente e necessitam que alguma versão antiga seja recuperada. Muitas vezes esse usuário é o próprio administrador da máquina, que modificou a configuração de um serviço resultando na total inoperância do mesmo e deseja restaurar o estado anterior.

Percebe-se então que o *backup* deixa de representar somente uma cópia extra dos dados para se tornar um aspecto essencial que consolida a segurança das informações de uma

organização. Segundo [Segurança da Informação 2006], os três elementos que constituem os pilares da segurança da informação são: integridade, disponibilidade, e confidencialidade. O *backup* está fortemente relacionado ao conceito de disponibilidade: *propriedade que garante que a informação esteja sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação*. É importante lembrar que todos os cenários citados acima podem ser provocados voluntariamente, ainda que muitas vezes sejam confundidos com acidentes. Por mais protegido que seja o sistema, com *Firewall*, *Intrusion Detection System (IDS)*, pouquíssimos serviços de rede disponíveis, nenhum usuário em desuso, senhas fortes, e todas as boas práticas recomendadas pelos padrões internacionais de segurança, se o atacante provocar a perda total do sistema, somente uma cópia de segurança poderá restaurar o sistema original. Ainda assim, o prejuízo será proporcional à idade do *backup* mais recente. Não havendo *backup*, faz-se inútil todo o investimento e a proteção aplicada ao sistema perdido.

Outro aspecto que deve ser considerado é a segurança de armazenamento dos *backups*. Em caso de desastre, se um *backup* estiver no mesmo espaço físico que o sistema, a probabilidade de que ele também seja danificado é grande. Portanto, deve-se pensar também num esquema que, além de realizar os *backups* periodicamente, transporte-os para um local diferente do que o sistema se encontra. E quanto mais automatizado for este processo, maior a sua eficiência e menor o seu custo operacional. Por esta razão, os *backups* remotos se consolidaram como uma solução bastante atraente, havendo inclusive empresas que prestam esse serviço através da Internet.

Estas e outras questões são contempladas no que se chama de política de *backups* de uma organização, que diz respeito ao planejamento dos procedimentos de *backup* e restauração dos sistemas monitorados. As variáveis envolvidas nestes procedimentos são, entre outras, a periodicidade e tipo dos *backups*, a quantidade de exemplares das cópias de segurança, o local de armazenamento das cópias, dispositivos físicos utilizados para armazenamento, método de rotação entre os dispositivos, compressão e encriptação dos dados, automatização dos procedimentos, etc.

Na definição de uma política de *backups*, um dos problemas recorrentes é o gerenciamento do espaço utilizado para o armazenamento de backups. Intuitivamente, a eficiência de uma política de *backups* depende diretamente da frequência com que os dados são copiados e por quanto tempo os *backups* são mantidos. Considerando que os *backups* são armazenados num dispositivo físico, quanto mais eficiente for a política, mais custosa será a solução de armazenamento.

É neste contexto que este trabalho se insere, apresentando uma nova proposta para o geren-

ciamento do espaço utilizado chamada BRF - Backup, Remember, and Forget [Simon 2002], e fazendo uma comparação entre esta e outras soluções mais populares atualmente. Desta forma, pretende-se difundir o BRF na comunidade, provendo uma documentação que incentive seu entendimento, utilização e desenvolvimento, somado à discussão sobre sua viabilidade e adequação através dos resultados dos experimentos aqui descritos.

O presente trabalho está organizado em sete capítulos, incluindo esta introdução. No capítulo 2 são apresentados alguns conceitos básicos necessários para o entendimento do trabalho. No capítulo 3 faz-se uma introdução ao BRF, a ferramenta que é o foco principal desta pesquisa. No capítulo 4, a metodologia adotada na realização dos experimentos é descrita. No capítulo 5 apresenta-se a primeira fase dos experimentos, na qual é realizada uma análise detalhada do funcionamento do BRF num sistema de grande porte. No capítulo 6 traz a segunda fase dos experimentos, onde o BRF é comparado com outras políticas de *backup* comumente utilizadas. Por fim, a conclusão do trabalho é apresentada no sétimo capítulo.

Todo o processo de pesquisa, implementação, testes, validação, publicação e apresentação deste trabalho foi concebido entre os meses de setembro e dezembro de 2006. É importante registrar que a comunicação com o orientador deu-se por completo através de trocas de email e videoconfêrencia, uma vez que a distância geográfica e compromissos em paralelo de ambas as partes inviabilizava encontros presenciais. O material que inclui textos e imagens foi construído e publicado gradualmente num ambiente colaborativo hospedado no portal Informação, Comunicação e a Sociedade do Conhecimento em [Camões 2006]. O co-orientador cumpriu com o papel de acompanhar as discussões via email e o desenvolvimento do trabalho localmente.

2 CONCEITOS IMPORTANTES

2.1 DEFINIÇÕES

A seguir são apresentados os significados de alguns termos utilizados ao longo do texto.

- **Evento:** um arquivo, diretório ou link presente no sistema de arquivos. Um evento pode ser representado como a tupla ⟨caminho, data de modificação, tamanho⟩, e se algum desses elementos for alterado tem-se como resultado outro evento. Portanto, versões diferentes de um mesmo arquivo são definidas como eventos distintos;

- **Volume:** a representação de um dispositivo físico para o sistema de *backup*. Pode ser uma mídia de armazenamento removível, uma partição de disco rígido, ou até mesmo um diretório do sistema de arquivos;

- ***Pool*:** um conjunto de volumes utilizados continuamente num procedimento de *backup*. Usando um *pool* é possível que um mesmo *backup* seja armazenado em diversos volumes sem que haja problema algum. Quando o espaço livre de um volume é esgotado no decorrer de um procedimento, outro volume do *pool* é disponibilizado imediatamente, possibilitando assim que o processo continue;

- **Precisão:** referente à distância entre o estado que se deseja recuperar e o estado disponível para recuperação num sistema de *backup*, no pior caso. Quanto menor for esta distância, mais preciso será o esquema. A precisão é portanto uma medida de qualidade de políticas de *backup* e será referenciada ao longo do trabalho;

- **Reciclagem:** reuso de um volume ou *pool* usado anteriormente para um procedimento de *backup* de dados;

- **Demanda de espaço:** espaço disponível necessário para que uma política de *backup* seja executada com sucesso. A demanda pode ser medida em valor absoluto ou relativo ao espaço ocupado pelos dados monitorados ao longo do tempo.

2.2 TIPOS DE *BACKUP*

Tendo como base as diferentes ferramentas de *backup* disponíveis atualmente, constata-se que não existe consenso na definição dos tipos básicos de *backup* que podem ser realizados num procedimento de cópia de dados. Este texto utiliza a definição de tipos de *backup* de [Cougias, Heiberger e Koop 2003], com a ressalva que não se faz distinção entre *backups* armazenados em disco ou dispositivo removível.

- **Completo (*Full backup*):**

  Todo o conteúdo é copiado do início ao fim, sem levar em consideração *backups* anteriores. Este *backup* tem como resultado um espelho do sistema monitorado. A recuperação dos dados é a mais rápida, visto que todos os dados podem ser obtidos numa mesma operação.

- **Diferencial (*Differential backup*):**

  Apenas os arquivos modificados entre o estado atual e o último *backup* completo são copiados. Este *backup* contém exatamente a diferença entre o estado atual e a última cópia completa realizada. O processo de recuperação é um pouco mais lento, já que é necessário que se faça duas operações: (1) recuperação do *backup* completo e (2) recuperação do *backup* diferencial. Neste caso a recuperação somente é possível se houver acesso ao *backup* completo.

- **Incremental (*Incremental backup*):**

  Neste tipo de *backup* apenas o conteúdo modificado no sistema desde o último *backup*, é copiado, independente do tipo do último *backup*. O resultado é a diferença entre o estado atual e a última cópia realizada. A recuperação neste caso é mais complexa, visto que é necessário que o sistema seja reconstituído até o estado anterior, o que pode representar diversas operações, para então ser aplicado o *backup* incremental. Para restaurar um *backup* incremental é necessário que estejam disponíveis, no mínimo, (1) o último *backup* completo; (2) o último diferencial entre o completo e o incremental que se deseja recuperar; (3) todos os incrementais desde o último diferencial; caso contrário, a recuperação é inviabilizada.

Um ciclo de *backup* tem início com um *backup* completo e prossegue com *backups* incrementais ou diferenciais subseqüentes. Quando um novo *backup* completo é realizado, um novo ciclo é iniciado.

2.3 MÉTODOS DE ROTAÇÃO DE DISPOSITIVOS

A rotação de dispositivos é um aspecto prioritário na definição de uma política de *backups*. Em situações onde a quantidade de dispositivos físicos a serem utilizados para *backups* é limitada, em algum momento será necessário que os dispositivos sejam reciclados, uns após os outros, seguindo uma lógica que definimos como o método de rotação de dispositivos. Se não houver rotação, o sistema de arquivos que armazena os *backups* crescerá indefinidamente, o que geralmente não é desejado. São apresentados a seguir os métodos clássicos de rotação segundo [Cougias, Heiberger e Koop 2003]. As descrições se baseam em esquemas diários e considera que o espaço livre nos volumes não se esgota no processo, entretanto o conceito de *pools* pode ser usado de forma análoga.

- **Rotação cíclica semanal:**

  São mantidos três volumes, havendo sempre um localmente e outros dois guardados em um lugar seguro, fisicamente distante do sistema. No início de cada semana é realizado um *backup* completo e diariamente um *backup* incremental. Após o último procedimento de *backup* da semana, um volume é trazido para o local do sistema e trocado com o que armazenou os *backups* da semana, o qual é tranferido para a outra localidade. A qualquer momento haverá pelo menos um volume local e um em outro lugar seguro, que deve ser usado em caso de desastre. A recuperação das cópias de segurança é simples e segura, basta recuperar o último *backup* completo disponível e em seguida todos os incrementais armazenados no mesmo volume. Cada *backup* é mantido por 3 semanas, e a cada semana se começa um novo ciclo.

- **Rotação ”Filho”:**

  Este é o primeiro conceito do esquema ”Avô, pai, filho”. Um *backup* completo é feito diariamente. Existem sete volumes para serem utilizados durante a semana, na qual um ciclo termina e outro começa a cada dia. Esta é forma mais simples de fazer *backup* e a recuperação é imediata, já que todos as cópias são completas. Por outro lado, é a mais custosa em termos de espaço e duração do procedimento de *backup*.

- **Rotação ”Pai, filho”:**

  Este esquema é uma mistura de *backups* completos e incrementais. Desta vez é considerado um mínimo de oito volumes, sendo dois para *backups* completos e seis para incrementais. Uma vez por semana é realizado um *backup* completo, e nos outros dias

são feitos incrementais. Os volumes incrementais são reciclados cada semana, exatamente no mesmo dia que ele foi usado na semana anterior, havendo um volume para segunda-feira, outro para terça-feira, e assim por diante. Já os volumes usados em *backups* completos são revezados e utilizados semana sim, semana não. Portanto, a qualquer momento pode-se recuperar um estado anterior do sistema com precisão máxima de um dia na última semana, e precisão de uma semana na semana anterior. Para aumentar o alcance deste método, pode-se aumentar o número de volumes utilizados em *backups*. Se houver um maior número para *backups* completos, o que tem um custo bastante elevado, será possível armazenar os dados por mais tempo, já que a reciclagem de dispositivos demorará um pouco mais para acontecer. Por outro lado, se houver mais volumes incrementais, estende-se o período em que se têm a precisão de um dia. Porém, se o número de volumes incrementais for muito grande, a reconstituição torna-se bastante lenta, visto que será necessário restaurar cada um dos *backups* incrementais entre o último completo e o que se deseja restaurar.

- **Rotação "Avô, pai, filho":**

  Neste esquema têm-se *backups* completos, incrementais e diferenciais. A cada mês é feito um *backup* completo, a cada semana é feito um diferencial e diariamente é feito um incremental. Para um esquema de *backups* diários, é necessário pelo menos doze volumes, sendo dois para *backups* completos, quatro para *backups* diferenciais e seis para incrementais. Com este cenário, consegue-se uma precisão de no máximo um dia na última semana, uma semana no último mês e um mês nos últimos dois meses. Analogamente ao esquema anterior, estes períodos podem variar de acordo com a quantidade de volumes disponibilizados para cópias completas, incrementais e diferenciais. Devido a sua eficiência e boa relação custo/benefício, este certamente é o método mais utilizado atualmente.

2.4 PERSISTÊNCIA DE DADOS

Aliado aos métodos de rotação de dispositivos é possível adotar uma medida para que alguns estados do sistema se tornem perenes, ou persistentes. Por exemplo, pode-se fazer uma cópia mensal do *backup* completo do sistema em outro dispositivo que não faça parte da rotação, o que caracteriza uma política persistente mensal. Numa política persistente total, ou simplesmente persistente, todos os volumes são perenes. Esses volumes são guardados separamente, até que um administrador decida descartá-los permanentemente, quando eles não forem mais

relevantes para a organização.

2.5 FERRAMENTAS DE *BACKUP*

São inúmeras as ferramentas de *backup* disponíveis atualmente. Como este trabalho limita-se à análise de soluções livres, são apresentadas abaixo duas das mais populares.

- **Amanda**

  O Amanda, acrônimo para *Advanced Maryland Automatic Network Disk Archiver*, é um sistema de *backup* que permite a configuração de um servidor mestre para realização de *backups* de múltiplos clientes numa rede, com a possibilidade de armazenamento em fita, disco ou dispositivo ótico (CD ou DVD) [Amanda 2006]. Ele usa o *dump* nativo do UNIX e/ou a ferramenta GNU *tar* e pode realizar *backups* de uma grande quantidade de máquinas rodando diferentes sistemas UNIX. Ele também usa *Samba* ou *Cygwin* para *backups* de sistemas Microsoft Windows. Esta foi por muito tempo a mais popular ferramenta livre para realização e gerencimento de *backups*, e ainda hoje é bastante usada. Numa busca por ferramentas de *backup* no *SourceForge* [SourceForge 2006], o Amanda aparece em segundo lugar, com cerca de 180.000 *downloads*.

- **Bacula**

  O Bacula é um sistema de *backup* distribuído para redes heterogêneas [Bacula 2006]. Suas funcionalidades são divididas em serviços que podem estar distribuídos na rede. O *Bacula Director* é o serviço central, que gerencia todas as ações no sistema. Ele se comunica e coordena as ações dos *Storage Daemons*, responsáveis pelo armazenamento dos dados, e dos *Client Daemons*, instalados nos sistemas monitorados. Além disso, o *Director* mantém um catálogo, uma base de dados que registra toda a evolução do sistema, com informações sobre os backups realizados, os arquivos presentes em cada volume de dados, os clientes, os *storages*, etc. O Bacula atualmente é a solução de *backup* mais popular do *SourceForge*, com cerca de 300.000 *downloads* [SourceForge 2006]. Uma das grandes vantagens do Bacula em relação a outros sistemas é que ele permite a realização de um *backup* em múltiplos volumes (dispositivos), utilizando o conceito de *pools*.

2.6 ESTRUTURAS DE DADOS

Nesta seção são apresentadas algumas estruturas de dados necessárias para o entendimento do funcionamento do BRF. Parte-se de conceitos fundamentais como grafo e árvore, para viabilizar em seguida a apresentação de heap binomial, estrutura na qual o BRF se baseia.

2.6.1 GRAFO

Um grafo é um par $\langle V, E \rangle$, onde V é um conjunto finito, denominado *conjunto de vértices*, e E consiste em pares de vértices (ordenados ou não-ordenados), chamado de *conjunto de arestas* [Cormen et al. 2002]. Graficamente, as arestas são representadas como ligações entre os vértices. Os grafos são *orientados* quando a ordem dos vértices na representação das arestas é significante, sendo $\langle u, v \rangle$ e $\langle v, u \rangle$ arestas distintas. Em grafos *não-orientados*, a relação entre dois vértices é bi-direcional, sendo assim, as arestas $\langle u, v \rangle$ e $\langle v, u \rangle$ têm o mesmo significado. Os grafos podem também ser caracterizados como *cíclicos*, se houver mais de um caminho possível entre quaisquer dois vértices do grafo, ou *acíclicos*, caso contrário. E ainda, *conectados*, se for possível alcançar todos os vértices do grafo a partir de um dos vértices, ou *desconectados*, caso exista um ou mais vértices desconectados dos demais.

Os vértices do grafo são identificados por *rótulos*, que representam o endereço do vértice na estrutura. Além do rótulo, cada vértice pode ter ou não conteúdo, que comumente é chamado de *chave* e representa seu valor.

2.6.2 ÁRVORE

Uma *árvore* (ou árvore livre) pode ser definida como um grafo acíclico não-orientado conectado [Cormen et al. 2002]. Grafos desconectados são chamados de *florestas*, já que representam um conjunto de árvores distintas. *Árvores com raiz* possuem um vértice que se distingue dos demais e é chamado de *raiz*. A *altura* de um vértice da árvore é definido pela quantidade de arestas que existe no caminho entre o vértice e a raiz da árvore e o *grau* de cada vértice é a quantidade de filhos que ele possui. As árvores são consideradas *ordenadas* quando existe ordem entre os filhos de um vértice (para vértices de grau k, existe o primeiro, o segundo, ..., até o k-ésimo vértice) e *posicionais* quando a posição dos filhos importa, não apenas a ordem deles entre si.

Uma *árvore binária* é uma árvore posicional onde cada vértice tem no máximo dois filhos, sendo um o *filho da esquerda* e outro o *filho da direta*. Se um vértice possuir apenas um

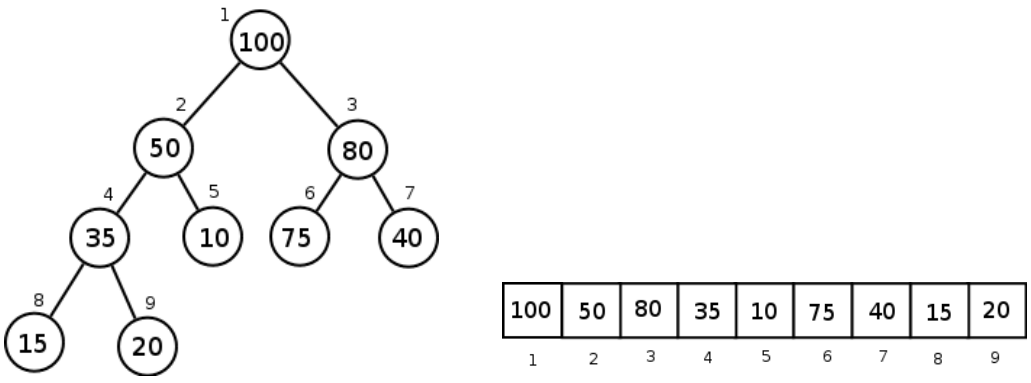
filho, o fato de ele ser o da direita ou o da esquerda é importante para a definição da árvore. Numa *árvore binária cheia* todos os nós têm grau 2 ou são folhas, e numa *árvore de alturas balanceadas* as alturas das folhas nunca diferem em mais de uma unidade.

2.6.3 HEAP

Heap é uma estrutura de dados que pode ser representada como uma árvore binária ou um vetor.

Considerando a heap como uma árvore binária, ela será completamente preenchida em todos os níveis, com exceção do nível mais baixo, onde as folhas são preenchidas da esquerda para a direita, resultando também numa árvore de alturas balanceadas. Os vértices da árvore são rotulados em ordem crescente a partir da raiz, e as chaves dos vértices obedecem à regra de crescimento monotônico da raiz para as folhas, ou das folhas para a raiz. Chama-se *heap de mínimo* uma heap em que todo nó da árvore é menor ou igual a seus filhos, sendo a raiz o menor nó da árvore. Analogamente, numa *heap de máximo*, todos os nós da árvore são maiores do que seus filhos e o maior elemento é a raiz.

Como as heaps são árvores binárias de alturas balaceadas e cheias (possivelmente com exceção do vértice da maior rótulo), elas podem ser representadas em um vetor, dispensando o uso de ponteiros explícitos para nós pai e filhos. O posicionamento dos elementos no vetor é equivalente ao rótulo de seus vértices na árvore. Desta forma, o pai de um elemento i estará na posição $\lfloor i/2 \rfloor$, seu filho da esquerda estará em $2i$ e seu filho da direita em $2i+1$.



(a) (b)

Figura 2.1: (a) Exemplo de uma heap de máximo (b) Representação da heap em vetor

Toda operação executada na heap deve considerar suas propriedades e garantir que elas ainda sejam válidas ao final da operação. Por este motivo, uma simples inserção ou edição

de um elemento da heap pode causar a realocação de vários outros elementos em diferentes vértices, a depender do valor do novo elemento e a sua relação com os outros elementos da heap.

2.6.4 ÁRVORE BINOMIAL

Árvores binomiais são árvores ordenadas, onde os filhos de cada nó aparecem da esquerda para a direita em ordem decrescente de grau, e podem ser definidas recursivamente da seguinte maneira:

1. Uma árvore binomial de ordem 0 tem somente 1 nó;
2. Uma árvore binomial de ordem k pode ser obtida anexando-se à raiz de uma árvore binomial de ordem $k-1$ uma outra árvore binomial de ordem $k-1$.

A figura 2.2 apresenta árvores binomiais de ordem 0 a 3 [BinomialTree 2006]. Cada árvore tem um nó raiz com todas as árvores de menor ordem como sub-árvores. Por exemplo, a árvore binomial de ordem 3 é ligada às árvores de ordem 2, 1 e 0 (destacadas em azul, verde e vermelho respectivamente).

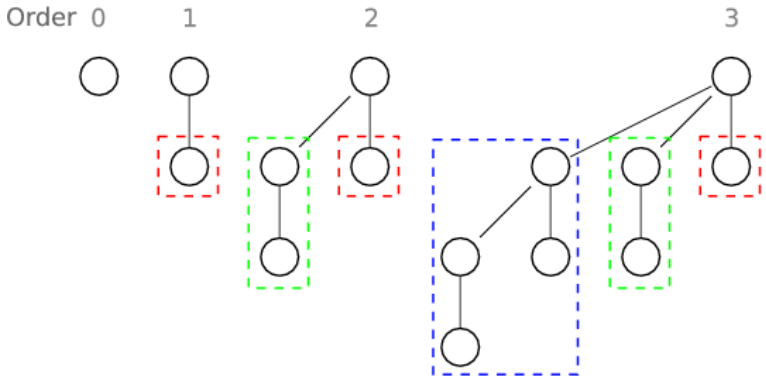


Figura 2.2: Árvores binomiais de ordem 0 a 3

A união ou *merge* de duas árvores binomiais de mesma ordem é a operação central da união de duas heaps binomiais, e pode ser deduzida a partir da própria definição de árvore binomial. A raiz de uma das árvores é escolhida para ser a raiz da nova árvore. Em seguida, a raiz da outra árvore é acoplada como filha da primeira raiz, se tornando então sub-árvore da nova árvore.

2.6.5 HEAP BINOMIAL

Uma heap binomial, também conhecida como heap intercalável ou *mergeable heap*, não é somente uma árvore, é um conjunto de árvores binomiais (uma floresta) que satisfaz às seguintes

propriedades:

1. Cada árvore binomial da heap é uma heap de mínimo;
2. A heap pode conter no máximo uma árvore binomial para cada ordem.

A definição acima descreve uma heap binomial de mínimo. Existe também uma estrutura dual, a heap binomial de máximo, na qual cada árvore binomial da heap é uma heap de máximo.

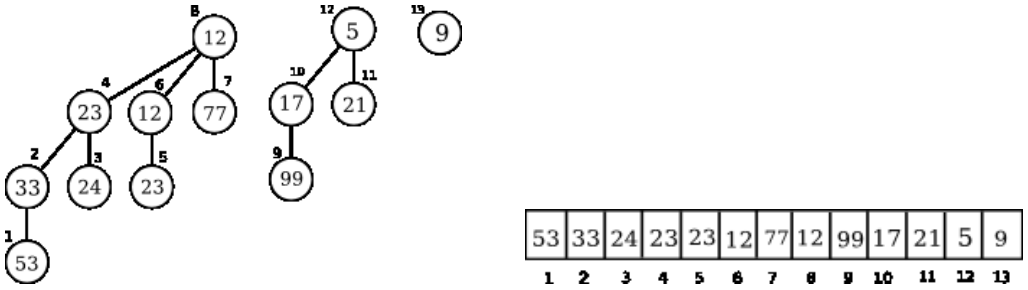
Os vértices de uma heap binomial devem ser rotulados obedecendo ao algoritmo de pesquisa em profundidade na heap binomial. Esta pesquisa realiza um percurso na heap que visita todos os filhos dos vértices antes de seus irmãos [Cormen et al. 2002]. Os rótulos não são definidos na ida, e sim na volta do percurso, no momento em que todos os filhos do vértice já foram rotulados. Deve-se escolher a raiz da árvore de maior ordem como primeira origem, fazer um percurso em profundidade até que todos os vértices da árvore tenham sido rotulados e em seguida partir para a próxima árvore de maior ordem, e assim por diante, até que todos os vértices da heap binomial tenham sido rotulados.

Considerando o endereçamento dos vértices como descrito acima, pode-se representar a heap num vetor, mapeando cada vértice numa posição do vetor de forma única. Este mapeamento, porém, não é tão trivial como o de uma heap simples.

Segundo o *Teorema Fundamental da Aritmética*, todo número natural maior do que um pode ser decomposto de forma única em fatores de números primos [Teorema Fundamental da Aritmética 2006]. Por exemplo, $10 = 2 \times 5$, $60 = 2^2 \times 3 \times 5$ e $100 = 2^2 \times 5^2$. Esta é a chamada fatoração canônica de um número. Já que 2 é o único número primo par, a fatoração canônica é composta por uma potência de 2 e potências de números ímpares. Multiplicando as potências de números ímpares tem-se um número ímpar como resultado, que será denominado i.

Segue uma análise do vértice de rótulo n. Considerando que n pode ser decomposto em $2^k \times i$, onde k é o índice da potência de dois e i é o produto das potências de números ímpares, o pai de n estará na posição $(i+1) \times 2^k$. Toma-se como exemplo o vértice 8. Oito pode ser decomposto em $2^3 \times 1^0$ (k=*3* e i=1), então o pai de 8 é $(1+1) \times 2^3 = 16$. Já para o vértice 11, tem-se k=*0*, já que 11 já é um número impar. Então i será igual a 11, e o pai de 11 será $(11+1) \times 2^0 = 12$. Estes e outros exemplos podem ser verificados na figura 2.3, onde uma heap binomial é representada em árvores binomiais e em um vetor.

A figura 2.3(a) traz um exemplo de heap binomial de mínimo contendo elementos de rótulo 1 a 13 [BinomialHeap 2006]. A heap consiste em 3 árvores binomiais de ordem 0, 2 e 3. Em 2.3(b) tem-se a representação da mesma heap em um vetor.



(a) (b)

Figura 2.3: (a) Exemplo de uma heap binomial de mínimo (b) Representação da heap em um vetor

O grande diferencial desta estrutura em comparação a uma heap simples é que o *merge* de duas heaps acontece em $O(log_2(n))$ no pior caso, contrastando com $O(n)$ pra uma heap simples. Então quando a operação de *merge* for necessária, é preferível que se use uma heap binomial, caso contrário, uma heap simples geralmente é suficiente.

O baixo custo da união de duas heaps, porém, só é garantido numa representacao com ponteiros. Representar esta estrutura sobre vetores, apesar de possível, demanda uma rearrumação no vetor que é muito custosa e muitas vezes não vale a pena. Entretanto, na situação particular onde as chaves são inseridas em ordem crescente para heap de máximo e decrescente para heap de mínimo, a representação sobre vetor se torna perfeitamente aceitável, visto que na união de duas árvores a chave inserida mais recentemente sempre se tornará a nova raiz, e não haverá necessidade de rearranjar elementos no vetor.

O *merge* acontece ordem a ordem, comecando pela ordem 0. Sempre que houver duas árvores de mesma ordem k, elas devem ser unidas, formando uma nova árvore de ordem $k+1$. Essa operação se repete até que só haja uma árvore binomial de cada ordem (propriedade 2 das heaps binomiais). Na operação de união de duas árvores de ordem k, deve-se levar em conta o valor das raízes para que a propriedade 1 das heaps binomiais seja preservada. Se as árvores forem heaps de mínimo, a que tiver o menor elemento entre as duas raízes deve se tornar a nova raiz. Analogamente, se forem heaps de máximo, a raiz que tiver o maior elemento será escolhida para ser a nova raiz.

2.6.6 HEAP BINOMIAL COM BURACOS

A propriedade 1 da heap binomial define que cada árvore binomial deve ser uma heap de mínimo ou de máximo. Mas a definição de heap supõe que todos os vértices têm conteúdo e

portanto podem ser comparados para verificação da propriedade de crescimento monotônico. A heap binomial com buracos se difere de uma heap binomial pela existência de vértices sem chave. Então as propriedades da heap precisam ser ligeiramente modificadas para permitir essa nova configuração.

Refere-se neste trabalho a *heap de mínimo com buracos* uma heap em que todo nó da árvore é menor ou igual a seus descendentes (não somente seus filhos, pois os filhos podem estar vazios), sendo a raiz o menor nó da árvore. Analogamente, numa *heap de máximo com buracos*, todos os nós da árvore são maiores do que seus descendentes, sendo a raiz o maior elemento.

Durante o *merge* de duas heaps binomiais com buracos, na união de duas árvores binomiais as raízes só podem ser comparadas se as duas chaves estiverem definidas. Se uma das raízes estiver vazia e a outra possuir conteúdo, ao final do processo a chave da que possui conteúdo deve ficar na nova raiz. No caso de as duas raízes estarem vazias, a união não demanda nenhuma operação de comparação ou realocação de chaves, as raízes são simplesmente unidas.

Como consequência da definição do *merge*, a heap binomial com buracos apresenta a seguinte propriedade:

1. Toda árvore com buracos não-vazia possui raiz não-vazia, ou seja, se qualquer um dos vértices de uma árvore tiver chave definida, a raiz da mesma não pode ser um buraco.

Sobre a representação da heap binomial com buracos, recai-se no mesmo problema da heap binomial simples. Se for possível garantir que as chaves sejam inseridas na ordem, a representação em vetor é eficiente e viável, caso contrário, deve-se optar pelo uso de ponteiros. Esta é a estrutura utilizada pelo BRF, e como no BRF essa condição é satisfeita, utiliza-se a representação em vetor sem problemas.

3 *BRF* - BACKUP, REMEMBER, AND FORGET

O BRF - Backup, Remember, and Forget, é um *software livre* escrito em 2002 sob autoria do prof. Imre Simon e tem uma proposta diferenciada dos sistemas de *backup* mais populares atualmente. O BRF otimiza o espaço utilizado por *backups* através de um mecanismo de esquecimento de dados. Alguns dados podem ser descartados em cada execução do sistema, resguardando informação suficiente para não perder a validade do esquema. Além disso, o BRF garante que a versão mais recente de qualquer arquivo que tenha sido removido ou substituído durante toda a execução do esquema jamais seja descartada. Para tanto, ele se sustenta num complexo esquema combinatório baseado em heaps binomiais com buracos.

Além da propriedade acima, a implementação atual conta com um esquema de esquecimento automático que permite a reconstrução total de aproximadamente $2 + log_2(n)$ estados anteriores do sistema, após o seu funcionamento por n etapas. Os estados anteriores reconstrutíveis estão logarítmicamente distribuídos, onde quanto mais antigo for um estado tanto menor será a probabilidade de que ele possa ser reconstruído. Esta política de preservação de dados adiante será chamada de Política de Reconstrução Logarítmica. Com uma aproximação grosseira, podemos dizer que na política logarítmica a probabilidade de que a etapa i esteja disponível para reconstrução na etapa n é $1/(n-i)$.

3.1 ARMAZENAMENTO DE DADOS

O modelo de armazenamento de dados utilizado pelo BRF é semelhante ao chamado de *espelho + incrementais reversos* [Backup 2006]. O BRF sempre guarda um espelho (cópia completa) do *backup* mais recente e os arquivos que são substituídos ao longo do tempo são estruturados como cópias incrementais reversas. Ou seja, para obter um estado anterior é necessário aplicar as diferenças incrementais reversas no *backup* atual. No entanto, os volumes armazenados não são exatamente volumes incrementais reversos, porque, além de conter os

eventos que saíram do *backup* na etapa que eles foram criados, eles podem guardar as versões mais recentes de alguns arquivos que saíram do *backup* em etapas anteriores.

3.2 *BACKUP, REMEMBER, AND FORGET*

O BRF funciona em etapas. A cada etapa o sistema de arquivos monitorado é ”fotografado”, e a fotografia gerada (a cópia do sistema) é armazenada no diretório *Backup*, que representa o B do BRF. A primeira fotografia é gerada na etapa 0, portanto o número da fotografia gerada na etapa n é $n+1$.

Os eventos que desaparecem do *Backup* ao longo do tempo, porque foram apagados ou substituídos, são guardados numa heap binomial com buracos que a partir deste momento denominaremos simplesmente de *heap*. Os rótulos dos vértices da heap representam a etapa em que os eventos nele armazenados foram removidos do *Backup*. Os conteúdos dos vértices são os eventos. Cada evento tem uma data de nascimento, que é a primeira etapa em que ele apareceu no *backup*, e pode ou não ter uma data de enterro, que é a etapa em que ele desapareceu do *backup* e entrou na heap. O parâmetro da heap (a chave usada para as comparações) é a data de enterro do evento. Um elemento é maior do que o outro quando ele saiu do *backup* numa etapa posterior à do outro. O BRF trabalha com heaps de máximo, portanto, à medida que aproxima-se das raízes, elementos cada vez maiores (isto é, mais recentemente enterrados) são encontrados.

Conceitualmente existe uma heap para cada conjunto de eventos de mesmo nome que aparecem no esquema (isto é, para todas as versões removidas de um mesmo arquivo). A cada etapa, um novo vértice contendo a versão do evento que acabou de sair do *Backup* é criado e unido à heap da etapa anterior. Desta maneira, a heap pode conter buracos (vértices vazios), considerando a possibilidade de o evento não ser modificado entre duas etapas subseqüentes.

Os vértices da heap dividem-se entre *Remember* e *Forget*, que representam R e F do BRF. As raízes de todas as árvores da heap ficam em *Remember*, e todos os outros vértices ficam em *Forget*. O esquema permite a reconstrução de todas as fotografias cujos números são os rótulos dos vértices de R e alguns de F, o que depende da política de reconstrução adotada (ver adiante). O BRF também garante que a versão mais recente de qualquer arquivo que foi removido do *Backup* estará sempre em R, portanto, quando os vértices em F são descartados o esquema não perde sua validade.

No momento da união de um novo vértice à heap de um arquivo é possível que uma versão anterior do mesmo seja ”promovida”, ascendendo um nível na árvore. Isto acontece quando não

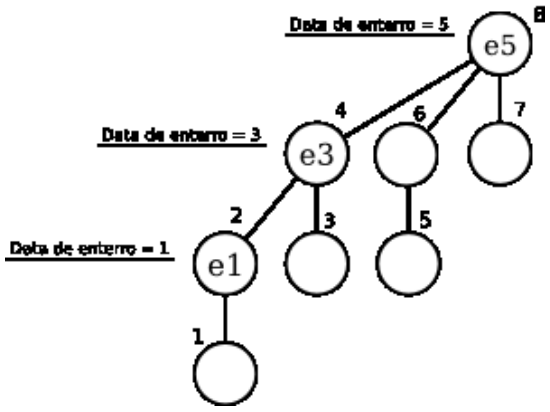


Figura 3.1: Heap binomial com buracos

existe uma nova versão de tal arquivo saindo de B para ser incorporada à heap, então a versão que já estava na heap pode ser a última versão do arquivo no esquema, portanto deve ser mantida em R (em uma raiz). O novo vértice vazio é unido à heap, e nas operações de união entre as árvores binomiais seu conteúdo sempre perde a competição e o conteúdo do outro vértice será necessariamente promovido para a raiz da árvore resultante. A figura 3.2 ilustra o momento da união de um vértice vazio à heap do arquivo *e*. O vértice de rótulo 4 é unido ao vértice de rótulo 3, pois ambos são árvores binomiais de ordem 0. Esta união resulta em uma árvore binomial de ordem 1. Neste momento, o evento *e*3 é promovido para o vértice 4, para que seja mantido em R. Por fim, as duas árvores de ordem 1 são unidas.

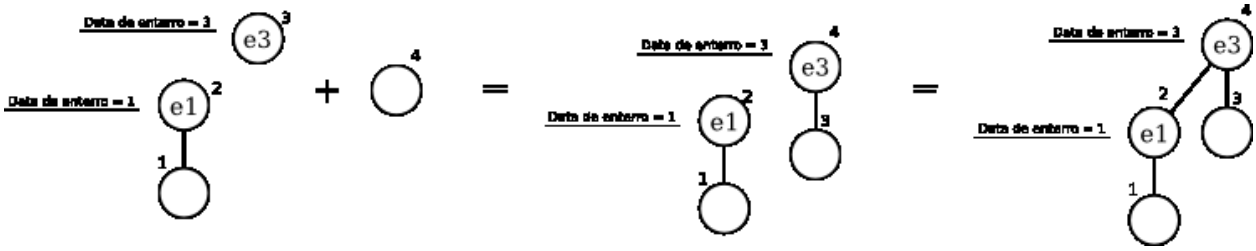


Figura 3.2: União de duas heaps binomiais com buracos

Considere que *x* e *y* são duas versões de um mesmo arquivo que competem por promoção. O *x* ganha e continua subindo, até encontrar outro arquivo *z* que pare sua subida. Entre *x* e *y*, assim como entre *x* e *z* existem ”buracos”, são vértices sem conteúdo. Estas operações podem ser melhor compreendidas na análise das figuras dos apêndices A, B e C, onde representamos o crescimento das heaps dos arquivos *a*, *b* e *c* ao longo de 22 execuções do BRF, assumindo que todos os arquivos são apagados na última etapa.

A propriedade invariante deste processo é a seguinte. Seja *x* o rótulo da raiz de uma das árvores na heap e seja *X* o intervalo de etapas constituído pelos rótulos dos vértices da árvore. Está na raiz da árvore a versão do arquivo mais recentemente enterrado durante as etapas em

X. A figura 3.3 representa o estado da heap do arquivo b na etapa 15. Neste momento existem 4 raízes na heap, que são os vértices de rótulo 8, 12, 14 e 15. O vértice 8 guarda a versão de b mais recentemente enterrada, entre as etapas 1 e 8, que é o evento $b5$. O vértice 14 guarda o $b8$, que é a versão mais recentemente enterrada entre as etapas 13 e 14. E os vértices 12 e 15 estão vazios, portanto suas árvores são vazias, indicando que nenhuma versão do arquivo foi enterrada entre as etapas 9 e 12, nem na etapa 15. A evolução completa desta heap pode ser acompanhado no apêndice B.

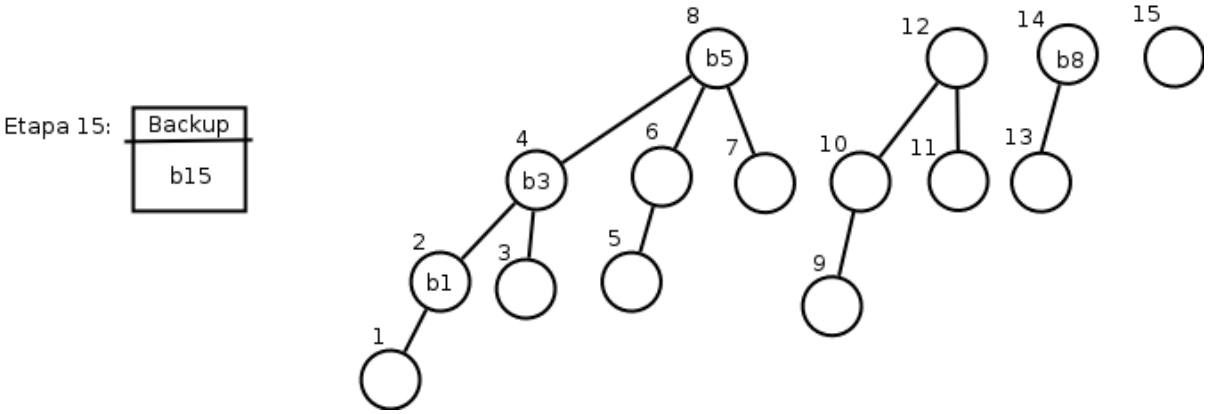


Figura 3.3: Etapa 15 da heap do arquivo b

Posicionando as heaps de todos os eventos paralelamente e projetando-as num plano, tem-se um sistema de arquivos que representará a evolução dos eventos no BRF e indicará onde os arquivos se encontram fisicamente. Nesse sistema de arquivos resultante, cada vértice da heap funciona como um volume de backup incremental reverso (adicionado de alguns arquivos promovidos de etapas anteriores). São inúmeras as possíveis configurações das heaps binomiais de um BRF, que depende da freqüência com que cada arquivo muda de versão. Os eventos são promovidos independentemente uns dos outros, e num dado momento nota-se eventos de diversas etapas diferentes que ocupam o mesmo vértice nas heaps.

Em uma determinada etapa do BRF um determinado evento pode estar em:

- **Backup**: se fizer parte da última fotografia;
- **Remember**: se estiver em uma das raízes da heap, podendo ser a versão mais recente de um arquivo na heap;
- **Forget**: se estiver em qualquer nó não-raiz da heap, e certamente não será a versão enterrada mais recente de nenhum arquivo, podendo ser descartada a qualquer momento.

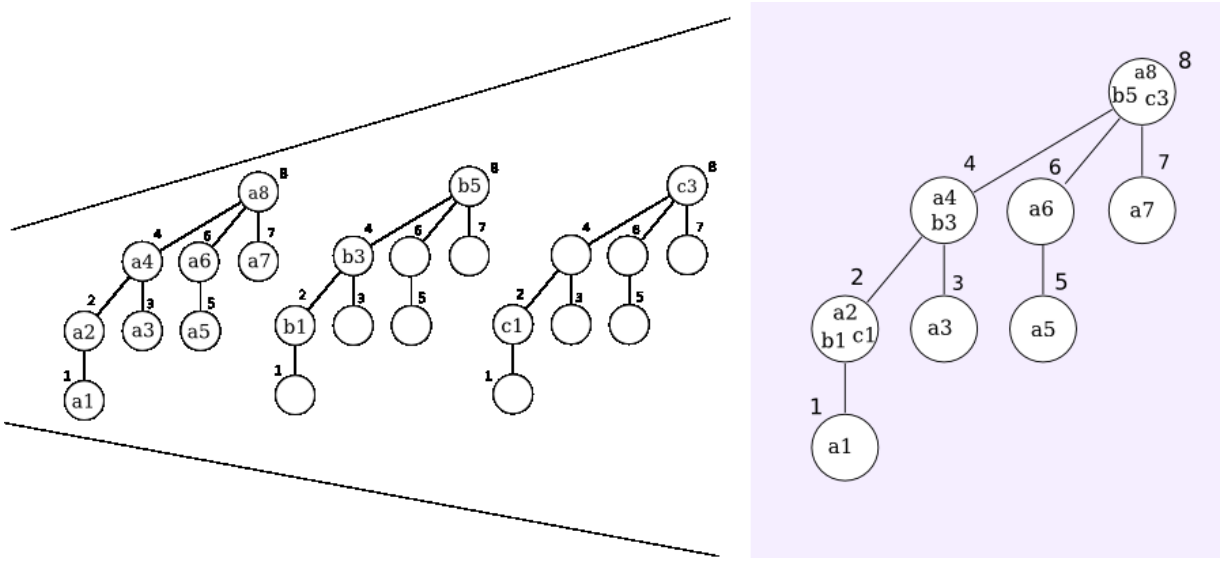


Figura 3.4: Projeção das heaps dos arquivos *a*, *b* e *c* sobre um plano, na etapa 8 do BRF

3.3 RECONSTRUÇÃO DE ESTADOS ANTERIORES

Para restaurar o sistema de arquivos de um estado anterior do *Backup*, referente à fotografia n que foi gerada na etapa $n-1$, é necessário que estejam disponíveis:

1. O *Backup* atual
2. Todos os sistemas de arquivos no caminho entre o vértice n e a raiz da árvore da heap que contém n, incluindo o próprio vértice n e a raiz.

O BRF mantém duas bases de dados durante toda a sua execução, ambas na forma de arquivos cujas linhas correspondem a eventos. A primeira contém todos os eventos que estão em *Backup* (B) na etapa atual, com informação de qual etapa cada um entrou em B (sua data de nascimento), e a outra contém todos os eventos que estão na heap (R+F), com informações sobre as etapas em que os eventos entraram, respectivamente, em B e na heap, isto é, as datas de nascimento e de enterro do evento.

A cada etapa do BRF, o *Backup* é substituído por uma nova fotografia e atualizações nessas bases de dados devem ser realizadas. Os eventos que estavam em B, R ou F e continuaram lá não devem sofrer modificação alguma. Os eventos que estavam em B e desapareceram devem ser retirados da base de dados do *Backup* e inseridos na base da heap, indicando as etapas em que ele entrou e saiu de B.

Para o entendimento dos passos a seguir considera-se um volume incremental reverso como o conteúdo de um vértice da heap resultante da união das heaps de todos os eventos.

A reconstrução de um estado do *Backup* anterior à execução da etapa n, ou seja, a reconstrução da fotografia n, gerada na etapa $n-1$, acontece da seguinte maneira:

1. O *Backup* atual é restaurado;
2. Todos os volumes incrementais reversos que se encontram no caminho entre a raiz da árvore e o vértice n devem ser aplicados ao *Backup* em ordem decrescente de rótulos, até a aplicação do incremental correspondente a n;
3. Através de consultas às bases de dados do BRF, deve-se apagar os arquivos excedentes do sistema resultante, que não faziam parte do *Backup* na etapa $n-1$.

3.4 POLÍTICAS DE RECONSTRUÇÃO

A parametrização do BRF se refere à disponibilidade de reconstruções de estados anteriores, que está relacionada diretamente com a quantidade de vértices em *Forget* que são mantidos ao longo da execução do BRF. Quanto maior for a preservação de *Forgets*, mais fotografias estarão disponíveis para reconstrução. A seguir são listadas algumas políticas que podem ser implementadas para o BRF, com gráficos que indicam as fotografias completamente reconstituíveis em cada etapa.

- Política de reconstrução Minimal:

  Esta é a política mais simples e mais econômica, onde nenhum *Forget* é mantido. Neste caso, apenas as fotografias presentes em R poderão ser recuperadas por completo. A figura 3.5 traz o gráfico de fotografias totalmente reconstituíveis para esta política.

- Política de reconstrução Maximal:

  Esta é a política mais custosa em termos de espaço, pois nenhum *Forget* é apagado. Com esta política, as fotografias de todas as etapas são disponibilizadas para recuperação total. A figura 3.6 traz o gráfico de fotografias totalmente reconstituíveis para a política Maximal.

- Política de reconstrução Logarítmica:

  Esta é a política que está implementada no BRF atualmente. Os vértices da heap são armazenados nos diretórios *remember-k*, sendo k igual ao grau do vértice. Apenas um *remember-k* para cada k é mantido, entre *Remembers* e *Forgets*, dando sempre prioridade aos *Remembers*. Chamamos de *remember-k* ativos os que guardam vértices em *Remember*

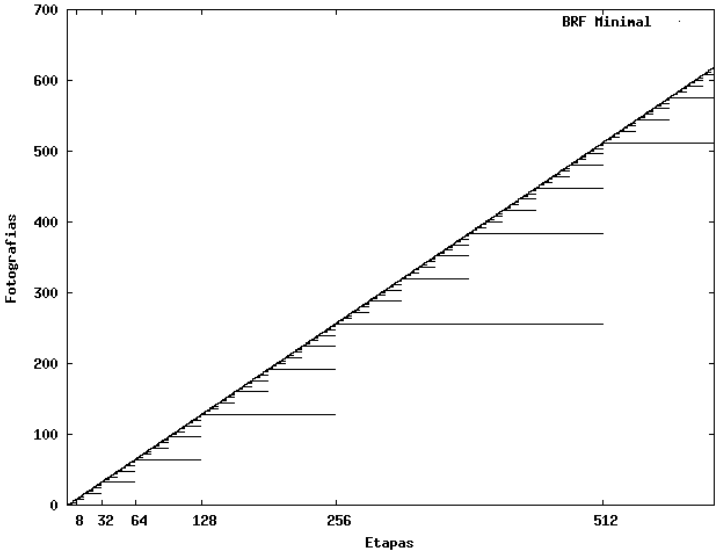


Figura 3.5: Fotografias reconstituíveis na *Política Minimal do BRF*

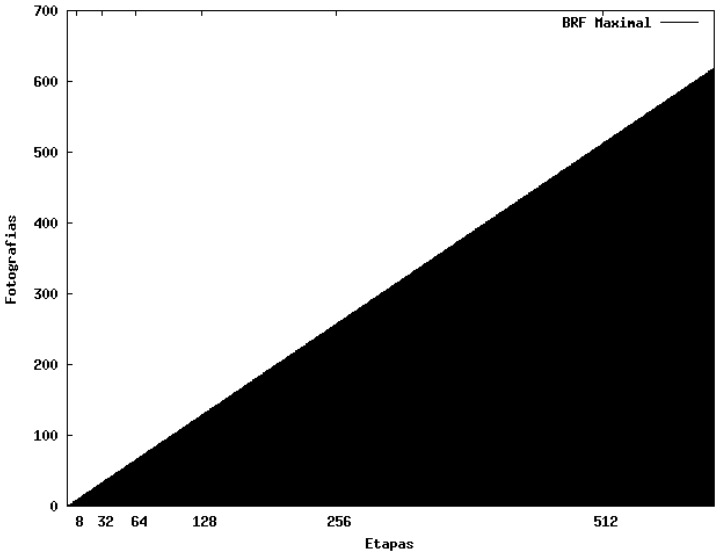


Figura 3.6: Fotografias reconstituíveis na *Política Maximal do BRF*

e inativos os que guardam vértices em *Forget*. Nesta política, a quantidade de fotografias disponíveis para reconstrução numa determinada etapa n é igual a $\lfloor log_2(n)+1 \rfloor$. Para saber quais são as fotografias disponíveis deve-se analisar a heap binomial de n vértices. Começando pelas árvores de menor ordem (criadas mais recentemente), seleciona-se o primeiro vértice encontrado de cada grau, entre 0 e $\lfloor log_2(n) \rfloor$.

A propriedade dos momentos completamente reconstituíveis da política logarítmica garante que o BRF sempre mantenha uma seqüência de reconstruções logaritmicamente distribuídas. Numa determinada etapa n, para cada l entre 0 e $\lfloor log_2(n) \rfloor$ existe uma fotografia que representa o estado do sistema a m etapas atrás e $\lfloor log_2(m) \rfloor$ é igual a l. Toma-se como exemplo a etapa 590 do BRF. É possível então restaurar os dados em B, que representam a fotografia 591, além de reconstruir as fotografias 590, 589, 588, 584, 576, 560, 544, 512, 384, e 256. Estas fotografias representam exatamente o estado do sistema há 1, 2, 3, 7, 15, 31, 47, 79, 207 e 335 etapas anteriores. Esses números se intercalam entre as potências de 2 numa seqüência crescente: $1 \leq 2^0 < 2 \leq 2^1 < 3 \leq 2^2 < 7 \leq 2^3 < 15 \leq 2^4 < 31 \leq 2^5 < 47 \leq 2^6 < 79 \leq 2^7 < 207 \leq 2^8 < 335 \leq 2^9$.

A figura 3.7 traz o gráfico de fotografias totalmente reconstituíveis para a política Logarítmica.

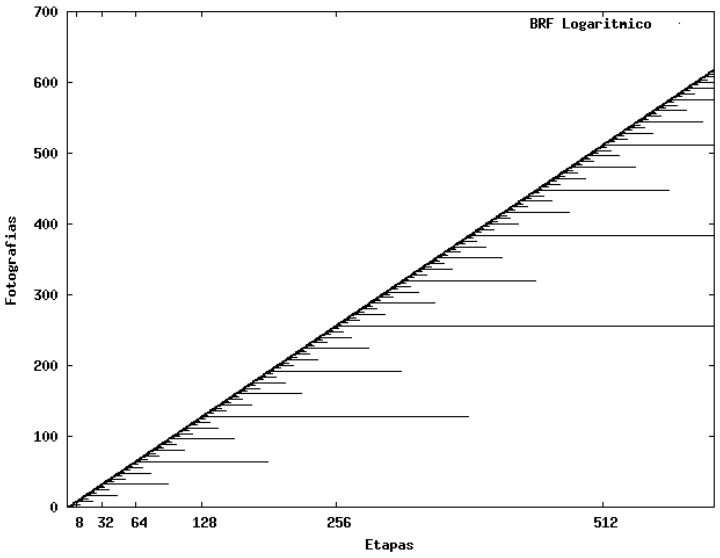


Figura 3.7: Fotografias reconstituíveis na *Política Logarítimica do BRF*

3.5 IMPLEMENTAÇÃO

Apesar da complexidade do esquema apresentado neste trabalho, o programa *rsync* [Rsync 2006] permite uma implementação bastante eficiente do mesmo. A implementação disponível

atualmente em [Simon 2002], escrita na linguagem *Perl* [Perl 2006], realiza em questão de minutos uma etapa de *backup* do BRF de um sistema de alguns *gigabytes* de dados.

4 METODOLOGIA DOS EXPERIMENTOS

A implementação do BRF disponível atualmente é apenas um piloto que serve como prova de conceito do esquema proposto, entretanto, o sistema em si ainda é rudimentar. No caso do Bacula, Amanda e outros sistemas populares, já existe uma camada de *software* para tratar de questões como a distribuição de *hosts* na rede, agendamento de tarefas, compressão de dados, entre outras funcionalidades, deixando para o usuário basicamente a tarefa de definir a política de *backup* a ser utilizada. Sendo assim, seria inadequado comparar as ferramentas no que diz respeito às suas funcionalidades. Os experimentos são concentrados na comparação de precisão da reconstrução, demanda de espaço exigido, número de reconstruções completas disponíveis e dispersão das reconstruções para cada uma das políticas analisadas.

A fim de simplificar nossa análise, a partir deste momento restingimos o conceito de política de *backup* para a combinação entre os fatores: (1) Sistema para gerenciamento de *backup* utilizado; (2) Método de rotação entre dispositivos e (3) persistência dos dados.

A análise realizada neste trabalho compara o esquema do BRF com as práticas habituais adotadas por usuários de sistemas de *backup*. O Bacula é utilizado como modelo deste estudo, pelo fato de ser um sistema bastante popular entre administradores de sistemas e atualmente de maior domínio pela autora deste trabalho.

A seguir são detalhados alguns aspectos importantes da realização dos experimentos.

4.1 HARDWARE DISPONÍVEL

Foram utilizadas três máquinas para preparação e execução de todos os experimentos:

1. *Sarah*: máquina pessoal de Imre Simon. Configuração: iBook da Apple com PowerPC G4 1.2Ghz, 1.25GB de memória RAM, disco rígido de 60GB.

2. *Camoes*: maquina pessoal de Tássia Camões. Configuração: Macbook Pro da Apple com

processador Intel Core Duo 2GHz, 512MB de memória RAM, disco de 60GB.

3. *Marco*: máquina de trabalho de Imre Simon no IME-USP. Configuração: PC com processador Intel Pentium IV 3GHz, 1GB de memória RAM e dois *HD*'s SATA de 250GB cada.

Todas as máquinas têm uma configuração moderada em termos de processamento, memória e disco. Diante do volume de dados que foi preciso tratar, a complexidade de espaço e tempo dos algoritmos usados sempre precisou ser levado em consideração. Para a manipulação inicial dos dados, que computacionalmente foi a parte mais pesada, *Marco* foi a máquina utilizada. E a execução dos experimentos propriamente dita foi realizada em *Camoes*. *Sarah* também foi usada em alguns momentos, para processamentos mais leves.

4.2 ANONIMIZAÇÃO DOS DADOS

Os dados de entrada dos experimentos são um conjunto de metadados de 10,9 milhões de eventos, que representam cada versão de cada arquivo que já fez parte do esquema. Eles foram coletados a partir de fotografias diárias de um sistema computacional do IME-USP, realizadas num período de 617 dias, entre os anos de 2002 e 2003. Para evitar qualquer possibilidade de problemas com questões de privacidade dos usuários, os dados foram anonimizados antes de serem disponibilizados para o experimento. Esta anonimização resultou na geração de nomes de arquivos numéricos para substituir os nomes reais. Por exemplo, o arquivo hipotético /home/maria/diretorio/arquivo poderia ser traduzido para /16/3466217/1014407/6135382.

4.3 FATORAÇÃO DOS DADOS

As fotografias foram produzidas utilizando o comando *find* do UNIX, e seus nomes são referentes às datas da observação do sistema de arquivos. O volume total das fotografias é de aproximadamente 90GB ou 14GB se comprimidos com gzip.

Cada linha das fotografias tem o formato: $\langle$ *tipo epoch tamanho nome* $\rangle$, onde *tipo* pode ser "f"(arquivo), "d"(diretório) ou "l"(link), *epoch* é a data de última modificaçao do evento (em segundos desde 00:00:00 01-01-1970 UTC), *tamanho* é o espaço ocupado pelo evento em *bytes* e *nome* é o caminho completo para o evento no sistema de arquivos. O exemplo abaixo representa um arquivo que foi modificado pela última vez em 14:05:33 29-08-2002 (1030640733 segundos após 00:00:00 01-01-1970 UTC), tem 1780 *bytes* de tamanho e /16/3466217/1014407/6135382

é o caminho do arquivo no sistema depois de anonimizado. O número um que se segue foi introduzido para facilitar algumas manobras com expressões regulares na manipulação dos dados, mas não têm relevância alguma no momento.

f 1030640733 1780 |/16/3466217/1014407/6135382|1

Devido ao grande volume de dados, a pura manipulação da entrada foi um problema computacional substancial. Trabalhar diretamente com as 617 fotografias, cada uma contendo cerca de 130MB de dados (20MB se compactados), mostrou-se inviável, diante do *hardware* disponível para realização dos experimentos. Então as fotografias foram fatoradas para um só arquivo, contendo uma linha para cada evento. Esta linha representa toda a sua trajetória no esquema. Desde o seu nascimento, morte, até sua acomodação num vértice da heap (usado somente para os experimentos com o BRF). Nesta forma fatorada, chamada de eventos acomodados, temos o seguinte formato de linha: $\langle$ *V: E S tipo epoch tamanho nome* $\rangle$, onde V é o vértice da heap que o evento está acomodado ao final do esquema, E é a etapa em que o evento entrou em backup, S é a etapa em que o evento saiu de backup, e os outros parâmetros têm o mesmo significado que eles tinham nas fotografias. A seguir é apresentada a linha do arquivo de eventos que representa o mesmo arquivo do exemplo anterior. Ele apareceu no *backup* na etapa 178, saiu do *backup* na etapa 311, quando entrou na heap, e foi promovido na heap diversas vezes até se acomodar no vértice 512.

512: 178 311 f 1030640733 1780 |/16/3466217/1014407/6135382|1

Esta linha no formato fatorado representa o aparecimento da linha do evento no formato original em todas as fotografias entre as etapas 178 e 310. Deste modo são eliminadas as repetições de linhas em fotografias consecutivas em que o evento não sofreu modificação e estava estacionário no espelho *Backup*. Após esse processo, a entrada foi reduzida para apenas um arquivo de 800MB que contém todos os dados necessários para o experimento, evitando a abertura dos 617 arquivos das fotografias, que se mostrou um problema.

4.4 DESENVOLVIMENTO DE SIMULADORES

Embora tenha-se feito referências às ferramentas BRF e Bacula, nenhum dos dois programas foi utilizado diretamente no experimento. As análises comparativas são baseadas em *softwares* que simulam os esquemas dos sistemas analisados, pois devido à limitação de tempo, a observação de sistemas reais em produção não foi viável.

Foram desenvolvidos simuladores para o Bacula e o BRF, na linguagem de programação

python [Python 2006], capazes de simular as diferentes políticas que desejávamos comparar. Os dois simuladores geram relatórios que servem como base para a análise, além de oferecerem informações próprias dos esquemas simulados por eles. No caso do BRF, por exemplo, o simulador informa o espaço ocupado por B, R e F em cada etapa.

4.5 REALIZAÇÃO DOS EXPERIMENTOS

Os experimentos foram divididos em duas fases. A primeira se concentra no estudo e análise do esquema do BRF e como ele se comporta diante de diferentes cenários. Na segunda fase é realizada uma análise comparativa entre diferentes políticas de backup que podem ser implementadas com o BRF ou Bacula. Após a execução dos experimentos, o Gnuplot [Gnuplot 2006] foi utilizado para produção de gráficos a fim de facilitar a análise e compreensão do leitor.

5 *EVOLUÇÃO DO BRF A LONGO PRAZO*

Na primeira fase dos experimentos foi analisada a evolução do BRF num sistema de grande porte, observando como ele se comporta a longo prazo. A princípio foram verificados todos os dados de uma só vez, mas notou-se que apenas esta análise não seria suficiente para entender a razão dos fenômenos que se apresentavam nos resultados. Então partiu-se para a tentativa de isolar alguns tipos de dados que tinham um comportamento diferenciado e assim foram obtidos resultados mais precisos.

Ao longo do texto será utilizado B para representar os dados em Backup, R para os vértices em Remember, F para os vértices que estão em Forget e L para um subconjunto de F que equivale aos vértices disponíveis para a política logarítmica, em cada etapa.

5.1 COMPORTAMENTO DO BRF

Os dois primeiros fatores analisados na adoção das diferentes políticas do BRF foram a quantidade de arquivos e o espaço ocupado pelos dados disponíveis para recuperação em cada etapa do processo. Para a política minimal, estes valores são equivalentes B+R, para a logarítmica são B+R+L, e para a maximal B+R+F.

Os gráficos das figuras 5.1 e 5.2 foram gerados a partir dos primeiros resultados obtidos. Nos dois casos, o *Backup* se comporta de forma estável, como uma linha praticamente horizontal. As outras políticas se apresentam através de curvas que se assemelham a retas, com inclinações crescentes na ordem: Minimal, Logarítmica e Maximal. As inclinações encontradas aqui representam o número de arquivos por etapa ou o espaço ocupado por etapa, a depender do que está sob análise.

Nota-se que as curvas Minimal e Logarítmica ficaram surpreendentemente próximas uma da outra, nos dois graficos. Para explicar este fenômeno foram gerados outros gráficos com

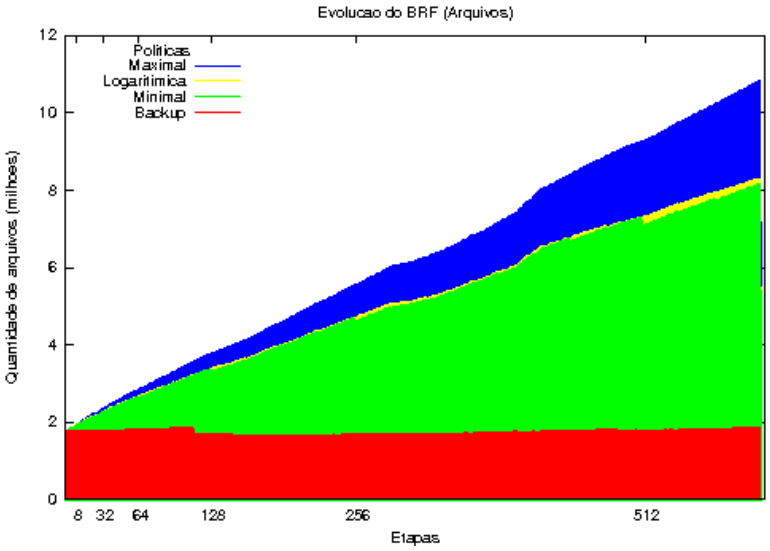


Figura 5.1: Evolução do BRF em quantidade de arquivos

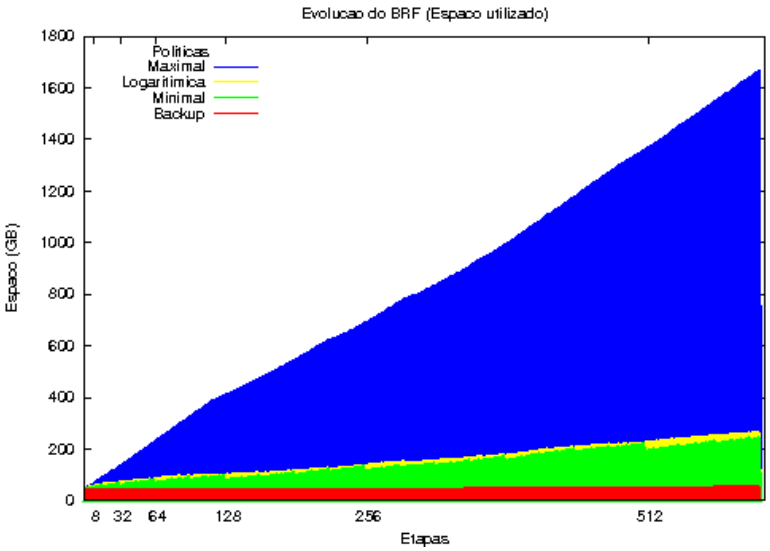


Figura 5.2: Evolução do BRF em espaço utilizado

as diferenças entre a política logarítmica e a minimal. O que ocorre no BRF é que R vai se aproximando de L a medida que o número da etapa encosta nas potências de 2, e exatamente nas potências de 2, R e L se distanciam ao máximo. Isso acontece porque nas etapas $2^k - 1$, para cada natural k, existem k árvores na heap, portanto, k vértices estão em R e os mesmos k vértices compõem L. Após uma etapa, nas etapas 2^k, só existe uma grande árvore na heap, portanto apenas um vértice está em R (a raiz), enquanto L é formado pela raiz e todos os seus filhos. Conforme as etapas vão passando, novas árvores vão sendo criadas e portanto novos vértices são incorporados a R. Até que, novamente numa etapa anterior a uma potência de 2, R possui exatamente os mesmos vértices de L, e na etapa seguinte eles se distanciam ao máximo novamente. Este comportamento acontece de forma recursiva, provocando uma natureza fractal no fenômeno, que pode ser percebida nos próximos gráficos.

Apesar de as diferenças entre as curvas verde e amarela dos gráficos 5.1 e 5.2 parecerem ser mínimas, na verdade elas são enormes, visto que as escalas dos gráficos são muito pequenas. O gráfico da figura 5.2 comportava o equivalente ao espaco de 30 discos rígidos de 60GB, enquanto na figura 5.4 a escala do gráfico é ampliada, e o espaço representado passa a comportar apenas um disco de 40GB. Para os gráficos de arquivos, a unidade de 5.1 era milhões de arquivos e na figura 5.3 passa a ser milhares.

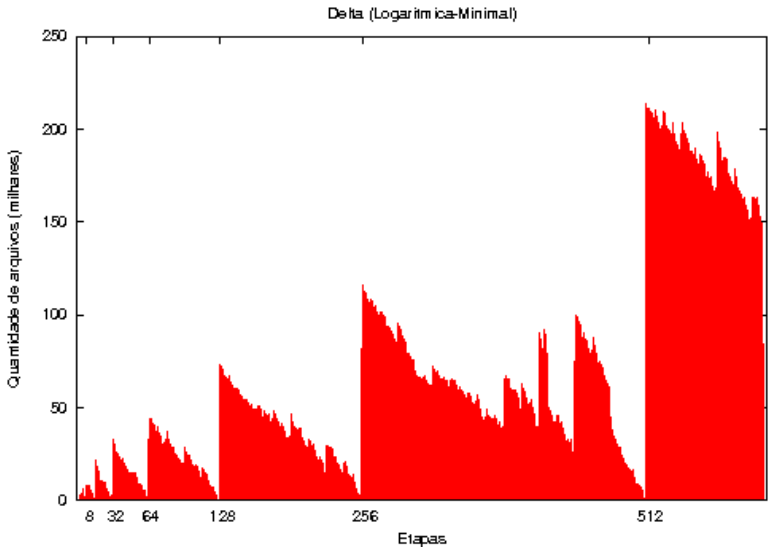


Figura 5.3: Delta de arquivos entre políticas Logarítmica e Minimal

Foram produzidos também alguns gráficos para acompanhar o crescimento dos volumes *remember-k* ao longo do tempo. Esses resultados são válidos apenas para a política logarítmica, visto que o a rotação dos *remember-k* apresentada aqui é equivalente à que está implementada atualmente no BRF. O estado *ativo* representa a etapa em que o *remember-k* foi alocado para armazenar os eventos de um vértice (de rótulo igual à etapa) e todas as etapas subseqüentes,

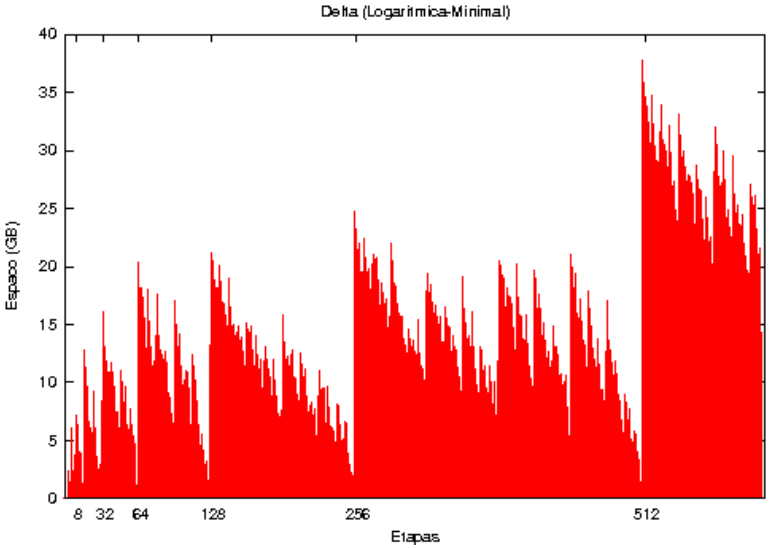


Figura 5.4: Delta de espaço entre políticas Logarítmica e Minimal

até que aconteça a promoção de alguns eventos para o vértice pai, na etapa em que o vértice pai é criado. Aí então seu estado muda para *inativo* e ele passa a conter apenas os eventos que não foram promovidos e podem ser descartados nas próximas etapas. Os gráficos 5.5 e 5.6 mostram, respectivamente, a quantidade de arquivos e o espaço ocupado por *remember-0* ao longo da etapas e os gráficos 5.7 e 5.8 trazem essas informações para *remember-4*. Os gráficos para todos os outros volumes estão disponíveis em [Camões e Simon 2006].

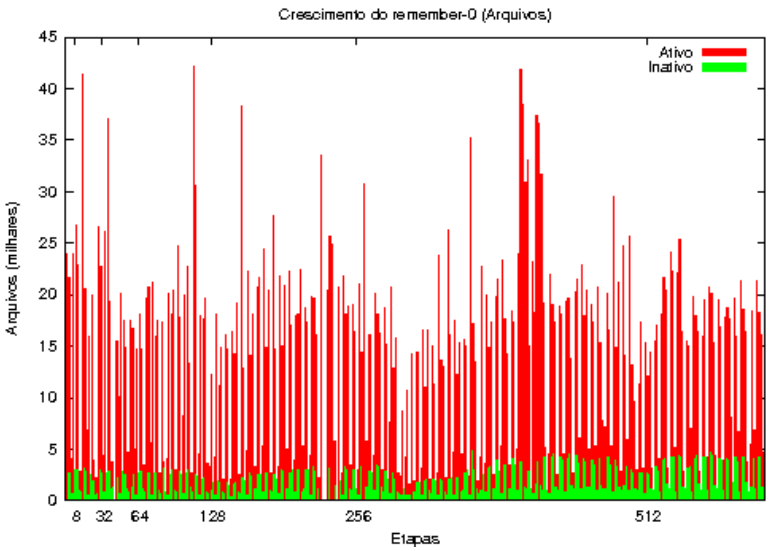


Figura 5.5: Evolução do remember-0 (arquivos)

Uma análise mais aprofundada neste tipo de gráfico poderia revelar propriedades interessantes do BRF, mas infelizmente não houve tempo hábil para o estudo neste trabalho. No entanto, aparentemente estes gráficos se comportam de forma regular para cada k. Se fosse possível

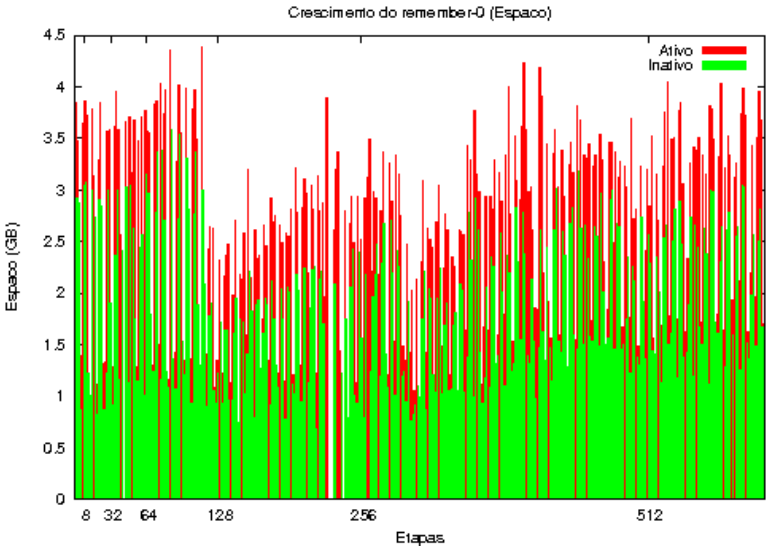


Figura 5.6: Evolução do remember-0 (espaço)

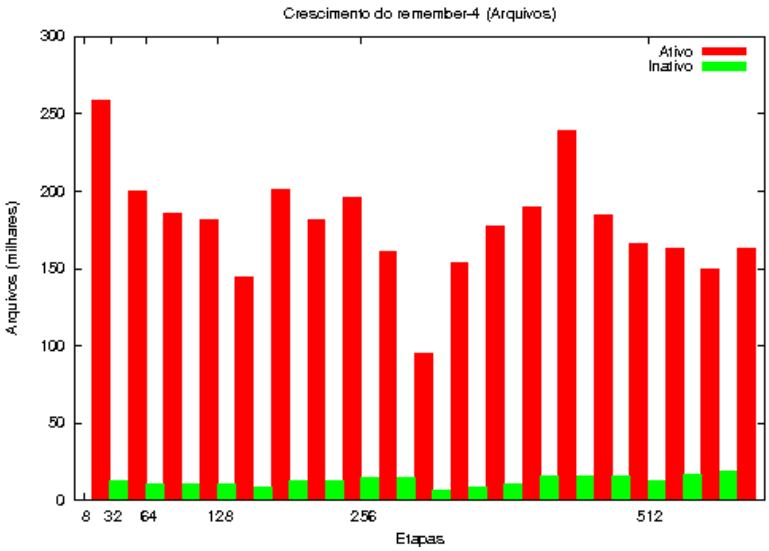


Figura 5.7: Evolução do remember-4 (arquivos)

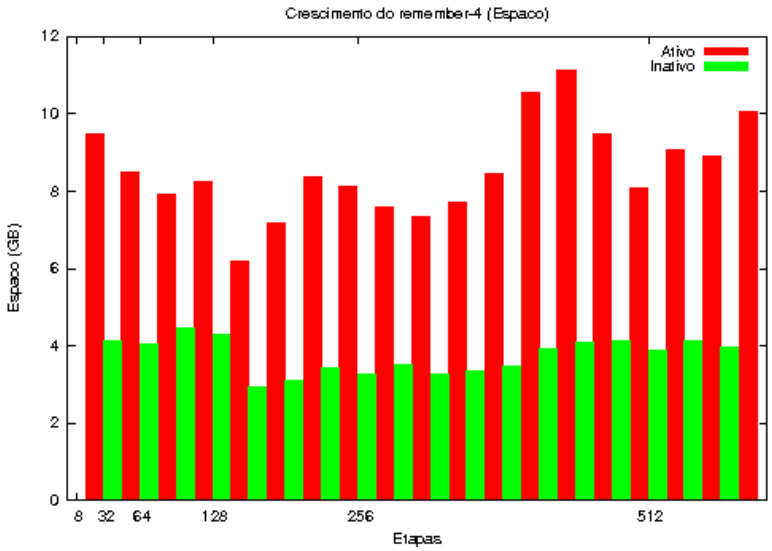


Figura 5.8: Evolução do remember-4 (espaço)

inferir sobre a variação do tamanho médio dos *remember-k* ativos e inativos para cada k, seria possível fazer previsões sobre a demanda de espaço para n etapas posteriores. Intuitivamente acredita-se que a demanda deve ter uma progressão linear, mas não é possível afirmar isso no momento.

Um detalhe importante, que talvez não esteja claro para o leitor até o momento, é que o tamanho médio dos arquivos varia bastante entre B, R, F e L para o conjunto de dados analisados. O gráfico 5.9 indica com o comprimento médio dos arquivos em B, R, F e L ao longo das etapas. Percebe-se que, justamente pelo fato de R e L se igualarem nas etapas $2^k - 1$, o comprimento médio dos arquivos de R e L também são iguais nestes momentos. E exatamente nas etapas 2^k os comprimentos médios de R e L se distanciam ao máximo, como era esperado.

Essa enorme variação entre os tamanhos médios dos arquivos não era esperada, e não pôde ser compreendida apenas com a análise quantitativa dos dados. Intuitivamente percebeu-se que tipos de dados diferentes provocam comportamentos diferenciados no BRF, e analisar o *backup* do sistema computacional do IME-USP como um todo não permitia perceber as variações sutis provocadas por cada tipo de dado. Então o conjunto de dados foi particionado em diversos subconjuntos e a mesma análise foi realizada separadamente.

5.2 SELEÇÃO DE DADOS

A seleção de dados não pôde ser feita diretamente, visto que os metadados disponíveis haviam sido anonimizados, impossibilitando qualquer separação semântica baseada nos nomes

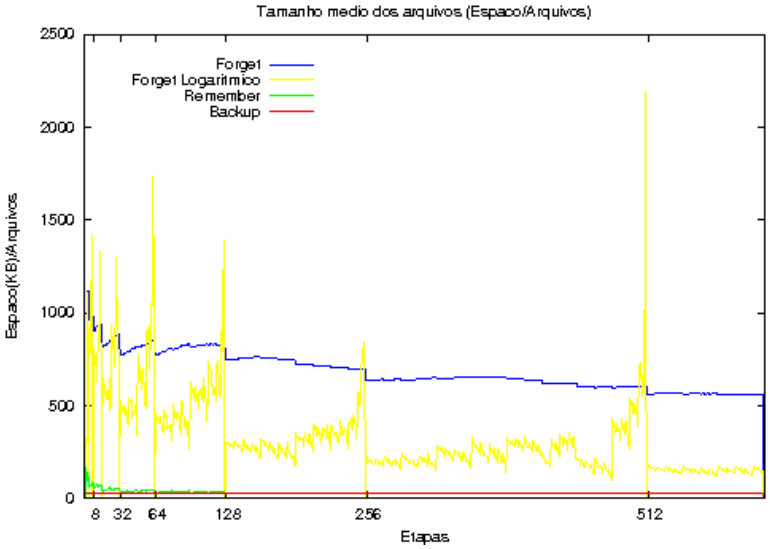


Figura 5.9: Tamanho médio dos arquivos (espaço / quantidade de arquivos)

dos arquivos. No entanto, esses subconjuntos de dados foram fornecidos para novos experimentos diretamente pela administração do servidor do IME-USP. Os subconjuntos foram gerados a partir de expressões regulares simples sobre os nomes dos arquivos, então a corretude desta classificação não é garantida. Apesar disso, acredita-se que foi obtida uma boa qualidade na classificação, que inclusive pôde ser verificada pela observação dos resultados.

Primeiramente foram criados dois subconjuntos complementares a partir dos dados completos, para separar arquivos de *e-mail* e *cache* dos demais, já que geralmente estes dados são modificados ou descartados numa frequência muito maior do que o resto do sistema. Dentro do conjunto de *e-mail* e *cache* foram criados dois novos subconjuntos para separar dados de usuários que usam *maildir* dos que usam *mbox* na sua configuração de *e-mail*.

Quando *mbox* é utilizado, todas as mensagens são armazenadas em um mesmo arquivo, uma após a outra. Cada usuário tem apenas um arquivo de *e-mail*, que é modificado cada vez que uma nova mensagem é recebida ou apagada. Quando este é o caso, a heap do BRF fica de certa forma equilibrada, já que os eventos são acomodados rapidamente pois são substituídos com freqüência.

Com *maildir* cada mensagem fica num arquivo separado, que nunca é modificado, podendo apenas ser criado e pagado. O diretório do *maildir* é dividido em *tmp*, *new* e *cur*. *tmp* é usado para armazenar arquivos temporários, assegurando confiabilidade no processo de entrega de mensagens. Quando as mensagens são entregues, elas são armazenadas em *new* com nome de arquivo *new/time.pid.host*, que caracteriza a mensagem unicamente no sistema e garante que ela jamais terá outra versão, visto que seu nome é amarrado ao momento de sua criação. Quando as

mensagens são lidas pelos clientes de *e-mail*, elas são transferidas para o diretório *cur*, podendo ser organizadas semanticamente pelo usuário em pastas. No caso de *maildir*, os eventos não têm versões diferentes, apenas aparecem e desaparecem do *Backup*, então eles são promovidos eternamente para a camada *Remember*, desde o momento que entram na heap.

O que acontece com *maildir* resulta num inchaço nas raízes das árvores que não é desejável, e é denominado *comportamento da bolha em Remember*, já que esses eventos sempre sobem na heap. Todos os arquivos que já foram apagados do *Backup* desde o início do BRF devem ter pelo menos uma versão em *Remember*, a mais recentemente apagada. E nas potências de 2, etapas 2^k, todos esses arquivos estão na raiz da heap, que contém apenas uma árvore. Se os arquivos considerados são úteis e é desejável que possamos recuperá-los posteriormente, estamos falando de uma grande qualidade do BRF. Mas no caso do *maildir*, os arquivos criados em *new* jamais serão requisitados para recuperação, já que os usuários mantém cópias dos mesmos em *cur*. Percebeu-se que esta granularidade de arquivos únicos muito elevada, em outras palavras, a criação de muitos arquivos que não terão novas versões, se configura num problema para o BRF, na medida em que esses arquivos nunca serão consultados para recuperação.

Na porção de dados para uso geral, que não contém *e-mail* nem *cache*, também foi identificado um caso particular e decidiu-se por isolá-lo. Haviam três usuários muito ”caros”em termos de *Remember*, cujos arquivos provocavam um certo distúrbio nos resultados. Estes usuários provavelmente são administradores da máquina, que por vezes criam arquivos grandes que logo são descartados e nunca mais reaparecem. Um bom exemplo disso são os fontes do *kernel* do Linux. De uma versão para outra, a maioria dos arquivos do *kernel* permanecem inalterados, mas o nome do pacote contém a versão do *kernel*. Os arquivos de cada versão são arrastados por toda a execução do BRF, o que caracteriza um comportamento de bolha mais uma vez, causado pela duplicação de arquivos com mesmo conteúdo, mesmo nome, mas em caminhos diferentes no sistema. Então estes três usuários foram separados dos demais, que, ao contrário desses três, apresentam um comportamento usual.

O *comportamento da bolha* é um problema que precisa ser melhor controlado num sistema em produção. Atualmente isso é feito pelo programa *purge*, que simula a criação instantânea de um arquivo de comprimento zero, apagando-o imediatamente a seguir. Estes dados são inseridos no mecanismo normal do BRF, o que faz com que oportunamente a cópia mais recentemente removida do arquivo tenha comprimento zero e não ocupe mais espaço em *Remember*. Há necessidade, porém, de oferecer mais ferramentas para que o usuário possa controlar e gerenciar a bolha com mais facilidade.

Abaixo são apresentados todos os subconjuntos que trabalhamos. Estes subconjuntos são

partições complementares no seu nível, ou seja, um evento jamais aparece nas duas partições do mesmo nível ao mesmo tempo.

- *E-mail* e *cache*
  - *E-mail* e *cache* com *maildir*
  - *E-mail* e *cache* com *mbox*
- Arquivos de uso geral no sistema (que não são *e-mail* nem *cache*)
  - Dados de três usuários muito caros em termos de *Remember*
  - Arquivos de uso geral e de usuários com comportamento usual

5.3 RESULTADOS OBTIDOS

A partir da observação dos resultados preliminares houve a tentativa de inferir sobre possíveis parâmetros que representariam a qualidade dos dados dados para uso no BRF, ou seja, parâmetros que, quando otimizados, ocasionassem um maior ganho na utilização do BRF.

Primeiramente foi definido o *fator de forma* como a razão do tamanho médio de arquivos em F pelo tamanho médio dos arquivos em R. Intuitivamente era esperado que quanto mais próximo de 1 fosse este fator, melhor seria o resultado obtido. Porém, essa hipótese não foi confirmada nos resultados. A exceção encontrada está na porção de dados de *e-mail* e *cache*. O conjunto de *mbox* teve resultados bem melhores que *maildir*, mas o fator de forma deste conjunto foi de 17.67, enquanto o *maildir* teve fator de forma de 5.43.

Foram investigados ainda outros parâmetros em busca do *fator de qualidade*, entre eles, a permanência média dos arquivos em B, a permanência média dos arquivos em R e a quantidade de versões que cada arquivo tem ao longo das etapas, para cada porção de dados. Houve também a tentativa de combinar alguns desses fatores, mas sem obtenção de sucesso a tempo de incluir este resultado no trabalho.

Para avaliar o custo do uso do BRF, utilizamos a demanda de espaço exigida, em cada porção de dados e para cada política. As demandas da tabela 5.1 devem ser interpretados da seguinte maneira: para executar o BRF por 617 dias utilizando uma determinada política são necessários x vezes o espaço de armazenamento dos dados em $Backup$.

Tipo de Dados	Tamanho do backup (GB)	Fator de forma	Demanda		
			Minimal	Logarítmica	Maximal
Sistema completo	58.06	17.67	4.91	5.34	33.34
E-mail e *cache*	12.62	78.52	8.82	10.36	130.20
E-mail e *cache* com *mbox*	8.82	17.41	4.32	6.15	165.14
E-mail e *cache* com *maildir*	3.80	5.43	21.98	22.31	30.90
Arquivos de uso geral	45.44	1.16	3.80	3.95	6.57
Arquivos de usuários caros	1.94	0.62	20.59	21.15	28.78
Arquivos de uso geral e de usuários com comportamento usual	43.50	1.41	2.73	2.85	5.15

Tabela 5.1: Performance do BRF para cada porção de dados

Como já era esperado, o conjunto que obteve melhor resultado no BRF foi o último analisado, *arquivos de uso geral e de usuários com comportamento usual*. O fator de forma neste caso também foi o melhor, apesar dele não funcionar perfeitamente como indicador de qualidade.

Considerando o conjunto de dados completos, conclui-se que a demanda de espaço na política logarítmica é perfeitamente aceitável, visto que com uma demanda de 5.34 pode-se usufruir de $\lfloor log_2(n) \rfloor$ reconstruções completas, além das versões mais recentes de todos os arquivos que já foram removidos do *Backup* durante toda a execução do esquema.

O fato de o fator de forma ter sido bastante elevado para o conjunto de *e-mail e cache com mbox* é explicado pela existência de muitas versões diferentes dos arquivos *mbox* na heap, a maioria delas fazendo parte de F. Como os *mbox* são arquivos muito grandes, eles interferem bastante no tamanho médio dos arquivo em F, e portanto provocam um crescimento considerável no fator de forma. Outra conseqüencia deste comportamento é a demanda da política Maximal para o mesmo conjunto de dados. Percebemos que a demanda é elevadíssima, inviabilizando esta política para este tipo de dados num sistema em produção.

Todos os gráficos gerados para o sistema completo apresentados na seção anterior também foram produzidos para cada subconjunto dos dados que analisamos. Por razões de espaço eles não foram apresentados neste documento, mas estão disponíveis em [Camões e Simon 2006].

6 ANÁLISE COMPARATIVA

Na segunda fase do experimento foi realizada uma análise comparativa de algumas políticas que podem ser adotadas usando o BRF e o Bacula, acompanhando o comportamento de cada uma delas ao longo das 617 etapas de *backup*.

6.1 PARÂMETROS PARA A COMPARAÇÃO

Foram utilizados os seguintes fatores como parâmetros para a comparação:

- Dispersão média dos estados reconstituíveis do sistema de arquivos para cada esquema. A dispersão será medida em termos de amplitude, ou seja, a diferença entre a primeira e a última etapas completamente reconstituíveis;
- Reconstrução relativa média, que é a proporção de reconstruções completas disponíveis em cada etapa relativa ao número de etapas passadas;
- Quantidade de estados reconstituíveis ao final das 617 etapas;
- Demanda de espaço exigida pela política, relativa ao tamanho do *backup*.

Uma comparação efetiva entre diferentes políticas de *backup* exige que sejam definidos cenários com características de reconstrução semelhantes. E para que o BRF tenha um poder de reconstrução semelhante aos métodos clássicos, foi definido um esquema de rotação de dispositivos para o BRF que se assemelha bastante com o método clássico.

Para recuperar estados antigos no BRF é preciso que se tenha todos os vértices ancestrais em Forget a partir do estado que se quer recuperar, até a raiz da árvore binomial (que é um Remember). O esquema Avô, pai, filho é construído com base na seguinte propriedade do BRF: considerando M como o conjunto dos p maiores múltiplos de 2^k, o pai de qualquer destes múltiplos também pertence ao conjunto M, pois o pai de um múltiplo de 2^k também é um

múltiplo de 2^k, e ainda maior. Então o que é necessário fazer é guardar os últimos p múltiplos de 2^k com a finalidade de reconstruir os backups correspondentes. Preservando sete múltiplos de 1 (2^0), três múltiplos de 8 (2^3) e dois múltiplos de 32 (2^5), tem-se como resultado uma reconstrutibilidade com a mesma precisão do esquema de rotação Avô, pai, filho com doze volumes, sendo 7 volumes para *backups* incrementais, 3 para *backups* diferenciais e 2 para *backups* completos.

Considera-se uma política como persistente quando, independente do esquema de rotação de dispositivos utilizado, os volumes de dados se tornam perenes no sistema, ou seja, não são jamais apagados. A persistência mensal é caracterizada pela preservação de apenas 1 estado do *backup* por mês. Então tornar o esquema *BRF Avô, pai, filho* persistente mensal, basta guardar todos os múltiplos de 32, ao invés de somente os dois últimos.

6.2 ESTADOS RECONSTITUÍVEIS

Alguns gráficos foram produzidos a fim de facilitar a compreensão do leitor acerca dos estados completamente reconstituíveis para cada política. Gráficos equivalentes para as políticas Minimal, Logarítmica e Maximal do BRF foram apresentados no capítulo anterior (figuras 3.5, 3.7 e 3.6) e devem ser consultados a título de comparação.

Percebe-se que o gráfico do esquema *Bacula Avô, pai, filho persistente* (figura 6.1) tem a mesma aparência do gráfico para *BRF Maximal* (figura 3.6), o que reforça a propriedade de reconstrução maximal das duas políticas.

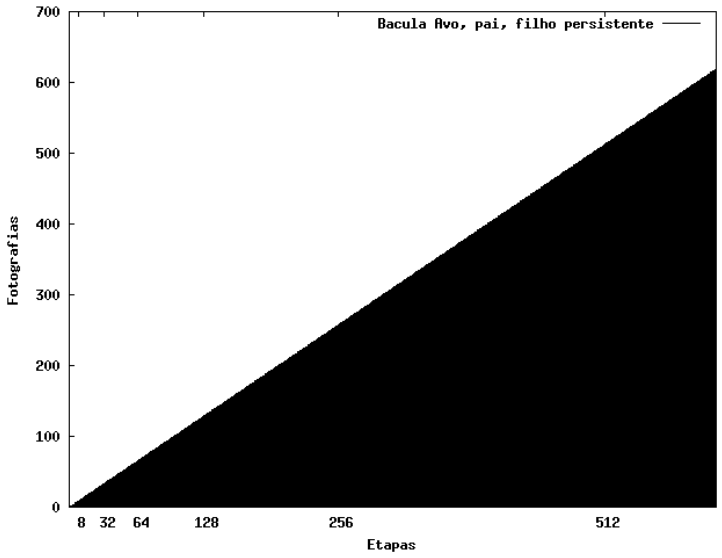


Figura 6.1: Fotografias reconstituíveis para a política *Bacula Avô, pai, filho persistente*

A propriedade comum das políticas persistentes mensais pode ser observada nos gráficos 6.2, 6.3 e 6.4. Nessas três políticas, a criação de estados persistentes acontece a cada 32 etapas. E para as políticas com o BRF, além dos múltiplos de 32, as fotografias 1, 2, 4, 8 e 16 também são perenes. Esta é a única forma de garantir que o estado inicial do *Backup* poderá ser recuperado, fazendo com que todos os sistemas de arquivos de vértices no caminho entre a raiz e o vértice 1 da heap estejam disponíveis em todas as etapas.

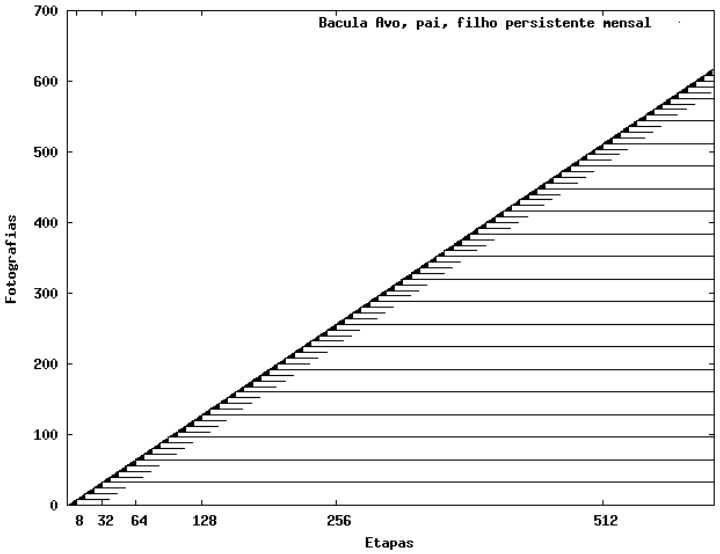


Figura 6.2: Fotografias reconstituíveis para a política *Bacula Avó, pai, filho persistente mensal*

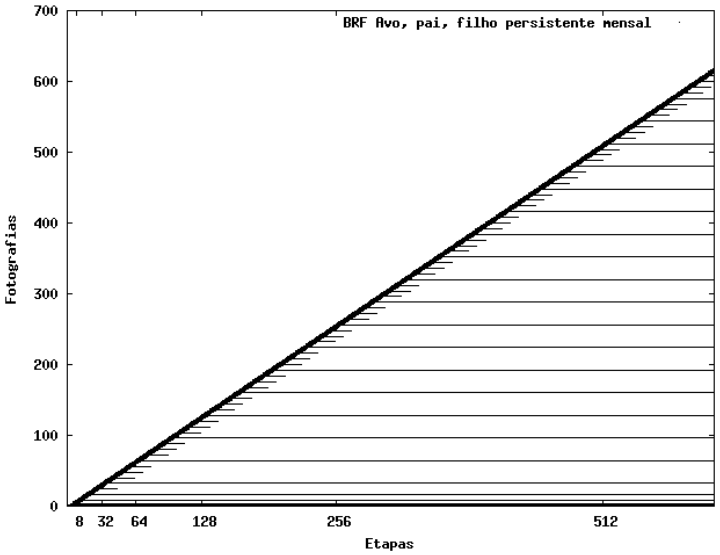


Figura 6.3: Fotografias reconstituíveis para a política *BRF Avô, pai, filho persistente mensal*

As figuras 6.5 e 6.6 apresentam os gráficos das políticas *Avô, pai, filho não persistente* para o bacula e para o BRF, respectivamente. Os gráficos são bastante semelhantes, mas o do BRF se destaca por manter como reconstituíveis todas as fotografias de vértices em R. Isso faz

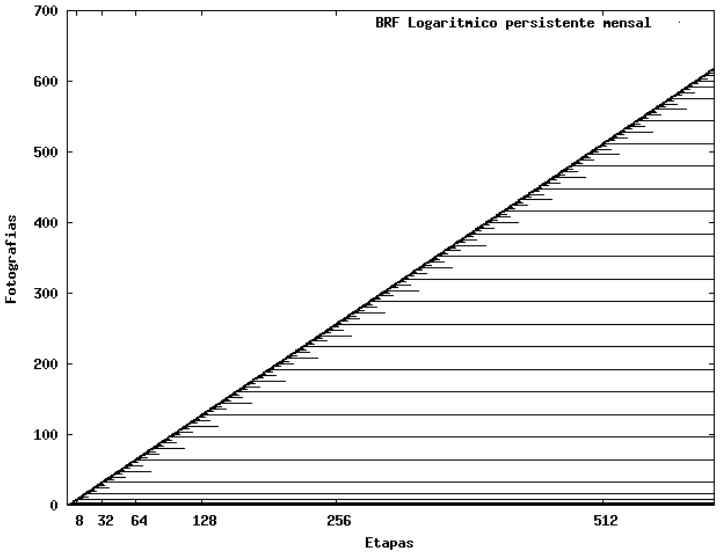


Figura 6.4: Fotografias reconstituíveis para a política *BRF Logarítmico persistente mensal*

com que o gráfico apresente características do gráfico para BRF Minimal misturadas com as características da política Avô, pai, filho não persistente.

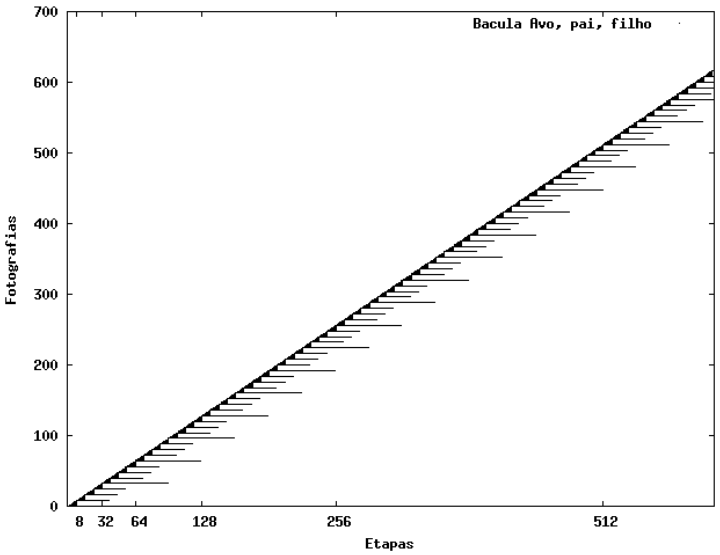


Figura 6.5: Fotografias reconstituíveis para a política *Bacula Avô, pai, filho não persistente*

O gráfico da figura 6.7 desenha as quantidades de etapas totalmente reconstrutíveis por etapa para cada política. As políticas maximais não foram consideradas neste momento porque a escala do gráfico ficaria muito pequena, dificultando assim a análise comparativa entre das políticas.

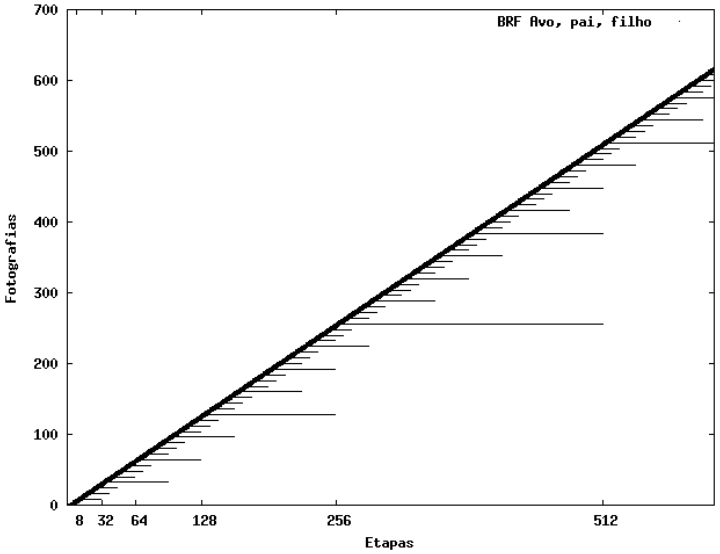


Figura 6.6: Fotografias reconstituíveis para a política *BRF Avô, pai, filho não persistente*

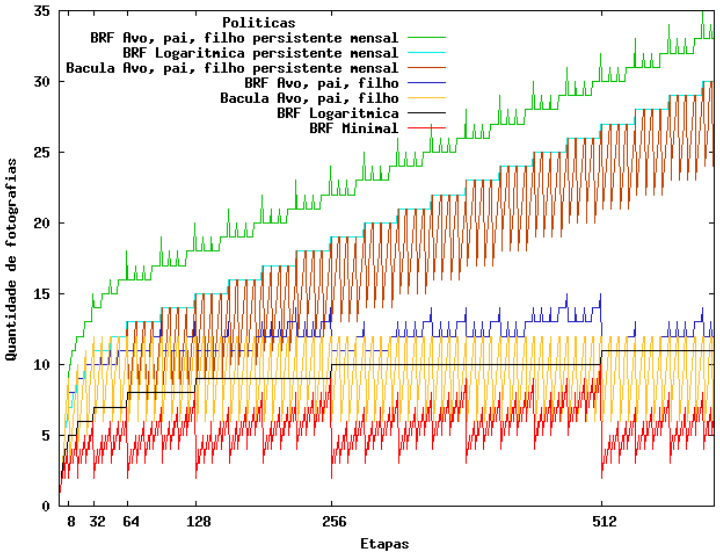


Figura 6.7: Quantidade de fotografias reconstituíveis (sem persistência total)

6.3 RESULTADOS OBTIDOS

Na tabela 6.1 são apresentados os resultados obtidos para cada política. As políticas foram agrupadas por precisão, indicando a distância entre o estado que se deseja recuperar e o estado disponível para recuperação, no pior caso. Executou-se primeiramente a simulação com o conjunto de dados completos, e em seguida selecionou-se a porção de dados que obteve melhor resultado na primeira fase dos experimentos (arquivos de uso geral e de usuários com comportamento usual). Desta forma viabiliza-se a comparação numa situação favorável e outra desfavorável para o BRF.

Ao analisar a tabela percebe-se que os resultados são bastante favoráveis para o BRF. Na maioria das situações ele consegue uma demanda de espaço menor do que a do Bacula, se políticas com a mesma precisão de recuperação forem comparadas. Isso acontece porque as redundâncias de arquivos que o Bacula mantém em cada volume de *backup* completo são eliminadas no BRF. O BRF não gera nenhuma redundância, devido ao armazenamento de dados baseado em *espelho* + *incrementais reversos*.

Por outro lado, o BRF guarda a versão mais recente de todos os arquivos que foram removidos do *Backup*, o que é uma propriedade valiosíssima, e não se mostrou muito custosa. Esta propriedade é mantida, não importa qual política é utilizada, pois estes arquivos estão sempre em *Remember*. Isto resulta num ligeiro aumento da demanda em comparação com o Bacula, mas em muitas vez o ganho da não-redundância supera esse custo extra de espaço.

Verifica-se também que a diferença entre as recuperações relativas de BRF e o Bacula é muito pequena. Este fator valida o experimento, visto que as capacidades de reconstrução foram bem aproximadas, portanto a comparação se deu de forma justa.

Política	Precisão	Amplitude Média de Dispersão	Reconstrução Relativa Média	Qtd. Rec. Final	Demanda	
					Dados Completos	Dados Selecionados
BRF Maximal	1 dia	309	100%	618	33.34	5.15
Bacula Avo, pai, filho		309	100%	618	55.65	26.58
BRF Logarítmico	$2^(k+1)-2^k$, para estados em $[2^k, 2^(k+1)]$	149	6.3%	11	5.34	2.85
BRF Logarítmico persistente mensal	$2^(k+1)-2^k$, para estados em $[2^k, 2^(k+1)]$ e no máximo de 1 mês	279	7.8%	25	13.03	5.98
BRF Avô, pai, filho persistente mensal	1 dia na última semana, 1 semana no último mês e 1 mês nos meses anteriores	279	9.8%	33	13.15	5.98
Bacula Avô, pai, filho persistente mensal		308	9.5%	24	20.40	20.14
BRF Avô, pai, filho não persistente	1 dia na última semana, 1 semana no último mês e 1 mês no penúltimo mês	87	8.6%	12	5.42	2.82
Bacula Avô, pai, filho não persistente		45	7.6%	6	2.35	2.12
BRF Minimal	$[n-(\lfloor log_2(n) \rfloor+1), n]$	80	4%	6	4.91	2.73

Tabela 6.1: Performance das diferentes políticas analisadas

7 CONCLUSÃO

Neste trabalho foi realizado um estudo detalhado do BRF, acompanhando seu comportamento em simulações a partir de dados reais. Além deste estudo, foi feita uma análise comparativa entre políticas implementadas no BRF e no Bacula.

A depender da natureza dos dados do sistema monitorado e como eles são criados, modificados e descartados ao longo do tempo, resultados melhores ou piores para o BRF são obtidos. No geral, os resultados foram bastante satisfatórios, confirmando as expectativas com relação ao espaço ocupado por ele ao longo do tempo para diferentes políticas [Simon 2002]. O custo de se manter as versões mais recentes de todos os arquivos que foram removidos ou substituídos desde o início do esquema é relativamente baixo, e certamente muitos administradores de sistemas pagariam este preço para ter um sistema com tamanha capacidade de recuperação.

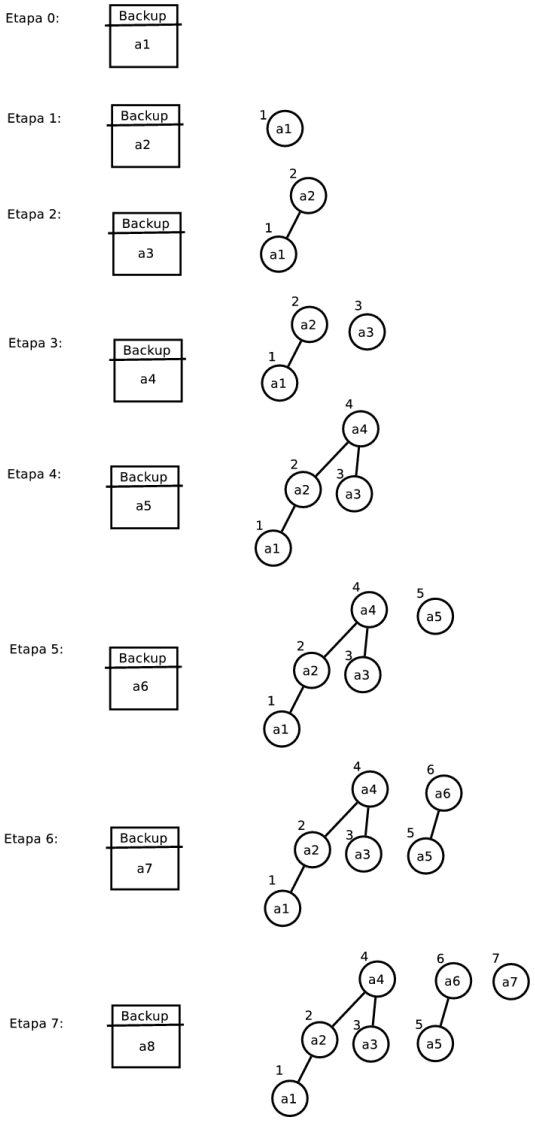
Todos os programas usados nos experimentos, assim como os dados de entrada, resultados, gráficos e tabelas geradas estão disponíveis em [Camões e Simon 2006]. Uma lista destes dados está disponível no apêndice D.

Como trabalhos futuros, pretende-se investir na remodelagem do BRF, possivelmente fazendo uma implementação na linguagem de programação Python [Python 2006]. Assim haverá a oportunidade de rever as funcionalidades do sistema, estender sua implementação para permitir que diferentes políticas sejam adotadas, melhorar a solução existente para controlar a bolha, etc. Outros aspectos teóricos do BRF também merecem ser investigados mais a fundo. A questão da previsibilidade da demanda de espaço do BRF ao longo do tempo e um fator de qualidade efetivo para os dados são temas aos quais pretende-se dedicar.

Certamente a maior contribuição deste trabalho foi a produção deste documento, que é o melhor disponível até o momento para o entendimento do BRF. Ele será útil para a divulgação do BRF na comunidade acadêmica e de Software Livre, podendo servir de incentivo para que outras pessoas passem a colaborar com o projeto.

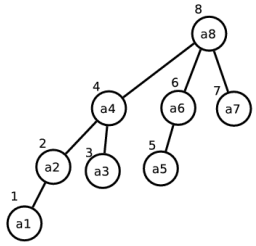
APÊNDICE A – HEAP DO ARQUIVO A

O arquivo *a* foi modificado nas etapas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 e 21. Considere *a*1, *a*2, *a*3, ..., *an* as diferentes versões do arquivo a.



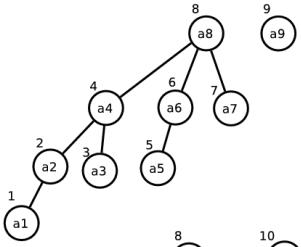
Etapa 8:

Backup
a9



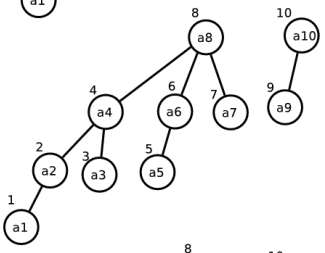
Etapa 9:

Backup
a10



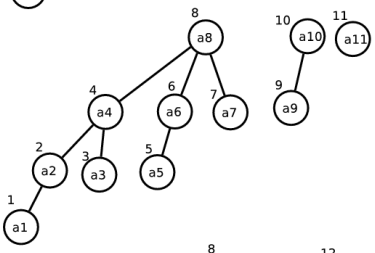
Etapa 10:

Backup
a11



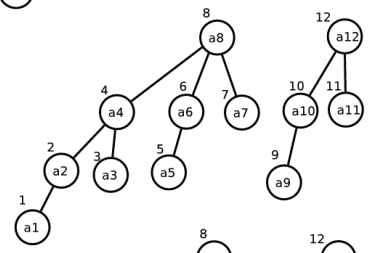
Etapa 11:

Backup
a12



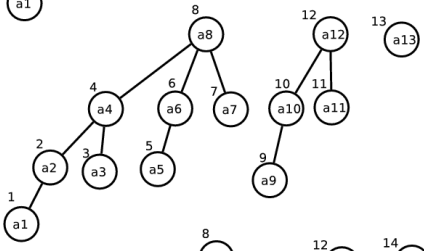
Etapa 12:

Backup
a13



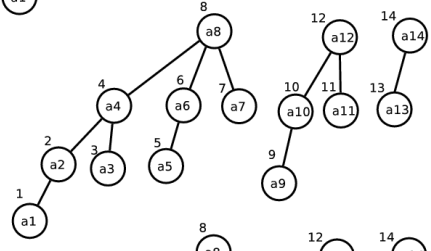
Etapa 13:

Backup
a14



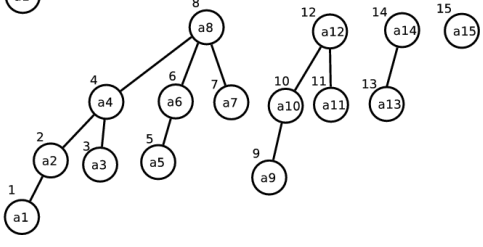
Etapa 14:

Backup
a15



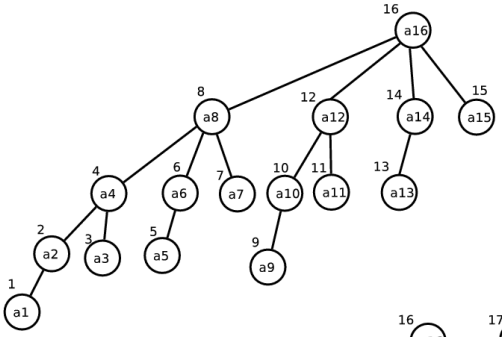
Etapa 15:

Backup
a16



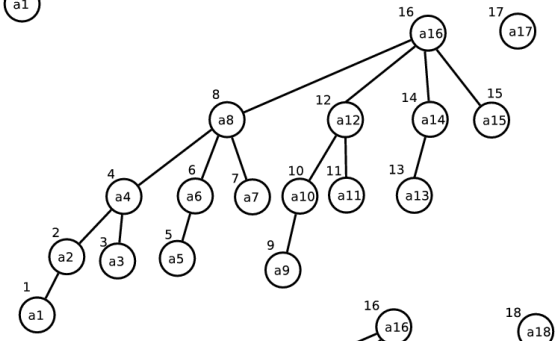
Etapa 16:

Backup
a17



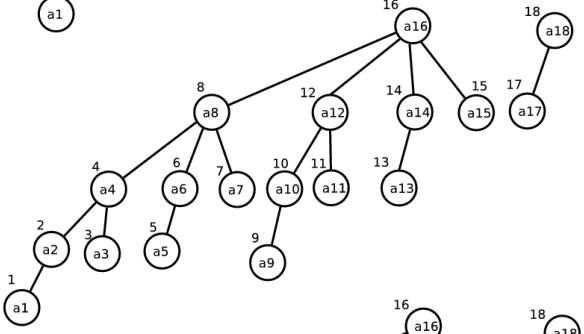
Etapa 17:

Backup
a18



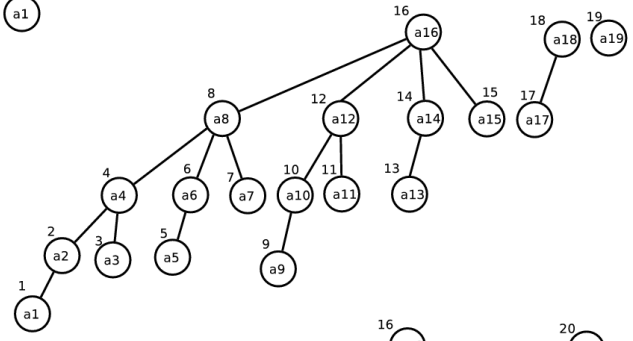
Etapa 18:

Backup
a19



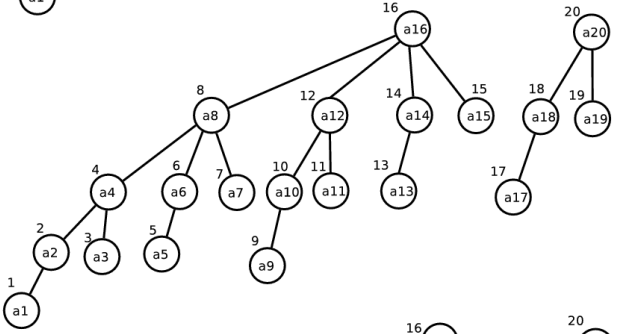
Etapa 19:

Backup
a20



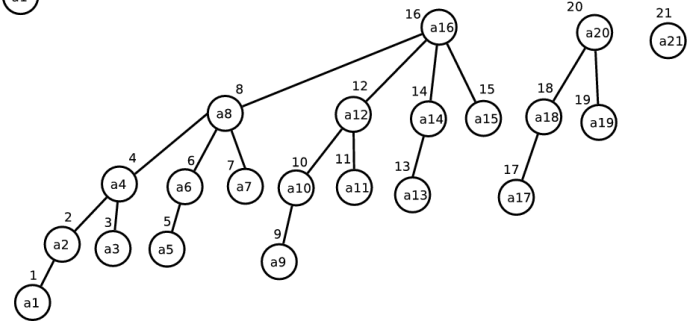
Etapa 20:

Backup
a21



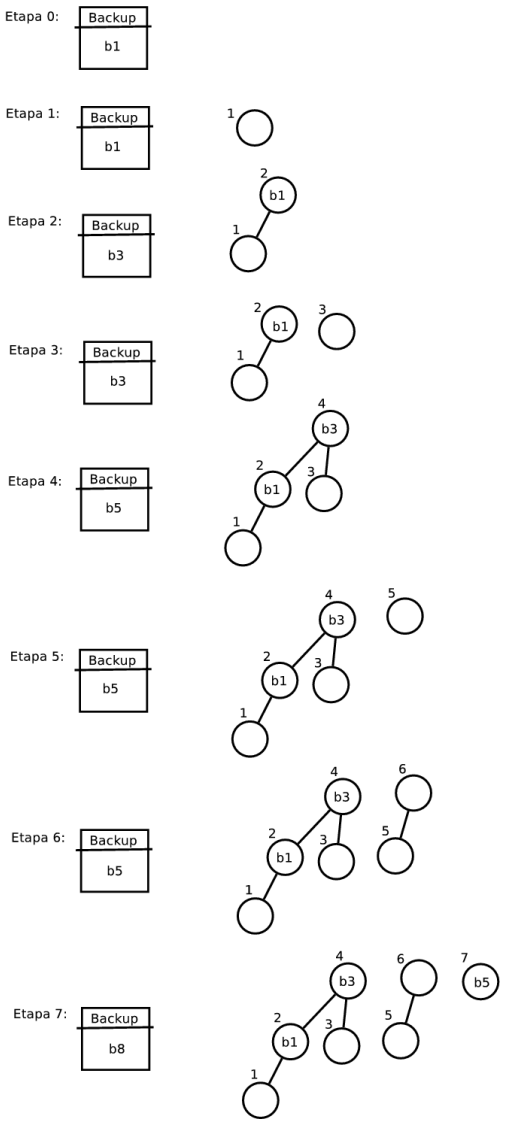
Etapa 21:

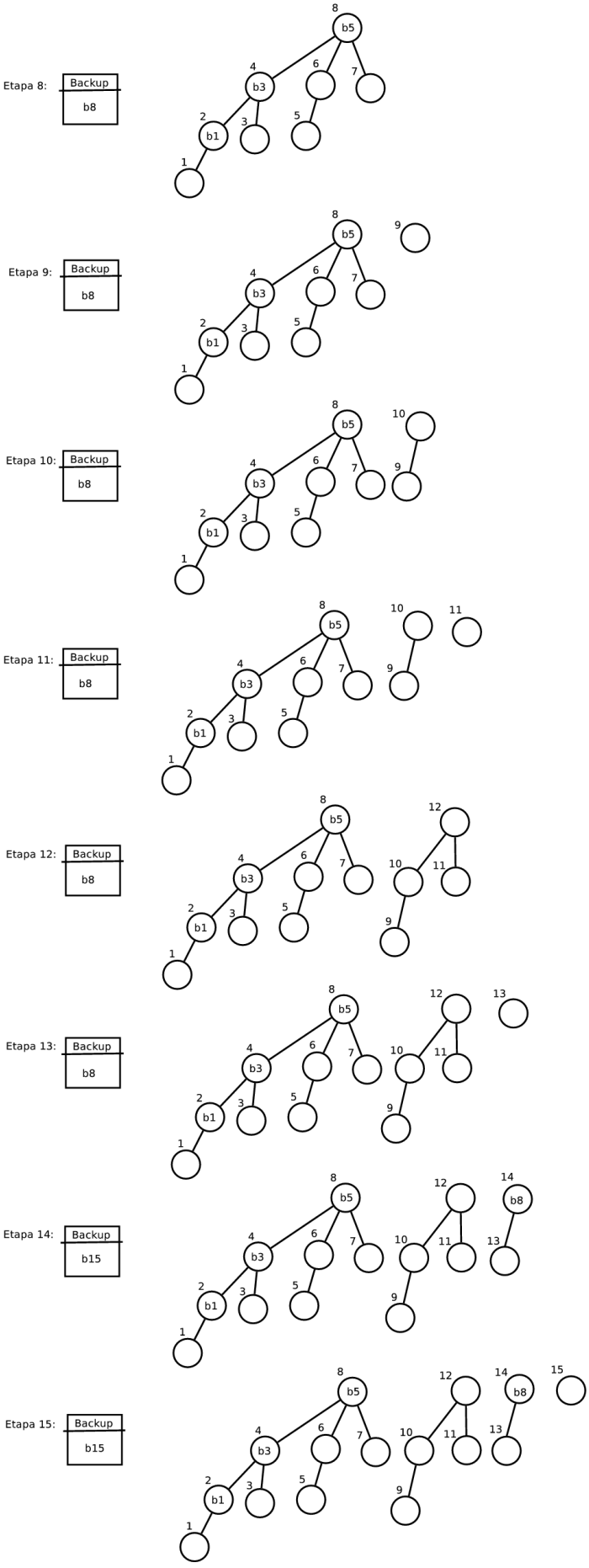
Backup

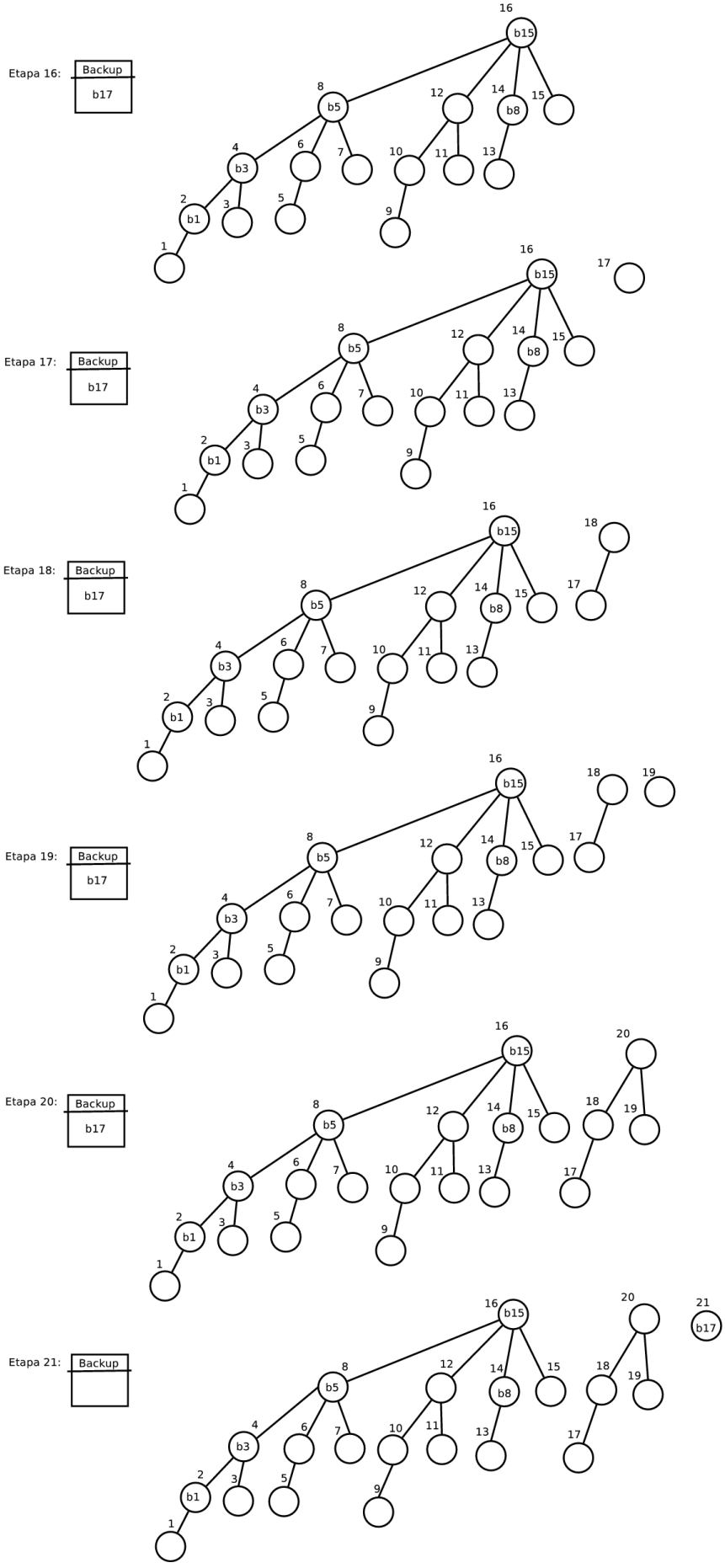


APÊNDICE B – HEAP DO ARQUIVO B

O arquivo *b* foi modificado nas etapas 1, 3, 5, 8, 15, 17 e 21. Considere $b1, b2, b3, ..., bn$ as diferentes versões do arquivo *b*.

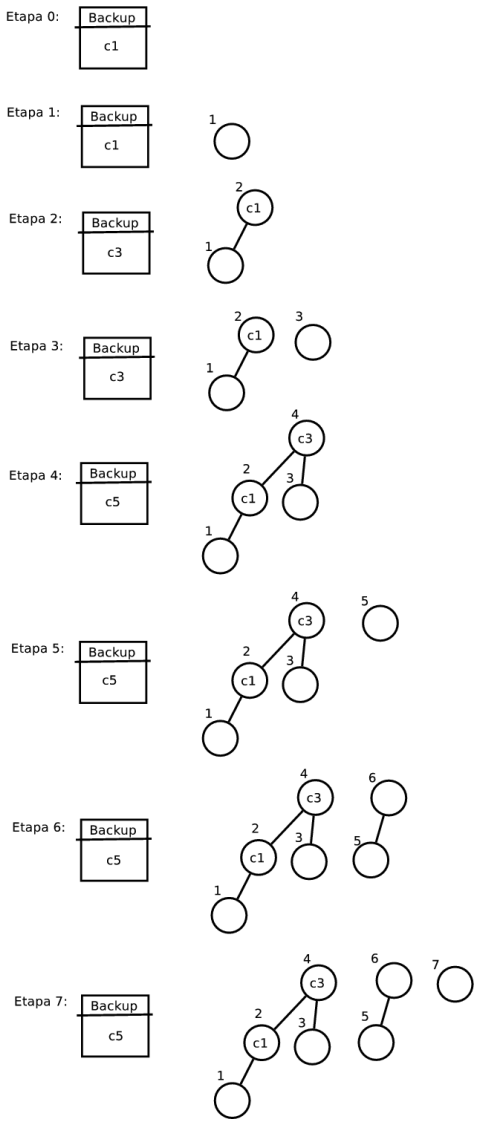


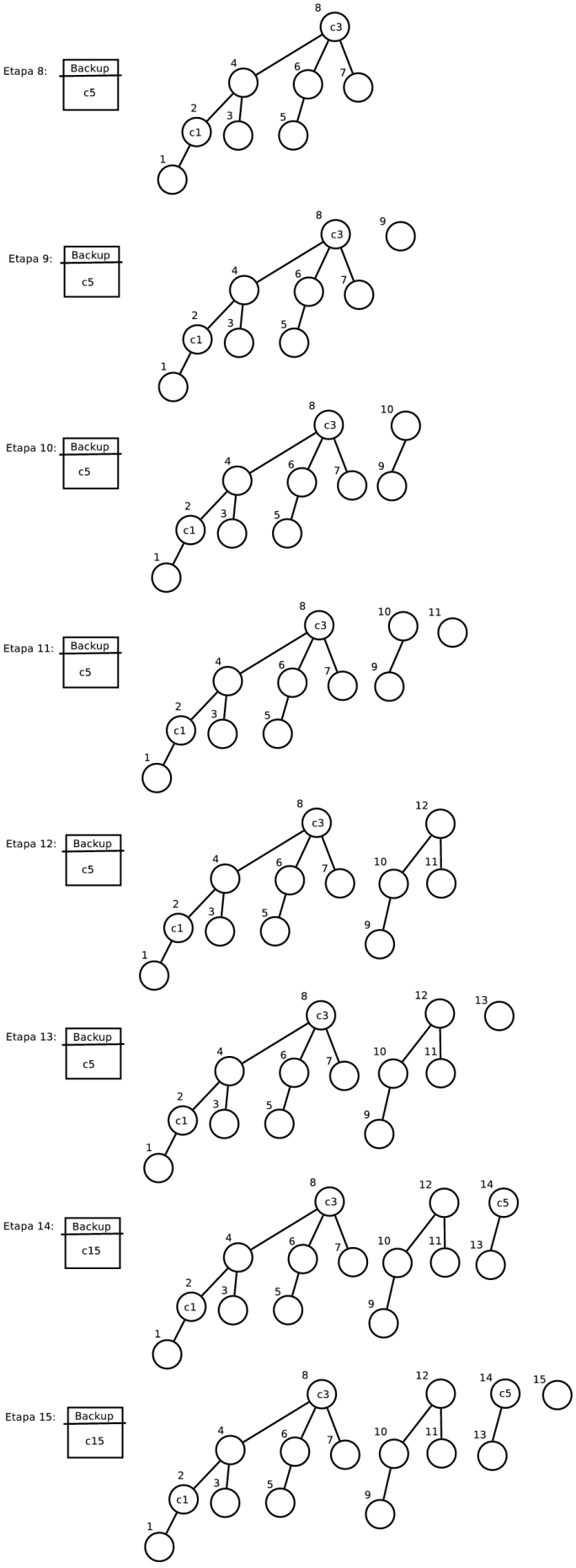

Etapa 8:
Backup
b8
8
b5
4
b3
6
7
2
b1
3
5
1
Etapa 9:
Backup
b8
8
b5
9
4
b3
6
7
2
b1
3
5
1
Etapa 10:
Backup
b8
8
b5
10
4
b3
6
7
9
2
b1
3
5
1
Etapa 11:
Backup
b8
8
b5
10
11
4
b3
6
7
9
2
b1
3
5
1
Etapa 12:
Backup
b8
8
b5
12
4
b3
6
7
10
11
2
b1
3
5
9
1
Etapa 13:
Backup
b8
8
b5
12
13
4
b3
6
7
10
11
2
b1
3
5
9
1
Etapa 14:
Backup
b15
8
b5
12
14
b8
4
b3
6
7
10
11
13
2
b1
3
5
9
1
Etapa 15:
Backup
b15
8
b5
12
14
b8
15
4
b3
6
7
10
11
13
2
b1
3
5
9
1

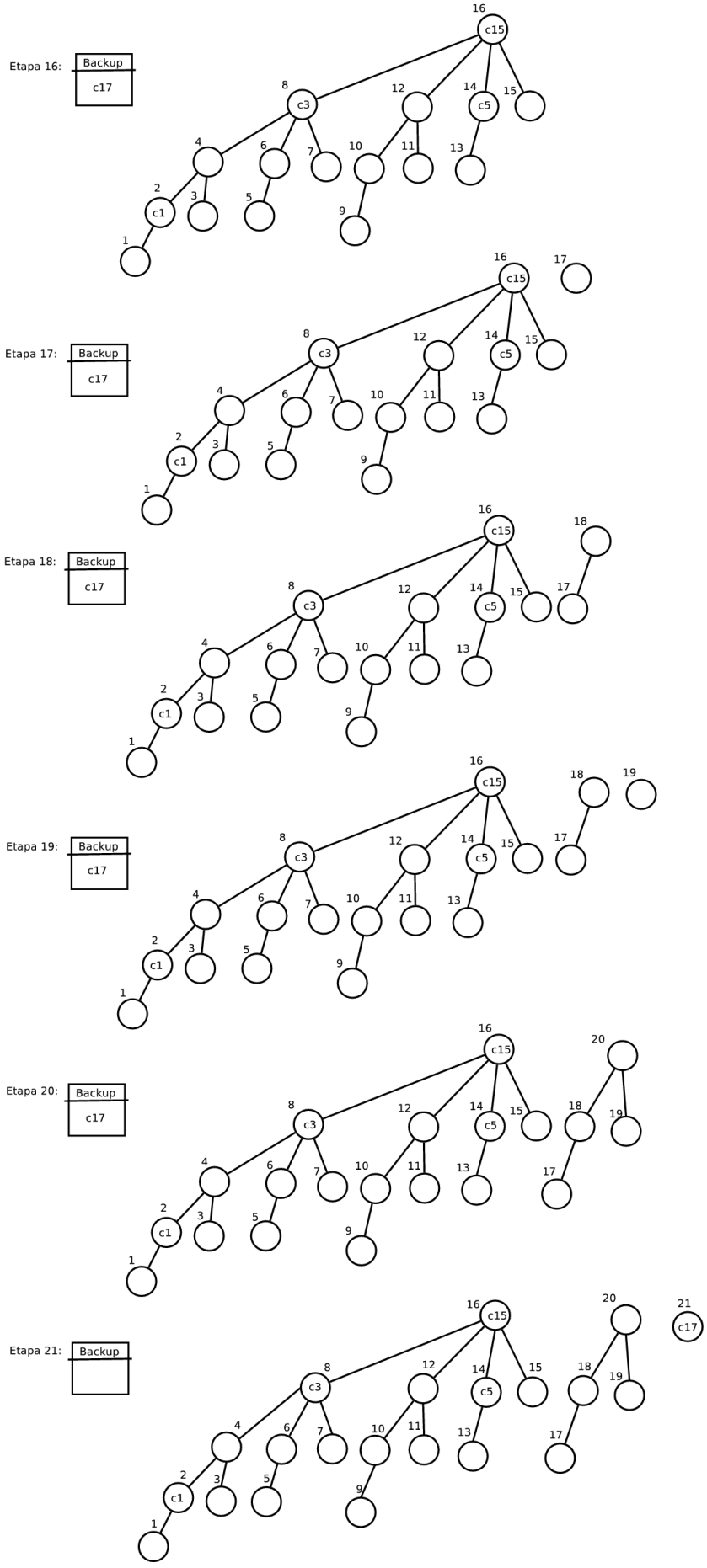

Etapa 16:
Backup
b17
Etapa 17:
Backup
b17
Etapa 18:
Backup
b17
Etapa 19:
Backup
b17
Etapa 20:
Backup
b17
Etapa 21:
Backup
b15
b5
b8
b3
b1
b17

APÊNDICE C – HEAP DO ARQUIVO C

O arquivo c foi modificado nas etapas 1, 3, 5, 15, 17 e 21. Considere $c1, c2, c3, ..., cn$ as diferentes versões do arquivo c.




Etapa 8:
Backup
c5
Etapa 9:
Backup
c5
Etapa 10:
Backup
c5
Etapa 11:
Backup
c5
Etapa 12:
Backup
c5
Etapa 13:
Backup
c5
Etapa 14:
Backup
c15
Etapa 15:
Backup
c15
c1
c3
c5


Etapa 16:
Backup
c17
Etapa 17:
Backup
c17
Etapa 18:
Backup
c17
Etapa 19:
Backup
c17
Etapa 20:
Backup
c17
Etapa 21:
Backup
c17

APÊNDICE D – MATERIAL ADICIONAL

Segue abaixo o material principal disponível em [Camões e Simon 2006] para *download*, entre programas, dados, gráficos, etc. Alguns arquivos foram omitidos nesta lista por questão de espaço. Existe um pacote chamado totaliza.tgz que contém o seguinte conteúdo:

- Scripts do Gnuplot [Gnuplot 2006] para geração de gráficos:
  - geraGrafico.gp
  - geraReconstituiveis.gp
- Scripts bash para automatização das duas fases dos experimentos:
  - parteII.sh
  - parteI.sh
- Simuladores do BRF e Bacula escritos em Python [Python 2006]:
  - totaliza-bacula.py
  - totaliza-brf.py
- Resultados da primeira etapa dos experimentos, separados por porção de dados:
  - Parte-I
    - Saida-completo
      - Graficos/
      - ime.totalizado
      - remember-gordo-0
      - remember-gordo-1
      - remember-magro-9
      - tempo

        * Saida-mail-cache/
        * Saida-mail-cache-cometa/
        * Saida-mail-cache-nao-cometa/
        * Saida-nao-mail-cache/
        * Saida-nao-mail-cache-nao-power/
        * Saida-nao-mail-cache-power/

- Resultados da segunda etapa dos experimentos:
    - Parte-II/
        * ime.totalizado.bacula
        * ime.totalizado.bacula.selecionado
        * ime.totalizado.brf
        * ime.totalizado.brf.selecionado

- Arquivos e gráficos com informações sobre as etapas reconstituíveis em cada política:
    - Reconstituiveis/
        * brf-max
        * brf-max.eps
        * brf-max.png
        * bacula-per
        * bacula-per.eps
        * bacula-per.png

- Dados de entrada dos experimentos:
    - Dados-ime/
        * cometa.codewords
        * eoca-618.acomodado
        * eoca-618.acomodado-grep-iFf-mail-cache.codewords
        * eoca-618.acomodado-grep-iFf-mail-cache.codewords-grep-iFf-cometa.codewords
        * eoca-618.acomodado-grep-iFf-mail-cache.codewords-grep-viFf-cometa.codewords
        * eoca-618.acomodado-grep-viFf-mail-cache.codewords

REFERÊNCIAS BIBLIOGRÁFICAS

AMANDA. 2006. *The Advanced Maryland Automatic Network Disk Archiver*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://www.amanda.org/>.

BACKUP. 2006. *Wikipedia, The Free Encyclopedia*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://en.wikipedia.org/w/index.php?title=Backup&oldid=91198495>.

BACULA. 2006. *The Network Backup Solution*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://www.bacula.org/>.

BINOMIALHEAP. 2006. *Wikipedia, The Free Encyclopedia*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://en.wikipedia.org/wiki/Image:Binomial-heap-13.svg>.

BINOMIALTREE. 2006. *Wikipedia, The Free Encyclopedia*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://en.wikipedia.org/wiki/Image:Binomial_Trees.svg>.

CAMõES, T. *Wiki utilizada para elaboração deste documento*. 2006. Hospedada no portal *Informação, Comunicação e a Sociedade do Conhecimento* da Incubadora da Fapesp. Disponível em: <http://conhecimento.incubadora.fapesp.br/portal/trabalhos/tassia/wiki/>.

CAMõES, T.; SIMON, I. *Dados, simuladores e resultados dos experimentos realizados neste trabalho*. 2006. Disponível em: <http://marco.ime.usp.br/~is/brf-sim/>.

CORMEN, T. H. et al. *Algoritmos — Tradução da 2a. edição americana*. Brasil: Editora Campus, 2002.

COUGIAS, D. J.; HEIBERGER, E. L.; KOOP, K. *The Backup Book: Disaster Recovery from Desktop to Data Center*. United States of America: Network Frontiers, LLC, 2003.

GNUPLOT. 2006. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://www.gnuplot.info/>.

PERL. 2006. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://perl.org>.

PYTHON. 2006. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://www.python.org/>.

RSYNC. 2006. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://samba.anu.edu.au/rsync/>.

SEGURANçA da Informação. 2006. Wikipédia, A Enciclopédia Livre. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Seguran%C3%A7a_da_informa%C3%A7%C3%A3o&oldid=3962379>.

SIMON, I. *BRF*. 2002. *Backup, Remember, and Forget*. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://www.ime.usp.br/~is/brf/>.

SOURCEFORGE. 2006. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://sourceforge.net/>.

TEOREMA Fundamental da Aritmética. 2006. Wikipédia, A Enciclopédia Livre. Último acesso em 04 de Dezembro de 2006. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Teorema_fundamental_da_aritm%C3%A9tica&oldid=3831020>.