Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Dissertação de Mestrado

Políticas de Alocação e Migração de Arquivos em Sistemas de Arquivos Distribuídos para Redes Locais

Thiago Emmanuel Pereira da Cunha Silva

Campina Grande, Paraíba, Brasil
Julho - 2010

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Políticas de Alocação e Migração de Arquivos em Sistemas de Arquivos Distribuídos para Redes Locais

Thiago Emmanuel Pereira da Cunha Silva

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

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas Distribuídos

Francisco Vilar Brasileiro
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S586p Silva, Thiago Emmanuel Pereira da Cunha.
Políticas de alocação e migração de arquivos em sistemas de arquivos distribuídos para redes locais / Thiago Emmanuel Pereira da Cunha Silva. ─ Campina Grande, 2010.
61f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientador: Profº. Drº. Francisco Vilar Brasileiro.

1. Sistemas de Processamento Distribuído. 2. Sistemas de Arquivos. 3. Algoritmos de Alocação. I. Título.

CDU – 004.75(043)

Resumo

Ao longo do tempo, sistemas de arquivos distribuídos têm sido implantados com sucesso em corporações e redes locais. Entretanto, apesar do sucesso obtido, ainda existem problemas. NFS, o estado-da-prática desta classe de sistemas, apresenta uma arquitetura cliente-servidor que limita sua escalabilidade. À medida em que ocorre um aumento na demanda de armazenamento, normalmente gerado pela adição de novos usuários à rede, a compra de novas unidades de armazenamento pode ser necessária. Uma alternativa consiste em agregar o espaço ocioso nos *desktops* que compõem a rede, uma solução factível desde que a capacidade dos discos rígidos continue maior do que a necessidade de boa parte dos usuários. Nessa direção, criou-se o Beehive File System (BeeFS), um sistema de arquivos distribuído que emprega uma arquitetura híbrida composta pelos seguintes componentes: um servidor central responsável por manter os metadados e resolver nomes, e um conjunto de servidores de dados implantados nos *desktops* que colaborativamente compõem o espaço de armazenamento. Neste contexto, o uso de máquinas não dedicadas cria um desafio adicional com relação ao desempenho do serviço, uma vez que *desktops* tipicamente têm desempenho inferior a servidores de propósito específico. Esta dissertação apresenta políticas de alocação e migração de arquivos que maximizam as chances de acesso local, melhorando o desempenho do sistema quando o mesmo é implantado de tal modo que sempre exista um servidor de dados localizado na mesma máquina do cliente. Estas políticas são avaliadas e têm seu desempenho contrastado com um modelo centralizado de armazenamento que emula um caso típico de uso do NFS para redes locais. Os resultados obtidos mostram que estas políticas, mesmo em cenários que induzem uma baixa localidade de acesso devido à alta mobilidade dos usuários da rede, têm desempenho equivalente a um servidor dedicado com poder computacional 3 vezes maior que os *desktops* que compõem a alternativa descentralizada correspondente ao BeeFS.

Abstract

Distributed file system has been successfully deployed on corporations and local area networks. However, state-of-the-practice distributed file systems still have some drawbacks. For instance, NFS uses a client-server architecture which inherently limits its scalability and availability; more importantly, dealing with a sudden increase on the capacity demand of the file system – normally triggered by the arrival of new users – is both costly and cumbersome, since it involves the acquisition of more disk, down time for data migration, and possibly changes in the administration activities – for instance, to accommodate new backup procedures. This work presents BeeFS, a distributed file system which focuses on the specific setting of local area networks desktops. It provides a global file namespace and location-transparent access to files which are stored in the disks of the participating machines. Such approach for storage is desirable and made feasible since the capacity of modern hard disks has outgrown the needs of many users, thus leaving them with much idle storage space in their desktops. Moreover, a notably fraction of LAN desktops shows low CPU and disk loads. BeeFS is designed following a hybrid architecture that employs a centralised server to store metadata and manage file replication – facilitating system design and administration – and distributed data storage servers that collaboratively store data, reducing the bottleneck on the central server, and allowing the incremental growth of the storage capacity. Data storage servers may be deployed on every desktop machine of the corporation to harness the spare disk space in these machines, increasing the utilisation of these disks at no noticeable extra cost. Access to files in this distributed file system is mediated by a client component that may also run at each desktop. Since the desktops may run both a client and a server component, this part of the system resembles a peer-to-peer system. This work proposes file allocation and migration policies which try to keep data as close as possible to the clients that access them, allowing the scalable growth of the file system. Those policies are evaluated and have their performance compared with a centralised storage model which resembles the typical NFS environment for local area networks. Our results show that those policies, even if exposed to a low locality scenario due to a high mobility pattern, give comparable performance in comparison with a centralised storage model 3 times more powerful than the desktops which composes the BeeFS alternative.

Agradecimentos

Agradeço ao Estado brasileiro e à rainha da Inglaterra, que financiaram meus estudos. Sinto-me obrigado, com muito grado, a exprimir a felicidade de ter trabalhado com Fubica e meus companheiros de bar e bancanda, especialmente os colegas do BeeFS que tiveram participação essencial neste trabalho. Por fim, vivas à minha família e minha namorada, por suportarem minha falta de humor com a infinita esperança de que ela fosse passageira.

Campinenses de todos os países, uní-vos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Demandas de armazenamento de uso geral têm sido providas tanto por sistemas de arquivos locais como ext3 [36] e NTFS [28], quanto por sistemas de arquivos distribuídos como NFS [31] e Coda [30]. Sistemas de arquivos locais se adequam bem às necessidades de usuários individuais mas são menos apropriados para uso em redes locais, uma vez que não se ajustam bem à atividades típicas deste ambiente, tais como o compartilhamento de dados e a mobilidade dos usuários[1].

Por sua vez, sistemas de arquivos distribuídos oferecem uma solução mais adequada para o ambiente de redes locais consequentemente têm sido amplamente utilizados. A despeito do grau de adoção destes sistemas, ainda persistem problemas: NFS, o estado-da-prática em sistemas de arquivos distribuídos para redes locais, usa uma arquitetura cliente-servidor que limita sua escalabilidade, disponibilidade e capacidade de crescimento incremental. Lidar com um aumento na demanda de armazenamento, muitas vezes causado pela adição de novos usuários à rede, costuma implicar em aumento nos custos de aquisição e administração. Compra de novos dispositivos de armazenamento e interrupções no serviço para redistribuição de dados são exemplos típicos destes custos. Sistemas como Coda [30] e AFS [40], embora aliviem o problema de escalabilidade ao dividir a carga entre múltiplos servidores, implicam em um aumento nos custos administrativos e de manutenção, e pouco alteram o problema do crescimento incremental da capacidade de armazenamento.

Enquanto soluções tradicionais falham ao lidar com o aumento da demanda de armazena-

[1] Mobilidade neste caso diz respeito à possibilidade de um usuário em um certo momento usar uma máquina da corporação para acessar o sistema de arquivos, e num momento futuro usar outra máquina.

mento, a capacidade dos discos nos *desktops* modernos costuma ser maior do que a necessidade dos usuários [10]. É provável que esta capacidade ociosa seja mantida no futuro: embora haja uma tendência por parte das aplicações de usar arquivos cada vez maiores, os fabricantes de dispositivos de armazenamento ainda são capazes de entregar dispositivos de maior capacidade com preços competitivos.

Este trabalho investiga a possibilidade de agregar este espaço subutilizado – e portanto barato – disponível nos *desktops* que compõem redes locais, como uma solução alternativa aos problemas causados pelo aumento de demanda descritos anteriormente.

A agregação de componentes de armazenamento de pequeno porte tem sido aplicada com sucesso por sistemas de arquivos para computação de alto desempenho, como GFS [15], Ceph [38], xFS [34] e HDFS [5]. Estes sistemas costumam adotar um modelo de distribuição que separa as funcionalidades de armazenamento de dados e metadados em componentes distintos, deste modo aliviando os gargalos de desempenho que apareceriam caso adotasse uma alternativa semelhante àquela adotada por Coda. Embora tenha uma motivação diferente, esta arquitetura pode ser utilizada também no contexto de interesse deste trabalho. Adicionalmente ao servidor de metadados e aos servidores de dados, uma coleção de *clientes* precisa ser implantada nos *desktops* que compõem a rede para mediar o acesso dos usuários ao sistema de arquivos.

O uso em redes locais da agregação de componentes não-dedicados e de baixa capacidade em detrimento da alternativa cliente-servidor clássica que requer máquinas de grande capacidade apresenta desafios: requisições remotas de acesso apresentam uma potencial queda de desempenho devido à diferença de capacidade entre *desktops* e servidores, além disto estas requisições podem afetar a experiência dos usuários nos *desktops* que servem estas requisições.

Por estes motivos, a viabilidade desta abordagem está intimamente ligada à probabilidade das operações sobre arquivos serem feitas sem acessar a rede. Realizar as operações de maneira local torna possível: *i*) reduzir a contenção na rede, diminuindo assim o impacto no servidor de metadados; *ii*) eliminar a latência da rede em operações de acesso aos dados, compensando a diferença de capacidade de processamento entre os *desktops* da solução descentralizada e as máquinas de alta capacidade da solução clássica de armazenamento; e *iii*) limitar a intrusividade do sistema, envolvendo apenas o *desktop* em que o usuário está

localizado nas operações de acesso aos dados.

1.1 Objetivos

Este trabalho é parte de uma pesquisa focada na construção de um sistema de arquivos, o Beehive File System (BeeFS) [2], que resolva o problema do crescimento da demanda de armazenamento em redes locais de maneira barata, utilizando o espaço de armazenamento ocioso dos *desktops* já implantados nesta rede local.

Neste contexto, o objetivo deste trabalho é apresentar uma estratégia de alocação de dados, aplicada neste sistema, que permita que o acesso aos arquivos seja feito de maneira eficiente, a despeito dos servidores de dados serem implantados em máquinas não dedicadas e de menor capacidade de processamento que aquelas usadas nas soluções tradicionais centralizadas.

A condição de sucesso é tornar este sistema equiparável em termos de desempenho às soluções de armazenamento mais utilizadas no ambiente de redes locais. Acreditamos que isto seja um habilitador para a adoção do sistema em questão.

1.2 Contribuições

As principais contribuições deste trabalho são o projeto e a avaliação de políticas de alocação de dados que favorecem a localidade de acesso como mecanismo de ganho de desempenho. Foram avaliadas duas políticas: a primeira é aplicada na criação de arquivos enquanto a segunda na migração de arquivos que estão sendo acessados remotamente. Os resultados desta avaliação apontam que quando associadas, estas políticas de fato promovem a localidade de acesso e diminuem o tempo de resposta percebido pelos clientes do sistema de arquivos. Uma segunda análise mostra que a aplicação destas políticas torna possível obter desempenho equivalente à uma solução centralizada que utiliza máquinas de capacidade 3 vezes maior do que os *desktops* que compõem a alternativa descentralizada considera neste trabalho.

Por fim, como prova de conceito dos algoritmos discutidos neste trabalho, a primeira

[2] http://www.lsd.ufcg.edu.br/beefs

política de alocação discutida anteriormente foi implementada no BeeFS. A implementação do BeeFS e desta política são discutidas bem como uma avaliação de desempenho e escalabilidade. Esta avaliação mostra que o BeeFS consegue agregar o espaço ocioso de *desktops* sem apresentar perda de desempenho em relação ao NFS,

1.3 Organização da Dissertação

No Capítulo 2, este trabalho é contextualizado com uma apresentação do estado-da-arte em alocação de dados para sistemas de arquivos distribuídos. Esta apresentação é dividida em duas partes: estratégias de alocação de dados usadas em sistemas de arquivos distribuídos em produção ou protótipos, e trabalhos teóricos sobre algoritmos e heurísticas de alocação de dados que não tiveram realização experimental.

No Capítulo 3, descreve-se o projeto de um sistema de arquivos que agrega o espaço disponível em *desktops* de uma rede local. Esta descrição, embora simplificada, serve de guia para os capítulos seguintes.

No Capítulo 4, uma política de alocação de dados aplicada durante a criação de arquivos é descrita e avaliada via simulação. Esta simulação considera um sistema de arquivos hipotético que segue o projeto descrito no Capítulo 3 numa rede local em que os usuários estão sujeitos a mobilidade – fato que altera a localidade de acesso.

No Capítulo 5, uma política de migração – aplicada em arquivos já existentes que estão sendo acessados remotamente – é avaliada. Tal como no Capítulo 4, esta política é avaliada via simulação, considerando o mesmo sistema hipotético e sob os mesmos critérios de mobilidade dos usuários.

No Capítulo 6, a capacidade de promoção da localidade de acesso da política de alocação descrita no Capítulo 4 e da política de migração descrita no Capítulo 5 é novamente avaliada. Esta avaliação considera se a adoção destas políticas permite atingir um desempenho equivalente às alternativas tradicionais de armazenamento centralizado.

No Capítulo 7, são descritos as considerações de projeto e os detalhes de implementação do BeeFS, um protótipo de sistema de arquivos distribuído utilizado como prova de conceito deste trabalho. Este protótipo, que implementa a política de alocação descrita no Capítulo 4, tem seu desempenho e escalabilidade avaliados e comparados com o NFS, o

estado-da-prática para sistemas de arquivos distribuídos de uso geral.

No Capítulo 8, reunimos o conjunto de conclusões desenvolvidas no decorrer deste trabalho bem como perspectivas de trabalhos futuros derivadas das contribuições alcançadas.

Por fim, no Capítulo 9 realizamos uma caracterização da carga de trabalho utilizada nas simulações descritas nos Capítulos 4, 5 e 6.

Capítulo 2

Trabalhos Relacionados

Neste capítulo é apresentado o estado-da-arte em alocação de dados para sistemas de arquivos distribuídos. Esta apresentação é dividida em duas partes: a primeira delas diz respeito a estratégias de alocação de dados implementadas em sistemas de arquivos distribuídos que entraram em produção ou que tiveram seu projeto realizado na forma de um protótipo; a segunda parte considera os estudos teóricos em algoritmos de alocação de dados – trabalhos que não tiveram necessariamente uma aplicação prática imediata. Enquanto a primeira parte desta apresentação reforça o trabalho de motivação feito no Capítulo 1 ao identificar lacunas existentes na literatura, a segunda parte permite o embasamento necessário paras as políticas de alocação investigadas nos capítulos seguintes.

2.1 Alocação de Dados em Sistemas de Arquivos Distribuídos

2.1.1 Sistemas de Arquivos Distribuídos de Uso Geral

O *Andrew File System* (AFS) [40], e sua continuação o Coda [30], foram criados para resolver problemas de escala existentes em sistemas de armazenamento para redes corporativas de grande escala. Do conjunto de decisões de projeto tomadas, três foram fundamentais para o sucesso destes sistemas:

1. O particionamento do serviço de armazenamento em múltiplos servidores.

2. O uso de *caching* agressivo por parte dos clientes, com o armazenamento de arquivos inteiros.

3. A aplicação de um protocolo de coerência de *cache* baseado em *callbacks*. Segundo este protocolo, os servidores de dados mantém estado sobre quais arquivos estão em *cache* por cada cliente, os quais recebem uma notificação quando outro cliente tenta atualizar o mesmo arquivo. Este protocolo elimina a necessidade de validação dos arquivos em *cache* por parte dos clientes antes de usá-los.

Nestes sistemas, disponibilidade é tão importante quanto escalabilidade. Por esta razão os arquivos são armazenados de maneira redundante e organizados em grupos de réplicas controlados por um protocolo otimista de replicação. Este protocolo permite que atualizações causem uma inconsistência nos dados, por exemplo quando falhas ocasionam uma partição na rede, em favor da disponibilidade do serviço.

O projeto e a implementação do AFS e Coda foram trabalhos bastante influentes na área de sistemas de arquivos distribuídos. Entretanto, considerando o problema de alocação de dados, ainda há espaço para desenvolvimento. Por exemplo, embora o espaço de armazenamento seja particionado entre múltiplos servidores, não há medidas automáticas de balanceamento desta distribuição.

Farsite [1], que provavelmente é a solução mais próxima em termos arquiteturais do cenário discutido neste trabalho, é um sistema de arquivos completamente descentralizado composto por coleções de *desktops* implantados em uma rede corporativa. Embora seu modelo de distribuição seja semelhante, o objetivo é bastante diferente: Farsite foi criado para prover alta disponibilidade, confiabilidade, segurança e tolerância a faltas para sistemas de armazenamento corporativos composto por máquinas assumidamente não confiáveis. Por esta razão, todos os arquivos armazenados são criptografados e armazenados de maneira redundante. Ainda, os servidores que assumem a função do serviço de diretórios trabalham conforme um protocolo tolerante a faltas bizantinas.

A estratégia de alocação de arquivos aplicada por Farsite tem como objetivo maximar a disponibilidade do serviço de armazenamento. Para tanto, como descrito por Douceur e Wattenhofer [11], Farsite implementa um serviço que monitora constantemente a disponibilidade das máquinas que compõem o sistema, e sucessivamente troca a localização das

réplicas dos arquivos entre estas máquinas.

2.1.2 Sistemas de Arquivos para *Clusters*

Frangipani [35] é um sistema de arquivos distribuído que agrega uma coleção de discos de múltiplas máquinas em um espaço único de armazenamento, tendo como objetivo a escalabilidade no armazenamento, objetivo este bastante semelhante à discutida neste trabalho. Apesar das semelhanças há uma diferença fundamental: este sistema utiliza máquinas dedicadas. Frangipani foi construído sobre Petal [19], um subsistema de armazenamento distribuído que provê a capacidade de tolerância a faltas, balanceamento dinâmico da carga entre os servidores e reconfiguração automática quando novos servidores são adicionados.

xFS [34], tal como o Farsite [1], é um sistema completamente descentralizado em que todas as funções que compõem o serviço de armazenamento são realizadas por estações de trabalho que operam de modo cooperativo. A implementação do xFS foi responsável por reunir em um sistema focado em alto desempenho e escalabilidade contribuições de outros trabalhos inovadores realizados à época, por exemplo: protocolos para consistência de *cache* escaláveis, *caching* cooperativo, *stripping* de disco e sistemas de arquivos baseados em *log*.

Ceph [38] é um sistema de arquivos para *clusters* criado para atender a carga gerada por aplicações em ambiente de peta-escala. Ceph adota uma arquitetura recentemente disseminada baseada em objetos [4] (*Object Storage Devices - OSDs*) que provê uma interface de mais alto nível do que aquela provida pelos dispositivos baseados em bloco comumente utilizados. Em sistemas que adotam esta arquitetura, clientes normalmente interagem com um servidor de metadados para realizar operações como *open*, *stat* e *rename*, enquanto se comunicam diretamente com os *OSDs* para realizar operações de leitura e escrita de dados, o que aumenta de maneira significativa a escalabilidade destes sistemas. Além desta decisão arquitetural, o Ceph também faz uso de funções de distribuição de dados baseadas em *hash* [39] como mecanismo de ganho de desempenho. Em outras palavras, a localização dos dados armazenados é calculada ao invés de ser resolvida por um serviço de diretórios.

Em boa parte dos sistemas de arquivos para *cluster* aqui descritos, e em outros omitidos nesta descrição por sua similaridade, a estratégia de alocação de dados adotada tem como objetivo principal o balanceamento da carga entre os nodos de armazenamento que são agregados pelo sistema. Em alguns destes sistemas também é comum a aplicação de

medidas para aumento da disponibilidade tais como RAID [24]. Em sistemas construídos mais recentemente, como Ceph e o Google File System [15], a localização dos dispositivos de armazenamento começa a ser levada em consideração nos procedimentos de alocação de dados. Réplicas de arquivos costumam ser alocadas em *racks* diferentes para evitar faltas dependentes, por exemplo ocasionadas por um defeito em um *switch* que conecta um *rack*.

Uma observação pertinente sobre os sistemas de arquivos discutidos nesta seção diz respeito à complexidade do modelo de distribuição adotado por estes. Alguns sistemas altamente descentralizados, como xFS e Farsite, embora tenham sido trabalhos de reconhecida relevância, não passaram da fase de protótipo. O xFS é um caso exemplar, seu protótipo não implementa algumas funcionalidades importantes como tolerância a faltas e reconfiguração. Por sua vez, Farsite adota um modelo tão complexo, devido à utilização de protocolos tolerantes à faltas bizantinas, que dificilmente teria um desempenho aceitável.

2.2 Algoritmos Ótimos de Alocação de Dados em Sistemas de Arquivos

Em 1969, é reportado o primeiro algoritmo relacionado à alocação ótima de dados [8]. Em linhas gerais, este estudo envolve a decisão de alocação de um dado conjunto de arquivos em um segundo conjunto composto por dispositivos de armazenamento, de modo a diminuir custos de operação. O problema é resolvido por um modelo de programação linear que considera: custos de armazenamento e transmissão, tamanho de arquivos, taxas de requisição e modificação, e finalmente, limites máximos para o tempo de acesso e capacidade de armazenamento. Posteriormente, este trabalho ficou conhecido como o problema da alocação múltipla de arquivos,

O problema da alocação múltipla de arquivos foi reexaminado em uma série de trabalhos [6, 21, 22] como uma coleção de alocações individuais. De maneira análoga à concepção original [8], estes trabalhos assumem que: as taxas médias de requisição e atualização para arquivos provenientes de cada nodo são conhecidas, cada requisição acessa somente uma cópia do arquivo por vez, e todas as cópias são acessadas durante uma atualização. Ainda, as requisições, atualizações e armazenamento têm custos conhecidos.

Esta nova abordagem, que recebeu o nome de *problema da alocação ótima de ar-*

quivos em oposição à alocação múltipla discutida anteriormente, recebeu um tratamento matemático [6] que anos mais tarde provou-se [13] ser um problema NP-Completo. Este tratamento, é sumarizado a seguir.

Sejam:

- I = conjunto de índices de nodos que contêm uma cópia do arquivo
- n = número de nodos no sistema
- ψ_j = taxa de atualização originada do nodo j
- λ_j = taxa de requisição originada do nodo j
- $d_{j,k}$ = custo de comunicação de uma requisição de acesso de j para k
- $d'_{j,k}$ = custo de comunicação de uma requisição de atualização de j para k
- σ_k = custo de armazenamento do arquivo no nodo k

A alocação ótima do arquivo k é definida como o conjunto de índices I que minimiza a seguinte função de custo:

$$C(I) = \sum_{j=1}^{n} \left[\sum_{k \in I} \psi_j d'_{j,k} Y_k + \lambda_j \min_{k \in I} d_{j,k} \right] + \sum_{k \in I} \sigma_k Y_k$$

Sendo Y uma variável de controle definida como:

$$Y_j = \begin{cases} 0 & \text{se } j \notin I; \\ 1 & \text{se } j \in I. \end{cases}$$

Novas direções para o problema da alocação ótima de arquivos têm sido estudadas desde então. Dentre estas podemos destacar a alocação conjunta de arquivos e de processos [14, 21, 22]. Outros exemplos são novos objetivos de otimização, tais como: tempo de acesso mínimo [26], tempos mínimos de execução e transferência [3], vazão máxima [18] e tempo mínimo de resposta [7]. Dowdy e Foster comparam estas e outras variantes [12].

Todas estas abordagens do problema de alocação têm uma característica em comum: assumem um ambiente estático e com informação completa. Entretanto, como reportado por Wah [37], taxas de acesso não são previsíveis em períodos curtos, podem existir variações

transitórias e é difícil extrapolar tendências a partir do comportamento passado. Ainda, os custos de comunicação e as variações transitórias tornam difícil a coleta acurada de taxas de requisição por um componente central responsável pela decisão de alocação.

Neste trabalho serão consideradas políticas para o problema da alocação de arquivos que atuam de maneira heurística. Embora as heurísticas normalmente sejam menos eficazes que as soluções ótimas, as limitações descritas anteriormente tornariam bastante complicada a implementação de soluções eficientes.

Capítulo 3

Projeto de um Sistema de Arquivos sobre Desktops

Como foi antecipado no Capítulo 1, este trabalho discute políticas de alocação de arquivos aplicadas a sistemas de arquivos para redes locais que agregam o espaço em disco livre de *desktops*. Neste capítulo descrevemos uma alternativa de projeto para um sistema deste tipo. Esta descrição contempla os grandes módulos, em que o sistema considerado pode ser dividido, as funcionalidades de cada um destes módulos bem como estes estão distribuídos na rede local. As políticas consideradas no decorrer deste trabalho consideram um sistema hipotético que segue o projeto descrito neste capítulo.

3.1 Considerações de Projeto

Como ilustra a Figura 3.1, consideramos um sistema de arquivos composto por três módulos principais: servidores de dados, clientes e um servidor de metadados.

Os servidores de dados são responsáveis por armazenar e prover acesso aos dados brutos que compõem o espaço de armazenamento. Eles executam nos *desktops* que compõem a rede local e tiram proveito do espaço em disco subutilizado nestas máquinas. Isto permite um incrementar a utilização destes componentes, e agregar capacidade de armazenamento ao sistema de arquivos a um custo baixo. Como este serviço de armazenamento é composto por servidores que executam em máquinas não confiáveis, por questões de tolerância a faltas, cada arquivo é armazenado de maneira redundante em um subconjunto destes servidores de

dados, que compõem o *grupo de replicação* deste arquivo.

O servidor de metadados é responsável por armazenar metadados (diretórios e atributos de arquivos) e executar operações sobre estes, resolver nomes, controlar o acesso aos dados, descobrir recursos, coordenar a alocação de réplicas dos arquivos e detectar e tolerar faltas dos servidores de dados. Devido à importância das operações que realiza, consideramos que o servidor de metadados executa em uma máquina dedicada e confiável.

O acesso aos dados é mediado pelos clientes, que de modo análogo aos servidores de dados devem ser implantados nos *desktops* que compõem a rede.

Enquanto a adoção de um componente centralizado facilita a implementação e administração do sistema, a distribuição do armazenamento de dados tem o potencial de promover o balanceamento da carga, além de permitir o crescimento incremental da capacidade total de armazenamento.

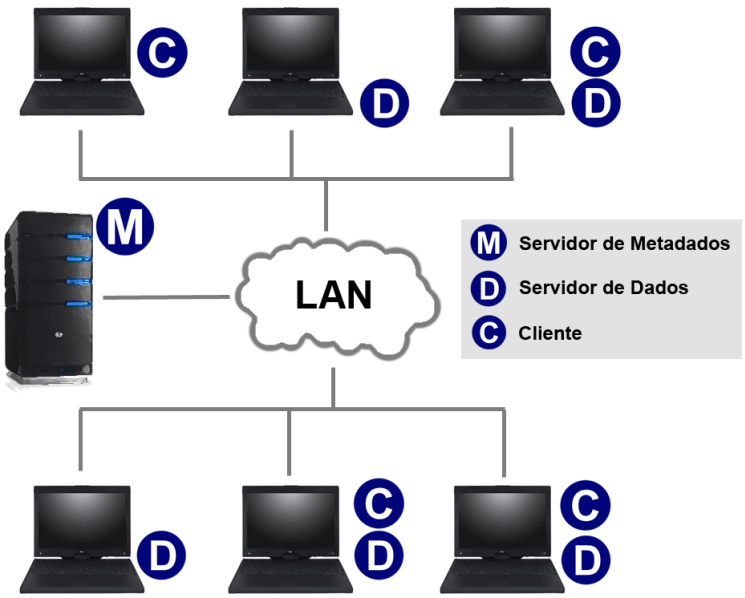

Figura 3.1: Componentes do sistema de arquivos e sua distribuição na rede local

3.2 Modelo do Sistema

Consideramos que o sistema de arquivos é composto por três serviços: cliente, servidor de dados e servidor de metadados, respectivamente descritos pelos símbolos c, d e md. A rede local de armazenamento é composta por *desktops* que possuem o mesmo poder de processa-

mento. Estes *desktops* são divididos em dois conjuntos independentes: um conjunto unitário D composto por uma máquina dedicada e outro conjunto ND composto por máquinas não-dedicadas. Sendo M o conjunto de k máquinas que compõem a rede temos que:

$M = \{m_1, m_2, ..., m_k\} = D \cup ND \wedge D \cap ND = \emptyset$;

$ND = \{m_1, m_2, ..., m_j\}$ com $j = k - 1$ e $D = M_k$

Temos ainda que o conjunto de serviços em execução em uma determinada máquina é descrito pelo símbolo σ. Sendo σ_i o conjunto de serviços em execução na máquina m_i.

Capítulo 4

Uma Política de Alocação para Novos Arquivos

Neste capítulo apresentamos e avaliamos uma política de alocação de dados aplicada no momento da criação dos arquivos. Os resultados da avaliação desta política mostram que mesmo quando os usuários da rede local estão submetidos a um regime de mobilidade bastante intenso, a localidade de acesso é favorecida – não menos que 73% dos dados são lidos localmente.

Na Seção 4.1 motivamos a necessidade da aplicação de uma política de alocação de arquivos ao discutir a origem de desvios de localidade em sistemas de arquivos para redes locais. Na Seção 4.2 a política de alocação analisada neste capítulo é descrita em detalhes. Na Seção 4.3 as simulações realizadas para avaliar a política descrita na Seção 4.2 são detalhadas. Por fim, os resultados destas simulações são analisados na Seção 4.4, mostrando que a política proposta na Seção 4.2 de fato favorece a localidade de acesso, e consequentemente o desempenho no acesso aos dados. Na Seção 4.5 reunimos as conclusões destas análises.

4.1 Desvios da Localidade de Acesso e sua Relação com a Alocação de Arquivos

O problema da alocação de arquivos surge quando a localidade de acesso aos dados não é perfeita. Em não havendo desvio na localidade de acesso não haverá problema de alocação.

Neste sentido, antes de se propor qualquer mecanismo mais sofisticado para a alocação de arquivos é preciso responder a seguinte pergunta: *"Dado o grau de localidade de acesso aos dados característico de uma rede local de armazenamento, há necessidade de um algoritmo de alocação?"*

Para responder esta pergunta, vamos considerar um sistema de arquivos distribuído para redes locais hipotético que segue o projeto descrito no Capítulo 3 e que implementa uma estratégia de alocação simplista, descrita na próxima seção. Este sistema será analisado via simulações que consideram desvios de localidade causados pela mobilidade dos usuários entre as máquinas que compõem a rede local de armazenamento. Este é um fenômeno comum em alguns ambientes, tais como laboratórios acadêmicos, onde não existe uma relação de posse entre usuário e máquina.

4.2 Política de Alocação

Como foi dito na seção anterior, o sistema de arquivos considerado neste trabalho terá seu desempenho analisado quando uma estratégia de alocação simplista de arquivos é adotada. Antes de descrever esta estratégia vale lembrar que no sistema de arquivos considerado, servidores de dados e clientes executam conjuntamente nos *desktops* que compõem a rede local, usando o modelo descrito na Seção 3.2, temos que: $\forall \sigma_i, \exists c, d | (c \in \sigma_i \wedge d \in \sigma_i) \vee (md \in m)$.

A política de alocação adotada funciona da seguinte forma: quando um arquivo é criado, este é alocado para o servidor de dados localizado na máquina onde a requisição foi feita. Todos os acessos sobre este arquivo serão atendidos por este servidor, mesmo que este arquivo seja acessado de maneira remota, eventualmente, em consequência da mobilidade dos usuários.

A razão que motiva esta estratégia é a seguinte: é conhecido que acesso concorrente é um evento raro em sistemas de arquivos para redes locais [20]; ainda, em alguns ambientes os usuários da rede frequentemente acessam o sistema de arquivos usando a mesma máquina. Deste modo, quando estas propriedades são válidas é esperado que esta política implique em um número baixo de acessos remotos. De fato, isto é confirmado pelos resultados das simulações executadas mesmo quando o cliente do sistema de arquivos está submetido a uma

taxa alta de mobilidade.

4.3 Modelo de Simulação

Para avaliar qual o impacto no desempenho que a adoção de uma solução simplista de alocação de arquivos exerce quando o sistema de arquivos considerado neste trabalho está exposto a mobilidade dos usuários, usaremos simulações. Adotamos esta metodologia por dois motivos: *i*) diminui os esforços de implementação, uma vez que é mais fácil codificar a política de alocação no simulador do que em um sistema real, *ii*) permite exercitar mais cenários em tempo hábil.

Na Seção 4.3.1 descrevemos a carga de trabalho considera pelo simulador. Na Seção 4.3.2 é explicado como o sistema de arquivos simulado trata os atributos e metadados do sistema. Na Seção 4.3.3, mais detalhes do modelo simulado são descritos bem como as variáveis aleatórias envolvidas na simulação.

4.3.1 Carga de Trabalho

Para construir a carga de trabalho utilizada pelo nosso simulador usamos rastros de utilização gerados pela instrumentação das chamadas ao sistema operacional de 13 computadores pessoais usados em um laboratório de pesquisa. Esta instrumentação, disponibilizada em http://iotta.snia.org/, foi realizada entre 2001 e 2002. Como resultado foram gerados 13 rastros independentes, um rastro para cada máquina instrumentada, cujas durações dos períodos cobertos variam entre 8 dias e 13 meses.

4.3.2 Metadados

Como apresentado anteriormente, os rastros utilizados não contêm informações sobre os metadados dos arquivos manipulados. Entretanto, os metadados dos arquivos são informações necessárias para a simulação. Dentre os metadados podemos incluir a hierarquia de nomes (os diretórios) e atributos dos arquivos, como por exemplo o tamanho. Estas informações costumam ser obtidas antes da coleta dos rastros para compor um *retrato* dos metadados, uma vez que muitas vezes os sistemas de arquivos instrumentados já existem

antes da coleta.

Como esta informação não foi disponibilizada pelos criadores dos rastros, adotou-se a seguinte estratégia para os registros que se referem a arquivos criados antes da instrumentação: quando a primeira referência para um arquivo no rastro é um *open* sem os parâmetros opcionais de criação, o arquivo é criado e seu tamanho é definido seguindo uma distribuição log-normal com parâmetros *média* =$8,46$ e *variância* = $2,38$. Estes parâmetros seguem os valores identificados em um trabalho de caracterização de sistema de arquivos [10].

4.3.3 Dinâmica da Simulação

A política de alocação de arquivos foi avaliada pela simulação do comportamento de um usuário da rede local de armazenamento. Para tanto, o rastro de simulação mais longo (13 meses) foi escolhido como mapeamento da carga de trabalho deste usuário. Consideramos que a utilização de um único rastro não implica em prejuízo na validade da nossa análise, pois como será descrito no Capítulo 9 todos estes rastros são semelhantes com respeito a duas características: o tempo de vida típico dos arquivos é curto e a distribuição da popularidade dos arquivos é extremamente desigual. Estas duas propriedades são as principais variáveis que impactam as soluções descritas no decorrer deste trabalho.

O número de máquinas existentes na rede local de armazenamento foi arbitrado em 50. Note que quanto maior o número de máquinas maior é o prejuízo para a política de alocação em termos da promoção da localidade de acesso: com o aumento da probabilidade de efetuar uma migração para uma máquina que não tenha sido utilizada antes, aumenta-se a chance de usar arquivos criados em outras máquinas que por consequência, precisam ser acessados remotamente. Consideramos que uma rede local com 50 máquinas seja maior do que as redes típicas.

Consideramos que em uma destas máquinas executa o servidor de metadados, nas 49 máquinas restantes executam um servidor de dados e um módulo cliente do sistema de arquivos. Usando a notação descrita na Seção 3.2 temos que:

$\forall \sigma_i, \forall m_i \in ND, \exists c, d | (c \in \sigma_i \wedge d \in \sigma_i)$

e,

$\forall \sigma_i, \forall m_i \in D, \exists md | (md \in \sigma_i)$

Do rastro de simulação construímos a abstração de *sessões de uso*. Uma sessão de uso

é definida como um conjunto de registros do rastro compreendidos entre períodos de inatividade. No início de cada uma destas sessões de uso, há uma probabilidade associada do usuário simulado realizar uma migração.

De maneira mais formal, considere que :

$$H = \{H_1, H_2, ..., H_{s-1}, H_s, H_{s+1}, ..., H_k\} \tag{4.1}$$

onde H é conjunto de máquinas que o usuário utilizou para ter acesso ao sistema de arquivos em cada uma das sessões de uso, e H_s a máquina em que o usuário estava localizado na s-ésima sessão. Temos que, $H_{s-1} \neq H_s$, para $1 < s \leq k$, com uma probabilidade p, chamada de probabilidade de migração.

Para entender como a política de alocação se comportaria para o caso extremo, adotou-se uma probabilidade de migração $p = 1$. Ainda, considerou-se um período de inatividade de 1 hora. A adoção destes valores implica que dois registros consecutivos que estejam separados por mais que 1 hora fazem parte necessariamente de sessões de uso que ocorreram em máquinas distintas.

Como apresentado no Capítulo 3, por questões de confiabilidade, dado que os servidores de dados executam em máquinas de uso geral, os arquivos são armazenados de maneira replicada. Para tanto, além da cópia selecionada através da política descrita na Seção 4.2, um conjunto de réplicas de cada arquivo é criado em outros servidores aleatoriamente escolhidos. Nas simulações descritas neste capítulo, o número de réplicas secundárias escolhido foi 2. Esta escolha considera a manutenção de um nível de confiabilidade e o impacto no desempenho associado ao aumento do número de réplicas, questões que foram avaliadas em um trabalho sobre tolerância a falhas [32] aplicado ao sistema de arquivos descrito no Capítulo 7.

Operações de atualização são realizadas de maneira bloqueante em todas as réplicas, como consequência o conteúdo entre estas é sempre o mesmo. Pelo mesmo motivo, quando um usuário realiza uma migração para uma máquina da rede que armazena réplicas secundárias de um determinado arquivo, estas poderão ser *promovidas* a réplicas primárias de modo imediato.

4.4 Avaliação dos Resultados Simulados

Para avaliar a eficiência da política de alocação simulada neste capítulo, adotamos como métrica de desempenho a proporção de dados acessados localmente. Acreditamos que esta métrica seja uma boa indicadora do desempenho de acesso aos dados porque foi simulado o comportamento de um único usuário, deste modo acessos locais serão mais rápidos do que remotos pois não há outras requisições em atendimento nos servidores.

A Figura 4.1 mostra a distribuição empírica acumulada da porcentagem de dados acessados localmente resultante da execução de 16 simulações do rastro. Estas execuções foram suficientes para alcançar um erro máximo de $\pm 0,5\%$ com confiança de 99%. Cada ponto neste gráfico refere-se à seguinte razão:

$$\frac{\Sigma L}{\Sigma L + \Sigma R} \tag{4.2}$$

onde ΣL indica o somatório dos *bytes* acessados em operações locais, enquanto ΣR o somatório dos *bytes* acessados em operações remotas, considerando uma simulação do rastro.

Como pode ser visto na Figura 4.1, mesmo com a probabilidade de migração máxima, a simulação do rastro que apresenta pior resultado na promoção da localidade de acesso tem cerca de 74% dos dados acessados localmente.

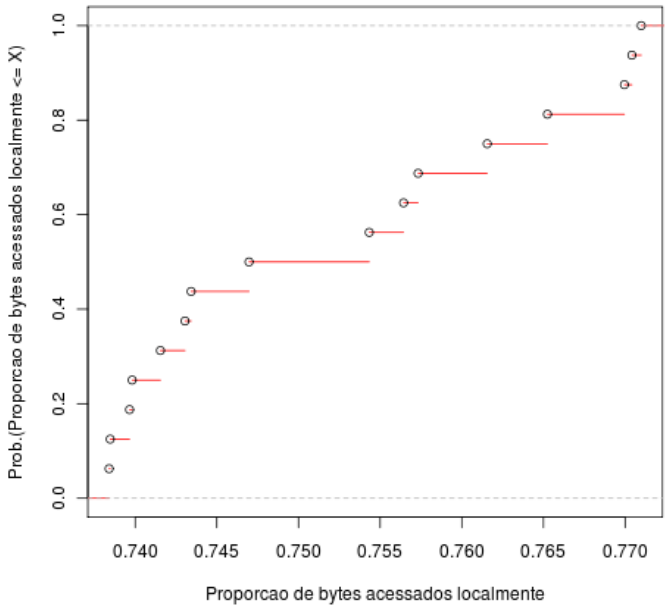

Figura 4.1: Função de Distribuição Acumulada para a Proporção de Bytes Acessados Localmente com probabilidade de migração $p = 1$ e período de inatividade de uma hora.

Como foi apontado na Seção 4.3.3 e discutido em mais detalhes no Capítulo 9, tipicamente os arquivos apresentam um tempo de vida curto o que permite atingir este grau de promoção de localidade. Entretanto, ainda existirão arquivos que serão acessados remotamente, e que desviam a métrica de localidade do ponto máximo. Para entender melhor como este padrão de acesso afeta o sistema simulado, vamos analisar os resultados das simulações numa perspectiva menos sumarizada. Para isto usaremos a mesma métrica descrita anteriormente – a proporção de *bytes* acessados localmente – desta vez aplicada a cada arquivo acessado durante uma simulação do rastro.

A Figura 4.2 mostra a frequência com que estes arquivos se distribuem quando categorizados na métrica descrita. Uma análise visual deste gráfico é suficiente para notar que a grande maioria dos arquivos que compõem o rastro são acessados tão somente de maneira local. Em valores numéricos a média e mediana das proporções de bytes acessados localmente são 0.974 e 1, respectivamente. De maneira complementar, isto indica a existência de um número bastante pequeno de arquivos com um tempo de vida longo e que são responsáveis por todo o desvio na localidade de acesso.

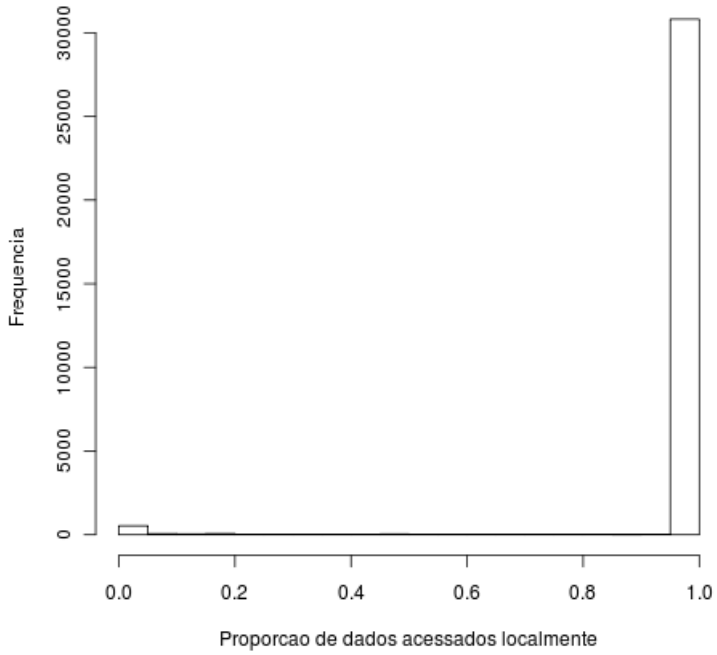

Figura 4.2: Frequência da distribuição dos arquivos quanto à proporção de dados acessados localmente

4.5 Conclusões Parciais

Neste capítulo mostramos quão eficiente pode ser o acesso aos dados em um sistema de arquivos para redes locais que implementa uma solução simplista de alocação, e que está sujeito a desvios na localidade de acesso causados pela mobilidade de seus usuários na rede.

Viu-se que apesar da simplicidade desta solução, que pode ser executada sem comunicação remota e com baixa necessidade de manutenção de estado, a maioria dos arquivos é acessada de maneira local independente do padrão de mobilidade dos usuários. No pior dos cenários avaliados, pelo menos 74% dos dados são acessados localmente. Além disso, 97% dos arquivos que compõem o rastro são acessados tão somente de maneira local.

Destes resultados pode-se derivar que ainda há um conjunto pequeno de arquivos que é responsável pela maior parte dos acessos remotos. Por seu grau de influência, estes arquivos são os candidatos naturais para novas políticas de alocação. No próximo capítulo descreve-

mos e avaliamos uma nova política que leva em consideração a existência de arquivos com sessões de uso maiores que o período de permanência em uma máquina.

Capítulo 5

Movimentando Dados

Neste capítulo analisamos como uma política de migração de arquivos pode ser associada à solução simplista de alocação descrita anteriormente, de modo a aumentar o grau de localidade no acesso aos dados apresentado pelo sistema. Esta política tem como alvo a classe de arquivos identificada no capítulo anterior que, embora tenha um tamanho reduzido, é responsável por grande parte do desvio da localidade de acesso.

A política proposta é experimentada via simulação, e avaliada através de duas métricas de desempenho: proporção de dados acessados localmente – tal como no capítulo anterior – e tempo de resposta. Os resultados obtidos permitem concluir que a associação destas duas política elevam a localidade de acesso próximo do máximo sem implicar em uma diminuição do tempo de resposta médio das operações de acesso aos dados.

Na Seção 5.1 o funcionamento da política de migração é descrito em detalhes. Na Seção 5.2 esta política de migração é avaliada através de simulações e tem sua eficácia comprovada em termos da promoção da localidade de acesso e da diminuição do tempo de resposta das operações de acesso aos dados. Na Seção 5.3 descrevemos as conclusões relacionadas à associação da política de migração descrita neste capítulo com a política de alocação descrita no capítulo anterior.

5.1 Política de Migração de Arquivos

Como foi dito na Seção 2.2 deste trabalho, os algoritmos ótimos de alocação de arquivos costumam depender de propriedades que dificilmente se confirmam na prática. Pode-se resumir

os fatores descritos naquela seção da seguinte maneira: i) não é possível prever o futuro; ii) é custoso obter informações acuradas sobre o passado recente.

As complexidades de ordem prática na adoção de soluções ótimas de alocação costumam ser um indicativo para a adoção de políticas que se adaptem aos desvios de localidade. Uma classe destas soluções são as políticas de migração de dados. De modo geral, estas políticas detectam mudanças nas taxas de acesso entre os nodos da rede de armazenamento para um determinado conjunto de dados, e por consequência alteram o conjunto de nodos responsáveis pelo armazenamento do dado em questão.

A política de migração estudada neste trabalho é aplicada em decorrência de um acesso remoto para um determinado arquivo. Após a ocorrência deste evento, deve-se decidir se este arquivo será migrado ou não. Para tanto, usou-se um critério de decisão baseado na continuidade de uso – caso o *tempo de atividade acumulado por sessão de uso* seja maior que δ unidades de tempo, este arquivo será migrado. O tempo acumulado por sessão de uso é definido como o intervalo entre o tempo atual e o primeiro acesso ao arquivo em questão na sessão de uso corrente. O valor atribuído a δ constitui o atraso de migração da política.

Quando o atraso para a migração é atingido, ou seja, quando o *tempo de atividade acumulado por sessão de uso* torna-se maior que o atraso de migração, a operação corrente é bloqueada até que todo o conteúdo do arquivo seja transferido para a máquina que originou o acesso remoto, ou seja, a máquina onde o usuário está localizado. Ao fim da transferência, umas das demais réplicas secundárias existentes do arquivo é removida, deste modo o nível de replicação do arquivo é mantido constante. Ademais, como nova integrante do grupo de replicação, a réplica criada pelo processo de migração será promovida à qualidade de primária, como descrito em 4.3.3.

Uma decisão é dita útil caso consiga melhorar o desempenho no acesso ao arquivo. Note que a transferência de dados resultante de uma migração pode acarretar em perda de desempenho. Para isto concorrem o tamanho do arquivo que precisará ser migrado e o padrão de acesso futuro, e portanto desconhecido, dos dados. Um atraso de migração suficientemente bem escolhido permite à política decidir de maneira útil sobre o conjunto de arquivos que terá maior impacto no desempenho percebido pelo cliente, bem como evitar a migração de arquivos que não são populares.

5.2 Modelo de Simulação e Análise de Resultados

Para analisar a estratégia de migração de dados, modificamos o simulador descrito na Seção 4.3.3 associando a política descrita na seção anterior à política de alocação de dados descrito no Capítulo 4 .

A estratégia de migração teve seu efeito avaliado através da métrica da proporção de dados lidos localmente, definida na Seção 4.4. Estes resultados foram comparados com aqueles obtidos na Seção 4.4 para aferir a capacidade da política de migração de incrementar a eficácia da política de alocação de dados. Ainda, o efeito da variação do atraso da migração foi analisado. A Tabela 5.1, sumariza esses resultados para um cenário composto pela política de alocação sem migração e para cenários com diferente atrasos de migração. Para os cenários que utilizam a política de migração foram adotados os valores $0, 30, 60, 120$ e 3.600 segundos para o atraso de migração. Cada um destes cenários é composto pelo resultado de 8 simulações do rastro usado no Capítulo 4. Em todos os cenários simulados, as execuções foram suficientes para alcançar um erro máximo de $\pm 0,5\%$ para a média com confiança de 99%. Nestas simulações adotou-se uma probabilidade de migração do usuário $p = 1$ e para o período de inatividade foi atribuído o valor de 1 hora.

Tabela 5.1: Sumário de dados para a proporção de dados acessados localmente em cenários compostos pela política de alocação e pela política de migração para diferentes atrasos de migração (δ), usando uma probabilidade de migração $p = 1$ e um período de inatividade de uma hora.

	Atraso de Migração (segundos)	Min	1 Qr.	Mediana	Media	3 Qr	Max
Sem Migração	–	0,7384	0,7411	0,7507	0,7523	0,7625	0,7710
Migração	0	1	1	1	1	1	1
Migração	30	0,9967	0,9967	0,9968	0,9969	0,9969	0,9975
Migração	60	0,9961	0,9961	0,9962	0,9962	0,9962	0,9966
Migração	120	0,9947	0,9948	0,9949	0,9950	0,9951	0,9957
Migração	3.600	0,9527	0,9528	0,9541	0,9541	0,9546	0,9567

Mesmo quando o atraso de migração assume um valor extremamente alto (3.600 segundos), o valor mínimo medido da proporção de *bytes* acessados localmente ($0,9527$) é consideravelmente maior que o valor máximo desta métrica quando não há migração ($0,7710$).

Outra observação importante diz respeito à pequena variação da métrica avaliada com respeito ao aumento dos valores assumidos pelo atraso de migração. Note que um aumento de 120 vezes nesta variável, de 30 para 3600, resulta em valores medianos de $0,9968$ e $0,9541$ respectivamente.

5.2.1 Migração de dados e o tempo de resposta percebido pelos clientes

Existe um compromisso na escolha do valor atribuído ao atraso de migração: se este valor for extremamente alto, nenhum arquivo é migrado e os dados continuam sendo acessados remotamente, e por consequência apresentando um tempo de resposta maior do que os acessos locais. Em oposição, quando o valor para este atraso é igual a zero, todos os arquivos são migrados e a proporção de dados acessados localmente atinge o valor máximo. Entretanto, isto pode implicar em migrações desnecessárias, e portanto, um aumento no tempo de resposta causado pela movimentação de réplicas, uma vez que a operação que causou a migração só é completada após a totalidade do arquivo ter sido transferida para a máquina em que o usuário está localizado.

Para analisar este compromisso na escolha do valor do atraso de migração, modificamos novamente o simulador descrito no Capítulo 4, desta vez para associar a noção do poder de processamento dos elementos que compõem o modelo simulado (servidor de metadados, servidor de dados e cliente). O poder de processamento deste elementos foi atribuído em termos da vazão e latência das operações que estes são responsáveis. Deste modo, o servidor de dados é capaz de ler e escrever dados em disco a uma vazão de $20MB/s$, ler e escrever dados no seu *cache* a uma vazão de $1GB/s$. As operações de metadados apresentam um tempo de resposta de $1ms$. A vazão da rede local foi atribuída em $12,5MB/s$.

Os valores para a vazão das operações de acesso ao disco e o tempo de resposta das operações sobre metadados foram escolhidos baseados na experiência obtida na construção do protótipo descrito no Capítulo 7. O valor da vazão de acesso ao *cache* foi escolhido baseado em *bechmarks* disponíveis em http://www.cs.virginia.edu/stream/peecee/Bandwidth.html. Foi assumido que a largura de banda típica para redes locais é de $100Mbits/s$, constituindo

assim o valor de vazão escolhido para a rede simulada. Estes valores estão descritos na Tabela 5.2.

Por adotar uma política que promove a localidade de acesso, o cliente não implementa nenhum mecanismo de *cache*. Por sua vez, os servidores de dados simulados usam um *cache* com uma política de substituição *LRU*, que é uma boa aproximação do algoritmo ótimo de substituição. Sistemas de arquivos modernos não costumam reservar uma região do sistema de memória quando implementam *caching*. No lugar desta alternativa competem com outros processos por todas as páginas disponibilizadas pelo sistema de memória virtual. Para simplificar a implementação do simulador adotou-se um *cache* com $64MB$ de capacidade, que corresponde a $\frac{1}{32}$ da capacidade de um *desktop* com $1GB$ de memória principal. Ainda, dados são incluídos no *cache* em blocos não menores do que $4KB$, tal como ocorre nas implementações de sistemas de arquivos.

Qualquer outro parâmetro da simulação omitido assume os mesmos valores descritos no Capítulo 4.

Tabela 5.2: Vazões e latências dos serviços modelados na simulação

Armazenamento	Metadados	Cache	Rede
Vazão (MB/s)	Tempo de resposta (ms)	Vazão (GB/s)	Vazão (MB/s)
20	1	1	12,5

Assim como nas simulações descritas na seção anterior, os cenários que utilizaram a política de migração consideram atrasos de migração com os valores $0, 30, 60, 120$ e 3.600 segundos. Além destes, também foi simulado o cenário que aplica apenas a política de alocação. Para cada um dos cenários foram realizadas 8 simulações do rastro utilizado no Capítulo 4.

Os resultados destas simulações foram analisados em termos do tempo de resposta percebido pelos clientes em operações de leitura. Diferente da métrica de promoção de localidade usada anteriormente, as operações de escrita são excluídas para tornar esta avaliação independente do mecanismo de atualização das réplicas dos dados.

Como ilustra a distribuição empírica acumulada do valores médios para tempo de resposta em operações de leitura, apresentada na Figura 5.1, não há diferença perceptível para

pelo menos 50% das operações realizadas. Assim como nas análises relacionadas à promoção da localidade de acesso, este resultado é decorrente da presença de um grande número de arquivos que apresentam um tempo de vida curto, e por consequência não são afetados pela mobilidade do usuário da rede de armazenamento.

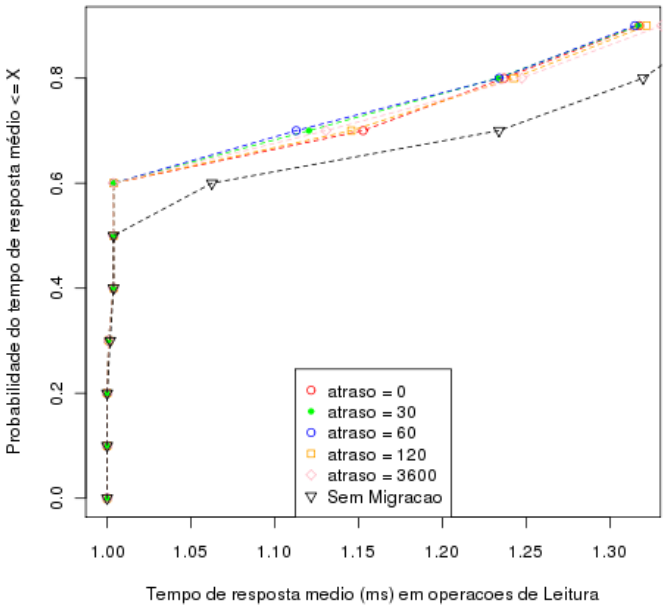

Figura 5.1: Comparação do tempo médio de resposta em operações de leitura em cenários compostos pela política de alocação e pela política de migração para diferentes atrasos de migração (δ), usando uma probabilidade de migração $p = 1$ e um período de inatividade de uma hora.

Para a outra metade das operações, a política de migração de dados permite a diminuição do tempo de resposta médio em comparação ao cenário que utiliza apenas a política de alocação para todos os valores testados do atraso de migração.

Para melhor visualização, a mesma distribuição empírica acumulada é ilustrada na Figura 5.2, desta vez ressaltando os quantis maiores (entre 0.8 e 0.99). Como era esperado, quanto maior o atraso de migração maior o tempo de resposta, pois arquivos que poderiam tirar proveito da migração continuam sendo acessados remotamente. Entretanto esta diferença só se torna clara para valores altos. Quando o valor do atraso de migração assume os valores $0, 30, 60$ e 120 não há diferença perceptível no tempo de resposta médio.

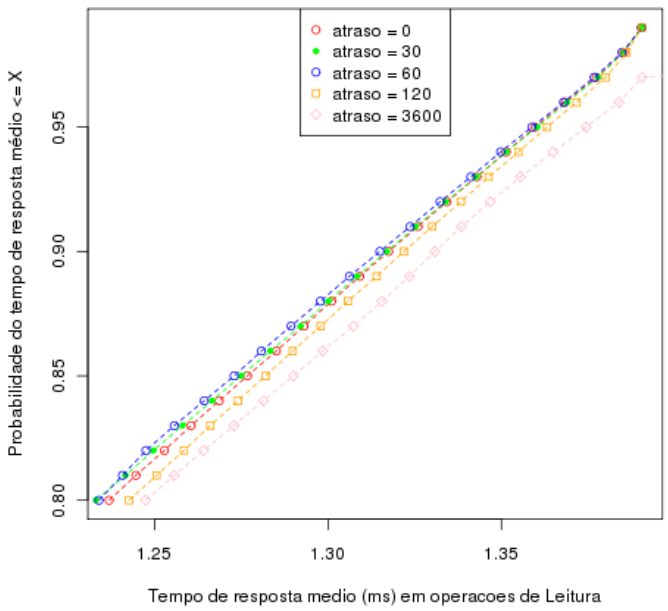

Figura 5.2: Comparação do tempo médio de resposta em operações de leitura em cenários compostos pela política de alocação e pela política de migração para diferentes atrasos de migração (δ), usando uma probabilidade de migração $p = 1$ e um período de inatividade de uma hora, ressaltando os maiores quantis.

Pode-se concluir que arquivos que apresentam tempo de vida suficientemente grande para serem acessados em sessões de uso distintas também têm potencial para serem acessados durante tempo suficiente após uma migração, e por consequência tornar o tempo de transferência das réplicas devido à migração insignificante. Ao considerar os compromissos relacionados à implementação desta política de migração, os resultados dão suporte à escolha da migração bloqueante (atraso de migração $= 0$).

5.3 Conclusões Parciais

Neste capítulo descrevemos e analisamos uma política de alocação de dados, que utilizando apenas informação local, é capaz de promover a localidade de acesso a valores próximos do máximo quando associada à política de alocação descrita no capítulo anterior. Também mostrou-se que a promoção da localidade de acesso é atingida sem prejuízos para o tempo

de resposta percebido pelo cliente do sistema de arquivos para os traços utilizados.

Capítulo 6

Uma alternativa a redes de armazenamento centralizadas

Recapitulando o que foi dito no Capítulo 3, o problema de alocação investigado neste trabalho considera uma rede de armazenamento composta pelos seguintes componentes: um servidor de metadados, servidores de dados e clientes. Considera-se que o servidor de metadados é implantado em uma máquina dedicada, enquanto os servidores de dados e clientes em máquinas de menor capacidade e de uso geral.

Neste modelo de distribuição, que diverge do modelo comum centralizado, o mecanismo de alocação de arquivos tem um papel fundamental. Cargas desbalanceadas devido à má alocação dos arquivos têm um impacto imediato tanto no tempo de resposta do sistema de armazenamento, bem como nos demais serviços em execução nas máquinas que armazenam os dados.

Embora tenha sido demonstrado nos Capítulo 4 e Capítulo 5 que as políticas de alocação e migração promovem de fato a localidade de acesso e consequentemente trazem ganhos de desempenho, os servidores de dados ainda são componentes não dedicados – tipicamente menos eficientes do que servidores utilizados em instalações que adotam o modelo centralizado. Este modelo de distribuição, embora tenha desvantagens apontadas em vários trabalhos [23, 40, 1, 16, 34, 35, 19, 29, 30], ainda é dominante para redes locais de armazenamento.

Diante disso, uma pergunta natural que surge é: seriam as políticas de migração e alocação capazes de incrementar o desempenho de um sistema de armazenamento composto por máquinas de uso geral a ponto de tornar seu desempenho equivalente às alternativas

centralizadas típicas?

Na Seção 6.1 descrevemos o modelo de simulação utilizado para responder esta pergunta. Por fim, os resultados destas simulações são discutidos na Seção 6.2. Conclusões acerca destes resultado são reunidas na Seção 6.3.

6.1 Modelo de Simulação

Para responder esta pergunta, submetemos o rastro de simulação utilizado no Capítulo 4 e 5 à mesma dinâmica descrita naqueles capítulos usando um modelo de simulação diferente. Neste novo modelo representado na Figura 6.1, as operações sobre metadados e dados são realizadas por um único servidor implantado em uma máquina de propósito específico. Este servidor é acessado por clientes distribuídos nas demais máquinas da rede.

Tanto o servidor central quanto os clientes implementam um *cache* com uma política de substituição LRU e cujos dados são inseridos em blocos não menores do que $4KB$. O *cache* no servidor tem capacidade de $256MB$, enquanto o *cache* dos clientes tem capacidade de $64MB$. Vale lembrar que no modelo adotado anteriormente descrito no Capítulo 3, os clientes não faziam uso de *cache*, já que as políticas de alocação promovem o acesso local como mecanismo de ganho de desempenho.

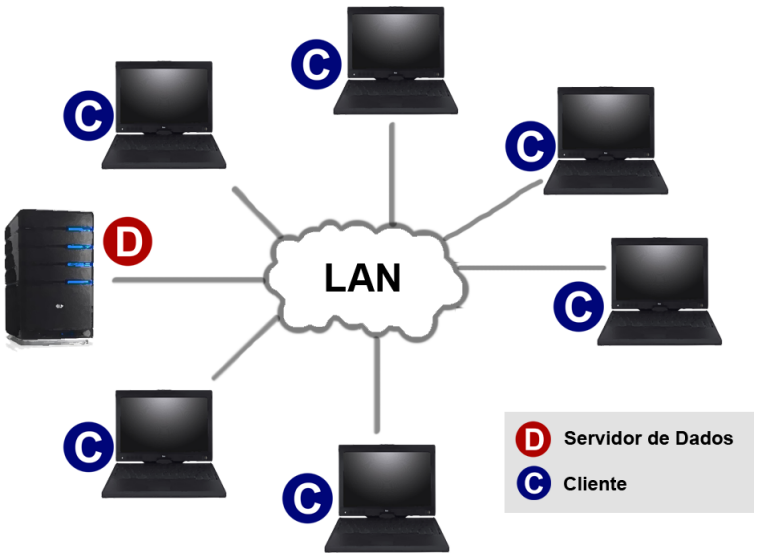

Figura 6.1: Rede de Armazenamento que Aplica o Modelo Cliente-Servidor.

Para encontrar um configuração centralizada equivalente à distribuída, simulamos servi-

dores com capacidades de processamento diversas. Estas e outras capacidades dos demais elementos simulados estão sumarizadas na Tabela 6.1. As justificativas para os valores adotados podem ser conferidas na Seção 5.2.1.

Tabela 6.1: Vazões e latências dos serviços simulados no modelo centralizado e distribuído

	Armazenamento	Metadados	Cache	Rede
	Vazão (MB/s)	Latência (ms)	Vazão (GB/s)	Vazão (MB/s)
Centralizado	40, 60, 80	1	1	12
Distribuído	20	1	1	12

6.2 Análise de Resultados

O modelo centralizado descrito na seção anterior foi analisado em comparação ao modelo usado no Capítulo 4 que utiliza a política de alocação, e ao modelo descrito no Capítulo 5, que utiliza as políticas de migração e alocação. Para os cenários que usam a política de migração adotou-se o valor 0 para o atraso de migração.

A Figura 6.2, reporta a função de distribuição acumulada para o tempo de resposta médio em operações de leitura resultante da simulação do rastro para os cenários descritos anteriormente.

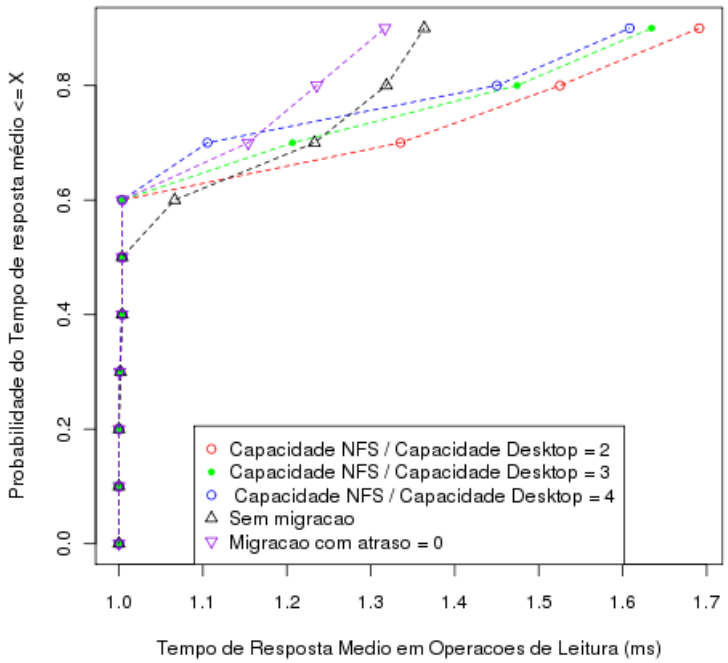

Figura 6.2: Comparação do tempo médio de resposta em operações de leitura em cenários compostos pelos modelos centralizado e descentralizado de armazenamento. Este último composto pela política de alocação e pela política de migração com atraso igual a zero, usando uma probabilidade de migração $p = 1$ e um período de inatividade de uma hora.

Vê-se que a adoção das políticas de alocação e migração permite atingir desempenho superior a um servidor centralizado de vazão 2 vezes maior do que o correspondente não dedicado. Ainda, em pelo menos 70% dos acessos estas políticas têm desempenho equivalente a um servidor dedicado de vazão 3 vezes maior. Por fim, em pelo menos 50% das operações de leitura não há diferença evidente de desempenho entre o modelo distribuído e qualquer uma das configurações centralizadas simuladas.

6.3 Conclusões Parciais

Embora instalações centralizadas de armazenamento sejam tipicamente constituídas por máquinas de desempenho elevado, este modelo tem um problema de escalabilidade inerente – todas as requisições são atendidas por um único servidor. É necessário ressaltar que

as simulações efetuadas neste trabalho não correspondem a uma carga típica – na medida em que apenas um cliente foi simulado. Este é mais um fator que contribui para uma conclusão favorável ao modelo distribuído suportado pelas políticas de alocação e migração desenvolvidas. O Capítulo 7 examina em mais detalhes a questão da escalabilidade em sistemas de armazenamento centralizados via a experimentação com um protótipo.

Capítulo 7

BeeFS

Este capítulo descreve os detalhes de implementação do BeeFS (Beehive File System). Este sistema de arquivos é uma prova de conceito do projeto descrito no Capítulo 3. Além disto, o BeeFS é um protótipo para a política de alocação avaliada no Capítulo 4.

Adicionalmente à descrição do funcionamento do BeeFS, este capítulo apresenta uma avaliação deste sistema com respeito ao desempenho no acesso aos dados em uma coleção de operações típicas para sistemas de arquivos de uso geral. Esta avaliação levou em consideração os dois cenários de acesso ao dados possíveis no BeeFS: no primeiro, todos os dados são armazenados na mesma máquinas em que o cliente usado no experimento está implantado; no segundo todos os dados são acessados de maneira remota. Estes resultados foram qualificados através de uma comparação com o NFS, que representa o estado-da-prática para sistemas de arquivos de uso geral em redes locais. Os resultados deste experimento mostram que o BeeFS é mais eficiente que o NFS para operações de escrita, leitura e acesso aos metadados para ambos os cenários experimentados. Desse modo, o ganho qualitativo do BeeFS – a diminuição de custos de propriedade resultante da agregação do espaço subutilizado dos *desktops* – é alcançado sem perda de desempenho.

O resto deste capítulo é organizado da seguinte forma. Na seção 7.1 são descritos detalhes da implementação do BeeFS. Em seguida na seção 7.2 a avaliação experimental é descrita, bem como os resultado obtidos. Na Seção 7.3 são discutidos, o propósito da construção do BeeFS, os resultados obtidos em sua avaliação e sua relação com as políticas desenvolvidas neste trabalho.

7.1 Arquitetura do BeeFS

7.1.1 Visão Geral

Como definido no Capítulo 3, o BeeFS possui três módulos principais: um servidor de metadados, servidores de dados e clientes. As funcionalidades destes componentes, bem como sua distribuição nas máquinas que fazem parte da rede local seguem exatamente aquilo que foi proposto no Capítulo 3. Daqui em diante, o servidor de metadados, os servidores de dados e os clientes podem adotar de maneira intercambiável os nomes *queen-bee*, *honeycomb* e *honeybee*, respectivamente. Esta correspondência entre os módulos contidos no projeto e os nomes adotados pelo BeeFS está ilustrada na Figura 7.1.

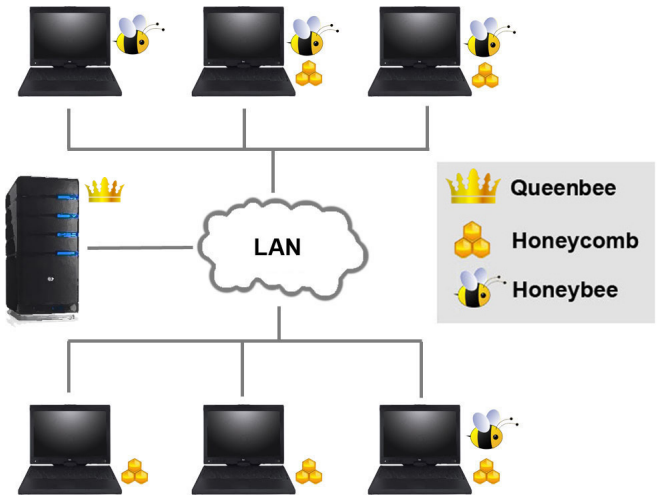

Figura 7.1: Visão geral dos módulos do BeeFS e sua ditribuição da rede local.

Todos os componentes do BeeFS foram implementados usando a linguagem Java. O único módulo dependente do sistema operacional é o cliente. O acoplamento do cliente com o VFS (*Virtual File System*) do Linux, que permite a utilização do BeeFS por aplicações legadas, é realizado através do FUSE, como esquematizado na Figura 7.2. Uma nova versão do módulo cliente que permite a utilização deste componente em ambiente Windows está em desenvolvimento.

A versão Linux do cliente BeeFS é aderente à especificação POSIX para sistemas de arquivos. A aderência ao padrão POSIX foi testada em um conjunto de testes de compatibil-

idade [1] originalmente criado para aferir a compatibilidade do ZFS para o sistema operacional FreeBSD.

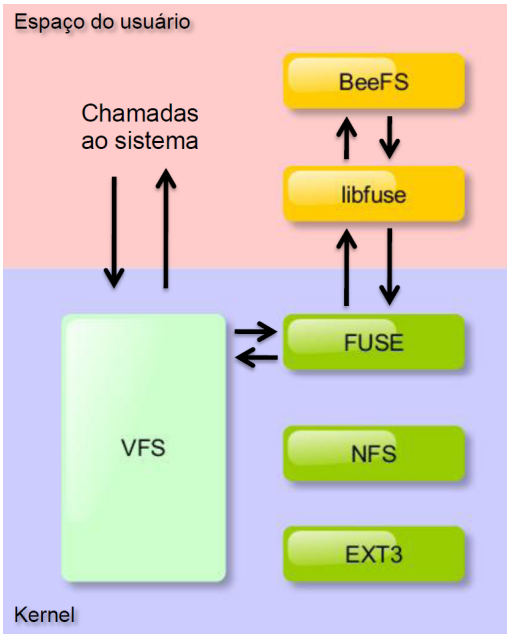

Figura 7.2: Acoplamento entre o cliente BeeFS e o *Virtual File System* - Fuse.

7.1.2 Dados, Metadados e Atributos dos Arquivos

No BeeFS, diferente de muitos sistemas de arquivos distribuídos, não há mecanismo de *caching* para os dados acessados pelo cliente. Por haver uma única cópia ativa para cada arquivo armazenado, qualquer sequência de operações sobre os dados têm um resultado equivalente à execução da mesma sequência em um sistema de armazenamento local [33].

A ausência de *caching* no *honeybee* pode acarretar uma perda de desempenho severa no acesso aos dados. Este problema é aliviado no BeeFS pela adoção da política de alocação de dados descrita no Capítulo 4, que promove a localidade de acesso ao levar em conta que *honeybees* e *honeycombs* podem ser implantados na mesma máquina.

Diferente das operações de acesso aos dados, o *honeybee* faz *cache* de metadados. Esta opção é apoiada por um trabalho de caracterização da carga em sistemas de arquivos [27] de uso geral, que mostrou que 98% das operações *stat* executadas nestes sistemas são seguidas

[1] (http://www.tuxera.com/community/posix-test-suite/

por outra operação *stat*. Em particular, a leitura de metadados de um diretório implica da na leitura dos metadados dos arquivos que estão contidos neste diretório. Metadados de um arquivo são excluídos do *cache* quando operações que modificam este arquivo são realizadas ou quando um *timeout* é atingido. Esta abordagem para *caching* de metadados é bastante semelhante àquela implementada no NFS [25].

Os metadados armazenados pelo servidor *queen-bee* podem ser divididos em duas classes: atributos comuns e estendidos. Os atributos comuns contêm propriedades de arquivos tais como tamanho, dono, grupos, permissões de acesso, estampilhas de tempos de acesso e modificação. Os atributos estendidos são coleções de duplas codificadas na forma de um identificador e um valor associado. Estes atributos permitem que aplicações cientes do BeeFS possam expandir a funcionalidade do sistema usando meta-informação associada aos arquivos. Também permite ao BeeFS, como ficará mais claro nas próxima seções, um meio de comunicação com aplicações ou usuários usando a *API* do sistema [2].

A quantidade de espaço necessária para manter estes atributos pode ser considerada pequena. Como aponta um estudo sobre o conteúdo de sistemas de arquivos [9], levando em consideração que usuários típicos armazenam cerca de 6000 arquivos e para cada arquivo fossem necessários 100 *bytes* para armazenar os atributos, ainda assim seria suficiente usar apenas $600KB$ por usuário do sistema. Em consequência, o BeeFS adota uma política similar ao GFS [15] para armazenar os metadados: mantém esta informação na memória e periodicamente envia as atualizações para o disco.

7.1.3 Tolerância a Faltas

Por serem implantados em máquinas não dedicadas, os *honeycombs* estão sujeitos à falhas e paradas não anunciadas de maneira mais frequente que as alternativas centralizadas tradicionais de armazenamento. Por este motivo, para tolerar faltas no serviço de armazenamento, o BeeFS emprega um modelo de replicação com cópia primária não-bloqueante. Cada arquivo é replicado em um conjunto de *honeycombs* que constituem o grupo de replicação para este arquivo. Como descrito no Capítulo 4, na formação do grupo de replicação, a réplica

[2]Embora não faça parte do padrão POSIX, atributos estendidos estão presentes em alguns sistemas de arquivos bastante usados como: ext2, ext3, ext4, JFS, ReiserFS, XFS e Btrfs para Linux, UFS1 e UFS2 para FreeBSD e ffs para OpenBSD

primária é escolhida considerando uma política de alocação que favorece a localidade de acesso, enquanto as réplicas secundárias são escolhidas aleatoriamente entre os conjunto de *honeycombs* disponíveis. O BeeFS permite que seja atribuído um nível de replicação distinto para cada arquivo do sistema.

A sequência de operações necessárias para a realização de uma operação sobre um determinado arquivo no BeeFS está sumarizada na Figura 7.3. Primeiro, uma mensagem de controle é enviada do *honeybee* para o servidor *queen-bee* quando uma chamada *open* é realizada. Em resposta à esta mensagem de controle, o servidor *queen-bee* retorna a identidade dos servidores que compõem o grupo de replicação associado ao arquivo para o qual a chamada *open* foi feita. De posse da identidade dos servidores do grupo de replicação, o *honeybee* realiza as operações requisitadas no servidor que contém a réplica primária.

Eventualmente, o conteúdo da réplica primária é atualizado para as réplicas secundárias [2]. A principal razão que sustenta esta abordagem é a seguinte: a maioria dos arquivos em um sistema de arquivos para uso tem um tempo de vida curto [27], portanto, haveria uma alta probabilidade de transferências desnecessárias caso as atualizações fossem realizadas atomicamente no conjunto completo de réplicas.

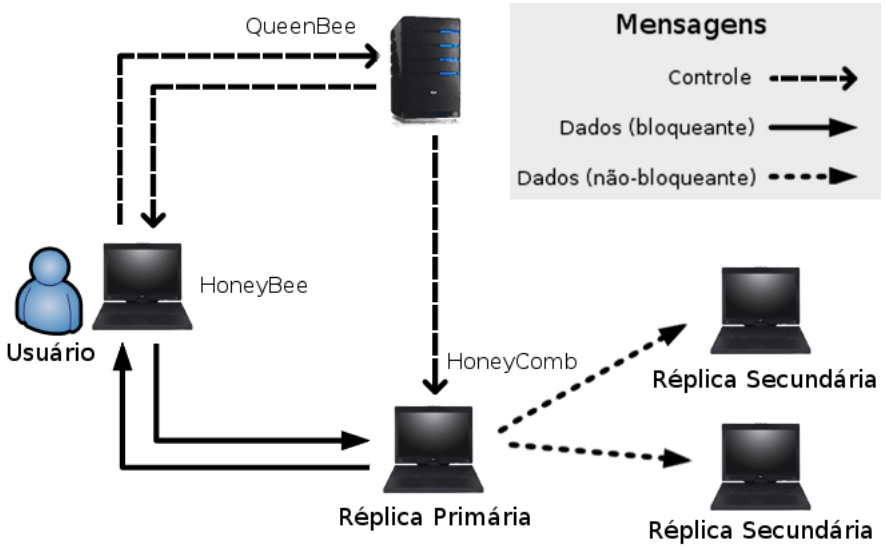

Figura 7.3: Modelo de replicação adotado pelo BeeFS.

O servidor *queen-bee* é responsável por coordenar este processo de atualização das réplicas secundárias. Para tanto, ele mantém uma visão do estado de cada grupo de replicação

conhecido. Esta visão contém a versão de cada réplica armazenada nos *honeycombs*. A versão de uma réplica primária é alterada e mantida pelos *honeycombs* após cada operação *write* ou *truncate* realizada sobre esta réplica. Após uma operação *close*, que pode ter sido precedida por várias modificações sobre determinado arquivo, este identificador de versão é enviado pelo *honeybee* para o servidor *queen-bee*, que deste modo atualiza sua visão sobre o grupo de replicação envolvido. Caso seja detectado que as réplicas possuem versões diferentes, o servidor *queen-bee* escalona um processo de atualização do conteúdo entre a réplica primária e as secundárias. Assim como o nível de replicação, o atraso entre a atualização da visão do estado de um grupo de replicação e o processo de coerência das réplicas secundárias é uma propriedade de cada arquivo armazenado pelo BeeFS.

Por fim, o servidor *queen-bee* também é responsável por monitorar os *honeycombs*. Para tanto, os *honeycombs* enviam mensagens de controle para o servidor *queen-bee* periodicamente. Quando o servidor *queen-bee* deixa de receber mensagens de controle de um determinado *honeycomb* por mais que N unidades de tempo assume-se que este *honeycomb* está indisponível. Quando a indisponibilidade do *honeycomb* é detectada, todos os grupos de replicação afetados precisam ser reorganizados para manter o nível de replicação desejado. Para os grupos de replicação em que o *honeycomb* indisponível era um *servidor* primário, a reorganização só será possível se houver um servidor *honeycomb* secundário com o mesmo número de versão que o primário. O acesso às replicas desatualizadas é sempre negado, até que a réplica primária esteja disponível novamente.

Com respeito ao servidor *queen-bee* são consideradas dois tipos de falhas: transientes e permanentes. Para falhas transientes, o BeeFS assume um modelo de falhas *crash-recovery*. Uma falha transiente impactará todas as operações que modificaram o estado do servidor *queen-bee* e que não tenham sido feitas permanentes. Como alguns sistemas de arquivos centralizados e distribuídos, o BeeFS periodicamente envia os metadados mantidos na memória do servidor *queen-bee* para o disco com o objetivo de diminuir a janela de inconsistência para falhas. Em recuperando-se, o servidor *queen-bee* realiza uma procedimento de análise de consistência. Para isto os metadados dos arquivos, armazenados tanto nos *honeycombs* quanto no servidor *queen-bee* são confrontados.

Falhas permanentes também são tratadas pela replicação do estado do servidor *queen-bee* nos servidores *honeycombs*. Neste caso é responsabilidade do administrador do sistema

a inicialização de uma nova instância deste, neste caso em modo de recuperação. Durante o procedimento de recuperação, os servidores *honeycombs* são contactados e o estado do servidor *queen-bee* é reconstituído [32].

Tanto no caso transiente como no permanente, não há garantia de que o sistema será totalmente recuperado. Entretanto é possível discernir entre os arquivos recuperados e aqueles que estão em um estado de inconsistência. Para este conjunto de dados é permitido ao administrador decidir sobre a reintegração.

7.1.4 Segurança

Os *honeycombs* que armazenam os arquivos servidos pelo BeeFS estão implantados em máquinas de uso não-exclusivo. Por este motivo, medidas que garantam a privacidade e integridade no acesso aos arquivos precisam ser adotadas. Farsite [1], outro sistema de arquivos que utiliza máquinas não dedicadas de redes locais, resolve este problema usando criptografia, técnica evitada pelo BeeFS devido ao impacto no desempenho. Embora não tão eficaz quanto criptografia, o BeeFS utiliza as capacidades de controle de acesso do sistema de arquivos sobre o qual o servidor *honeycomb* está instalado, ao atribuir um conjunto de permissões aos arquivos que permite apenas aos processos que compõem o serviço provido pelo BeeFS acessá-los.

7.2 Avaliação experimental

Nessa seção é descrita a avaliação de desempenho do BeeFS. Esta avaliação foi composta por uma série de operações típicas em sistemas de arquivos de uso geral e levou em consideração os dois cenários de acesso aos dados possíveis no BeeFS: no primeiro, todos os dados são armazenados nas mesmas máquinas em que o cliente usado no experimento está implantado, enquanto no segundo todos os dados são acessados de maneira remota. Estes resultados foram qualificados através de uma comparação com o NFS, que representa o estado-da-prática para sistemas de arquivos de uso geral em redes locais. Os resultados deste experimento mostram que o BeeFS é mais eficiente que o NFS para operações de escrita, leitura e acesso aos metadados para ambos os cenários experimentados. Deste modo, o ganho qualitativo do BeeFS – a diminuição de custos de propriedade resultante da agregação

do espaço sub-utilizado dos *desktops* – é alcançado sem perda de desempenho.

A seguir, é descrito o *benchmark* utilizado para executar os testes de desempenho, apresentação dos resultados obtidos e análise dos mesmos.

7.2.1 O *benchmark* Andrew

Com objetivo de medir o desempenho do BeeFS para uma série de operações típicas em sistemas de arquivos de uso geral, adotou-se o Andrew Benchmark [17]. Este *benchmark* simula a carga de trabalho de compilação de um código fonte. Ele é composto por 5 fases: `mkdir`, `copy`, `stat`, `grep` e `compile`.

1. A fase **Mkdir** cria a hierarquia de diretórios do código-fonte original no sistema de arquivos que está sendo testado;

2. A fase **Copy** copia todos os arquivos da árvore original para a árvore criada;

3. A fase **Stat** busca o *status* de todos os arquivos na árvore sem examinar seu conteúdo;

4. A fase **Grep** lê todo o conteúdo dos arquivos criados durante a fase de cópia;

5. A fase **Compile** compila e liga os arquivos.

O *benchmark* Andrew foi executado usando como entrada o código fonte do aplicativo Nagios 3.0.6 (disponível em `http://www.nagios.org/`), o qual contém 21 diretórios, 515 arquivos e 9 Mbytes de informação.

Foram consideradas duas configurações para os experimentos com o BeeFS e uma configuração utilizando NFS, que são descritas a seguir:

- **BeeFS co-alocado**, a qual contém o servidor *queenbee* rodando numa máquina e o cliente *honeybee* e o servidor *honeycomb* co-alocados em outra máquina;

- **BeeFS não co-alocado**, a qual consiste no servidor *queenbee*, o cliente *honeybee* e o servidor *honeycomb* instalados em máquinas diferentes;

- **NFS**, a qual contém o servidor NFS executando numa máquina e o cliente NFS em outra máquina.

Todas as medidas de desempenho foram feitas em máquinas com processadores de dois núcleos 3.2 GHz Intel e 2 Gbytes de memória RAM cada, executando Ubuntu 9.04 *kernel* $2.6.28-11$-vserver. Todos os nós foram conectados através de uma rede 100 Mbit Ethernet que foi isolada do tráfego pesado durante a execução do experimento.[3]

A Tabela 7.1 mostra o tempo médio da execução do *benchmark* para as 3 configurações descritas, bem como o percentual de melhoria do tempo de execução para BeeFS Co-alocado e BeeFS Não Co-alocado em relação ao NFS. O *bechmark* foi executado um número suficiente de vezes que garante um nível de confiança de 95%. Para todas as configurações do BeeFS, ambas as fases **Mkdir** e **Stat** possuem melhorias semelhantes — em torno de 40% para `mkdir` e 26% para `stat` —, isto acontece porque essas fases consistem em um fluxo de mensagens de controle para o servidor *queenbee*.

Tabela 7.1: Média do tempo de execução (em ms) para cada fase individual para cada configuração: NFS, BeeFS co-alocado e BeeFS não co-alocado.

Fase	NFS	BeeFS	
		BeeFS co-alocado	BeeeFS não co-alocado
Mkdir	29.4	17.8 (40%)	18.4 (37%)
Copy	9,118.5	2,362.3 (74%)	3,954.8 (56%)
Stat	2,434.4	1,794.9 (26%)	1,712.9 (29%)
Grep	3,073.2	2,190.6 (29%)	2,481.5 (20%)
Compile	43,075.0	38,537.3 (10%)	43,472.5 (-1%)
Total	57,730.5	44,902.9 (22%)	51,640.1 (10%)

Por outro lado, as fases de leitura e escrita intensa em disco (`copy` e `grep`) tiveram melhorias diferentes, refletindo o benefício da co-alocação; enquanto configurações co-alocadas realizam operações no disco local, máquinas não co-alocadas e NFS transmitem os dados pela rede.

[3] Alguns serviços distribuídos como o **NTP** e **IMAP** estavam rodando durante a execução do experimento. Entretanto, o tráfego de rede gerado por esses serviços não é substancial e não gera impacto significativo nos resultados do experimento.

7.2.2 Avaliação de Escalabilidade

A propriedade de escalabilidade do BeeFS foi avaliada por meio de cenários em que múltiplos clientes executaram simultaneamente o *benchmark* em diferentes sub-árvores do sistema de arquivos. Esse experimento usou as configurações *NFS* e *BeeFS Co-alocado* descritas na seção anterior.

A Figura 7.4 ilustra a comparação da média dos tempos de execução percebido pelo cliente do *benchmark* Andrew, para ambas configurações, em função do número corrente de clientes. Destes resultados nota-se que o tempo de execução do *benchmark* tem um crescimento menos acelerado no BeeFS do que no NFS. Esse resultado confirma a suposição que a arquitetura híbrida diminui o gargalo do servidor central. Assumindo um comportamento de um modelo linear desse experimento, quando comparado com o NFS, o BeeFS possui performance semelhante àquela do NFS quando exposto a uma carga 4 vezes menor.

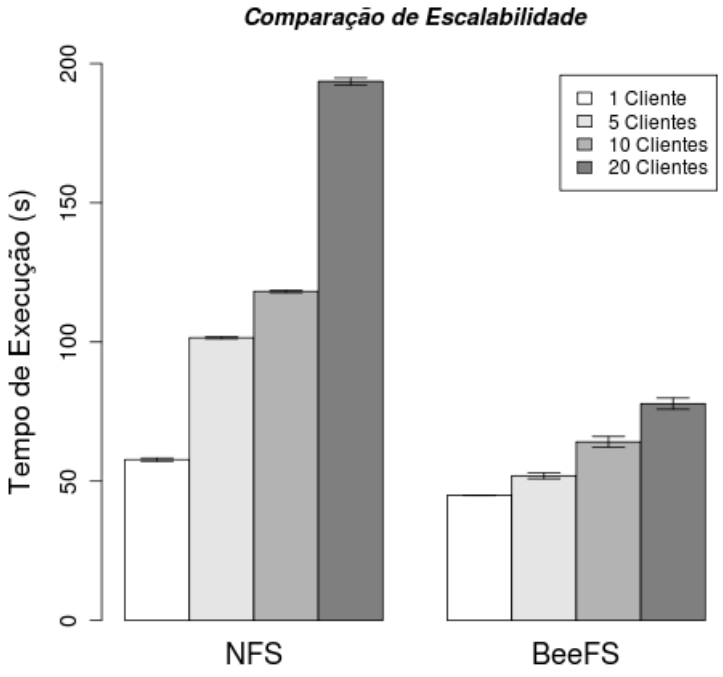

Figura 7.4: Média do tempo total de execução (em segundos) para NFS e BeeFS co-alocado em função do número de clientes.

7.3 Conclusões Parciais

Neste capítulo apresentamos o projeto e detalhes de implementação do BeeFS, um sistema de arquivos que utiliza o espaço livre em disco nos *desktop* existentes em redes locais. O BeeFS foi desenvolvido como prova do conceito da utilização deste espaço ocioso como solução para o problema do crescimento da demanda de armazenamento descrito no Capítulo 1.

Como descrito na Seção 7.2.1, o BeeFS apresenta desempenho superior ao NFS quando os clientes executam na mesma máquina que os servidores de dados. O resultado desta avaliação, quando associado aos resultados obtidos nos capítulos anteriores, que mostram uma promoção do acesso local próxima ao máximo, indicam que o problema do crescimento da demanda de armazenamento pode ser resolvido pelo BeeFS sem perda de desempenho. Os resultados descritos na Seção 6.2, que apontam uma equivalência de desempenho entre a alternativa representada pelo BeeFS e um sistema centralizado de capacidade duas vezes maior, reforçam esta indicação, uma vez que os resultados da avaliação contida neste capítulo consideram máquinas de mesma capacidade.

Capítulo 8

Conclusões e Trabalhos Futuros

Este capítulo contém considerações, apresentadas na Seção 8.1, acerca dos resultados descritos ao longo desta dissertação. Ainda, desdobramentos destes resultados em termos de trabalhos futuros são discutidos na Seção 8.2.

8.1 Conclusões

Nesta dissertação mostrou-se que é possível tornar eficiente o acesso aos dados em sistemas de arquivos que agregam o espaço ocioso de *desktops* através da aplicação de políticas de alocação e migração de arquivos. Estas políticas consideram um sistema de arquivos em que servidores de dados e clientes executam na mesma máquina, e tiram proveito deste fato para promover a localidade de acesso, e consequentemente obter ganhos de desempenho. A promoção de acessos locais implica em ganhos de desempenho ao evitar as latências envolvidas na transferência remota de dados e diminuir a contenção da rede. Adicionalmente aos ganhos de desempenho, acessar dados localmente permite limitar a intrusividade do sistema. Ambas as política, de alocação e migração, utilizam apenas informação local, o que facilita bastante a sua implementação.

Como prova de conceito da capacidade de agregar o espaço ocioso dos *desktops*, criamos o BeeFS (Beehive File System). Consideramos que associada à capacidade de agregar este espaço ocioso, provada com a construção do BeeFS, obter desempenho no mínimo equivalente às soluções convencionais para redes locais é uma condição necessária para a adoção do BeeFS. Nesta direção, a política de alocação de arquivos foi incorporada ao BeeFS, e

avaliada neste trabalho. Estas avaliações apontam que o BeeFS tem desempenho em geral superior, e nos piores casos equivalente ao NFS, o estado-da-prática para sistemas de arquivos em redes locais.

8.2 Trabalhos Futuros

8.2.1 Um modelo de simulação mais acurado

A avaliação das políticas consideradas neste trabalho foi feita via simulação. Como descrito no Capítulo 4, a carga de trabalho simulada foi gerada por rastros de utilização. Entretanto estes rastros não são completos para os fins da nossa análise, por exemplo, não contêm as informações referentes aos metadados do sistema. Na falta desta informação, geramos o conteúdo dos metadados baseando-se em caracterizações de sistemas de arquivos existentes na literatura. Outra limitação dos rastros utilizados é o seu tamanho em termos do número de computadores instrumentados: é provável que os usuários destas máquinas tenham uma carga de trabalho semelhante uma vez que participavam de um mesmo grupo de trabalho, entretanto é discutível a adoção desta carga de trabalho como representante fiel para toda a classe de sistemas de arquivos de uso geral para redes locais. Como solução para estes problemas estamos desenvolvendo um arcabouço próprio para a coleta e análise de rastros de utilização. A princípio, estes rastros serão coletados em três laboratórios da Universidade Federal de Campina Grande, sendo um laboratório de pesquisa e dois de ensino. Estes laboratórios agregam cerca de 120 *desktops*.

Por fim, utilizamos estes rastros também para inferir o modelo de mobilidade dos usuários da rede local. Quando a ausência de registros no rastro se prolongava por um intervalo de tempo determinado considerávamos que o usuário associado àquela carga de trabalho iniciava uma nova sessão de uso. Por sua vez, nosso modelo considerava uma probabilidade associada de migração a cada uma destas sessões de uso. Uma solução mais adequada seria instrumentar os registros de início e término das sessões iniciadas pelos usuários da rede local. Isto será realizado como parte do arcabouço de instrumentação citado anteriormente.

8.2.2 Análise de utilidade das políticas de alocação de arquivos

Neste trabalho mostramos que é possível promover a localidade de acesso, e consequentemente obter ganhos de desempenho ao adotar políticas simples de alocação e migração de arquivos. Entretanto, pelas análises que realizamos não é possível discernir quão distantes as soluções providas por estas políticas estão das soluções ótimas em termos do desempenho no acesso aos dados. Vale lembrar que atingir o resultado máximo de promoção da localidade de acesso, tal como ocorre com a política de migração quando não há atraso de migração, não implica necessariamente em alcançar o valor ótimo possível de ganho de desempenho.

Para investigar esta questão consideramos adotar duas alternativas. A primeira delas consiste em simplificar o nosso modelo à medida em que seja possível resolver as políticas analiticamente e compará-las com os algoritmos ótimos de alocação de dados discutidos no Capítulo 1. Ao adotar esta alternativa, como de costume, faz-se necessário discutir os efeitos causados pela simplificação do modelo. Uma segunda alternativa, que não responde completamente a pergunta inicial desta seção mas é simples de ser executada, é fundamentada na análise de utilidade das decisões tomadas pelas política. Como dissemos no Capítulo 5, uma decisão é considerada útil caso consiga melhorar o desempenho no acesso ao arquivo envolvido. Como os acessos futuros são descritos pelos registros do rastro, as escolhas feitas em cada decisão podem ser confrontadas com o resultado alternativo de sua contrapartida.

8.2.3 Avaliação de novas políticas de alocação de arquivos

As análises sugeridas na seção anterior podem indicar a necessidade de políticas menos simplistas do que as estudadas neste trabalho. As políticas que consideramos até agora não utilizam nenhuma informação sobre os arquivos que estão sendo manipulados. Nossa intuição é que informações como tamanho dos arquivos, grau de compartilhamento de arquivos e o conjunto de trabalho associado à carga de trabalho dos usuários podem ser utilizadas para incrementar o resultado das políticas descritas anteriormente.

8.2.4 Implementação das políticas no BeeFS

Como dissemos no Capítulo 1, este trabalho faz parte de um esforço de pesquisa em torno da construção do BeeFS. Por esta razão, é importante que os novos algoritmos de alocação

sugeridos neste capítulo bem como o algoritmo de migração de dados descrito no Capítulo 5 tenham sua implementação considera no BeeFS, que em sua versão corrente já implementa a política de alocação descrita no Capítulo 4.

Capítulo 9

Caracterização dos Rastros de Simulação

Neste capítulo analisamos os rastros de simulação descritos no Capítulo 4 quanto ao tempo de vida e popularidade dos arquivos. Na Seção 9.1 mostramos que a maioria dos arquivos referenciados nos rastro têm um tempo de vida curto. Na Seção 9.2 mostramos que há uma desigualdade na distribuição de popularidade dos arquivos: muitos arquivos são pouco populares e poucos arquivos são bastante populares. Estas duas características estão presentes em todos os rastros que possuem duração maior que um mês.

9.1 Desigualdade no Tempo de Vida dos Arquivos

Ao analisar os rastros descritos na Seção 4.3.1, foi possível identificar que a maioria dos arquivos tem um tempo de vida – o período entre a primeira e a última chamada – muito curto.

Por exemplo, no rastro mais longo (13 meses), verificou-se que o tempo de vida de 86% dos arquivos não dura mais que 1 milissegundo. Nesta mesma direção, um número bastante reduzido de arquivos contidos neste rastro apresenta um tempo de vida maior do que algumas horas, como mostra a Figura 9.1.

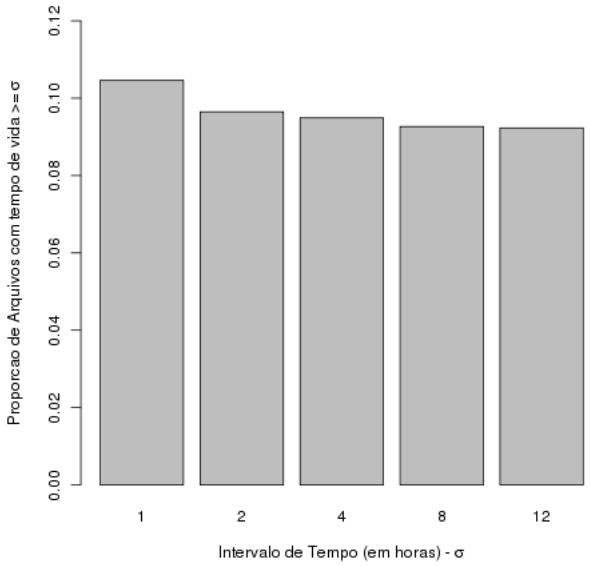

Figura 9.1: Proporção de arquivos com tempo de vida maior que um determinado intervalo de tempo expresso em horas.

Como mostra a Tabela 9.2 que reúne a informação ilustrada na Figura 9.1 para todos os rastros de utilização, a característica de vida curta dos arquivos é mantida tal como no rastro mais longo – ressaltado na tabela. Vê-se que para aqueles rastros com duração maior que um mês, a proporção de arquivos com tempo de vida maior que uma hora varia entre 9% e 18%. Estes resultados sobre a curta duração do tempo de vida típico estão em acordo com um trabalho prévio de caracterização do uso de sistemas de arquivo [20].

Por consequência, espera-se que poucos arquivos sejam afetados pelo fenômeno da mobilidade, uma vez que o período que um usuário permanece em uma máquina provavelmente será maior do que o tempo de vida da maioria dos arquivos acessados.

9.2 Desigualdade na Popularidade dos Arquivos

Na seção anterior, vimos que grande parte dos arquivos contidos nos rastros têm um tempo de vida bastante curto. Por ter um tempo de vida tão curto, estes arquivos tendem a ser referenciados (criados e a acessados) durante uma única sessão de uso, deste modo não são afetados pelas migrações do usuário e não desviam a localidade de acesso.

Tabela 9.1: Proporção de arquivos com tempo de vida maior que um determinado intervalo de tempo expresso em horas para o conjunto de rastros

Duração do Rastro	1	2	3	4	8	12
33 dias	0,18	0,17	0,17	0,17	0,17	0,16
9 meses	0,18	0,18	0,17	0,17	0,17	0,17
4 meses	0,15	0,12	0,12	0,12	0,12	0,10
13 **meses**	**0,10**	**0,09**	**0,09**	**0,09**	**0,09**	**0,08**
8 dias	0,29	0,26	0,25	0,20	0,20	0,19
3 meses	0,15	0,15	0,14	0,13	0,13	0,13
4 meses	0,18	0,18	0,17	0,17	0,16	0,15
3 meses	0,09	0,07	0,07	0,07	0,07	0,06
7 meses	0,13	0,12	0,12	0,12	0,12	0,11
9 meses	0,10	0,10	0,09	0,08	0,08	0,07
7 meses	0,16	0,16	0,14	0,14	0,14	0,14
8 meses	0,12	0,11	0,10	0,10	0,09	0,09

Por outro lado, existe um número reduzido de arquivos que ao ter um tempo de vida suficientemente grande para serem acessados em mais de uma sessão de uso, tornam-se responsáveis por todo o desvio na localidade de acesso. A Figura 9.2, ilustra o *ranking* do arquivos que possuem registros no rastro simulado no Capítulo 4 quanto ao número de operações realizadas sobre cada arquivo. Quanto maior o número de operações sobre determinado arquivo maior sua popularidade. Vê-se que há poucos arquivos com muitas operações, enquanto o número de arquivos com poucas operações é elevado.

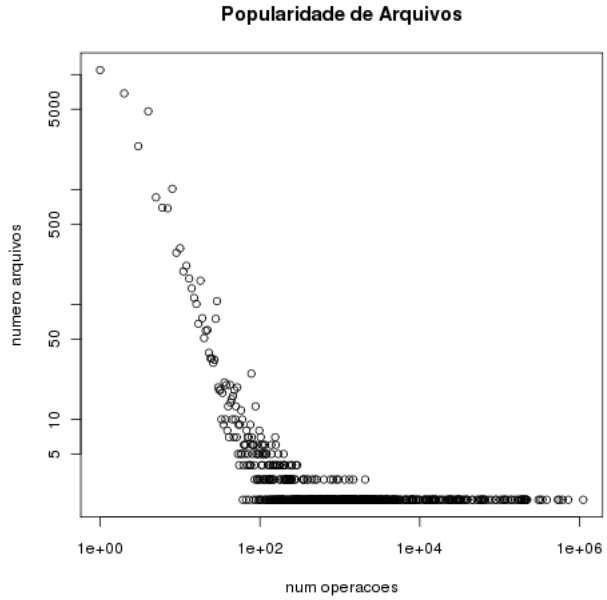

Figura 9.2: Popularidade dos arquivos simulados no rastro em termos do número de operações realizadas sobre estes.

A Tabela 9.2 sumariza a desigualdade, mensurada pelo valor de seu *skewness*, da distruibuição do número de operações sobre os arquivos para todos os rastros. Vê-se que a maioria dos rastros com duração maior que um mês apresentam um grau de desigualdade maior do que rastro simulado no Capítulo 4. Vale lembrar que esta é uma medida do grande número de arquivos, que dominam a distribuição, sobre os quais foram relizadas poucas operações. Um colorário a isto, e que reforça os dados apresentados na Tabela 9.2 é a existência de um conjunto pequeno de arquivos que têm um grande número de acessos, e portanto, quando devidamente tratados podem ter grande efeito sobre o resultado das políticas estudadas.

Tabela 9.2: Valores obtidos para o *Skewness* da distruibuição do número de operações realizadas sobre os arquivos nos rastros de simulação

Duração do Rastro	*Skewness*
33 dias	12,22
9 meses	20,29
4 meses	59,80
13 meses	**36.64**
8 dias	9,75
3 meses	121,68
4 meses	128,86
3 meses	38,90
7 meses	78,54
9 meses	31,49
7 meses	128,15
8 meses	147,57

Bibliografia

[1] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. 2002.

[2] Peter A. Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 562–570, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[3] SR Arora and A. Gallo. Optimization of static loading and sizing of multilevel memory systems. *Journal of the ACM (JACM)*, 20(2):307–319, 1973.

[4] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, and L. Yerushalmi. Towards an object store. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings*, pages 165–176, 2003.

[5] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. *Document on Hadoop Wiki*, 2008.

[6] RG Casey. Allocation of copies of a file in an information network. In *Proceedings of the November 16-18, 1971, fall joint computer conference*, pages 617–625. ACM, 1971.

[7] P.P.S. Chen. Optimal file allocation in multi-level storage systems. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pages 277–282. ACM, 1973.

[8] WW Chu. Optimal file allocation in a multiple computer system. *IEEE Transactions on Computers*, 100(18):885–889, 1969.

[9] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. *SIGMETRICS Perform. Eval. Rev.*, 27(1):59–70, 1999.

[10] J.R. Douceur and W.J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 59–70. ACM New York, NY, USA, 1999.

[11] J.R. Douceur and R.P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings*, pages 311–319, 2001.

[12] L.W. Dowdy and D.V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys (CSUR)*, 14(2):313, 1982.

[13] K.P. Eswaran. Placement of records in a file and file allocation in a computer network. *Information Processing*, 74:304–307, 1974.

[14] M.L. Fisher and D.S. Hochbaum. Database location in computer networks. *Journal of the ACM (JACM)*, 27(4):718–735, 1980.

[15] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[16] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[17] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, February 1988.

[18] PH Hughes and G. Moe. A structural approach to computer performance analysis. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pages 109–120. ACM, 1973.

[19] E.K. Lee and C.A. Thekkath. Petal: Distributed Virtual Disks. *Computer Architecture News*, 24:84–92, 1996.

[20] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. Measurement and analysis of large-scale network file system workloads. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 213–226, Berkeley, CA, USA, 2008. USENIX Association.

[21] K.D. Levin. Organizing Distributed Data Bases in Computer Networks., 1974.

[22] H.L. Morgan and K.D. Levin. Optimal program and data locations in computer networks. *Communications of the ACM*, 20(5):322, 1977.

[23] M.N. Nelson, B.B. Welch, and J.K. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):134–154, 1988.

[24] D.A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Record*, 17(3):109–116, 1988.

[25] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3 design and implementation. In *Proceedings of the Summer USENIX Conference*, pages 137–152, 1994.

[26] CV Ramamoorthy and KM Chandy. Optimization of memory hierarchies in multiprogrammed systems. *Journal of the ACM (JACM)*, 17(3):426–445, 1970.

[27] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. A comparison of file system workloads. In *In Proceedings of the 2000 USENIX Annual Technical Conference*, pages 41–54, 2000.

[28] M. Russinovich. Inside Win2K NTFS. *Windows & .NET Magazine, November–December*, 2000.

[29] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. *ACM SIGPLAN Notices*, 39(11):48–58, 2004.

[30] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: a highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[31] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS Version 4 Protocol. Technical report, Technical Report RFC 3010, Network Working Group, December, 2000.

[32] Alexandro S. Soares, Thiago Emmanuel Pereira, Jonhnny Silva, and Francisco Vilar Brasileiro. Um modelo de armazenamento de metadatados tolerante a falhas para o DDGfs (in portuguese) . In *WSCAD-SSC 2009: Proceedings of the 10th Computational Systems Symposium*, October 2009.

[33] Swarup. *Concurrency Control and Recovery in Database Systems*. 1987.

[34] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the Annual Technical Conference on USENIX 1996 Annual Technical Conference table of contents*, pages 1–1. USENIX Association Berkeley, CA, USA, 1996.

[35] C.A. Thekkath, T. Mann, and E.K. Lee. Frangipani: a scalable distributed file system. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 224–237. ACM New York, NY, USA, 1997.

[36] S. Tweedie. EXT3, Journaling File System. In *Ottawa Linux Symposium*, pages 24–29, 2000.

[37] B.W. Wah. File placement on distributed computer systems. *IEEE Computer*, 17(1):23–32, 1984.

[38] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System.

[39] S.A. Weil, S.A. Brandt, E.L. Miller, and C. Maltzahn. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122. ACM, 2006.

[40] E.R. Zayas and A.R. Reserved. AFS-3 Programmer's Reference: Architectural Overview. *Transarc Corporation, Pittsburgh, PA*, 1, 1991.