UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UMA CACHE DE ARQUIVOS REMOTOS EM MODO USUÁRIO

TRABALHO DE GRADUAÇÃO

João Vicente Ferreira Lima

Santa Maria, RS, Brasil

2007

DESENVOLVIMENTO DE UMA CACHE DE ARQUIVOS REMOTOS EM MODO USUÁRIO

por

João Vicente Ferreira Lima

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Dr. Benhur de Oliveira Stein

Trabalho de Graduação Nº 227
Santa Maria, RS, Brasil

2007

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UMA CACHE DE ARQUIVOS REMOTOS EM MODO USUÁRIO

elaborado por
João Vicente Ferreira Lima

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Benhur de Oliveira Stein
(Presidente/Orientador)

Prof. Antonio Marcos de Oliveira Candia (UFSM)

Prof. MSc. Diego Luíz Kreutz (UNIPAMPA)

Santa Maria, 2 de março de 2007.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a minha família que fez o possível e o impossível para que eu chegasse até aqui. Outra pessoa que eu gostaria de agradecer em especial é minha namorada Liana que me apoiou constantamente e que compreendeu minhas dificuldades quando eu estava "grudado" em frente ao computador.

Gostaria de agrader a meu orientador Benhur Stein que esteve sempre disposto a ajudar, esclarecer dúvidas e solucionar os problemas encontrados no desenvolvimento deste trabalho e na administração da rede.

Quero agradecer aos professores deste curso pelos ensinamentos e às vezes "lições de vida" dados. Em especial, aos professores Antonio Candia e Diego Kreutz que participaram da banca avaliadora e contribuiram muito para o desenvolvimento deste trabalho.

Aos meus colegas de graduação e amigos, que contribuiram nas "risadas", na cerveja e na amizade durante minha graduação e elaboração deste trabalho. Em especial cito Giovani (maaaaquina), Eduardo, Leandro Sacchet, Tiago Nonoai, Tiago Scheid e Rafael Tesser.

E também agradeço a meus colegas e ex-colegas da administração da rede no NCC, além de amigos que freqüentam este lugar como o Elias Junior, Valdir Junior e muitos outros.

Um abraço a todos e meus sinceros agradecimentos.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UMA CACHE DE ARQUIVOS REMOTOS EM MODO USUÁRIO

Autor: João Vicente Ferreira Lima
Orientador: Prof. Dr. Benhur de Oliveira Stein
Local e data da defesa: Santa Maria, 2 de março de 2007.

O presente trabalho situa-se na área de sistemas operacionais. Quando se tem um conjunto de computadores semelhantes com sistemas instalados independentemente, como em um laboratório, geralmente se tem problemas para realizar a atualização dos arquivos locais, a fim de manter esses computadores com versões coerentes de seus sistemas operacionais e seus diversos programas instalados. Este trabalho apresenta uma solução para este problema, através da centralização da instalação e atualização de sistemas e da implementação de um mecanismo de *cache* de sistema de arquivos para disponibilizar essa instalação comum sem a necessidade de reinstalar cada equipamento. No desenrolar dos capítulos são apresentados os trabalhos relacionados, as ferramentas utilizadas para o desenvolvimento do trabalho, a implementação realizada e os principais resultados obtidos.

Palavras-chave: Sistemas operacionais, sistemas de arquivos, redes de computadores.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FUSE	File System in Userspace
GB	Gigabytes
MB	Megabytes
NFS	Network File System
NLM	Network Lock Manager
RCFS	Remote Cache File System
VFS	Virtual File System
KB	Kilobytes

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores deixaram de ser privilégio de cursos de computação ou centros tecnológicos para fazer parte do cotidiano. Cada vez mais o número de usuários e suas necessidades de acesso crescem a medida que as demais áreas de pesquisa começam a apresentar conteúdos digitais publicados na *Internet*. Com este crescimento, as redes necessitam de um controle escalável e eficiente de recursos, como os de *software* por exemplo.

Esse controle é uma das tarefas à qual os administradores devem prestar uma atenção especial, principalmente em ambientes com usuários avançados. Pode-se citar como exemplo os laboratórios de computação onde o controle de programas básicos ao desenvolvimento de aplicativos é fundamental. Este controle ocorre tanto por atualizações, instalação de novas ferramentas ou alterações em suas configurações.

Para manter o controle dos sistemas de diversos computadores, há basicamente duas soluções possíveis:

- instalar as novas versões de sistema ou programas em todos os equipamentos cada vez que uma atualização se faz necessária;
- instalar as novas versões em um servidor centralizado, que será acessado remotamente pelos demais equipamentos.

A primeira solução representa um grande trabalho manual, enquanto a segunda pode representar uma sobrecarga, tanto ao servidor quanto à rede de interligação. Uma solução intermediária é criar um mecanismo capaz de realizar cópias de arquivos remotos para o disco local e atender as futuras requisições localmente.

Baseado nesta idéia, o objetivo deste trabalho é apresentar um sistema de arquivos virtual responsável por esse mecanismo de armazenamento de arquivos remotos em disco.

Sua implementação consiste em um aplicativo em modo usuário a fim de facilitar sua depuração e evitar a modificações dos sistemas clientes. Este programa é chamado de *Remote Cache File System* (RCFS) que torna as atualizações livres da intervenção manual em cada equipamento. Após uma primeira leitura, os acessos subseqüentes serão realizados localmente com verificações de consistência à medida que for necessário. As operações permitidas em seus arquivos serão somente de acesso e leitura, sem suporte a escrita e modificação.

O restante do capítulo apresenta uma breve descrição do que será abordado neste trabalho. Inicialmente, são descritos alguns trabalhos relacionados, seguido da apresentação das ferramentas utilizadas no desenvolvimento da solução proposta. Por fim, é abordada a implementação realizada e os principais resultados obtidos.

Trabalhos relacionados. O problema de se ter arquivos compartilhados com um bom desempenho não é recente, e diversos trabalhos já foram realizados para solucionar este problema. Entre eles temos o NFS (*Network File System*) (SANDBERG et al., 1988; CALLAGHAN; PAWLOWSKI; STAUBACH, 1995; PAWLOWSKI et al., 2000) e o FS-Cache (HOWELLS, 2006). Quanto ao NFS, é um sistema tipicamente distribuído com mecanismos de escrita e protocolos definidos para comunicação entre servidores e clientes. O FS-Cache trata-se de uma camada extra no núcleo Linux que faz o armazenamento de arquivos recentemente acessados em disco local, além de proporcionar o gerenciamento das cópias locais.

Ferramentas utilizadas. Neste trabalho é utilizado o FUSE (*File System in Userspace*), que é um biblioteca que proporciona a programas em modo usuário exportar um sistema de arquivos virtual ao núcleo Linux. Sua característica principal é permitir montagens não-privilegiadas de forma simples e segura. Através dessa ferramenta as políticas de acesso e consistência do trabalho são definidas.

Estudo de caso. O estudo de caso deste trabalho está centrado no Núcleo de Ciência da Computação (NCC) da Universidade Federal de Santa Maria (UFSM). Num primeiro momento, as duas soluções usadas atualmente no NCC são descritas. O primeiro método é através da sincronização de dois sistemas com o programa *rsync*. No segundo, um *CDROM* realiza a cópia dos arquivos remotos em disco local. Por fim, será demonstrado como a proposta deste trabalho atuará nos computadores do NCC.

Implementação proposta. O capítulo 3 descreve a solução proposta e discrimina os

objetivos do trabalho. Após, é descrita a implementação realizada, com os mecanismos de armazenamento e validação em *cache*.

Resultados obtidos. No capítulo 4 serão demostrados os principais resultados obtidos da solução proposta, tanto sob aspecto do cliente quanto do servidor.

Conclusão e trabalhos futuros. Por fim, serão mostrados os principais ganhos e algumas inferências sobre trabalho futuros que podem melhorar o funcionamento da implementação realizada.

2 CONTEXTUALIZAÇÃO

Este capítulo apresenta alguns conceitos básicos sobre sistemas de arquivos, trabalhos relacionados e ferramentas aplicadas no desenvolvimento da solução proposta, que será discutida no capítulo 3. Na seção 2.1 é introduzido o funcionamento dos sistemas de arquivos em Linux. Em seguida, são apresentadas as ferramentas relacionadas com os propósitos deste trabalho. Por fim, a seção 2.3 descreve o FUSE e algumas de suas características importantes para o desenvolvimento deste trabalho.

2.1 Sistema de arquivos

Todos os aplicativos necessitam armazenar e recuperar informações, mesmo após o término de sua execução. Como solução existem os arquivos, unidades que residem em discos ou mídias externas capazes de armazenar e manter os dados persistentes (TANENBAUM; WOODHULL, 1997). O responsável pelo gerenciamento destas unidades é o sistema operacional, e a parte de estruturação é realizada pelo sistema de arquivos.

No Linux, essa estrutura é baseada nas melhores decisões de desenvolvimento de outros sistemas operacionais que seguem os conceitos da primeira versão do UNIX (RITCHIE; THOMPSON, 1974), criado para computadores PDP-11/40 e 11/45. A seguir, são apresentados os conceitos básicos de sistemas de arquivos em Linux, seguido de uma descrição sobre a camada que suporta e abstrai diferentes implementações, o sistema de arquivos virtual.

2.1.1 Conceitos básicos

Os sistemas de arquivos em Linux são compostos, em sua maioria, por três componentes básicos: arquivos comuns, arquivos especiais e diretórios.

Arquivos comuns são locais onde o usuário armazena informações como texto ou da-

dos no formato binário. O sistema não espera qualquer estrutura destes objetos, apenas uma seqüência de *bytes*. Esta característica permite uma política aberta para os programas atribuírem uma estrutura sem qualquer interferência ou limitação por parte do sistema (RITCHIE; THOMPSON, 1974).

Os arquivos especiais constituem uma das inovações que foram propostas na versão original do UNIX (RITCHIE; THOMPSON, 1974). Cada dispositivo do sistema é associado a pelo menos um arquivo, na maioria das vezes localizado do diretório */dev*. A leitura e escrita destes é semelhante a de arquivos comuns, mas os dados têm como origem e destino os dispositivos correspondentes. O controle de acesso aos dispositivos é facilitado pelas permissões do arquivo que o representa.

Um diretório é o mapeamento entre o nome dos arquivos e os arquivos propriamente ditos. A estrutura de diretórios cria uma árvore, como na figura 2.1, onde cada um é ligado a seu único superior. Além dessa ligação, representada por um "..", existe uma entrada especial chamada de "." que representa o diretório atual.

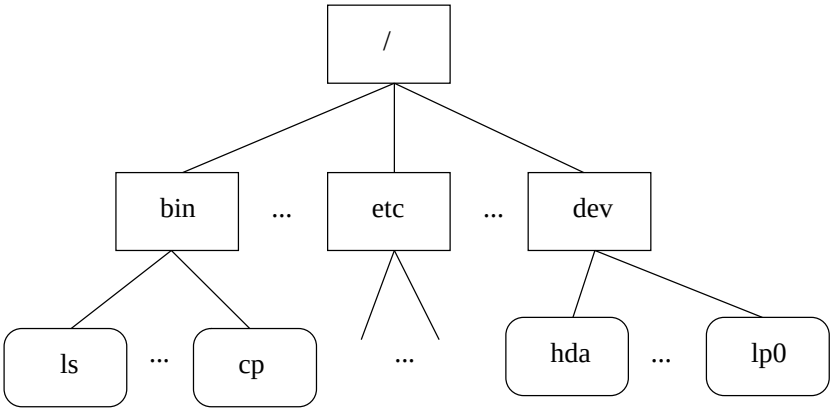

Figura 2.1: Árvore de diretórios do Linux.

2.1.2 Sistema de arquivos virtual

O *Virtual File System* (VFS), também conhecido como *Virtual Filesystem Switch*, é uma camada no núcleo Linux que proporciona a representação dos diferentes sistemas de arquivos em um modelo comum. Esse modelo facilita a criação e manutenção de um sistema de arquivos em núcleo ao separar a parte dependente e independente de implementação (KLEIMAN, 1986). Além disso, suas estruturas abstraem a representação de objetos fundamentais para a representação de um sistema de arquivos, o que permite uma visão comum entre os vários sistemas de arquivos suportados (BOVET; CESATI, 2002).

Como exemplos de objetos, ou estruturas, que o VFS abstrai tem-se o *superblock* e o *inode*. O primeiro descreve as informações sobre um sistema de arquivos montado, que no caso dos baseados em disco representará o seu bloco descritor. O segundo contém informações gerais a respeito de um arquivo específico, que pode ser um diretório ou arquivo comum.

A figura 2.2 ilustra o papel do VFS na cópia de um arquivo entre diferentes sistemas de arquivos. O programa *cp* primeiramente comunica-se com o VFS (1) para a requisição da operação. Em seguida, o VFS faz um reconhecimento para saber a qual sistema de arquivos pertence a origem e o destino, a fim de chamar as funções de leitura (2) e escrita (3) correspondentes. Em ambos os procedimentos, o conhecimento sobre o funcionamento interno de cada sistema de arquivos é desconhecido tanto do VFS quanto do aplicativo *cp*.

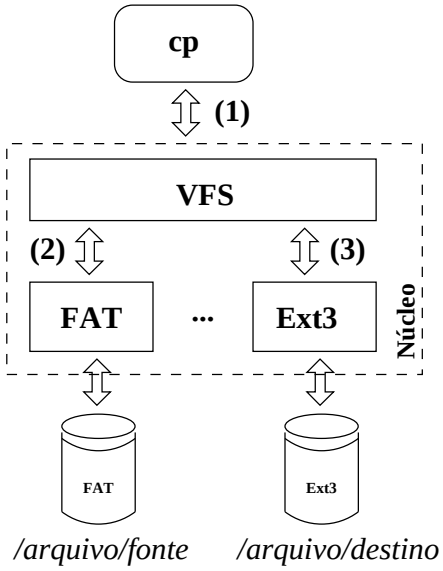

Figura 2.2: Papel do VFS entre diferentes sistemas de arquivos.

2.2 Trabalhos relacionados

2.2.1 NFS

O *Network File System* (NFS) permite que um sistema compartilhe seus arquivos e diretórios com outros através de uma rede de forma que acessos a estes arquivos pareçam locais ao invés de remotos. Com o uso do programa *mount*, a árvore de diretórios remota é anexada em um diretório local. A figura 2.3 demonstra, de forma simplificada, o acesso de um arquivo remoto por NFS.

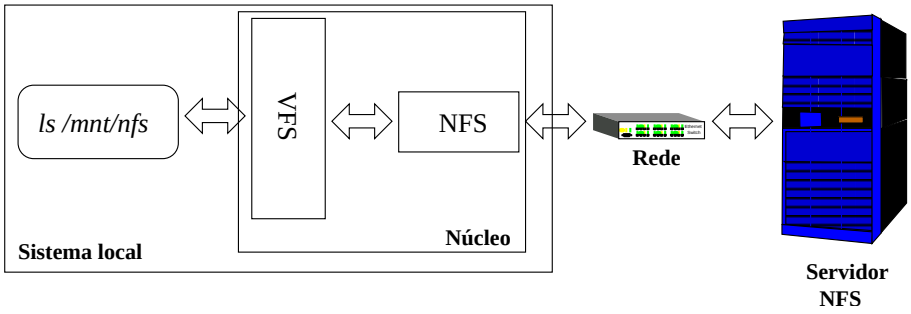

Figura 2.3: Comunicação entre o NFS e o servidor remoto.

Versões existentes

A primeira versão existiu somente na *Sun Microsystems* e nunca foi publicada (PAWLOWSKI et al., 1994). Em 1984, a versão 2 foi implementada no SunOS 2.0 (SANDBERG et al., 1988) e oficializa as características básicas do protocolo, que prevalecem até hoje.

A segunda versão (SUN MICROSYSTEMS, 1989) busca a independência de arquitetura, sistema operacional e camada de transporte. Com um protocolo *stateless*, sem armazenar qualquer informação sobre acessos anteriores no servidor, o NFS proporciona simplicidade na recuperação de falhas. Nestes casos, o cliente é responsável por refazer a operação até que o servidor envie um retorno. Mecanismos de *cache* e *locking* não são especificados nesta versão.

As limitações da versão 2 levaram a criação do NFS versão 3 (CALLAGHAN; PAWLOWSKI; STAUBACH, 1995), com melhorias em desempenho, exclusão mútua e *cache* em clientes. Em desempenho, a introdução de procedimentos assíncronos e arquivos maiores, representados por 64 *bits*, acelerou a vasão de dados (PAWLOWSKI et al., 1994). A exclusão mútua faz parte de outro protocolo, especificado juntamente com o padrão NFS, chamado de *Network Lock Manager* (NLM). Quanto a *cache*, esta versão é a primeira a apresentar procedimentos que facilitam a implementação nos clientes, tanto em memória quanto em disco, mas o protocolo não especifica mecanismos ou padrões.

A terceira versão, todavia, mantinha carências quanto a segurança e *cache* de arquivos (PAWLOWSKI et al., 1994) . A versão 4 (SHEPLER et al., 2003), criada pela *Internet Engineering Task Force* (IETF), propôs melhorias nestes dois aspectos e incorpora mecanismos de exclusão mútua, o que leva o protocolo a ser *statefull*. Entretanto, o NFS versão 4 continua sem um mecanismo padrão de *cache*.

Conclusões sobre o NFS

Com a versão 4, as possibilidades de *cache* são significativas mas dependentes da implementação em questão. O problema apresentado neste trabalho trata de mecanismos de *cache* que não estão presentes no protocolo, como o armazenamento em disco dos acessos mais recentes. Dessa forma, o NFS não atende as necessidades anteriormente discutidas.

Particularmente, a implementação do sistema NFS no núcleo Linux versão 2.6.18 (INC., 2007) armazena algumas leituras e escritas em memória como forma de ganho de desempenho, mas esta *cache* poderá ser limitada à memória do computador, caso seja considerável a demanda por arquivos grandes. Para sistemas inteiramente remotos, como mencionado no capítulo 1, a memória será limitante se os arquivos acessados ultrapassarem a capacidade do cliente em questão.

2.2.2 FS-Cache

O FS-Cache (HOWELLS, 2006) é uma ferramenta que consiste em uma camada no núcleo Linux para *cache* de arquivos remotos ou mídias lentas em disco. Ele permite que, após uma leitura inicial, as leituras subseqüentes aconteçam localmente sem a necessidade de recorrer novamente a origem dos arquivos.

Estrutura e funcionamento

O FS-Cache atua no núcleo Linux como uma camada acima do VFS, cuja finalidade é armazenar os acessos mais recentes localmente. Seu funcionamento consiste em um aplicativo de configuração, o *cachefilesd*, e em alterações no núcleo responsáveis pelo controle da *cache*.

O *cachefilesd* permite a especificação do local de *cache*, onde serão armazenados os arquivos, e a configuração de alguns parâmetros da parte em núcleo. Estes parâmetros se resumem ao controle de uso mínimo e máximo de espaço em disco.

O núcleo realiza as operações de controle e armazenamento dos arquivos em *cache*, como a verificação de consistência entre a fonte e o armazenamento local. Sua estrutura está organizada em três partes. A primeira é o próprio FS-Cache que abstrai do sistema de arquivos alvo a implementação do armazenamento físico, ao fazer a seleção do módulo de controle. Em seguida o *CacheFS* e o *CacheFiles* tratam da leitura e escrita em um

dispositivo de bloco, que pode ser um disco, ou em um diretório previamente montado respectivamente.

A figura 2.4 demonstra os pontos de interação entre as partes do FS-Cache e do NFS. O programa em modo usuário, primeiramente, faz uma requisição de leitura ao sistema (1), que pode ser atendida em memória ou não. Caso não for possível será feito um pedido ao NFS que consulta o FS-Cache sobre a existência do arquivo em *cache* localmente ou não (3). O FS-Cache pode requisitar este arquivo tanto do *CacheFS* (4) quanto do *CacheFiles* (5). No caso do arquivo não existir em *cache*, o NFS fará um acesso remoto (2).

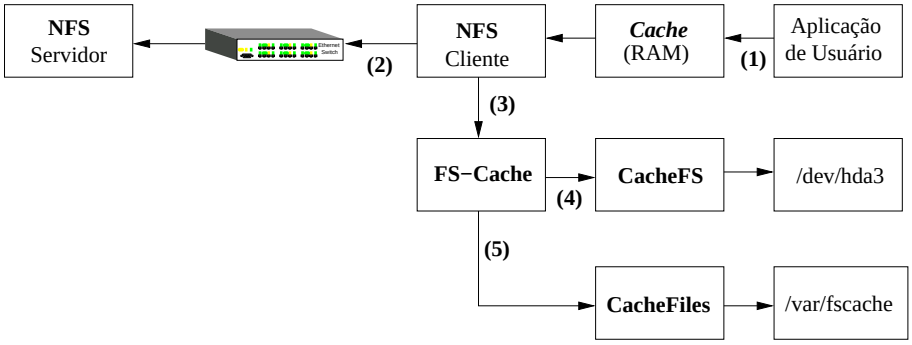

Figura 2.4: Interações do FS-Cache com o NFS e um aplicativo de usuário.

Conclusões sobre o FS-Cache

O FS-Cache apresenta facilidades como permitir várias *caches* independentes e gerenciáveis, atuar diretamente no núcleo Linux com algum ganho em desempenho e ter suporte à modificação de seus arquivos em *cache*. Todavia, as alterações no sistema Linux são extensas, o que não é bem aceito por seus desenvolvedores.

Do ponto de vista dos sistemas de arquivos suportados, o FS-Cache não permite um mecanismo uniforme para *cache* de um sistema de arquivos porque deve-se modificar a implementação em questão, a fim de ter suporte a este sistema. Os programas em modo usuário também devem ser alterados devido a parâmetros extras necessários para definir o interesse em fazer *cache* de seus arquivos.

O exemplo mais claro dessas alterações mencionadas é o NFS com suporte ao FS-Cache. A implementação em núcleo, assim como os aplicativos de sistema, devem ser alterados. O primeiro deve conter as chamadas ao sistema de *cache* necessárias, enquanto o segundo recebe parâmetros extras para escolher entre fazer ou não *cache* de seus arquivos.

Na prática, o sistema com suporte ao FS-Cache é de difícil manutenção devido a inibir qualquer atualização que a distribuição GNU/Linux hospedeira oferecer. Pode-se citar o exemplo do `nfs-utils` (PROJECT, 2007) responsável por conter os aplicativos em modo usuário que manipulam o NFS. Caso o gerenciador de pacotes da distribuição GNU/Linux em questão concluir que o `nfs-utils` necessita ser atualizado, os binários modificados para o uso do FS-Cache serão perdidos. Nesta situação, a solução trivial seria deixar de atualizar os aplicativos do sistema em questão. Outra forma de contornar este problema é realizar as atualização e instalar novamente os binários modificados.

Atualmente, sua última versão disponível foi lançada na versão 2.6.19-rc5-mm2, núcleo Linux mantido por Andrew Morton e que objetiva o teste de novas funcionalidades (INC., 2007). A partir desta versão o FS-Cache deixou de ser mantido.

2.3 Ferramentas utilizadas

Nesta seção o FUSE (SZEREDI, 2007), utilizado para solução proposta no capítulo 3, será apresentado. Através dessa ferramenta, a especificação de um sistema de arquivos virtual que reside em modo usuário pode ser feita.

A programação de aplicativos com o FUSE consiste em implementar funções que descreverão os dados e metadados do sistema de arquivos. Primeiro, deve-se atribuir essas funções às estruturas básicas do FUSE e passar o controle de execução aos procedimentos da biblioteca *libfuse*.

As partes que compõem o FUSE são o programa *fusermount*, a biblioteca *libfuse* e o módulo do núcleo Linux *fuse*. A seguir, a descrição em detalhes de seus componentes é apresentada.

2.3.1 O utilitário *fusermount*

Ele permite que a biblioteca *libfuse* realize a abertura do dispositivo */dev/fuse*, assim como a montagem do sistema de arquivos. Além disso, o aplicativo possibilita a desmontagem dos diretórios. A justificativa para usar-se o *fusermount* para a montagem é a independência entre a biblioteca, o módulo em núcleo e a montagem do sistema de arquivos.

2.3.2 O módulo *fuse*

Este módulo é responsável por criar o dispositivo */dev/fuse* e realizar a comunicação entre o VFS, discutido na seção 2.1.2, e os aplicativos em modo usuário através do dispositivo. Em cada requisição gerada pelo VFS, o módulo cria uma estrutura e a envia para o dispositivo, onde o aplicativo em modo usuário aguarda o seu recebimento.

A parte em núcleo possui uma opção de *cache* das leituras mais recentes em memória. Esse mecanismo consiste em manter algumas páginas de memória após os término das requisições. No caso da opção estar desabilitada, ao final da requisição as páginas em memória do arquivo são marcadas como inválidas. O gerenciamento desta *cache* não é feita pelo FUSE mas é mantido pelo VFS.

2.3.3 A biblioteca *libfuse*

A *libfuse* define os procedimentos básicos que manipulam as estruturas do FUSE e permitem a comunicação do aplicativo com o núcleo Linux. Através dela é possível, em modo usuário, definir as funções que serão parte do sistema de arquivos virtual e chamar o procedimento principal, o `fuse_main`.

As tarefas necessárias para que o aplicativo, em modo usuário, defina um sistema de arquivos são simplificadas e organizadas pela biblioteca. Primeiramente, o programa implementa e defini os procedimentos que descreverão os dados e metadados desse sistema. Em seguida, a estrutura `fuse_operations` é preenchida com as funções previamente definidas e o procedimento `fuse_main` controla o restante da execução.

A figura 2.5 ilustra o funcionamento do FUSE a partir de uma requisição de um programa em modo usuário. Num primeiro momento, esse programa faz um pedido de informações sobre um diretório (1) que é captado pelo VFS. Por sua vez, o VFS identifica que o caminho especificado pertence ao FUSE e o notifica ao mesmo tempo que envia as informações necessárias (2). No FUSE, estas informações são tratadas e enviadas para o aplicativo em modo usuário (3), através da *libfuse*, que atenderá a requisição. Por fim, a resposta do aplicativo (3) é enviada pelo caminho inverso, como ilustram as setas da figura 2.5.

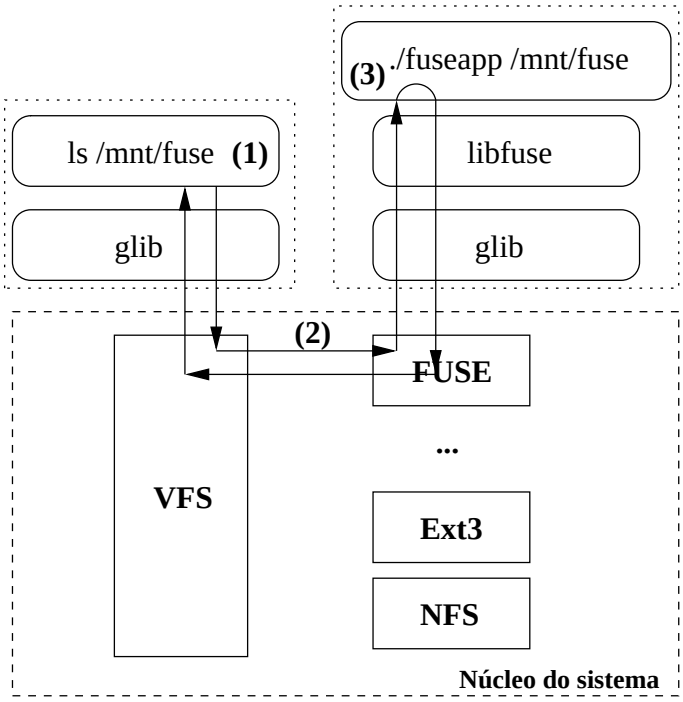

Figura 2.5: Caminho de dados do FUSE.

2.3.4 Vantagens e desvantagens do FUSE

Como mencionado anteriormente, os dados e metadados do sistema de arquivos são fornecidos pelo aplicativo em modo usuário. Essa facilidade é importante para o propósito deste trabalho devido a simplicidade na manutenção e depuração de programas em modo usuário.

A atualização de um sistema leva a modificações de interfaces e bibliotecas. Todavia, é mais conveniente utilizar ferramentas que tenham menos efeitos colaterais quando este fato acontecer. Para tanto, o FUSE permite uma interface independente de sua biblioteca e do núcleo do sistema operacional, o que inibe a necessidade de alterações nos códigos-fonte que a utilizam.

A depuração em fase de desenvolvimento também é facilitada, pois pode-se recorrer a depuradores em modo usuário para detectar falhas de programação. Sistemas de arquivos em núcleo exigem métodos de depuração complexos e dependentes de arquitetura. Em caso de falhas, o método em modo usuário depende apenas do término do processo, enquanto em núcleo leva ao término do sistema inteiro.

O problema maior do aplicativo em modo usuário é a troca de contexto entre ele e o núcleo do sistema. Essa troca é ocasionada por requisições geradas por outros programas que passam pelo núcleo e após ao sistema de arquivos descrito. Outro problema que o

FUSE apresenta é a falta de exclusão mútua para os sistemas de arquivos que o utilizam.

2.4 Conclusão

Após os conceitos básicos sobre sistemas de arquivos e trabalhos relacionados, conclui-se que as ferramentas apresentadas solucionam parcialmente o problema apresentado. O NFS não especifica mecanismos de *cache* em disco no seu protocolo, ao passo que o FS-Cache soluciona o problema de uma forma inadequada, ao propor alterações no sistema. No próximo capítulo, serão discutidos estes problemas e uma proposta para solucioná-los.

3 SOLUÇÃO PROPOSTA

Este capítulo apresenta a proposta para solucionar os problemas de atualização dos sistemas mencionados anteriormente, chamada de RCFS. Primeiramente serão apresentados os objetivos do projeto, seguido dos detalhes de implementação. Por fim, um estudo de caso no NCC da UFSM será demonstrado.

3.1 Objetivos do trabalho

Solução intermediária. Como foi descrito na introdução, a atualização de um conjunto de computadores independentes pode ser feita basicamente de duas maneiras. A primeira é instalar em todos os equipamentos uma nova versão de sistema ou programas quando necessário, e a outra é manter um servidor centralizado com estas novas versões acessíveis a cada cliente. Todavia, as duas soluções apresentam problemas.

Na primeira, tem-se o trabalho de instalar cada sistema manualmente à medida que as atualizações estão disponíveis. Essa tarefa pode representar um grande trabalho manual e demanda tempo, conforme o número de equipamentos que necessitam de manutenção. Na segunda, a proposta é centralizar uma instalação acessível remotamente aos clientes da rede, geralmente associada ao uso do NFS para o compartilhamento. Estes acessos remotos podem representar uma sobrecarga ao servidor e a rede de interligação em cenários onde o número tanto de acessos a arquivos quanto de participantes do compartilhamento é grande.

Este trabalho busca uma solução intermediária entre as duas soluções comentadas. A proposta é criar um sistema de arquivos capaz de acessar arquivos remotos e armazenar as leituras mais recentes em disco. Dessa forma, pretende-se eliminar o trabalho manual em cada atualização e reduzir a sobrecarga dos acessos inteiramente remotos.

A proposta apresentada é semelhante ao FS-Cache, mas com a diferença de que o

sistema de arquivos proposto é independente de alterações tanto do núcleo Linux quanto dos programas de sistema e aplicativos em geral. Os pré-requisitos são as ferramentas utilizadas, descritas no capítulo 2.

Transparência. Uma característica importante, objetivo deste trabalho, é a transparência do mecanismo proposto tanto para usuários, administradores e programas. Assim o RCFS deve ser inserido de forma que sua manutenção não interfira nos programas de sistema.

O RCFS exige uma manutenção mínima por parte dos administradores na tarefa de atualização dos sistemas clientes. O requisito necessário é a especificação do diretório montado remotamente e o diretório a ser montado localmente. No momento que o sistema de arquivos verificar que os arquivos locais devem ser atualizados ele será o encarregado desta iniciativa.

3.2 Implementação proposta

Estrutura básica

A proposta do trabalho supõe a existência de um servidor que exporta alguns diretórios e um número de computadores como clientes. O servidor permanecerá inalterado, ao passo que os clientes serão hospedeiros do mecanismo de *cache*.

O servidor manterá um sistema de arquivos compartilhado entre seus clientes através do NFS, cujas características foram discutidas no capítulo 2. Estes arquivos terão origem em um de seus clientes, fonte da instalação GNU/Linux atualizada e padrão para as suas semelhantes. Em casos de modificações nestes arquivos, os administradores serão responsáveis pela substituição. Neste trabalho não será desenvolvido o mecanismo de atualização dos arquivos em servidor.

Nos clientes, temos o desenvolvimento central da solução. Eles terão de montar este sistema remoto e realizar as devidas atualizações dos arquivos. Após um primeiro acesso, que é responsável pela cópia do arquivo para o disco local, as próximas leituras passarão a ser atendidas localmente.

A proposta de implementação consiste em descrever um sistema de arquivos virtual, o RCFS, em modo usuário que realizará a tarefa de sincronização entre dois diretórios distintos, nesse caso o remoto e o local. A linguagem de programação C (KERNIGHAN; RITCHIE, 1988) será utilizada, juntamente com o FUSE, discutido no capítulo 2.

Funcionamento externo

A figura 3.1 mostra a forma como o RCFS interage com o núcleo do sistema através de uma requisição para um diretório montado com o FUSE. Cada operação inicia com a requisição de um programa (1) que o núcleo se encarrega de passar ao sistema de arquivos apropriado (2) que neste caso é o FUSE. Ele, por sua vez, notifica o RCFS que é responsável pela montagem deste diretório (3). No RCFS é decidido se o arquivo existe localmente (4) ou remotamente (5), que no caso de existir somente no segundo caso deverá ser copiado para o disco.

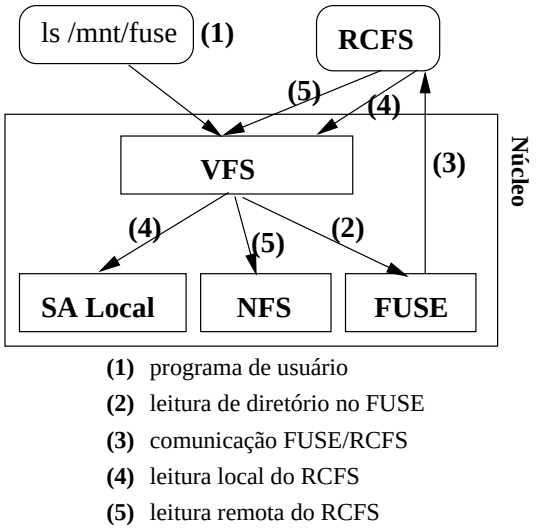

Figura 3.1: Caminho de dados da solução proposta.

Na descrição acima, o FUSE participa somente como um comunicador entre as requisições de sistema e os dados do sistema de arquivos descrito, sem realizar qualquer entrada e saída diretamente.

Funcionamento interno

O armazenamento em *cache* consiste em manter uma cópia idêntica de sua correspondente remota no disco local sem acrescentar atributos extras. Este mecanismo simplifica tanto a cópia para a *cache* quanto a consulta de existência do arquivo.

O algoritmo principal do RCFS está resumido na figura 3.2 onde se encontra as etapas iniciais, com a chegada da requisição, e o retorno de resultados. Nas fases demonstradas, acessos locais acontecem em (1), (5) e (6). As leituras remotas estão centradas em (2), (4), (5). As partes (3), (6) e (4) que tratam dos atributos usados para invalidar e diferenciar arquivos serão explicados a seguir.

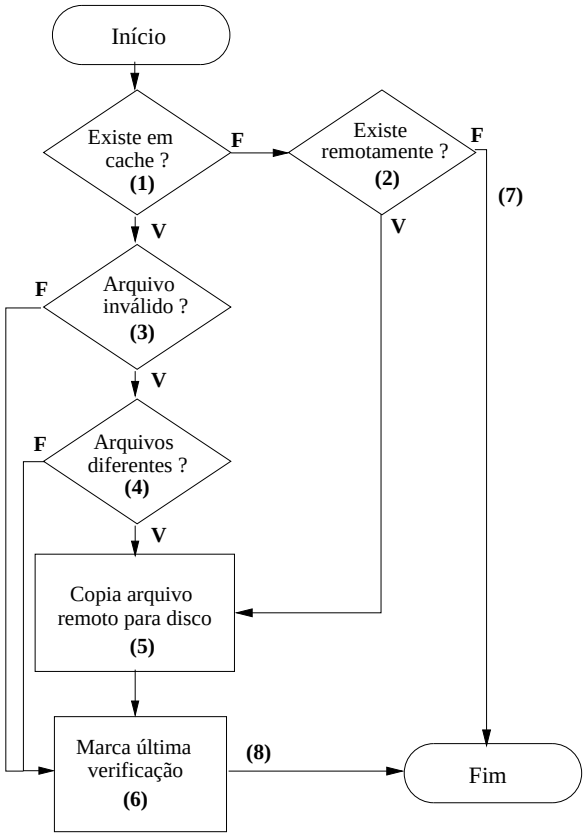

Figura 3.2: Fluxograma do funcionamento interno do RCFS.

Validação de arquivos

A validação de arquivos em *cache* será feita conforme o tempo da última verificação remota, armazenada na data de acesso de cada arquivo. A cada requisição é verificado se a diferença entre a data da última verificação e a data atual ultrapassa um limite. Esse limite é um parâmetro do RCFS chamado *timeout*.

Nos casos onde esse limite é ultrapassado, é feito um acesso remoto ao servidor para obter a data da última modificação do arquivo. Caso esta data seja diferente da data da última modificação do arquivo local, a versão em *cache* deverá ser atualizada.

Estrutura da implementação

O RCFS é composto por diversos subsistemas: *mem*, *iocache* e *rcfs*, descritos a seguir.

Especificação do sistema de arquivos *rcfs*. Este módulo é o responsável pelo controle e ligação dos anteriores. Nele é descrito o comportamento das operações realizadas e o controle de arquivos em *cache* e remotos.

A entrada e saída *iocache*. O *iocache* consiste nas funções responsáveis pela manipulação de arquivos que o aplicativo necessitar. Pode-se, com ele, otimizar e escolher as

melhores formas de realizar a leitura ou escrita de arquivos e diretórios.

O alocador de memória *mem*. O alocador permite ao aplicativo uma interface simples, padrão e controlável de requisição de memória ao sistema operacional. Em modo de depuração, a contagem de alocações e liberações permite o controle do uso de memória e a definição de pontos com altas requisições. Caso a depuração esteja desabilitada, os procedimentos são apenas apelidos, ou macros, para as chamadas padrão do sistema Linux. Essa última escolha de implementação elimina qualquer sobrecarga da chamada de outra função com apenas uma instrução, que seria o caso sem a depuração.

A figura 3.3 ilustra as interações entre o *rcfs* e os demais módulos. Em uma requisição do FUSE (1), o *rcfs* realizará os devidos acessos que podem ser remotos (3), quando um arquivo é inválido ou inexistente na *cache*, ou locais (4) através do *iocache*. A decisão dos tipos de acessos é feita pelo *rcfs* conforme as verificações de validade ou diferença de arquivos. A validade se faz por tempos de acesso, como descrito anteriormente. Na diferença, é comparado o tamanho do arquivo remoto e do local.

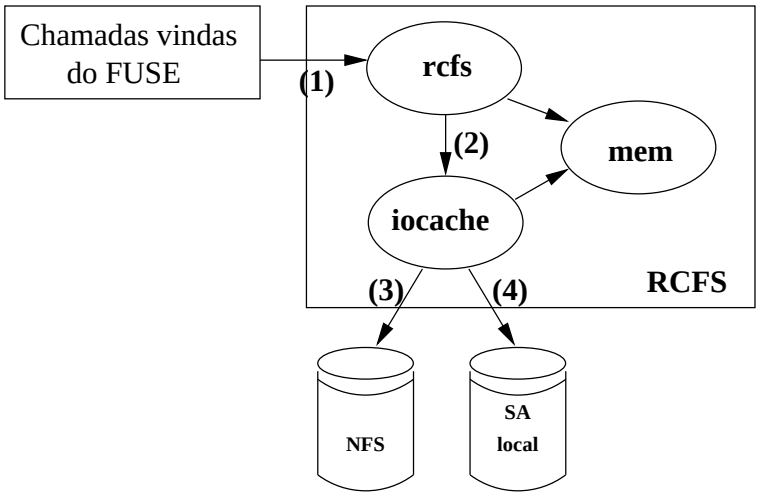

Figura 3.3: Estrutura da implementação proposta.

3.3 Estudo de caso

O estudo de caso desta seção trata sobre o funcionamento dos sistemas compartilhados do NCC na UFSM. O NCC conta com alguns grupos de máquinas semelhantes onde cada um possui aproximadamente dez computadores. Estes são compartilhados entre os usuários do Curso de Ciência da Computação com o propósito de realizarem os trabalhos de graduação e outras atividades. A seguir, é relatado como a administração do NCC mantém os sistemas homogêneos atualmente, e em seguida são discutidos os problemas

enfrentados. Por fim, será apresentada a forma que o trabalho proposto atuará na atualização dos clientes.

3.3.1 Atualização de sistemas no NCC

No grupo de computadores mais recentes, é realizada a instalação do sistema localmente em uma das máquinas e as demais são atualizadas a partir desta. Duas maneiras distintas de realizar essa tarefa já foram utilizadas no NCC.

A primeira consiste em fazer a atualização de um cliente e depois sincronizar as restantes com o programa *rsync* (TRIDGELL, 1999). A execução do processo necessita da especificação dos diretórios que devem ser comparados e dos clientes que sofrerão o processo.

Na outra, utilizada atualmente, a atualização dos equipamentos depende da reinstalação do sistema em cada um deles. Para tanto é utilizado um *CDROM*, chamado *livecd*, que ao ser iniciado carrega um sistema GNU/Linux com os comandos para a tarefa. Primeiro ele refaz as partições e o sistema de arquivos do disco local, e logo após copia os arquivos de um servidor.

3.3.2 Problemas atuais

As formas de atualização apresentadas são eficientes mas ambas apresentam um problema em comum. Em cada atualização de sistema ou programas, o administrador deve ir até o computador em questão e inserir o *CDROM* ou aplicar o *rsync* e esperar o término da tarefa. Dessa forma, o problema principal é automatizar a tarefa de sincronizar os sistemas dos computadores com instalação local sem a intervenção direta do administrador.

3.3.3 Situação proposta

O trabalho proposto pretende atualizar um grupo de clientes através de um programa em modo usuário de forma que seja desnecessária a intervenção dos administradores. Ele deverá atualizar o sistema de um cliente qualquer e copiar os arquivos que deseja compartilhar a um servidor. Em seguida, bastará esperar que os cliente realizem as atualizações necessárias.

No caso de uma nova atualização estar disponível, ela será visível aos clientes após um período especificado pelo parâmetro *timeout*, responsável pelo tempo de validade de cada arquivo nos clientes.

4 RESULTADOS OBTIDOS

Este capítulo apresenta os resultados obtidos com a solução descrita no capítulo anterior. A fim de facilitar a compreensão, serão utilizados gráficos e tabelas que ilustram os resultados analisados, além das algumas conclusões acerca da implementação realizada.

4.1 Avaliação realizada

A solução proposta tem como objetivo a melhora das atualizações entre equipamentos e a diminuição da sobrecarga de acessos remotos. A fim de se realizar medidas que permitam avaliar os resultados obtidos, um *micro-benchmark* foi desenvolvido.

A avaliação feita por esse *micro-benchmark* é por meio de leituras aos arquivos desejados. Seus parâmetros são o número de arquivos a serem lidos, quantas leituras serão realizadas e o número de *bytes* que devem ser lidos. Dessa forma, ele realiza leituras aleatórias aos arquivos até alcançar o limite especificado. Seu caráter aleatório deve-se ao fato de que este estudo de desempenho não segue um padrão de acessos aos arquivos.

O tempo obtido é desde a abertura do arquivo, com a chamada *open*, a leitura dos *bytes* especificados e o fechamento, com o *close*. Para a consulta da hora de sistema, usa-se a chamada *gettimeofday*. O resultado é impresso na saída padrão e é constituído da média entre as leituras aleatórias realizadas.

Para os testes das próximas seções, os tempos individuais são desconsiderados. O número de leituras realizadas e de execuções feitas em cada avaliação por tamanho de arquivo é demonstrada na tabela 4.1. Para se obter os números apresentados é feito uma média do número de leituras em 50 arquivos do mesmo tamanho e depois uma média das execuções realizadas.

Os equipamentos usados são computadores pertencentes ao NCC com configurações bem definidas. Os cliente são computadores com processador Pentium IV de 2.0GHz,

Tabela 4.1: Número de leituras e execuções

Tam. de arquivos	Leituras	Execuções
de 1KB à 512KB	de 1000 à 200000	100
de 1MB à 8MB	de 100 à 500	10

512MB de memória RAM e disco rígido de 40GB. Como servidor, um computador Pentium IV de 2.4GHz, 2GB de memória RAM e disco de 80GB. Ambos os discos possuem uma *cache* de 8MB, taxa de transferência de 100 MB/s e velocidades de rotação de 7200 rotações por minuto (RPM). O meio de comunicação entre os equipamentos é uma rede Fast Ethernet.

4.2 Resultados com relação ao cliente

Nesta seção são apresentados os principais resultados obtidos do RCFS com relação aos acessos específicos entre um cliente e um servidor, sem considerar a sobrecarga de rede. Nos testes, são utilizados 50 arquivos de um tamanho específico, que variam de 1KB à 8MB.

4.2.1 Acessos locais *versus* acessos com o FUSE

Os gráficos das figuras 4.1 e 4.2 apresentam uma comparação entre leituras de arquivos locais e estas mesmas leituras feitas através do FUSE, como uma camada a mais no acesso aos arquivos em disco.

Como pode-se observar nos gráficos, o uso do FUSE impõe um custo significativo para arquivos pequenos. Na figura 4.2 o desempenho da versão com a opção de *cache* no FUSE é semelhante aos resultados obtidos com leituras diretas em disco.

Esta avaliação não se preocupa em detalhar o motivo pelo qual o FUSE apresenta estes resultados com relação a arquivos pequenos e grandes. Os valores podem ser influenciados tanto pela implementação do módulo em núcleo quanto pela biblioteca em modo usuário ou pelos problemas mencionadas anteriormente no capítulo 2.

4.2.2 Acessos remotos, locais e com o RCFS

Os gráficos das figuras 4.3 e 4.4 demostram os resultados obtidos nas leituras de arquivos locais, remotos com o NFS e com o RCFS. No RCFS, é utilizado um *timeout* de 480 segundos que durante as leituras não foi alcançado propositalmente.

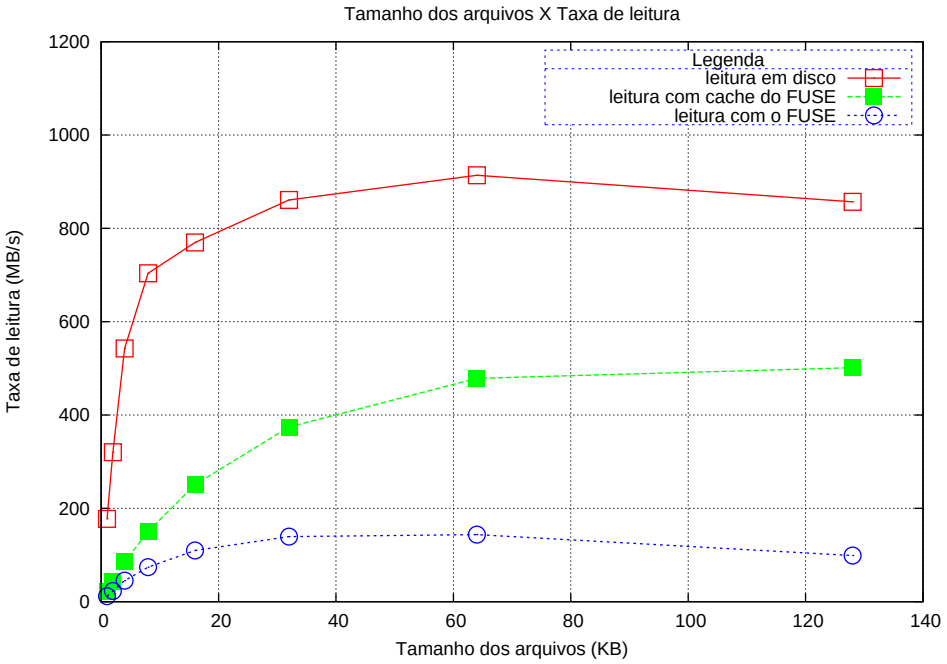

Figura 4.1: Avaliação de desempenho entre leituras locais e leituras locais com o FUSE para arquivos pequenos.

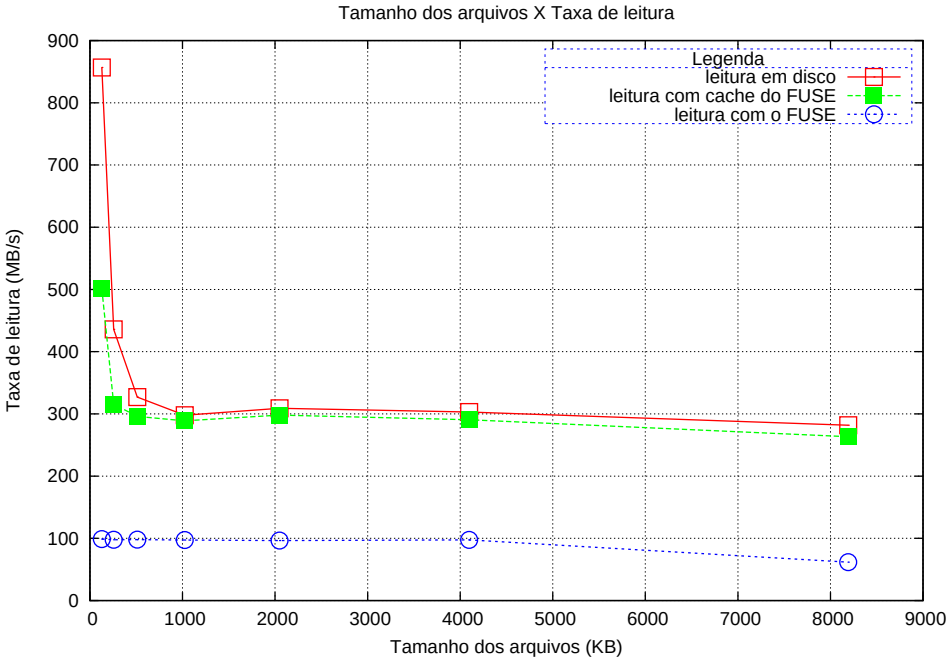

Figura 4.2: Avaliação de desempenho entre leituras locais e leituras locais com o FUSE para arquivos grandes.

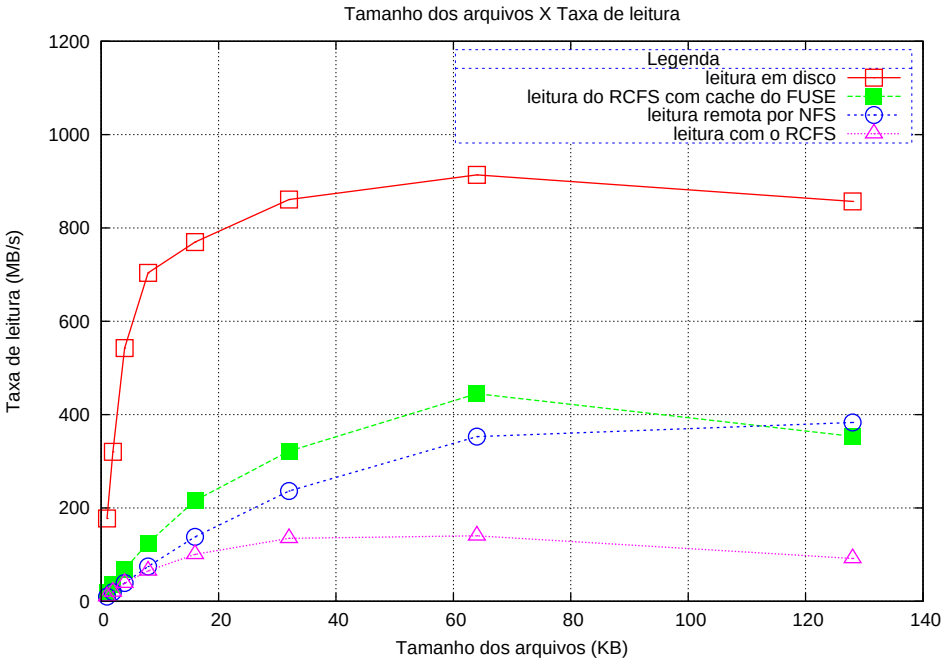

Figura 4.3: Comparação entre leituras locais, remotas e com o RCFS para arquivos pequenos.

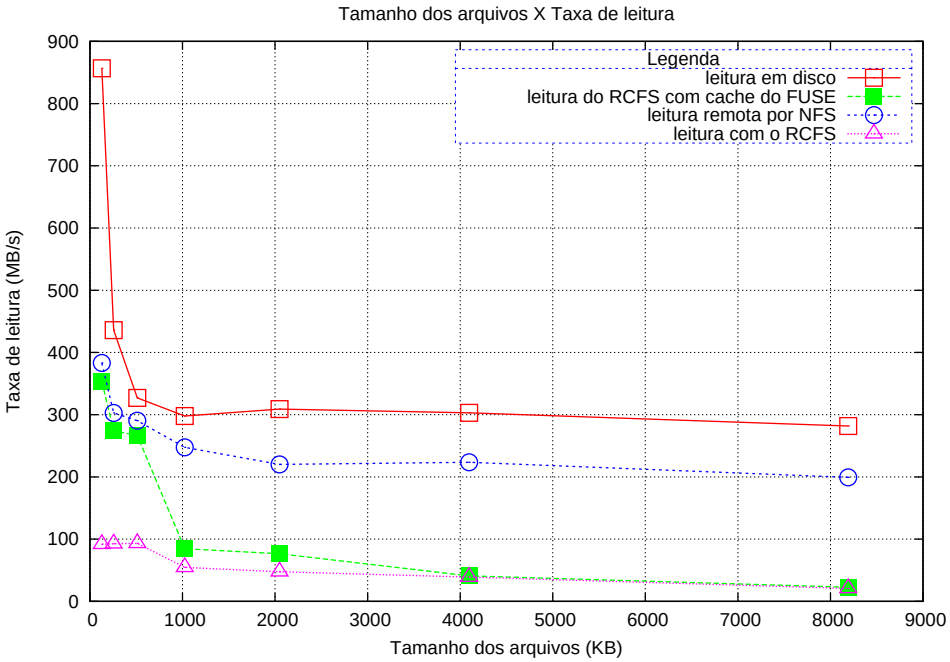

Figura 4.4: Comparação entre leituras locais, remotas e com o RCFS para arquivos grandes.

O gráfico da figura 4.3 apresenta um desempenho melhor do RCFS com relação ao NFS para arquivos entre 1KB e 64KB e uma queda com arquivos maiores que 128KB. Este resultado é justificável devido ao módulo em núcleo do FUSE realizar leituras de, no máximo, 32 páginas de memória ou 128KB no caso da configuração utilizada nestes testes.

Na figura 4.4, o gráfico apresenta uma queda de desempenho significativa entre 512KB e 1MB para o RCFS com a opção de *cache*. Esse resultado é devido ao fato do RCFS gerar um tráfego de rede elevado no primeiro acesso aos arquivos remotos, como será demonstrado na seção seguinte.

4.3 Resultados com relação ao tráfego de rede

Nesta seção são apresentadas duas comparações realizadas para avaliar o tráfego de rede gerado pelo RCFS e pelo NFS. Primeiramente, são mostrados os resultados obtidos com os testes apresentados anteriormente entre um servidor e um cliente. A seguir, são mostrados os resultados obtidos da execução do *micro-benchmark* descrito anteriormente entre alguns clientes.

4.3.1 Tráfego de rede com relação ao cliente

Os gráficos das figuras 4.5 e 4.6 mostram o tráfego de rede gerado pelas operações apresentadas na seção 4.2. O gráfico da figura 4.5 mostram uma diferença significativa do RCFS com relação ao NFS para arquivos menores de 256KB. Esta vantagem confirma os objetivos esperados de se ter uma *cache* de arquivos em disco localmente.

A desvantagem dos arquivos grandes, ilustrado no gráfico da figura 4.6, demonstra que o RCFS apresenta dificuldades com relação a arquivos maiores que 1MB. Este tráfego extra se deve à implementação atual copiar inteiramente um arquivo ao disco local no primeiro acesso, mesmo que a leitura seja realizada em apenas parte do requisitado.

4.3.2 Tráfego de rede com relação ao servidor

O testes realizados nesta categoria objetivam demonstrar o tráfego de rede gerado pelo RCFS e pelo NFS com acessos de mais de um cliente. Os números apresentados são uma média de 10 execuções do *micro-benchmark* descrito anteriormente ao ser lidos 300 arquivos de um tamanho específico, entre 4 clientes com as mesmas configurações apresentadas na seção 4.1. Estes tamanhos variam entre 1KB e 8MB.

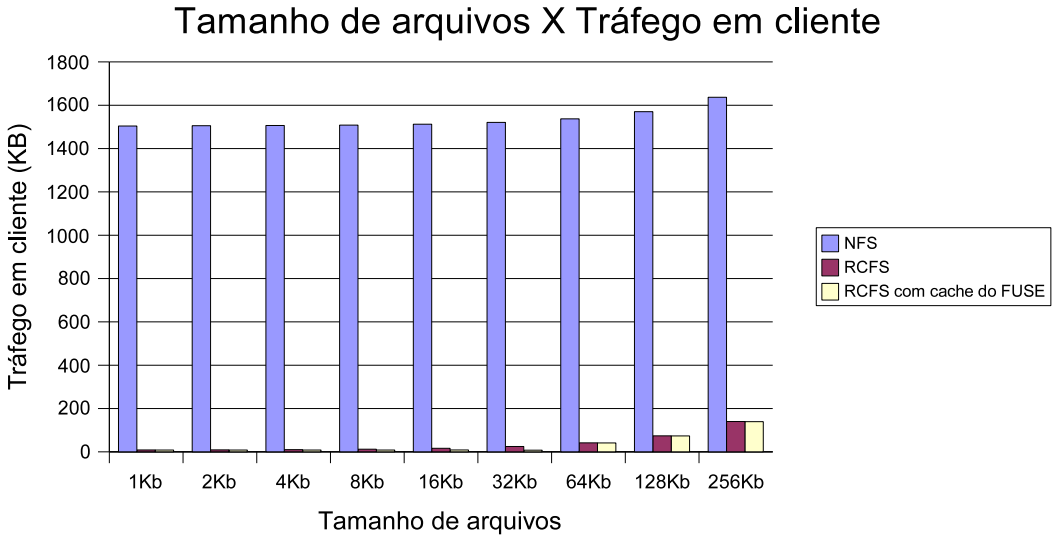

Figura 4.5: Avaliação de tráfego feita por um cliente RCFS e NFS para arquivos pequenos.

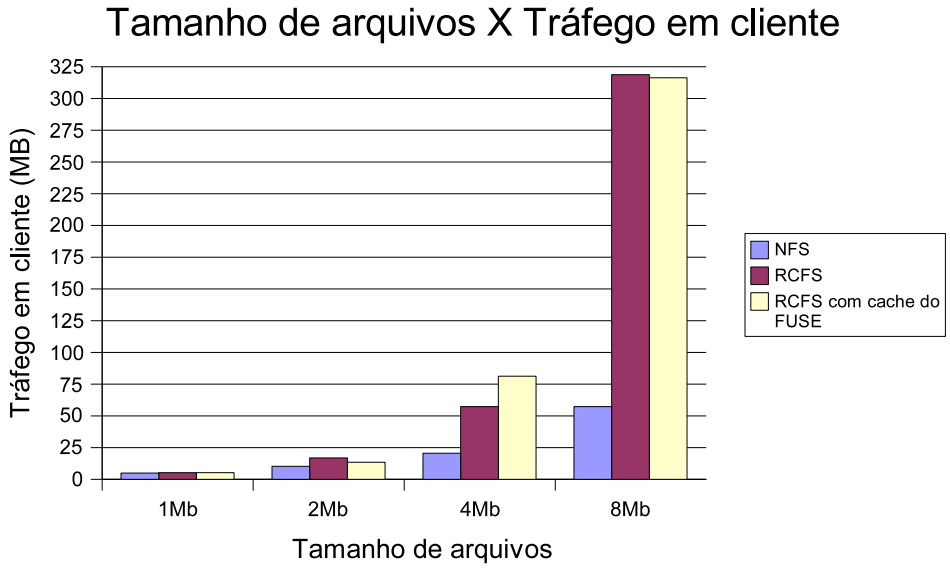

Figura 4.6: Avaliação de tráfego feita por um cliente RCFS e NFS para arquivos grandes.

As figuras 4.7 e 4.8 apresentam os gráficos com os resultados obtidos com diferentes tamanhos de arquivos. Tanto com arquivos pequenos quanto grandes, ilustrados nas figuras, o RCFS apresenta um tráfego menor que o NFS em servidor. Os resultados ilustrados demonstram a vantagem do RCFS quanto a redução de carga do servidor e do meio de comunicação utilizado.

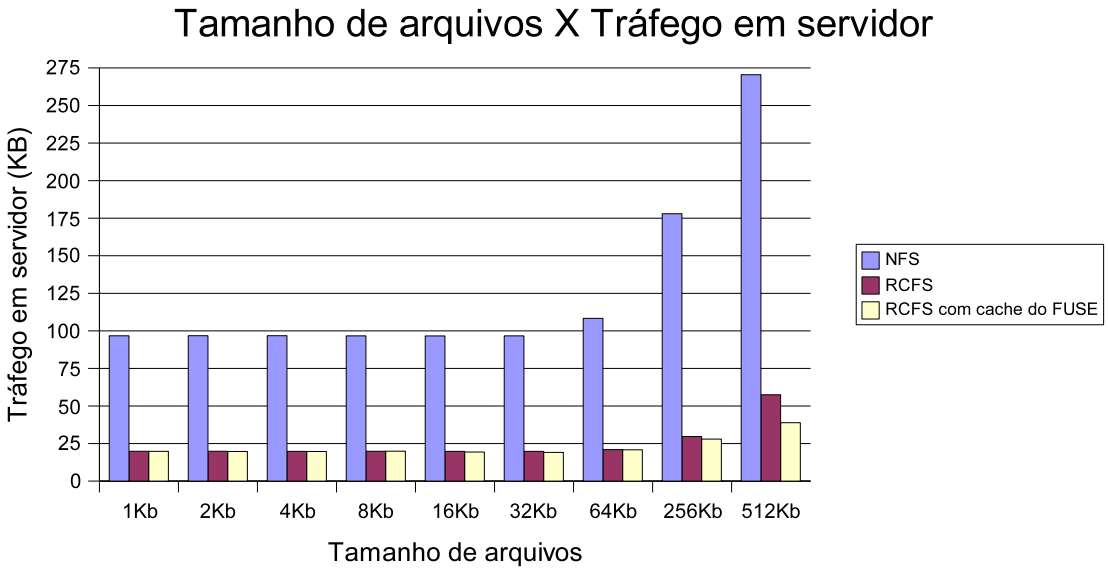

Figura 4.7: Avaliação de tráfego entre um cliente e um servidor com o NFS e o RCFS para arquivos pequenos.

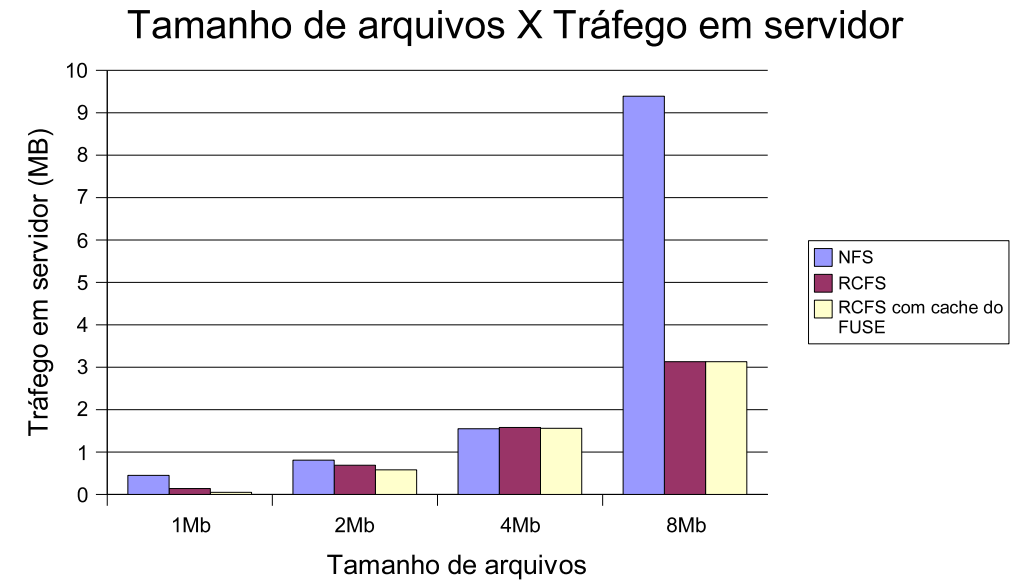

Figura 4.8: Avaliação de tráfego entre um cliente e um servidor com o NFS e o RCFS para arquivos grandes.

5 CONCLUSÃO E TRABALHOS FUTUROS

Um dos problemas enfrentados por administradores de rede é a atualização de computadores com sistemas instalados localmente. Neste tipo de problema, a tarefa de realizar tais atualizações pode-se tornar muito manual ou cara em termos de recursos como o meio de comunicação utilizado.

Alguns trabalhos já tentaram solucionar este problema, como o NFS e o FS-Cache. Todavia, o primeiro causa sobrecarga do meio de comunicação entre os equipamentos caso seu número cresça significativamente. No segundo, a idéia de armazenar os acessos mais recentes em disco o favorece mas seu funcionamento depende de alterações no sistema que o utiliza, dificultando futuras atualizações do sistema.

Neste trabalho, a proposta foi proporcionar um meio simples e transparente de atualizar os computadores de uma rede. Baseado em uma idéia mista de acessos remotos e *cache* em disco, a implementação de um sistema de arquivos em modo usuário com o FUSE, chamado de RCFS, permite a cópia dos acessos mais recentes para o disco. Ao mesmo tempo, verificações de novas versões de arquivos remotos são realizadas sem o conhecimento dos administradores ou programas do sistema. Este foi um dos objetivos que a implementação realizada alcançou.

Os resultados apresentados demonstraram um custo elevado com o uso do FUSE e um tráfego elevado para arquivos acima de 1MB por parte do RCFS. Da mesma forma, o uso em alguns clientes deste sistema de arquivos mostrou que o tráfego de rede está abaixo do gerado pelo NFS com acesso inteiramente remotos. Assim conclui-se que os primeiros acessos com arquivos grandes geram um custo maior no RCFS, devido ao acesso remoto, mas o tráfego médio comparado com o NFS é baixo.

Trabalhos Futuros

Um dos pontos que este trabalho constatou mas não justificou é a causa do custo dos acessos feitos através do FUSE. Uma investigação a cerca dos custos gerados pelo uso do FUSE, assim como a identificação das partes responsáveis por estes resultados, pode ser realizada.

Os acessos remotos realizados pelo RCFS podem ser melhorados na medida que arquivos de tamanhos acima de 1MB possam ser acessados em partes ao invés de serem buscados inteiramente. Da mesma forma, essa modificação poderá levar a uma alteração na forma de armazenar os arquivos em *cache* localmente a fim de reconhecer as partes presentes em disco.

Ainda na implementação, um mecanismo de *cache* em memória de entrada e saída mais elaborado poderá colaborar para algum ganho em desempenho dos acessos realizados localmente na *cache* em disco.

Um outro ponto não tratado neste trabalho, é a parte de tolerância a falhas onde pode-se fazer mecanismos eficientes de recuperação de arquivos antigos, ou algoritmos eficientes para verificação de recursos de disco disponíveis.

Por fim, a atualização dos clientes com RCFS pode ser melhorada com um mecanismo que possibilite a notificação de que novos arquivos estão disponíveis, ou que o sistema de arquivos remoto foi alterado. Essa alteração leva a criação de uma ferramenta remota que comunique-se com os clientes.

REFERÊNCIAS

BOVET, D. P.; CESATI, M. **Understanding the Linux Kernel, 2nd Edition**. United States: O'Reilly, 2002.

CALLAGHAN, B.; PAWLOWSKI, B.; STAUBACH, P. **NFS Version 3 Protocol Specification**. Disponível em: http://www.ietf.org/rfc/rfc1813.txt, RFC 1813 (Informational).

HOWELLS, D. FS-Cache: A Network-Filesystem Caching Facility. **Preceedings of the Linux Symposium**, Ottawa, Ontario, Canada, v.1, p.427–440, 2006.

INC., L. K. O. **The Linux Kernel Archives**. `http://www.kernel.org`.

KERNIGHAN, B. W.; RITCHIE, D. M. **The C Programming Language**. second.ed. United States: Prentice Hall, 1988.

KLEIMAN, S. R. Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In: USENIX SUMMER, 1986. **Anais...** USENIX, 1986. p.238–247.

PAWLOWSKI, B.; JUSZCZAK, C.; STAUBACH, P.; SMITH, C.; LEBEL, D.; HITZ, D. NFS Version 3: Design and Implementation. In: USENIX SUMMER, 1994. **Anais...** USENIX, 1994. p.137–152.

PAWLOWSKI, B.; SHEPLER, S.; BEAME, C.; CALLAGHAN, B.; EISLER, M.; NOVECK, D.; ROBINSON, D.; THURLOW, R. The NFS Version 4 Protocol. **Proceedings of the 2nd international system administration and networking conference (SANE2000)**, United States, p.94, 2000.

PROJECT, T. L. N. **Linux NFS faq**. `http://nfs.sourceforge.net`.

RITCHIE, D. M.; THOMPSON, K. The UNIX Time-Sharing System. **Communications of the ACM**, New York, NY, USA, v.17, n.7, jul 1974.

SANDBERG, R.; GOLGBERG, D.; KLEIMAN, S.; WALSH, D.; LYON, B. Design and Implementation of the Sun Network Filesystem. **Innovations in Internetworking**, Norwood, MA, USA, p.379–390, 1988.

SHEPLER, S.; CALLAGHAN, B.; ROBINSON, D.; THURLOW, R.; BEAME, C.; NOVECK, M. E. D. **Network File Ssystem (NFS) version 4 Protocol**. Disponível em: http://www.ietf.org/rfc/rfc3530.txt, RFC 3530 (Standards Track).

SUN MICROSYSTEMS, I. **NFS**: Network System Protocol Specification. Disponível em: http://www.ietf.org/rfc/rfc1094.txt, RFC 1094 (Informational).

SZEREDI, M. **FUSE**: Filesytem in Userspace. `http://fuse.sourceforge.net`.

TANEMBAUM, A. S.; WOODHULL, A. S. **Operating Systems Design and Implementation**. second.ed. United States: Prentice Hall, 1997.

TRIDGELL, A. **Efficient Algorithms for Sorting and Synchronization**. 1999. Tese (Doutorado) — . `http://samba.org/~tridge/phd_thesis.pdf`.