UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Thiago Kenji Okada

Metodologia para Recuperação de Falhas e Garantia de Disponibilidade no FlexA

São José do Rio Preto
2013

Thiago Kenji Okada

Metodologia para Recuperação de Falhas e Garantia de Disponibilidade no FlexA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Prof. Dr. Aleardo Manacero Jr. Thiago Kenji Okada

Banca Avaliadora:

Prof. Dr. Leandro Alves Neves
Prof. Dr. Norian Marranghello

São José do Rio Preto
2013

Okada, Thiago Kenji.
Metodologia para recuperação de falhas e garantia de disponibilidade no FlexA / Thiago Kenji Okada. -- São José do Rio Preto, 2013
74 f. : il., gráfs., tabs.

Orientador: Aleardo Manacero Junior
Trabalho de conclusão de curso (bacharelado – Ciência da Computação) – Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Sistemas operacionais distribuídos (Computadores) 3. Sistemas de arquivos distribuídos. I. Manacero Junior, Aleardo. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.
CDU – 681.3

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Campus de São José do Rio Preto

Ao meus pais Mauro e Eliana

Aos meus queridos avós Makoto e Katsue, Satoshi e Marico

Ao meu irmãozinho Felipe

Meu muito obrigado

"Duas coisas são infinitas: o universo e a estupidez humana.
Mas, no que respeita ao universo, ainda não adquiri a certeza absoluta."

-Albert Einstein

Agradecimentos

Agradeço primeiramente ao meu pai, Mauro Yukio Okada, que sempre lutou para dar ao seu filho do bom e do melhor; e a minha mãe, Eliana Mayumi Hieda por sempre apoiar as minhas decisões e cuidar de mim praticamente sozinha durante toda a minha adolescência. E claro ao meu irmãozinho Felipe, que sempre está disposto a brincar.

Agradeço também aos meus avós paternos Makoto e Katsue Okada, por estarem sempre presentes quando eu precisava. Também agradeço a minha avó materna Marico por sua honestidade, além do meu avô Satoshi que eu não cheguei a conhecer.

Gostaria de agradecer também a Prof.ª Renata Spolon Lobato, por coordenar o projeto de iniciação científica em que este projeto faz parte, ao Prof. Aleardo Manacero Jr. pela co-orientação e suporte sempre que minha orientadora não pode estar presente, além da FAPESP pela bolsa de iniciação científica (processo 2012/24861-2) além dos equipamentos cedidos (processos 04/01340-0 e 2012/02926-5).

Por fim não posso deixar de agradecer a todas as amizades que construi antes e durante a graduação: Tinen (Bruno), Musashi, Dudous (Eduardo) e Yudy (Guilherme), Woompa-Loompa (Gabriel 1), Mavalu (Gabriel 2), Napa (Matheus), Lúcio, Pai (Rafael Stabile), Cassião (Cássio), Leandro & Leonardo, Denilson (Denison), Gordo (Diogo), Ex-Gordo (Danilo), Mario Camara, Fatal (Arthur), Amarelo (Willian), Suvaco (Heitor), Mari (Mariana), Jão (João Marcos), Gustavo, Mano (Vinícius 1), Galhardi (Vinícius 2), João Matheus, Raphel Campos, Dan Alec, Gordo 2 (Vitor Strazzi). Se não fosse as conversas jogadas foras com todos vocês provavelmente não estaria aqui agora.

Resumo

O aumento no volume de dados e a incerteza dos recursos de *hardware* e *software* junto com a descentralização dos dados em sistemas de arquivos surgiu como uma forma de minimizar a probabilidade de perda total desses dados. Considerando isto, este trabalho modificou um sistema de arquivos existente para que o mesmo tenha uma melhor tolerância quanto a falhas, melhorando a disponilidade do sistema estudado. Também foram feitos testes comparando o sistema modificado com sua versão original e com outros sistemas de arquivos distribuído disponíveis, para verificar o impacto das modificações no sistema.

Palavras-chave: Sistemas de Arquivos Distribuídos, Tolerância a Falhas, Réplicas

Abstract

The increasing amount of data and uncertainty of available hardware and software resources with the decentralization of data in file systems came as a method to minimize data loss. Thus this paper proposes a modification of an existing file system to garantee security against data loss and better availability. Tests made comparing the modified system, the original version and other distributed file system available, to verify the impact of the modifications on this system.

Keywords: Distributed File Systems, Fault Tolerancy, Replication

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AFS Andrew File System

AVSG Accesible Volume Storage Group

CPU Central Processing Unit

FlexA Flexible and Adaptable Distributed File System

GB Gigabyte

GFS Google File System

GHz Gigahertz

GSPD Grupo de Sistemas Paralelos e Distribuídos

HDD Hard Disk Drive

HDFS Hadoop Distributed File System

LAN Local Area Network

MB Megabyte

MB/s Megabytes por segundo

Mbps Megabits por segundo

MDS (Ceph) Metadata Daemon Server

MDS (Lustre) Metadata Server

MDT Metadata Target

Mem. Memória

ND Não disponível

NFS Network File System

NIC Network Interface Controller

OSD Object Storage Device

OSS Object Storage Server

OST Object Storage Target

PG Placement Group

Proc. Processador

RAM Random Access Memory

RPC Remote Procedure Call

SAD Sistema de Arquivo Distribuído

SD Sistema Distribuído

Tahoe-LAFS Tahoe Least-Authority File System

VSG Volume Storage Group

Capítulo 1

Introdução

Neste capítulo é mostrada a motivação para o trabalho feito, seus objetivos e a organização geral do texto apresentado.

1.1 Motivação

A evolução tecnológica das últimas décadas permitiu o crescimento rápido do poder de processamento computacional enquanto tornou os preços cada vez mais acessíveis. O surgimento dos microprocessadores, mais baratos e com poder de processamento superior aos antigos *mainframes*, e das redes locais de alta velocidade (LAN), permitiram conectar um grande número de computadores numa rede e facilitando assim o desenvolvimento de sistemas distribuídos (TANENBAUM; STEEN, 2007).

Com o volume cada vez maior de dados, sistemas convencionais não suportam mais o crescente número de usuários. Além disso, falhas de *software* e *hardware* são comuns, o que pode levar à interrupção de serviços e à insatisfação dos usuários.

A construção de sistemas distribuídos vem da motivação de se compartilhar recursos computacionais. Essa motivação envolve diversos desafios como concorrência, inexistência de um relógio global, diversos tipos de falhas e outros problemas que serão mostrados nos capítulos posteriores (COULOURIS et al., 2011).

Nesse contexto, sistema de arquivos distribuídos devem promover uma solução para o gerenciamento de arquivos oferecendo, na medida do possível, características desejadas em sistemas distribuídos: transparência, escalabilidade, desempenho, controle de concorrência, tolerância a falhas e segurança.

Desse modo, existem diversos sistemas que procuram suprir as principais características de um sistema distribuído para arquivos. Porém, a agregação de mais recursos torna o conjunto difícil de ser administrado. Com base nisso, este trabalho busca expandir as características de um sistema de arquivos distribuído em desenvolvimento, de modo a expandir características desejadas como tolerância a falhas e segurança dos

dados armazenados.

1.2 Objetivo

Neste trabalho é apresentado o desenvolvimento de um grupo de servidores secundários com o objetivo de replicação de dados no sistema de arquivos distribuído FlexA e garantir a disponibilidade do sistema a partir da eleição de servidores secundários como primários.

1.3 Organização do texto

Este texto está organizado da seguinte forma:

- No Capítulo 2 é apresentado a fundamentação teórica do trabalho desenvolvido, uma visão geral de sistemas distribuídos e seus problemas, uma visão geral de sistemas de arquivos distribuídos e seus principais representantes, fechando com uma fundamentação teórica de disponibilidade em sistemas distribuídos;
- No Capítulo 3 é descrito a metodologia e o desenvolvimento do trabalho, que aborda a implementação de novas características ao sistema de arquivos FlexA;
- No Capítulo 4 é apresentado os testes e a validação da implementação descrita no capítulo anterior;
- No Capítulo 5 é apresentado as conclusões e direções futuras deste trabalho.

Capítulo 2

Revisão bibliográfica

O presente capítulo aborda uma breve revisão dos conceitos de Sistemas Distribuídos (SDs), seus problemas e recuperação de falhas nesse tipo de sistema. A seguir, há uma revisão do que são Sistemas de Arquivos Distribuídos (SADs) e uma breve descrição dos seus principais representantes, além de uma breve introdução a disponibilidade em SADs.

2.1 Sistemas distribuídos

Segundo Tanenbaum e Steen (2007), "Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente". Por serem independentes, os computadores em um sistema distribuído são interligados em rede para se comunicarem e coordenam suas ações através da passagem de mensagens.

Por serem naturalmente heterogêneos, os sistemas distribuídos apresentam diversos desafios como mostrado em Coulouris et al. (2011):

1. Heterogeneidade: sistemas distribuídos comumente são executados em diferentes configurações de *hardware* e *software*, devendo, assim, ser capazes de contornar essas diferenças;

2. Sistemas abertos: um sistema é considerado aberto quando ele pode ser estendido e reimplementado de várias maneiras. O desenvolvimento de um padrão é necessário a fim de tornar sistemas diferentes compatíveis;

3. Segurança: os dados armazenados em SDs têm um alto valor intrínseco para seus usuários. Portanto a sua segurança é de considerável importância.

   A segurança de recursos de informação tem três componentes: confidencialidade (proteção contra exposição de pessoas não autorizadas), integridade (pro-

teção contra alteração ou dano) e disponibilidade (proteção contra interferência com os meios de acesso aos recursos);

4. Escalabilidade: sistemas distribuídos devem funcionar de forma efetiva e eficaz em muitas escalas diferentes. Para isto, o SD deve ser capaz de aumentar o desempenho com o aumento no número de computadores no sistema;

5. Tratamento a falhas: os sistemas de computador falham com frequência. Porém graças a heterogeneidade dos SDs, as falhas em sistemas distribuídos são parciais, isto é, alguns componentes falham, outros continuam funcionando. Os SDs devem ser capazes de detectar falhas e se recuperar delas.

   Mais detalhes sobre tratamento de falhas serão apresentados na Seção 2.2;

6. Concorrência: como SDs trabalham com recursos compartilhados, existe a possibilidade de vários clientes tentarem acessar um recurso compartilhado ao mesmo tempo. Portanto, é necessário que um recurso compartilhado em um sistema distribuído opere corretamente em um ambiente concorrente;

7. Transparência: os SDs têm que ser desenvolvidos de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes. Este conceito pode ser dividido em vários tópicos:

   - Transparência de acesso: clientes não precisam saber onde os arquivos estão;
   - Transparência de localização: clientes devem ver um sistema de arquivos uniforme;
   - Transparência de mobilidade: nem o cliente nem o administrador devem precisam ser modificados quando os arquivos forem movidos;
   - Transparência de desempenho: os clientes devem ter desempenho satisfatório mesmo quando a carga de serviço varia dentro de um limite especificado;
   - Transparência de escalabilidade: o serviço deve ser expandir para atender diferentes cargas e tamanhos de rede.

2.1.1 Modelo de falhas

As falhas possíveis em um sistema distribuído podem ser classificadas de acordo com a lista abaixo:

- Travamento: o servidor trava, mas estava funcionando corretamente antes da falha;

- Omissão: o servidor para de responder às requisições;
- Temporização: a resposta do servidor está fora do tempo especificado;
- Falha de resposta: a resposta do servidor está incorreta;
- Falha arbitrária: o servidor pode produzir respostas arbitrárias em tempos arbitrários.

Um **travamento** ocorre quando o servidor para prematuramente. Um importante aspecto dessa falha é que uma vez que o servidor travou, não há nenhuma resposta dele.

Uma falha por **omissão** ocorre quando o servidor não responde a uma requisição. No caso de uma falha de recebimento, o servidor pode não ter recebido a requisição, ou uma falha ao iniciar a *thread* que ouviria as requisições. Nesse caso, a falha geralmente não afeta o estado atual do servidor, já que o mesmo não recebeu nenhuma mensagem. Por outro lado, o servidor pode ter feito seu trabalho mas a falha ocorreu ao enviar a resposta. Um outro tipo de falha de omissão pode acontecer em erros de *software* como laços infinitos ou gerenciamento impróprio da memória.

Uma falha de **temporização** pode ocorrer quando a resposta chega em um tempo fora do especificado, geralmente causado por problemas de desempenho do sistema.

Uma falha de **resposta** acontece quando a resposta do servidor simplesmente é incorreta. Nesse caso, tanto a requisição pode estar errada (o que pode gerar problemas em servidores mal configurados) quanto a resposta do servidor simplesmente estar errada para determinada requisição.

Uma falha **arbitrária**, também conhecida como *Bizantina*, é uma falha em que o sistema não estava preparado para tratar. Esses são os tipos de falhas mais perigosas em um sistema (TANENBAUM; STEEN, 2007).

2.2 Tolerância a falhas

Uma característica que distingue sistemas distribuídos de sistemas simples são as falhas parciais. Uma falha parcial pode acontecer quando um componente do sistema distribuído falha, enquanto em sistemas não distribuídos uma falha geralmente afeta todos os componentes, tornando fácil derrubar todo o sistema. Por isso é importante construir um sistema distribuído que consiga se recuperar de falhas parciais sem afetar o desempenho (TANENBAUM; STEEN, 2007).

2.2.1 Mascaramento de falhas por replicação

Se um sistema é feito para ser tolerante a falhas, o melhor que se pode fazer é tentar esconder as falhas de outros processos. Uma das maneiras de se fazer isso é usando

redundância (COULOURIS et al., 2011).

Para arquivos, podemos fazer sua replicação para garantir a redundância. As técnicas de replicação podem ser desde ações simples como o uso de *caches* no cliente ou recursos mais avançados, como o uso de gerenciadores de réplicas. Nesse último caso, a replicação pode atuar de dois modos: passivo ou ativo.

Na replicação passiva, também conhecida como *backup* primário para tolerância a falhas, existe um grupo de gerenciadores de réplicas, sendo um o gerente primário e o restante secundários. O gerente primário executa operações e envia cópias do dado para os secundários. Se o primário falha, um dos *backups* assumem como primário.

Na replicação ativa, os computadores do grupo de réplicas desempenham todos a mesma funcionalidade, ou seja, as requisições são enviadas a todos os gerenciadores que processam e respondem independentemente. Nesse modelo, a falha de algum gerenciador não implica no desempenho do serviço, já que os gerenciadores de réplica restantes continuam respondendo normalmente.

2.3 Eleição de nós em sistemas distribuídos

Considerando um sistema distribuído, para eleger qualquer nó da rede para uma nova função (por exemplo, de servidor secundário para primário) podemos utilizar de algoritmos de eleição. Uma breve descrição dos algoritmos clássicos de eleição na literatura aparecem a seguir (COULOURIS et al., 2011) (TANENBAUM; STEEN, 2007):

- Algoritmo *bully*: o processo P inicia uma eleição. Ele envia uma mensagem a todos os nós com prioridade maior que ele. Se ninguém responder, o P ganha a eleição. Se alguém responder, o P desiste.

- Algoritmo anel: cada processo sabe qual é o seu posterior (por alguma métrica a ser definida, como por exemplo o ID do processo). Quando qualquer processo descobre que o coordenador não funciona, ele inicia uma eleição enviando uma mensagem contendo seu número de processo e envia ao seu sucessor. Se o sucessor está indisponível, o processo que enviou pula o sucessor até um processo funcional ser achado. A cada passo o processo que envia a mensagem adiciona seu próprio número de processo. Quando a mensagem chegar no primeiro nó, a eleição acaba e uma nova mensagem enviando o nó eleito (com o maior número da lista) é passada até chegar ao final do anel.

No pior caso, o algoritmo *bully* envia $O(n^2)$ mensagens, enquanto no melhor caso são enviados $O(n)$ mensagens. O algoritmo anel sempre envia $2(n-1)$ mensagens independente de quem inicia a eleição e de possíveis falhas no meio (JHAVERI; SHAH, 2011).

2.4 Sistemas de arquivos distribuídos (SAD)

De acordo com Coulouris et al. (2011), "Um sistema de arquivos distribuído (SAD) permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais, possibilitando que os usuários acessem arquivos a partir de qualquer computador em uma rede.".

Consequentemente, um SAD deve possuir as operações convencionais de leitura, escrita e listagem de arquivos. Os conceitos de sistemas distribuídos vistos anteriormente também se aplicam aos SADs (COULOURIS et al., 2011).

Nesta Seção alguns SADs são apresentados, analisando suas características e qualidades, a fim de agregar suas melhores qualidades no trabalho. Além disso, será mostrado conceitos sobre disponibilidade em SADs.

2.4.1 *Network File System* (NFS)

O *Network File System* foi criado pela Sun Microsystem e provavelmente é o sistema de arquivos distribuído mais popular. Para estimular sua adoção como padrão, as definições das interfaces foram colocadas em domínio público, permitindo várias implementações diferentes mas compatíveis. Atualmente, ele é suportado por muitos fornecedores e o protocolo NFS é um padrão da Internet, definido no RFC 1813 (COULOURIS et al., 2011).

A arquitetura do NFS consiste em três partes: o protocolo, o servidor e o cliente.

O protocolo que o NFS usa é o *Remote Procedure Call* (RPC). Ele facilita a especificação da definição, organização e implementação de serviços remotos. O protocolo é definido com procedimentos, seus argumentos e efeitos. Os RPCs são síncronos.

O NFS é um protocolo *stateless*: os parâmetros de cada procedimento contém toda a informação necessária para completar a requisição. Caso o servidor falhe, o cliente refaz a requisição até uma resposta ser recebida. Por ser *stateless*, quando uma requisição é feita, o servidor tem que tratar de todos os dados modificados antes de retornar uma resposta (OSADZINSKI, 1988). Sua arquitetura é apresentada na Figura 2.1.

O NFS não tem gerenciamento nativo de réplicas (COULOURIS et al., 2011), mas uma implementação de réplicas nesse sistema pode ser encontrado em Liskov et al. (1990), onde é usado o método da cópia primária.

Nesse trabalho, as requisições do cliente são enviadas para um servidor definido como primário. Se a operação do cliente não envolver modificações no sistema de arquivos, essa operação é feita completamente no primário; nenhuma comunicação com o resto dos servidores (conhecidos como *backups*) precisa ser feita.

Operações de modificação ocorrem em duas fases, na primeira fase o primário informa os *backups* sobre a operação. Depois que o *backup* avisa que recebeu mensagem, a operação é de fato executada. Neste ponto, o primário retorna a informação ao

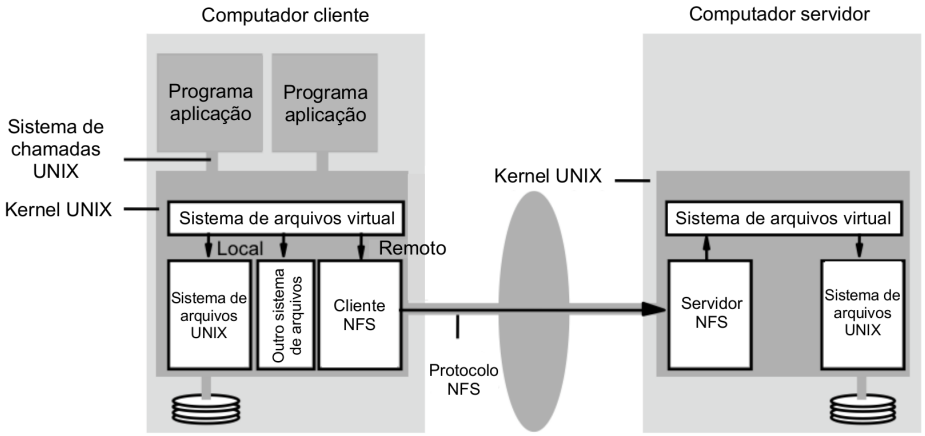

Figura 2.1: Arquitetura do NFS por Tanenbaum e Steen (2007) adaptado por Fernandes (2012)

ciente; os *backups* são informados para executar a operação em segundo plano (essa é a fase 2).

Tanto o primários quanto as réplicas guardam um registro das operações. A informação é removida do registro apenas após a escrita no sistema de arquivos tanto do primário quanto dos *backups*.

2.4.2 *Andrew File System* (AFS)

O AFS é compatível com o NFS, porém difere no seu projeto e implementação. As diferenças são atribuídas principalmente à identificação da escalabilidade como objetivo mais importante do projeto. O AFS foi desenvolvido com um grande número de usuários em mente.

A arquitetura do AFS é composta de dois componentes de *software*, o *Vice* (executado no servidor) e o *Venus* (executado nos clientes). Os arquivos compartilhados são armazenados em servidores, e cópias deles são armazenadas na *cache* dos discos locais das estações de trabalho. O *Venus* gerencia a *cache*, removendo arquivos usados menos recentemente para caso seja necessário. Sua arquitetura pode ser vista na Figura 2.2.

Quando o *Vice* fornece uma cópia de um arquivo para um processo *Venus*, ele também fornece uma *promessa de callback*, um *token*, garantindo que notificará o processo do cliente quando qualquer outro cliente modificar o arquivo. Desta forma, o servidor realiza uma requisição de atualização, ele avisa todos os processos *Venus* que contém o *token* que versão do arquivo no *cache* foi invalidada.

O AFS permite réplicas apenas no modo somente-leitura, em diretórios que são raramente modificados como o */bin* e */usr/bin* do UNIX. Neste modo, somente uma

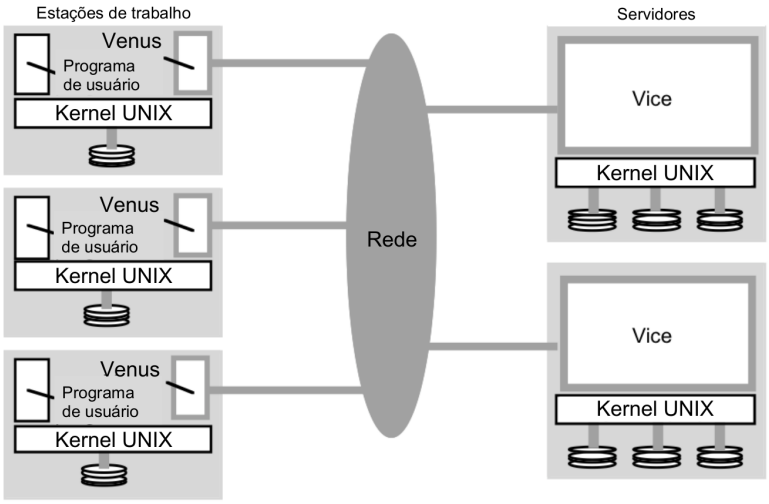

Figura 2.2: Arquitetura do AFS por Coulouris et al. (2011) adaptado por Fernandes (2012)

réplica tem permissão de escrita e todas as outras réplicas são atualizadas diretamente dele (COULOURIS et al., 2011).

2.4.3 Coda

Muitos aspectos do Coda são inerentes do AFS. Cada cliente executa um processo chamado *Venus*, que se encarrega das requisições remotas do sistema usando um disco local como *cache*. Na ausência de recursos do Coda, seu modelo de consistência é idêntico do AFS.

A unidade de replicação do Coda é um *volume*. Um volume é uma coleção de arquivos que são guardados em um servidor e formam uma subárvore com o espaço de nomes de arquivo compartilhado. O grupo de servidores que contém as réplicas do volume é um grupo de armazenamento de volume (**VSG**, ou *Volume Storage Group*). Dados que não precisam de alta disponibilidade podem ser guardados em volumes não replicáveis. Coda também suporta replicação de volumes somente leitura, recurso derivado do AFS.

Para cada volume que existe dados no *cache*, *Venus* mantém um subconjunto dos VSGs atualmente disponíveis. Esse subconjunto é chamado de AVSG (*Accessible Volume Storage Group*), e é idêntico ao VSG na falta de falhas. Um servidor é deletado do AVSG quando expira o tempo de operação e é adicionado de volta quando o *Venus* consegue conexão com o servidor.

A estratégia de replicação usada é o *read one, write-all*. Quando um arquivo é fechado após uma modificação, ele é transferido para todos os membros do AVSG.

Essa estratégia é simples de implementar e garante que todas as réplicas tem o dado atual (SATYANARAYANAN et al., 1990).

2.4.4 Tahoe-LAFS

O Tahoe-LAFS (*Tahoe Least-Authority File System*) foi desenvolvido segundo o princípio da menor autoridade, onde cada usuário ou processo têm que ser feito sem que se precise mais autoridade que o necessário.

A arquitetura do Tahoe-LAFS é composta de: os clientes, os servidores de armazenamento e o servidor *introducer*. O servidor *introducer* procura apenas conhecer a localização dos nós, delegando toda a transferência de dados para interações direta entre o cliente e os servidores de armazenamento. Uma visão geral da arquitetura pode ser vista na Figura 2.3.

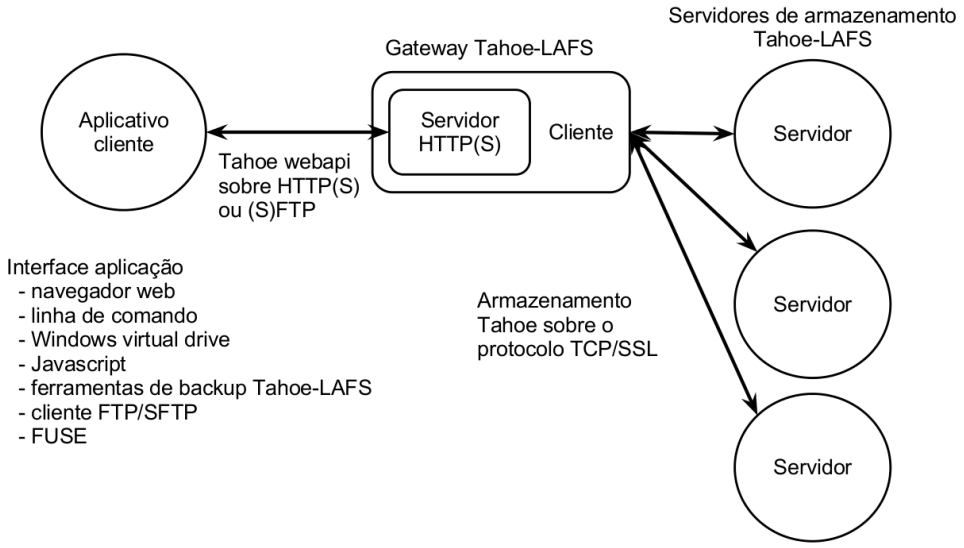

Figura 2.3: Arquitetura do Tahoe-LAFS por Wilcox-O'Hearn e Warner (2008) adaptado por Fernandes (2012)

O sistema distribui os dados e metadados entre os servidores usando criptografia e *erasure coding*. O processo de distribuição segue dois parâmetros: N quantos servidores são usados para guardar o arquivo, e K, o número de servidores necessários para o arquivo estar disponível. Esses parâmetros variam de $1 \leq K \leq N$, sendo que o valor padrão é $K = 3$ e $N = 10$.

Se o número de computadores ativos for menor que K, os arquivos que foram submetidos anteriormente sem condições normais não são recuperados, isso devido à quantidade inferior de nós de armazenamento necessários para recompor o arquivo

original. Entretanto, a escrita de novos arquivos é permitida mesmo com nós ativos abaixo de *K*, porém esses arquivos não serão replicados para os demais nós *N* quando estiverem ativos, limitando-se o acesso a esses arquivos somente aos nós que receberam (WILCOX-O'HEARN; WARNER, 2008).

2.4.5 *Google File System* (GFS)

A arquitetura do *Google File System* consiste de um servidor mestre e vários servidores de porção que são acessados por múltiplos clientes. Desde que os recursos da máquina permitam, é possível executar ambos o servidor de porção e o cliente na mesma máquina.

O servidor mestre guarda os metadados de todo o sistema, se comunicando periodicamente com os servidores de replicação por *HeartBeat* para enviar instruções e coletar os estados. Uma visão geral da arquitetura do GFS pode ser vista em Figura 2.4.

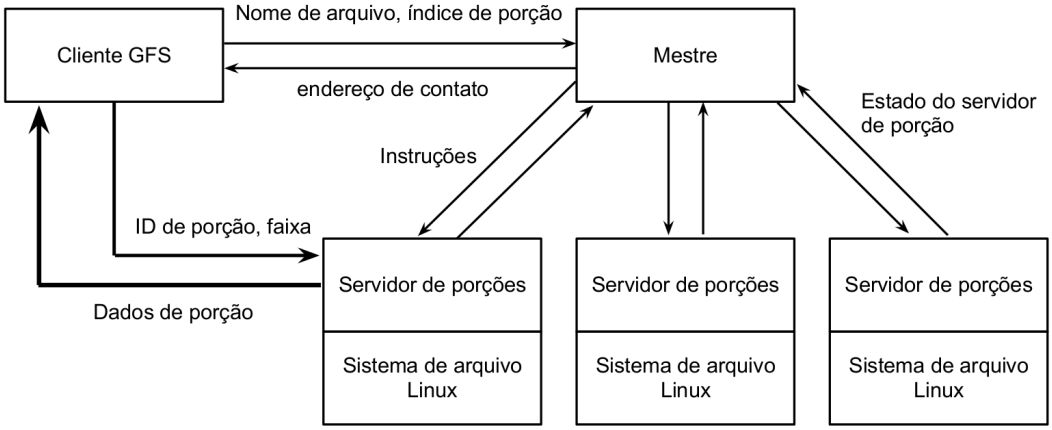

Figura 2.4: Arquitetura do GFS por Ghemawat, Gobioff e Leung (2003) adaptado por Fernandes (2012)

Quando um cliente precisa ter acesso aos dados, ele interage com o mestre para descobrir quais servidores de porção possuem a informação. Depois disso, ele se comunica apenas com os servidores de porção, minimizando o impacto que o acesso a disco teria no servidor mestre e permitindo o sistema ser escalável.

Para garantir a disponibilidade, as porções são replicadas pelos próprios servidores de réplicas, sem interagirem com o mestre. Quando o cliente faz uma operação de atualização, ele interage com o servidor de porção mais próximo e envia o arquivo para o mesmo. Este servidor vai enviar a atualização para o servidor mais próximo e assim por diante. Quando todas as atualizações terminarem, o cliente vai entrar em

contato com o servidor de porção primário que vai dar um número de sequência para a operação de atualização e vai passar para os *backups*.

O número de réplicas das porções podem ser modificadas pelo usuário (o padrão é três). O estado do mestre também é replicado e caso o mesmo falhe, ele pode ser restaurado quase instantaneamente ou um novo processo mestre pode ser iniciado a partir do registro de operações replicado do mestre (GHEMAWAT; GOBIOFF; LEUNG, 2003).

2.4.6 Apache Hadoop

O HDFS (*Hadoop Distributed File System*) é um sistema de código aberto desenvolvido pela Apache. Ele armazena os metadados em um servidor de metadados dedicado, chamado *NameNode* e os dados em outros servidores chamados *DataNodes*. Os dados no HDFS são replicados em vários *DataNodes* para garantir a confiabilidade.

Os nomes e diretórios são armazenados no *NameNode*. Cada arquivo é dividido em grandes blocos e cada bloco é replicado independentemente em vários *DataNodes*. O *NameNode* guarda a estrutura de diretórios e os mapeamentos dos blocos dos *DataNodes*.

Um cliente que deseja ler no sistema de arquivos primeiro requisita ao *NameNode* a localização dos blocos correspondente ao arquivo e lê os blocos do *DataNode* mais perto dele. Na escrita, o cliente requisita ao *NameNode* a escolha de três *DataNodes* para receber os blocos de réplicas. O cliente então escreve os dados usando um *pipeline*. Uma visão geral da arquitetura pode ser vista em Figura 2.5

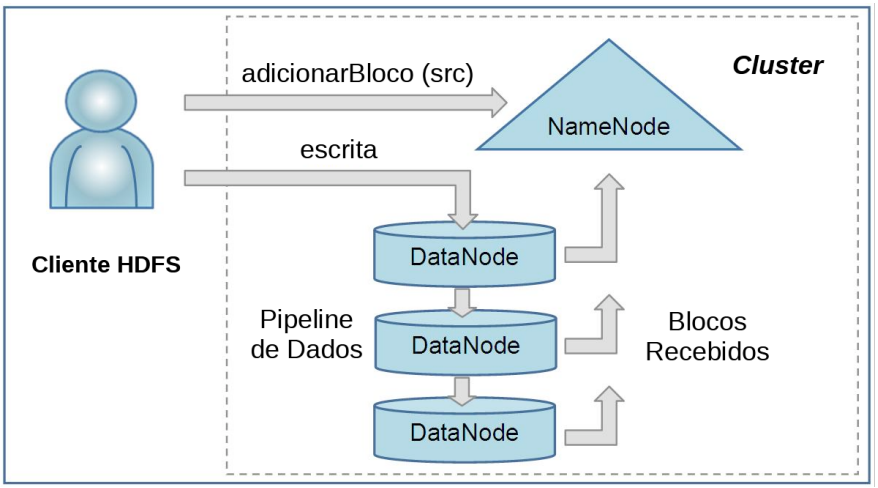

Figura 2.5: Arquitetura do HDFS por Shvachko et al. (2010)

O HDFS guarda o espaço de nomes inteiro na RAM. Os dados e a lista de blocos de cada arquivo são chamados de *image*. Uma cópia persistente do image é armazenado

no sistema de arquivos local do servidor é chamado de *checkpoint*. O *NameNode* armazena um registro de modificações chamado de *journal* no sistema de arquivos local. Para melhorar a durabilidade dos dados, cópias redundantes do checkpoint e *journal* podem ser feitas em outros servidores. No início o *NameNode* recupera o espaço de nomes lendo o espaço de nomes e carregando o *journal* (SHVACHKO et al., 2010).

2.4.7 Ceph

O sistema de arquivos Ceph apresenta uma arquitetura composta por: cliente; um *cluster* não confiável de dispositivos de armazenamento de objetos (**OSDs**, ou *Object Storage Devices*) que coleta e armazena todos os dados e metadados; e um *cluster* de servidores de metadados (**MDSs**, ou *Metadata Daemon Server*) que gerencia os nomes de arquivos e diretórios enquanto coordena a segurança, consistência e coerência. Uma visão geral da arquitetura do Ceph pode ser vista em 2.6.

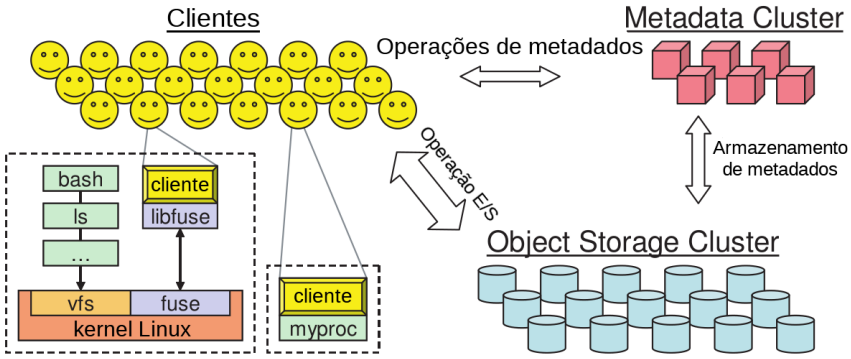

Figura 2.6: Arquitetura do Ceph por Weil et al. (2006)

O Ceph maximiza a separação entre os arquivos e seus metadados. Operações de metadados (abrir, renomear, etc.) são gerenciados pelo *cluster* de metadados, enquanto os clientes interagem diretamente com os OSDs nas operações de entrada e saída.

Nesse sistema, os objetos são colocados em *placement groups* (PGs) usando uma função de hash simples. Para localizar um objeto, é necessário apenas o PG e um mapa do cluster de OSDs.

Na questão da disponibilidade no Ceph, os dados são replicados em grupos. O cliente envia todos os dados para o primeiro OSD funcional em um objeto PG (o primário), que designa uma nova versão para o objeto e redireciona o dado a todos os OSDs de réplica. Depois que cada réplica aplicou a atualização e respondeu ao primário, o primário aplica a atualização local e avisa ao cliente que a operação foi um sucesso. Leituras são feitas diretamente ao primário.

Em caso de falhas, o Ceph manipula todas elas da mesma maneira. Para facilitar a recuperação, OSDs mantém a versão de cada objeto e um registro das mudanças recentes em cada PG. Se um servidor de um PG primário ficar indisponível, uma réplica assume como primário do PG. Quando o servidor primário inicial voltar, a réplica que assumiu envia a versão do objeto e o registro de mudanças, permitindo que o primário requisite as informações que faltam (WEIL et al., 2006).

2.4.8 Lustre

A arquitetura do Lustre é composta por: clientes; servidores de armazenamento de objetos (**OSSs**, ou*Object Storage Servers*), que provêm o serviço de entrada e saída; os servidores de metadados (**MDSs**, ou*Metadata Server*), que gerenciam os nomes e diretórios no sistema de arquivos. Uma visão geral da arquitetura pode ser vista em Figura 2.7.

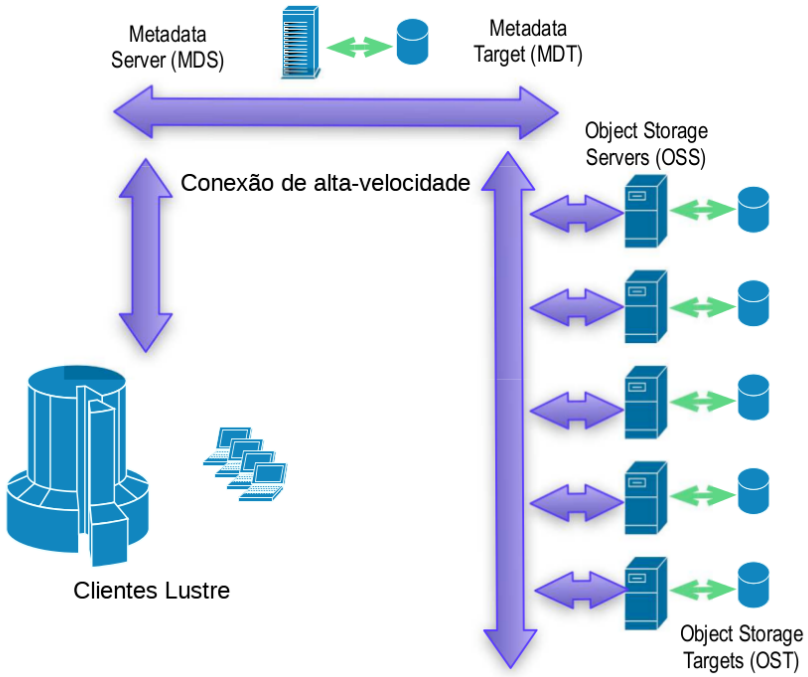

Figura 2.7: Arquitetura do Lustre por Wang et al. (2009)

Cada OSS podem gerenciar um ou mais OSTs (*Object Storage Target*, que são quem de fato armazenam o arquivo, enquanto cada MDS gerência um MDT (*Metadata Target* por vez, que também são quem de fato armazenam os metadados.

Todas as requisições no Lustre são feitas com requisições RPC. Antes de ler um arquivo, o cliente envia uma requisição ao MDS e é informado com qual OST ele

precisa se comunicar para executar a operação (WANG et al., 2009).

2.4.9 *Flexible and Adaptable Distributed File System* (FlexA)

O FlexA utiliza diversos módulos que se encarregam de identificar e administrar as comunicações entre os nós da rede. Nesse modelo, os computadores pertencentes ao SAD são administrados em grupos específicos, o qual podem ser de três tipos: grupo de escrita, grupo de clientes e um grupo de réplicas que não foi implementado no projeto original. Uma visão geral do sistema, incluindo a ideia original do grupo de réplicas, pode ser vista em 2.8.

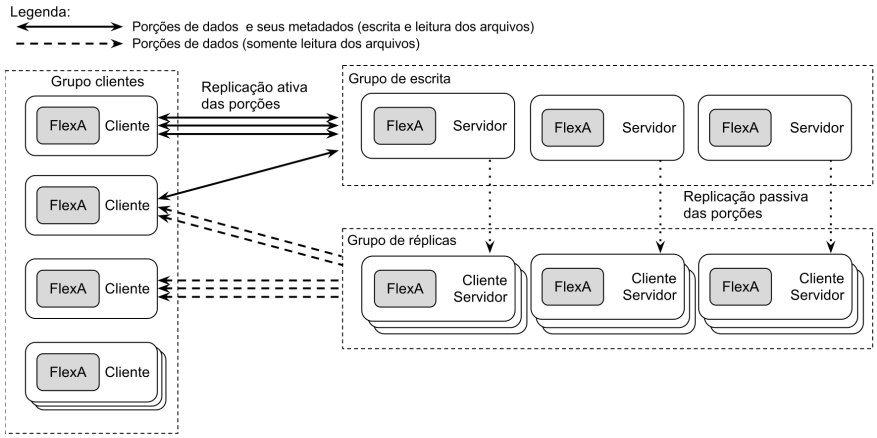

Figura 2.8: Arquitetura do FlexA por Fernandes (2012)

O grupo de escrita ou também chamados de servidores primários, compreende as estações com o processo servidor ativo, sendo responsáveis em administrar e armazenar os arquivos e seus metadados.

As porções são formadas a partir da divisão do arquivo criptografado em três partes, e cada um dos três servidores da versão original do sistema recebia duas dessas porções; caso um dos servidores caíssem ainda era possível recuperar essas informações, porém o sistema não permitia a recuperação em caso de falhas.

Processos mais custosos em termos de processamento como criptografia e divisão de arquivos foram movidos para o cliente, aliviando o consumo de recursos nos servidores.

O sistema tem como objetivo ser flexível e adaptável, onde por exemplo um cliente pode executar o processo servidor caso necessário. Ele funciona no espaço de usuário e tem o objetivo de executar em *hardware* de baixo custo (FERNANDES, 2012).

2.4.10 Comparativo entre os SADs analisados

Cada SAD estudado nesta Seção traz alguma particularidade que pode ser agregada para garantir a disponibilidade do sistema. Na Tabela 2.1 é mostrada uma comparação entre as características desses SADs.

Tabela 2.1: Comparação entre os SADs analisados

SADs	Arquitetura	Tolerância a Falhas	Garantia de Disponibilidade
NFS	Cliente-servidor	Sem replicação De arquivos	Ausente
AFS	Cliente-servidor	Uso de réplicas no Modo somente-leitura	Modo somente-leitura
Coda	Separação entre Metadados e dados	Replicação de porções Entre os servidores De armazenamento	Vários volumes de Replicação + modo Somente leitura (derivado Do AFS)
Tahoe-LAFS	Descentralizada	Distribuição de porções Do arquivo entre os nós De armazenamento	Vários servidores de Armazenamento (porém Apenas um *introducer*)
GFS	Separação entre Metadados e dados	Replicação de porções Entre os servidores De armazenamento	*Cluster* de armazenamento (porém apenas um Servidor mestre)
HDFS	Separação entre Metadados e dados	Replicação de porções Entre os servidores De armazenamento	*Cluster* de armazenamento (porém apenas um Servidor mestre)
Ceph	Separação entre Metadados e dados	Replicação de porções Entre os servidores De armazenamento	*Cluster* de armazenamento Tanto de dados como Metadados
Lustre	Separação entre Metadados e dados	Replicação de porções Entre os servidores De armazenamento	Vários servidores de Armazenamento (porém Apenas um de metadado)
FlexA	Descentralizada	Distribuição de porções Do arquivo entre os nós De armazenamento	Somente na leitura

2.5 Disponibilidade em SADs

Em sistemas de arquivos distribuídos, a disponibilidade faz com que na falha de um ou mais nós, outros nós estejam aptos a prover a mesma funcionalidade ao sistema (BZOCH; SAFARIK, 2011).

Com base nisso, o desafio é estimar a quantidade de falhas que podem ocorrer durante o período de ciclo de vida de um sistema, de tal maneira que os procedimentos possam a ser tomados de forma a garantir uma maior disponibilidade no sistema.

Nessa linha, Schroeder e Gibson (2006) citam um estudo realizado com vinte e dois sistemas de alto desempenho durante os anos de 1996 e 2005, totalizando a análise de funcionamento de 4750 servidores e 2410 processadores para a identificação de falhas em sistemas. A Tabela 2.2 apresenta os resultados analisados. Foram utilizadas letras em substituição ao nome de fabricante do *hardware*.

Tabela 2.2: Visão geral dos sistemas analisados por Schroeder e Gibson (2006)

(I) Informações do Sistema				(II) Informações por categoria			
HW	ID	N. Nós	N. Proc.	N. Proc./N. Nós	Tempo de produção	Mem.	NICs
A	1	1	8	8	ND – 12/99	16	0
B	2	1	32	32	ND – 12/03	8	1
C	3	1	4	4	ND – 04/03	1	0
D	4	164	328	2	04/01 – hoje	1	1
				2	12/02 – hoje	1	1
E	5	256	1024	4	12/01 – hoje	16	2
	6	128	512	4	09/01 – 01/02	16	2
	7	1024	4096	4	05/02 – hoje	8	2
				4	05/02 – hoje	16	2
				4	05/02 – hoje	32	2
				4	05/02 – hoje	352	2
	8	1024	4096	4	10/02 – hoje	8	2
				4	10/02 – hoje	16	2
				4	10/02 – hoje	32	2
	9	128	512	4	09/03 – hoje	4	1
	10	128	512	4	09/03 – hoje	4	1
	11	128	512	4	09/03 – hoje	4	1
	12	32	128	4	09/03 – hoje	4	1
				4	09/03 – hoje	16	1
F	13	128	256	2	09/03 – hoje	4	1
	14	256	512	2	09/03 – hoje	4	1
	15	256	5112	2	09/03 – hoje	4	1
	16	256	512	2	09/03 – hoje	4	1
	17	256	512	2	09/03 – hoje	4	1
	18	512	1024	2	09/03 – hoje	4	1
				2	03/05 – 06/05	4	1
G	19	16	2048	128	12/96 – 09/02	32	4
				128	12/96 – 09/02	64	4
	20	49	6152	128	01/97 – hoje	128	12
				128	01/97 – 11/05	32	12
				80	06/05 – hoje	80	0
	21	5	544	128	10/98 – 12/04	128	4
				32	01/98 – 12/04	16	4
				128	11/02 – hoje	64	4
				128	11/05 – 12/04	32	4
H	22	1	256	256	11/04 – hoje	1024	0

Legenda:	
HW:	Fabricante do *hardware*
ID:	Identificador do sistema
N. Nós	Número de nós
N. Proc.:	Número de processadores
Mem.:	Memória em GB
NICs:	Número de interfaces de rede
ND:	Não disponível

É importante a classificação das falhas que podem ocorrer em um sistema para análise posterior. Coulouris et al. (2011) e Tanenbaum e Steen (2007) classificam as falhas que podem ocorrer em um sistema distribuído como x, y, z. Já no estudo proposto por Schroeder e Gibson (2006), as falhas foram classificadas como falha humana, falha de ambiente, falha de rede, falha de *software* e falha desconhecida. O percentual de falhas em cada uma das categorias é apresentado na Figura 2.9.

Este mesmo estudo analisou a quantidade de falhas durante um ano para os sistemas A - H da Tabela 2.2. Após esse estudo do número de falhas gerou-se o Gráfico visto em 2.10.

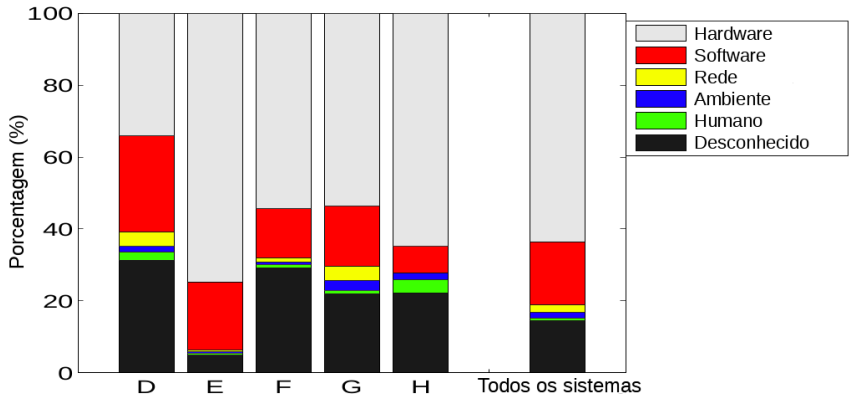

Figura 2.9: Principais motivos de falhas em sistemas distribuídos por Schroeder e Gibson (2006)

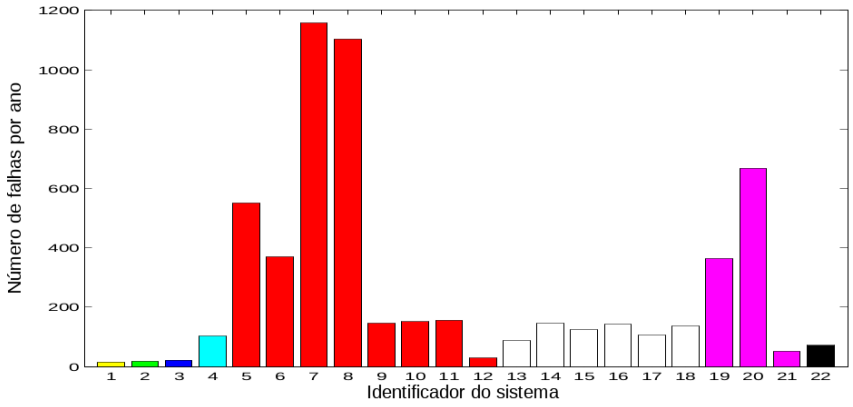

Figura 2.10: Número de falhas ocorridas em cada sistema em um ano por Schroeder e Gibson (2006)

Capítulo 3

Descrição e desenvolvimento do projeto

Após a fundamentação teórica apresentada no Capítulo 2, o Capítulo atual tem como objetivo descrever o trabalho desenvolvido a partir do estudo feito sobre disponibilidade em sistemas de arquivos distribuídos.

É apresentado a implementação dos servidores secundários no sistema de arquivos FlexA e depois os métodos usados para eleger um servidor secundário como primário, já que arquitetura original do FlexA (FERNANDES, 2012) citava, mas não especificava ou implementava o grupo de servidores secundários.

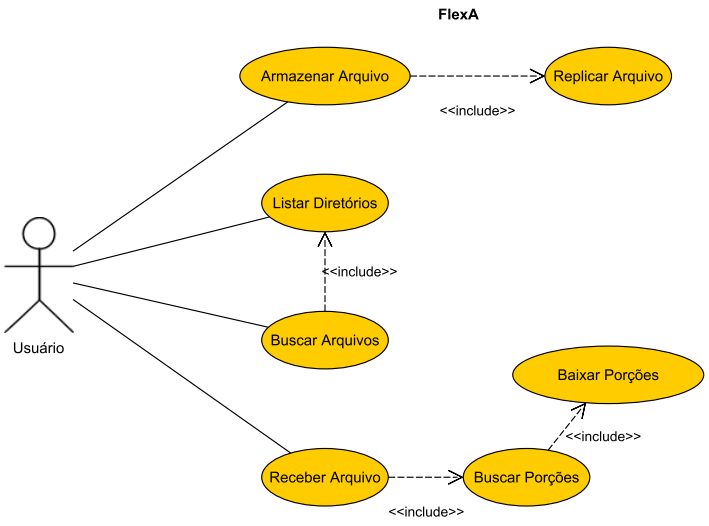

Figura 3.1: Diagrama de casos de uso da implementação proposta

3.1 Implementação do grupo de servidores secundários

É apresentado na Figura 3.1 o diagrama de casos de uso da implementação de servidores secundários para replicação de arquivos em um sistema de arquivos distribuídos.

Para implementar o grupo de servidores secundários, foram necessárias várias modificações nos servidores primários, apresentadas na Seção 3.1.1. Depois foi necessário implementar o módulo **coletor_replica**, que permite um nó da rede se tornar servidor secundário ao executá-lo; a implementação deste módulo é apresentada em detalhes na Seção 3.1.2. Por fim, algumas modificações menores foram feitas no módulo utilizado pelo cliente, apresentadas na Seção 3.1.3.

Nas Figura 3.2 é possível ver um resumo da arquitetura do FlexA após as modificações. Na nova arquitetura o cliente continua dividindo o arquivo em porções e enviando as porções junto com os metadados ao grupo de escrita (servidores primários). A diferença está nos servidores primários, que agregaram a sincronia de metadados relativo a localização das réplicas.

Os servidores primários também são responsáveis por replicar as porções ao grupo de leitura (servidores secundários), que está implementado de maneira diferente do que era citado em Fernandes (2012). Com as modificações outro cliente pode requisitar os metadados do arquivo a partir do grupo de escrita, e requisitar as porções tanto do grupo de escrita como do grupo de leitura.

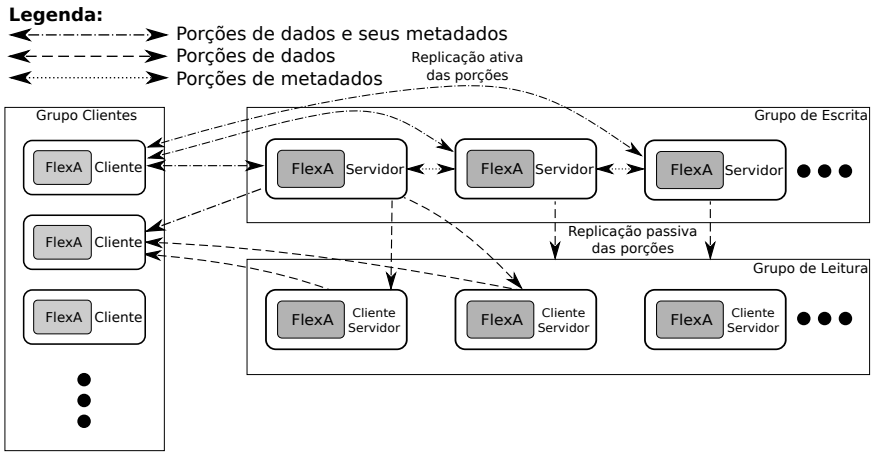

Figura 3.2: Arquitetura do FlexA após as modificações

Nas próximas seções uma descrição mais detalhada das modificações será feita.

3.1.1 Modificações nos servidores primários

O armazenamento de metadados e as operações de escrita ficaram centralizadas no grupo de servidores primários, de maneira que o cliente precisa solicitar (no caso de leitura) ou fornecer (no caso de escrita) os metadados ao grupo primário, além das porções do arquivo caso seja uma operação de escrita. Os metadados são constituídos pelas referências de localização de cada porção nos servidores primários e/ou réplicas.

Os servidores do grupo primário, por sua vez, sincronizam os metadados, garantindo que as referências de localização das porções estejam disponíveis para a solicitações de metadados dos clientes.

Para garantir que os clientes tenham transparência em relação às operações realizadas sobre os servidores secundários, toda a tarefa de localização de servidores secundários na rede e sua comunicação é também feita pelos servidores primários.

A cada 30 segundos (ou um tempo arbitrário definido pelo administrador do sistema) os servidores primários varrem a rede a procura de novos servidores secundários e armazenam isso em um arquivo. Quando ele recebe uma nova porção do cliente, ele usa este arquivo para localizar os servidores secundários disponíveis e envia a porção para os dois servidores secundários com mais recursos disponíveis (usando as métricas da Seção 3.2.2).

No momento em que o cliente realiza a requisição de metadados para uma operação de *download*, o primário envia a localização de todas as porções na rede, incluindo as porções encontradas nos servidores secundários.

De acordo com Fernandes (2012) o sistema de arquivos FlexA tem como um dos seus objetivos executar os servidores em *hardware* de baixo custo. As mensagens do cliente são retransmitidas sem alteração para os servidores secundários, o que evita o *overhead* gerado ao se criar uma mensagem, principalmente pelo fato que toda a tarefa de divisão de arquivos em porções e encriptação das mesmas continuam sendo feitas exclusivamente pelos clientes.

3.1.2 Módulo coletor_replica

Como mostrado em 2.4.9, os **módulos Coletores** são responsáveis pela distinção da função de cada nó na rede (ou seja, se ele é um servidor primário, secundário ou cliente). Assim o nó que estivesse executando o módulo coletor_réplica ficaria encarregado das funções de replicação de dados. Esse módulo não existia na versão original do FlexA, então foi necessário criá-lo.

A funcionalidade principal desse módulo é receber a porção e os metadados do respectivo arquivo dos servidores primários e armazená-los. Caso um cliente faça uma requisição de *download* da porção ao servidor secundário, ele se comporta semelhantemente ao servidor primário, devolvendo a porção junto com um cabeçalho com algumas informações do sistema.

Esse módulo foi desenvolvido a partir do módulo coletor_servidor, considerando que ambos os módulos têm a função de retornar os dados pedidos pelo cliente (apesar do servidor secundário não retornar metadados com as informações do arquivo). A semelhança entre os dois módulos é importante pois o servidor secundário poderá virar um servidor primário, como visto na Seção 2.3, além de garantir que o controle de acesso seja feito da mesma maneira atual, sem precisar de várias modificações no sistema.

Com a implementação do grupo de servidores secundários, eles continuam sendo usados de maneira opcional, mas passam a ser necessários para se ter redundância de arquivos no sistema, diferente da versão original do FlexA onde a redundância era garantida mesmo com apenas os servidores primários rodando. O grupo de servidores secundários também aumenta a tolerância a falhas no sistema de arquivos, algo que era previsto na arquitetura do FlexA mas não estava implementado na versão original do sistema (FERNANDES, 2012).

Caso sejam usados, é recomendado ter mais servidores secundários que servidores primários na rede, considerando que, para cada arquivo enviado, são geradas até seis réplicas das porções deste arquivo. Se existirem mais servidores primários que secundários isso pode gerar uma sobrecarga de requisições nos servidores secundários. Porém se existirem mais servidores secundários que primários, o algoritmo de balanceamento se encarrega em distribuir a carga igualmente, evitando a sobrecarga.

3.1.3 Modificação nos clientes

O cliente do FlexA fazia a divisão dos arquivos em porções de forma estática (daí a necessidade de no mínimo três servidores primários na versão original do FlexA). O sistema foi reestruturado para suportar um número variável de servidores primários, desde que existam no mínimo 3 para garantir o número de porções necessárias para o sistema funcionar. O servidor primário não é mais responsável pela redundância de arquivos (é enviado apenas uma porção para cada servidor primário, contra duas porções na versão original). A redundância de arquivos ficará a cargo do servidores secundários, conforme visto na seção 3.1.2.

Após a divisão dos arquivos em porções, cada porção do mesmo é encriptada usando uma chave AES no modo CBC (FERNANDES, 2012) em conjunto com sequências de *hash*. Uma mensagem então é gerada para cada uma das porções e enviada aos servidores primários.

Ao requisitar os metadados do arquivo, o cliente recebe a localização de todas as porções do arquivo na rede, incluindo as localizações dos servidores primários e secundários que contêm determinada porção. O cliente então poderá enviar uma segunda mensagem, pedindo o arquivo dessa vez, para tanto o servidor primário quanto o secundário, que devolverá a porção mais um cabeçalho com informações do sistema.

O sistema é bastante flexível, a ponto do cliente, na hora do *download*, conseguir baixar algumas porções de determinado arquivo dos servidores primários e outras porções dos servidores secundários. Além disso, o cliente pode se tornar um servidor primário ou secundário: basta ele executar, respectivamente, o módulo **coletor_servidor** ou **coletor_replica**.

3.2 Indicadores de disponibilidade na eleição de servidores secundários

A versão original do FlexA não era tolerante a falhas. Apesar de ser possível recuperar as informações no caso da falha de um servidor, o sistema não se recuperava automaticamente da falha. Em caso da falha de dois servidores, o sistema não conseguia se recuperar.

Uma solução para esse problema é eleger servidores secundários como servidores primários. Isso permite a recuperação do sistema em caso de falhas, além de permitir o funcionamento do sistema em caso de falhas de múltiplos servidores, pois com a eleição o sistema consegue manter o número de primários mínimos para garantir seu funcionamento.

A detecção de queda de um servidor primário é feita pelos próprios primários, que enviam uma mensagem a um dos servidores secundários para iniciar uma eleição. Como parâmetro da eleição é feito um cálculo de métricas combinando banda e espaço em disco. A eleição em si é uma modificação do algoritmo anel. Uma descrição mais detalhada é vista nas seções abaixo.

3.2.1 Verificando se o servidor está ativo

Para verificar se um servidor primário está ativo ou não, é necessário localizar os servidores primários que estão disponíveis para os clientes. Após isto, cada servidor faz requisições de *ping* de tempos em tempos, para verificar se os servidores vizinhos ainda estão ativos. Esse processo é chamado de sondagem de servidores.

Caso um servidor não responda a requisição, o servidor que detectou o problema envia uma mensagem a todos os servidores, informando da situação. Essa mensagem contém o IP do servidor atual mais o IP dos servidores que responderam a requisição de *ping* (o campo contendo o endereço do servidor inativo fica vazio nesse caso).

Ao receber a mensagem, o servidor vizinho compara com seu próprio resultado de requisição. Se ele perceber que o nó está ativo, não é necessário convocar uma eleição pois o servidor ainda está funcionando (provavelmente ocorreu um problema parcial na rede ou o sistema do servidor estava muito ocupado na hora da requisição). Caso contrário, é aguardado um tempo (por padrão 10 segundos) para garantir que

o servidor está inativo. Após esse tempo é feita outra requisição de *ping*, e caso o servidor não tenha respondido então a máquina realmente está inativa. Neste caso, um dos servidores ainda ativos envia uma mensagem a um dos servidores secundários para iniciar o processo eleição descrito em 3.2.3.

Um exemplo deste processo é apresentado na Figura 3.3. No caso o servidor com IP 192.168.1.1 está fora do ar e os servidores 192.168.1.2 e 192.168.1.3 estão sondando a rede. O servidor 192.168.1.3 detecta durante a sondagem que o servidor 192.168.1.1 está inativo (pois o retorno do comando *ping* é 0 em caso de sucesso; 1 caso o endereço foi encontrado e mas a máquina não respondeu; 2 no caso do endereço não ter sido encontrado), enviando uma mensagem ao resto do servidores ativos contendo o IP do servidor inativo, o que para a sondagem de servidores durante um tempo.

Após 10 segundos, temos a situação da Figura 3.4. O servidor envia uma requisição de *ping* mais uma vez ao servidor detectado como inativo. Se ele não responder o servidor 192.168.1.3 envia uma mensagem para avisar ao servidor secundário escolhido (nesse caso, 192.168.1.11) a iniciar a eleição.

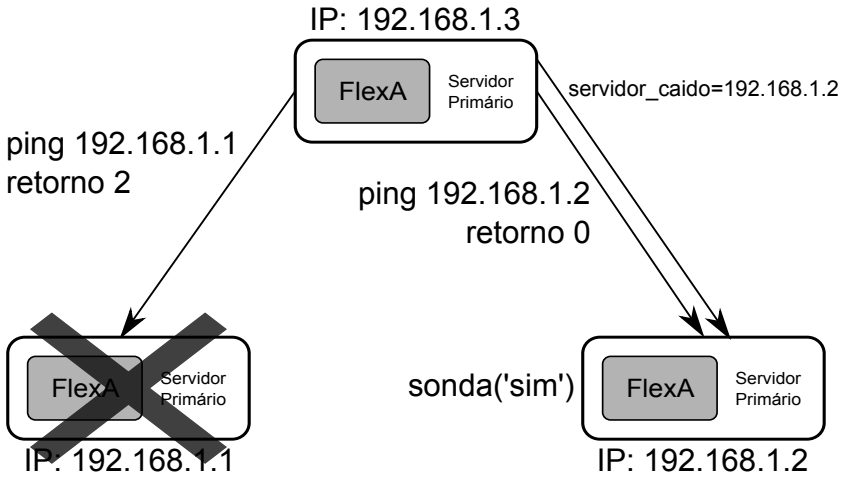

Figura 3.3: Envio de pings entre os servidores primários

3.2.2 Cálculo de métricas

Graças à arquitetura do FlexA, que executa os processos mais custosos em termos de processamento no cliente, os servidores do sistema tem um uso mais intenso de entrada/saída do que de CPU. Por isso, é importante eleger um servidor que esteja com menor uso de disco e banda do que um servidor com menor uso de CPU.

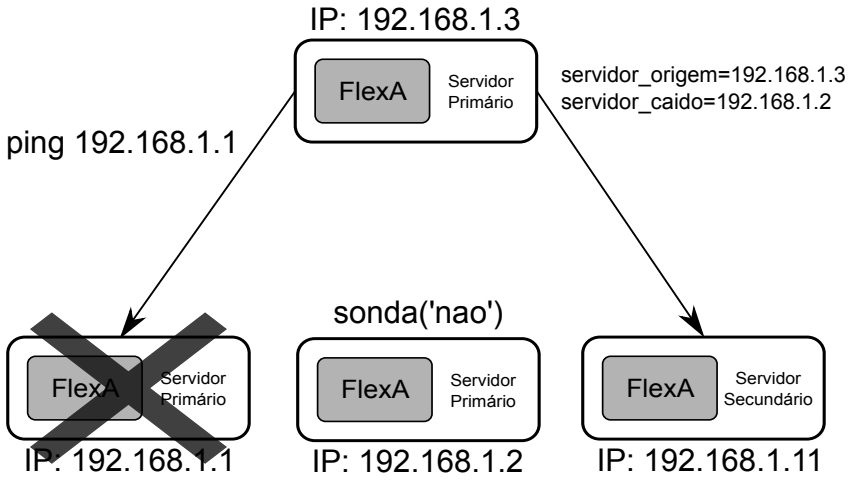

Figura 3.4: Envio de mensagem após detectar uma queda

A eleição é feita com a utilização de métricas para escolher o servidor secundário menos ocupado, já que ao ser eleito um secundário passará a ter mais responsabilidades.

A implementação desse cálculo envolveu duas partes. A primeira foi implementar módulos que obtêm o uso de disco e banda. Como a banda varia de acordo com o tempo, foi necessário implementar um *daemon* que obtém a quantidade de banda usada em um determinado período (por padrão 30 segundos).

Depois foi necessário normalizar os valores para eles poderem ser usados no cálculo. O valor conseguido de disco foi normalizado em relação ao espaço total de disco disponível, enquanto a banda foi normalizada em relação à banda total teórica da interface (por exemplo, redes 100Mbps tem 100Mbps de banda teórica). Com os valores normalizados uma métrica é calculada em determinado instante no nó, usando a Equação 3.1.

$$recursos = 1 - (\frac{disco}{3} + \frac{download}{3} + \frac{upload}{3}) \tag{3.1}$$

onde:

$$0 \leq disco \leq 1$$

$$0 \leq download \leq 1$$

$$0 \leq upload \leq 1$$

$$0 \leq recursos \leq 1$$

Para evitar eleger uma máquina com a rede sobrecarregada ou com pouco espaço em disco disponível, é usado como limite superior o valor de 0,8 para cada um dos parâmetros usados para o cálculo de métricas. Caso qualquer um dos parâmetros seja maior que esse valor, a métrica resultante é 0 (zero).

3.2.3 Tornando um servidor secundário em primário

Para fazer a eleição do sistema, é necessário implementar um algoritmo de eleição. O algoritmo usado foi uma modificação do algoritmo anel citado no Capítulo 2. Ele foi escolhido pois os nós no FlexA já são naturalmente organizados em um anel considerando que cada servidor tem seu IP na rede (o IP 192.168.1.2 é claramente menor que o IP 192.168.1.3).

Apesar do algoritmo anel enviar sempre $2(n-1)$ mensagens, o algoritmo usado nessa implementação faz o nó que iniciou a eleição enviar diretamente a mensagem que o nó foi eleito ao invés de passar pelo anel outra vez, enviando a mensagem que determinado nó ganhou a eleição. Isto não é necessário já que quem indica que um nó está atuando em determinada função é o processo coletor que ele está rodando Fernandes (2012). Isso reduz o número de envio de mensagens para $n+1$.

Após receber uma mensagem de eleição de um servidor primário, o servidor secundário inicia o processo de eleição, que ocorre da seguinte maneira:

1. O primeiro nó da eleição adiciona suas métricas descritas na Seção 3.2.2 à mensagem e envia para o próximo nó da eleição, que é determinado pelo IP. Por exemplo, se o nó atual tem o IP 192.168.1.15 e existe um IP 192.168.1.16, este será o próximo nó da eleição. Caso ele seja o último IP e a eleição não tenha terminado ainda, a mensagem é enviada para o nó com o primeiro IP da rede.

2. O nó que recebeu a mensagem calcula suas próprias métricas e compara com as que ele recebeu. Se suas métricas forem melhores ele adiciona suas métricas à mensagem (substituindo as anteriores) e envia. Caso contrário ele envia a mensagem do jeito que recebeu.

3. Isso continua até a mensagem voltar ao primeiro nó da eleição, que verifica o vencedor e envia uma mensagem para o nó eleito avisando que ele venceu.

Um esquema da eleição é apresentado na Figura 3.5.

Após o processo de eleição o servidor secundário eleito precisa sincronizar os seus metadados com o resto dos servidores primários para que ele consiga atuar como primário. O nó eleito então envia uma mensagem ao servidor que originou a eleição solicitando o seu banco de dados. O servidor primário retorna o seu banco de dados junto com uma soma de verificação MD5 para garantir a consistência dos dados. Não

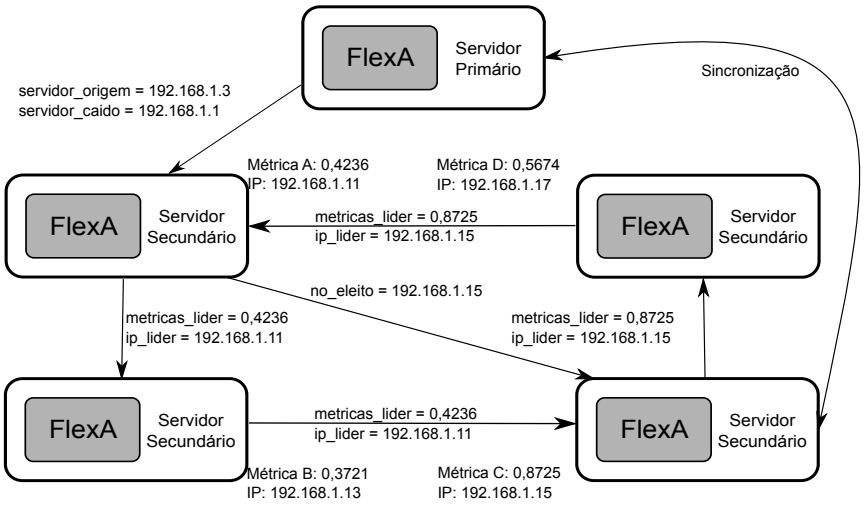

Figura 3.5: Esquema de eleição em anel

há a necessidade de sincronizar as porções de dados faltantes, já que existem réplicas das porções nos servidores secundários e isso evita um grande uso da rede após a eleição.

Com o banco de dados do primário o servidor eleito inicia o processo **coletor_servidor**, sem deixar de servir suas funções de servidor secundário, já que se ele simplesmente parasse de servir suas funções de replicação o sistema poderia ficar em um estado inconsistente.

A responsabilidade de achar o novo nó primário da rede fica por conta de um *daemon* que executa a busca de novos nós de tempos em tempos.

3.3 Índice de disponibilidade no sistema

A disponibilidade de um SAD como o FlexA está condicionado ao tempo em que o sistema fica disponível mesmo após a ocorrência de falhas. Um sistema rodando o FlexA deve possuir no mínimo 3 servidores primários como mostrado na Seção 3.1.3. Não há número mínimo de servidores secundários, mas é recomendado ter mais servidores secundários que primários para o sistema funcionar bem, como mostrado na Seção 3.1.2. Para este cálculo vamos assumir 5 servidores secundários.

Para determinar um índice de disponibilidade no sistema estudado, é necessário analisar os tipos de falhas que podem ocorrer em determinadas situações. Nas situações de leitura, considerando falhas aleatórias, o índice de disponibilidade chega nos 100% uma vez que, se um servidor primário ficar indisponível, outros poderão responder com a localização dos arquivos; mesmo na queda de um servidor secundário, outros também poderão oferecer essa porção.

No caso da escrita, caso aconteça uma queda de um servidor primário, o sistema ficará indisponível para essa operação até um servidor secundário se tornar primário (assumindo que o sistema está sendo usado com o número mínimo de nós primários). As características do FlexA se assemelham mais ao sistema D apresentado em Schroeder e Gibson (2006). Um cálculo de forma proporcional resulta em uma média de 3,9 quedas de servidores por ano relacionados a um sistema rodando o FlexA.

Como é mostrado no Capítulo 4, o tempo médio de eleição no sistema é 13,01577 segundos. Se o sistema falha com uma frequência de 3,9 vezes em um ano, considerando que todas essas quedas são de servidores primários, temos que o sistema fica **50,76148453** segundos fora do ar em um ano. Ou seja, seu índice de disponibilidade seria de **0,999998391** ou **>99,999%**.

Capítulo 4

Testes e validação da implementação proposta

Este capítulo apresenta uma descrição dos testes e resultados obtidos com o objetivo de comparar o desempenho do sistema proposto com sistemas de arquivos distribuídos existentes. Além disso, são apresentados testes e resultados com o objetivo de validar a implementação proposta no Capítulo 3.

4.1 Testes de desempenho do sistema

Neste experimento foi medido o desempenho do sistema de arquivos FlexA comparado com os sistemas de arquivos NFS e Tahoe-LAFS, além de ter sido realizada uma comparação com a versão anterior do FlexA sem os servidores secundários, para verificar o impacto no desempenho das modificações feitas no sistema.

Os testes foram feitos simulando 1, 2, 4, 8, 16 e 32 clientes em cada sistema, além de 8 nós servidores nos sistemas de arquivos que suportam mais de um nó com essa funcionalidade.

4.1.1 Ambiente de teste e considerações iniciais

Para a realização de tais testes foi utilizado o *cluster Beowulf* instalado no laboratório do Grupo de Sistemas Paralelos e Distribuídos (GSPD) com as configurações abaixo:

- Nós novos: 8 nós com a seguinte configuração:
  - Processador Intel®Core ™i7-3770 CPU de 4 núcleos e 8 *thread* @ 3,40GHz;
  - Memória RAM de 16GB;
  - HDD de 500GB 7200RPM SATA II;

  - Rede Gigabit Ethernet (1000 Mbps) ;
  - Sistema Operacional Debian GNU/Linux 7.1.0 e Linux kernel versão 3.2.0-4-amd64.

- Nós antigos: 8 nós com a seguinte configuração:
  - Processador Intel®Pentium®Dual CPU E2160 @ 1,8Ghz;
  - Memória RAM de 2GB;
  - HDD de 40GB 7200RPM IDE;
  - Rede Gigabit Ethernet (1000 Mbps);
  - Sistema Operacional Debian GNU/Linux 7.1.0 e Linux kernel versão 3.2.0-4-amd64.

Para simular mais clientes foi utilizado o programa de virtualização VirtualBox versão 4.1.18 rodando nos novos nós do *cluster* e com a seguinte configuração em cada máquina virtual:

- Um processador;
- 1GB de RAM;
- HDD de 10GB
- Rede Gigabit Ethernet (1000 Mbps);
- Sistema Operacional Debian GNU/Linux 7.1.0 e Linux kernel versão 3.2.0-4-amd64.

Cada nó do novo *cluster* rodou 4 máquinas virtuais como configuradas acima, resultando nos 32 clientes necessários para os testes, enquanto os nós do *cluster* antigo rodavam os servidores dos respectivos sistemas de arquivos a serem testados.

Entre cada escrita e leitura de um arquivo foi colocado uma pausa em segundos de tempo aleatório conseguido a partir de uma distribuição normal com parâmetros $\mu = 15$ e $\sigma = 3$. Isso serve para simular melhor uma situação real de uso, onde os clientes enviam seus arquivos em tempos aleatórios e não necessariamente em rajada (*burst*).

Os tamanhos dos arquivos usados nos testes foram 1MB, 5MB, 10MB, 25MB, 50MB e 100MB. Cada teste foi repetido 20 vezes. Após isso a média e o desvio padrão foram calculados. Para obter os valores de vazão o tamanho do arquivo foi dividido pela média dos tempos obtidos, o que resultou em valores em MB/s.

Os sistemas de arquivos testados e as suas respectivas configurações são apresentadas a seguir:

- FlexA com as modificações propostas: 3 servidores primários e 5 servidores secundários
- FlexA original: 3 servidores primários
- Tahoe-LAFS: 1 *introducer* e 7 servidores de armazenamento (com um deles atuando como *helper*)
- NFS: 1 servidor

Nos sistemas que suportavam alguma configuração com oito servidores esse número foi usado. Os sistemas que não suportam oito servidores foram testados com o maior número de servidores possíveis: a versão original do FlexA usa três servidores fixos, e o NFSv4 suporta apenas um servidor. Com exceção das modificações feitas para o uso de oito servidores, as configurações usadas em cada SAD foram as padrões.

A versão original do FlexA não completou com sucesso o teste com 32 clientes, por isso os respectivos resultados não foram apresentados. Foram encontrados diversos problemas que impossibilitaram a execução dos testes com este número de clientes; vários destes problemas foram corrigidos durante o desenvolvimento do trabalho.

Os testes de leitura feitos no NFS foram feitos realizando uma operação de desmontagem (*umount*) e montagem (*mount*) da partição usada nos testes. Isso foi feito para prevenir a geração do *cache* de leitura do NFS, o que diminuía consideravelmente o tempo das operações seguintes de leitura. Foi utilizada a versão 4 do NFS nos testes, usando a implementação de servidor disponível no kernel Linux 3.2.0-4 do Debian GNU/Linux 7.

A versão usada do Tahoe-LAFS foi a 1.9.2-1, disponível nos repositórios do Debian GNU/Linux 7.

Os resultados individuais de desempenho de cada sistema de arquivos podem ser vistos no Apêndice A.

4.1.2 Comparação de desempenho

Começando com os testes com apenas um cliente (Figuras 4.1 e 4.2), o desempenho do FlexA original e o FlexA com as modificações é semelhante, o que mostra que mesmo após as modificações não foi adicionado um *overhead* significativo no sistema. Além disso, a arquitetura usando vários servidores é interessante para operações de escrita, onde o FlexA apresenta desempenho superior aos outros sistemas testados (uma vantagem que aumenta nos testes posteriores).

O fato de ser necessário baixar o arquivo, juntar e desencriptar antes de poder ler é provavelmente o fato que faz o FlexA ser mais lento que o NFS em situações com poucos clientes (no NFS basta ler o arquivo diretamente), mas o FlexA foi consistentemente mais rápido que o Tahoe-LAFS que também usa mais de um servidor.

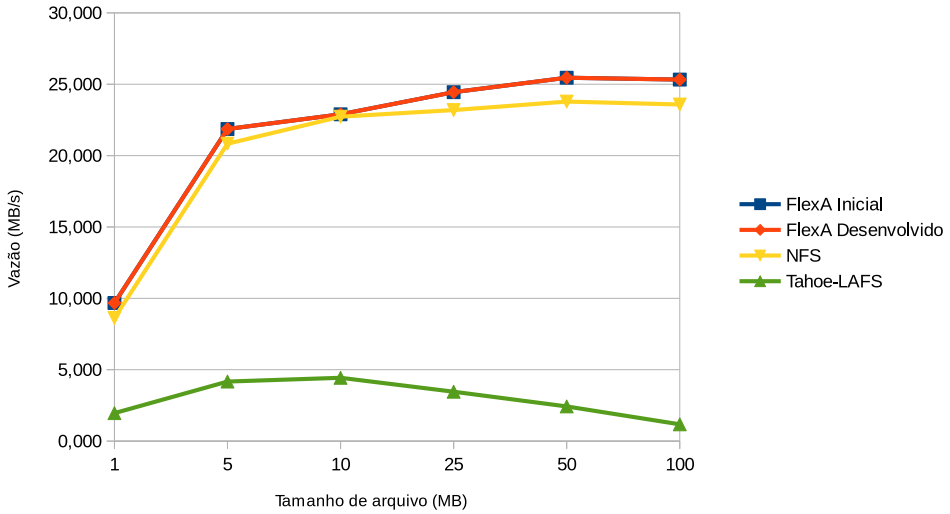

Figura 4.1: Comparativo de vazão na escrita com 1 cliente

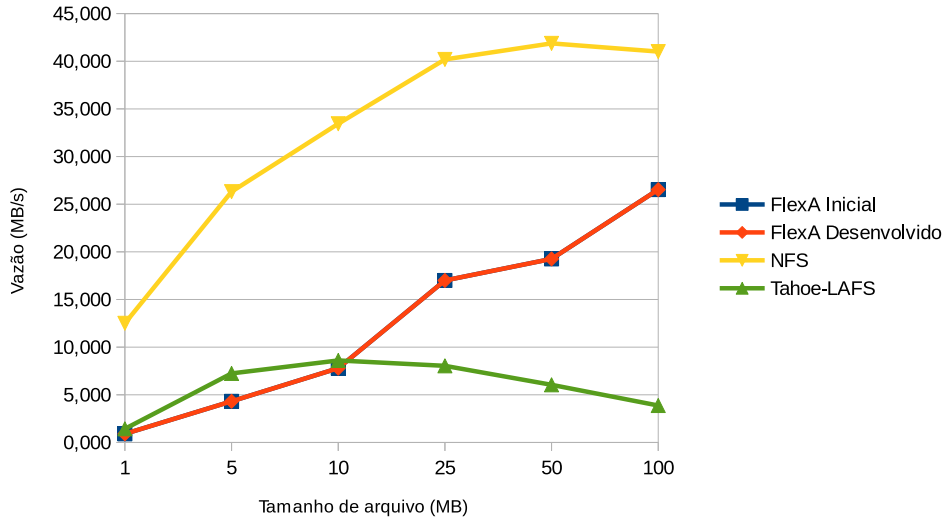

Figura 4.2: Comparativo de vazão na leitura com 1 cliente

Com dois e quatro clientes (Figuras 4.3, 4.4, 4.5 e 4.6) o FlexA com modificações é mais rápido que a versão original nas operações de escrita de arquivos maiores e praticamente igual nas operações de leitura. Isso provavelmente ocorreu por causa das mudanças que foram feitas para melhorar o desempenho do sistema. O NFS ainda

tem a melhores taxas de leitura, mais uma vez provavelmente por não necessitar fazer nenhuma operação prévia antes de ler um arquivo. O Tahoe-LAFS continua em último.

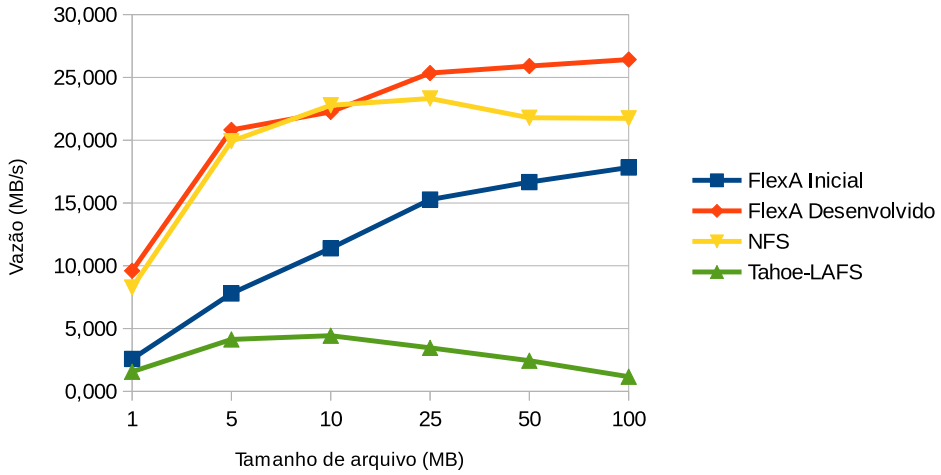

Figura 4.3: Comparativo de vazão na escrita com 2 clientes

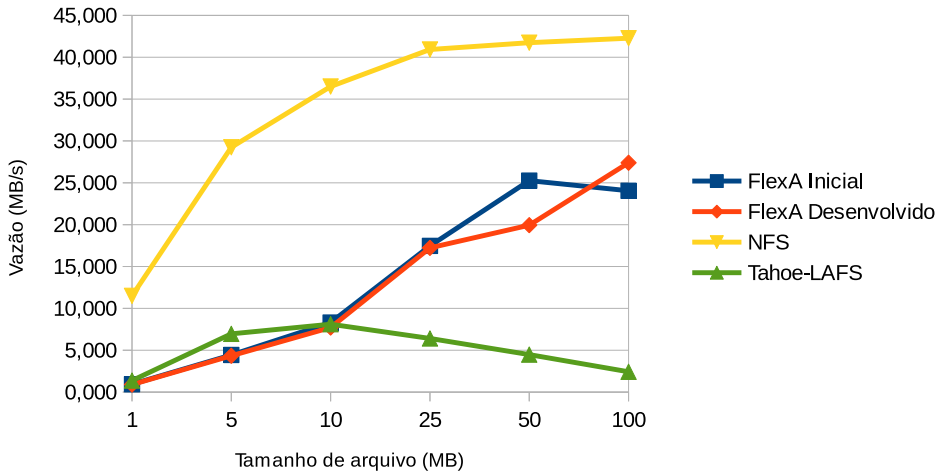

Figura 4.4: Comparativo de vazão na leitura com 2 clientes

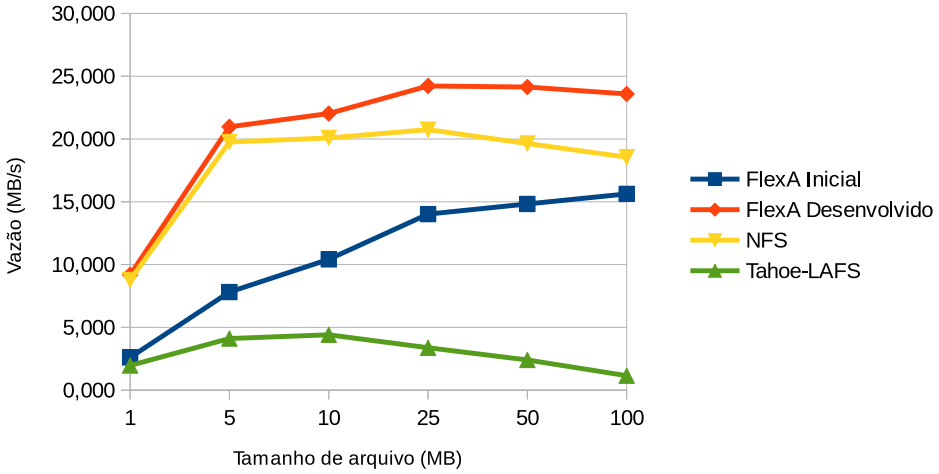

Figura 4.5: Comparativo de vazão na escrita com 4 clientes

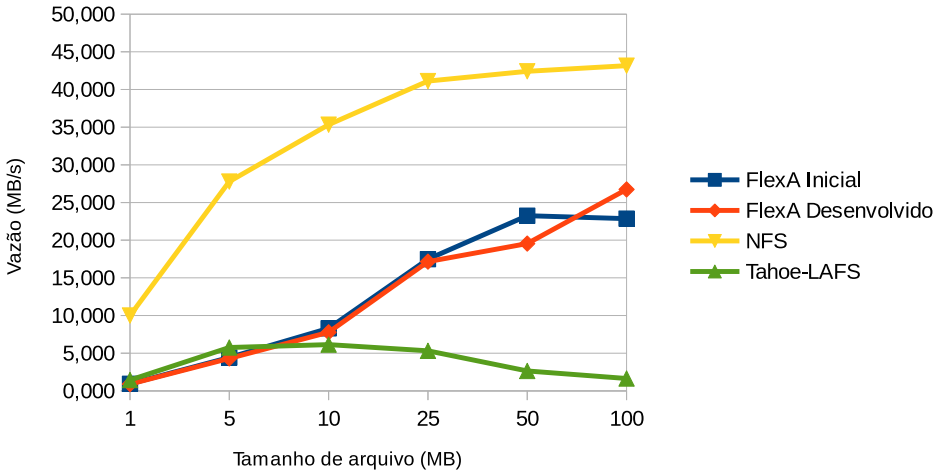

Figura 4.6: Comparativo de vazão na leitura com 4 clientes

No teste com oito e dezesseis clientes (Figuras 4.7, 4.8, 4.9 e 4.10) o desempenho de escrita do NFS para arquivos maiores caiu bastante, provavelmente por causa do número de clientes tentando escrever no mesmo servidor ao mesmo tempo, porém o NFS continua sendo o mais rápido em operações de leitura.

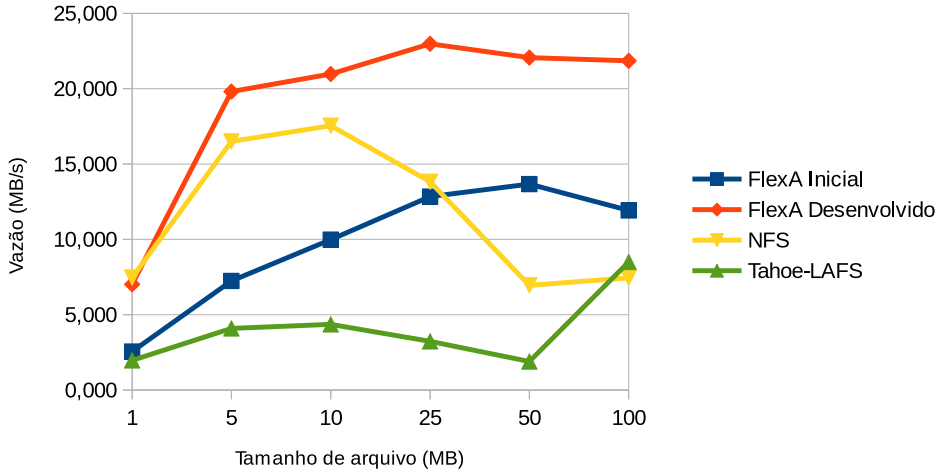

Figura 4.7: Comparativo de vazão na escrita com 8 clientes

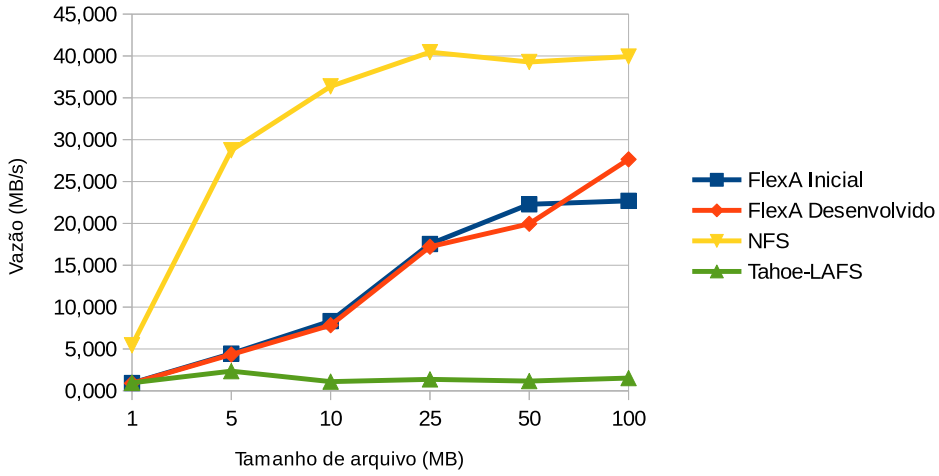

Figura 4.8: Comparativo de vazão na leitura com 8 clientes

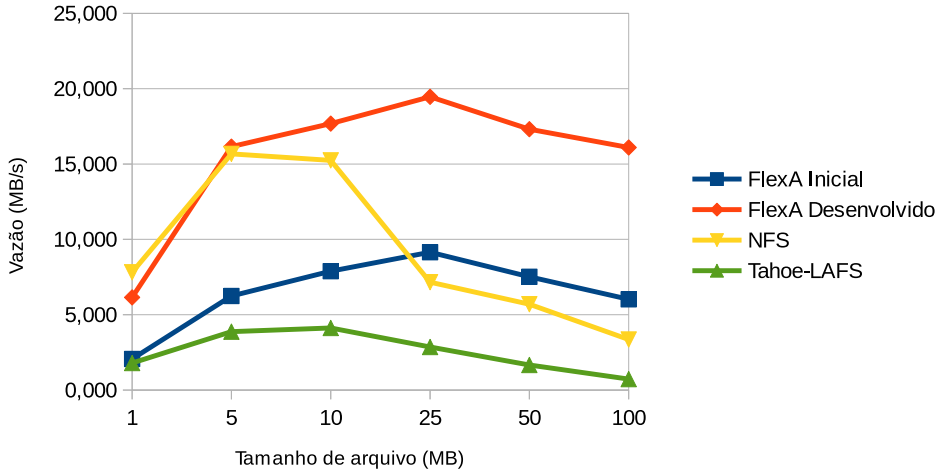

Figura 4.9: Comparativo de vazão na escrita com 16 clientes

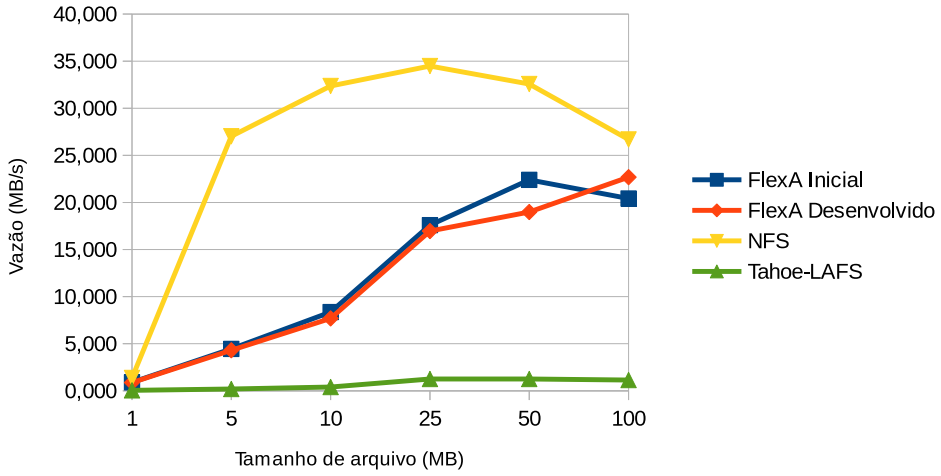

Figura 4.10: Comparativo de vazão na leitura com 16 clientes

No último teste com 32 clientes (Figuras 4.11 e 4.12), o FlexA começou a ganhar do NFS mesmo em situações de leitura com arquivos maiores, o que mostra a vantagem da arquitetura descentralizada do FlexA.

O NFS ganhou na escrita com arquivos menores, provavelmente por ele terminar rapidamente a escrita dos arquivos enquanto os outros sistemas perderam bastante

tempo tentando escrever com tantos clientes, porém com arquivos maiores o FlexA modificado mais uma vez é o vencedor.

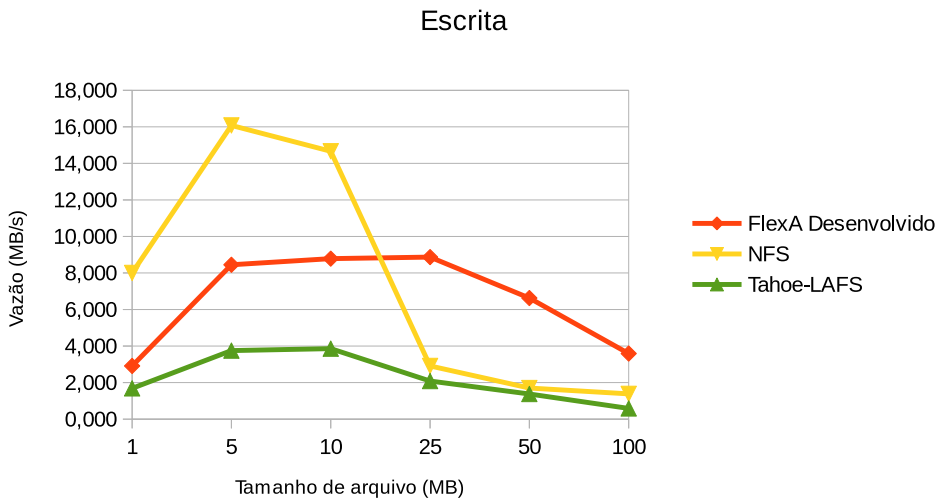

Figura 4.11: Comparativo de vazão na escrita com 32 clientes

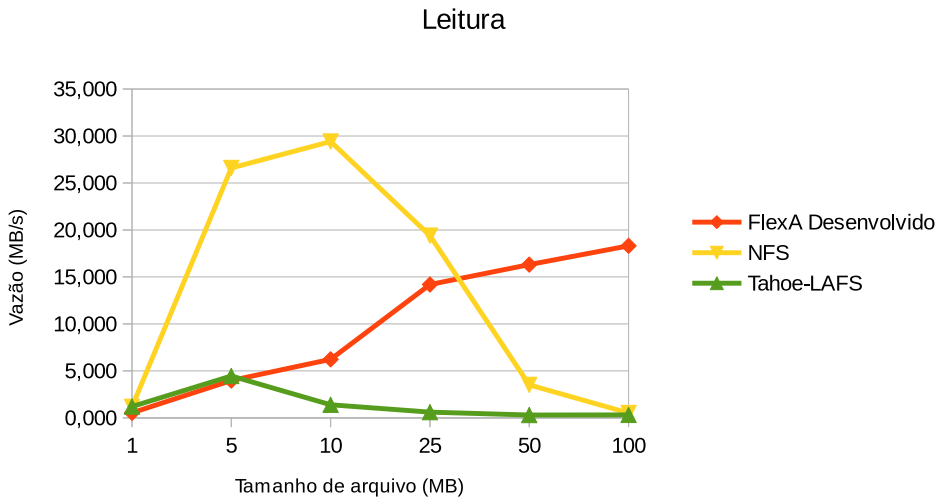

Figura 4.12: Comparativo de vazão na leitura com 32 clientes

4.2 Validação e desempenho da implementação

Nesta Seção serão apresentadas os testes de validação das funcionalidades implementadas no Capítulo 3 no sistema de arquivos FlexA.

Para este exemplo foram configurados três servidores primários, quatro servidores secundários e um cliente como mostrado no Apêndice B. Os servidores foram configurados dentro de uma máquina virtual no programa VirtualBox, enquanto o cliente foi executado diretamente na máquina *host* utilizada nos testes. Foram usadas as configurações abaixo:

- Máquina *host*:
  - Processador Intel®Core ™i7-3610QM CPU de 4 núcleos e 8 *thread* @ 2,30GHz;
  - Memória RAM de 8GB;
  - HDD de 1TB 7200RPM SATA II;
  - Rede Gigabit Ethernet (1000 Mbps);
  - Sistema Operacional Arch Linux e o Linux kernel versão 3.12.1-ARCH;
  - VirtualBox versão 4.3.2.
- Máquinas virtuais:
  - Um processador;
  - 1GB de RAM;
  - HDD de 10GB;
  - Rede Gigabit Ethernet (1000 Mbps);
  - Sistema Operacional Debian GNU/Linux 7.1.0 e Linux kernel versão 3.2.0-4-amd64.

A validação da implementação foi feita simulando uma queda num servidor primário a cada minuto. O servidor primário ficava fora do ar durante 30 segundos antes de voltar para o próximo teste. Nesse período era verificado se os outros servidores primários conseguiram detectar a queda do servidor, e se os servidores secundários conseguiam eleger com sucesso um novo servidor primário. Este teste foi repetido 200 vezes, e além de verificar se um servidor foi eleito com sucesso ou não. Depois foram recolhidos os tempos de execução para analisar o desempenho da implementação.

Os resultados dos testes são apresentados na Tabela 4.1. É possível observar que o algoritmo de eleição e o processo de sincronização após a eleição de um servidor secundário são desprezíveis, e que o processo de sondagem dos servidores ativos é o que toma mais tempo do sistema.

Esse tempo de sondagem é explicado pois o sistema espera até 10 segundos para o servidor que caiu voltar, como explicado no Capítulo 3.

Tabela 4.1: Estatísticas sobre recuperação de falhas no primário

Sondagem	
Número de quedas	200
Número de vezes que uma queda foi detectada	200
Porcentagem de sucesso	100,00%
Eleição	
Número de vezes que foi requisitada uma eleição	200
Número de vezes que o servidor secundário 1 iniciou a eleição	51
Número de vezes que o servidor secundário 2 iniciou a eleição	48
Número de vezes que o servidor secundário 3 iniciou a eleição	50
Número de vezes que o servidor secundário 4 iniciou a eleição	51
Total	200
Porcentagem de sucesso	100,00%
Sincronização	
Número de vezes que foi requisitada uma sincronização	200
Número de vezes que o servidor secundário 1 foi eleito	59
Número de vezes que o servidor secundário 2 foi eleito	69
Número de vezes que o servidor secundário 3 foi eleito	0
Número de vezes que o servidor secundário 4 foi eleito	72
Total	200
Porcentagem de sucesso	100,00%

Na Tabela 4.2 é mostrado o desempenho da implementação, com os tempos médios individuais de sondagem, eleição e sincronização de dados.

Tabela 4.2: Tempo médio em segundos de sondagem, eleição e sincronização

	Sondagem	Eleição	Sincronização	Total
Média	13,00773	0,00019	0,00784	13,01577
Desvio padrão	0,00176	0,00006	0,05466	1,05539

O número de clientes pouco influencia no algoritmo de eleição. Em outro teste utilizando as máquinas virtuais do *cluster* visto na Seção 4.1.1, foi feito um teste com

20 repetições e com um número variável de servidores secundários (1, 2, 4, 8, 16 e 32). O resultado deste teste é apresentado na Tabela 4.3.

O tempo de sincronização pode ser afetado pelo tamanho do arquivo de banco de dados a ser sincronizado, porém esse arquivo é normalmente pequeno (na ordem de KB). Considerando os testes de desempenho de escrita nos arquivos (onde são usados arquivos bem maiores), é de se esperar que normalmente o banco de dados não vai influenciar muito no tempo total do processo em tornar primário um servidor secundário.

Tabela 4.3: Tempos médios em milisegundos de eleição com número variável de servidores secundários

Servidores	Tempo Médio (s)	Desvio Padrão
1	0,0954151	0,0548357
2	0,0793338	0,0246791
4	0,0693560	0,0035235
8	0,0693560	0,0035235
16	0,0759721	0,0093576
32	0,1221180	0,1949884

Capítulo 5

Conclusões

Este texto apresentou uma visão geral sobre sistemas de arquivos distribuídos, seus problemas e possíveis soluções, focando principalmente na tolerância a falhas e garantia de disponibilidade. Também foi apresentado o sistema de arquivos distribuído e adaptável FlexA, discutindo possíveis alterações que permitissem tornar o sistema mais robusto quanto a falhas de modo a garantir a disponibilidade do sistema.

5.1 Considerações finais

O sistema FlexA têm como intuito principal permitir a criação de um sistema de arquivos distribuído e adaptável usando o espaço de usuário e que use hardware de baixo custo, seja escalável, tolerante a falhas e disponível. Neste contexto, este trabalho contribuiu para a melhoria dessas características no sistema. As principais conclusões que podem ser tiradas desse trabalho são:

- Aumento disponibilidade do sistema pela implementação de um grupo de servidores secundários, o que permite um usuário recuperar um arquivo mesmo em casos de queda de um servidor primário;
- Aumento da tolerância a falhas do sistema por permitir que um servidor secundário seja eleito caso seja detectado uma queda de um servidor primário;
- Os servidores secundários não necessitam de hardware especial ou de grandes recursos computacionais, já que o eles não fazem tarefas de uso computacional intenso como criptografia e divisão de arquivos;
- O sistema se tornou mais escalável por permitir um número qualquer de servidores primários e secundários, ou seja, se for necessário suportar mais clientes basta adicionar mais nós na rede.

5.2 Problemas encontrados

Dentre os problemas encontrados durante a realização deste trabalho destacam-se principalmente:

- A falta de estruturação do código original dificultou a implementação dos novos recursos;
- Problemas de desempenho causados pela escolha do Banco de Dados SQLite3 e a forma que os *sockets* foram implementados na versão original;
- Diversas dificuldades encontradas durante a execução dos testes, entre elas: falha no roteador usado no *cluster*, desligamento arbitrário das máquinas, problemas nas configurações padrões usadas em alguns SADs, etc.

5.3 Trabalhos futuros

Dentre as possíveis direções que este trabalho pode tomar no futuro, destacam-se as seguintes atividades:

- modificação do processo de eleição, para permitir que o servidor secundário eleito abdique das suas funções de primário caso o servidor primário que caiu volte a funcionar;
- modificação das métricas para que elas passem a considerar outros parâmetros, como a memória RAM disponível ou número de clientes conectados no servidor;
- permitir que um servidor secundário seja eleito também em situações em que os servidores primário estão com sobrecarga.

5.4 Publicações

O trabalho gerou uma série de publicações de iniciação científica. Além disso, o trabalho está incluso dentro de um projeto mais amplo de desenvolvimento do sistema de arquivos FlexA e isso gerou outra publicação, como pode ser visto abaixo:

1. *FlexA - Grupo de Réplicas em um Sistema de Arquivos Distribuídos*. Apresentado na IV Escola Regional de Alto Desempenho de São Paulo (ERAD-SP 2013) (OKADA et al., 2013);
2. *Desenvolvimento de um Sistema de Arquivos Distribuído Adaptável - Disponibilidade de Servidores*. Apresentado no XXV Congresso de Iniciação Científica da UNESP (CIC 2013) (OKADA; LOBATO; MANACERO, 2013a);

3. *Disponibilidade de Servidores em um Sistema de Arquivos Distribuído e Adaptável*. Apresentado no XXI Simpósio Internacional de Iniciação Científica da USP (SIICUSP 2013) (OKADA; LOBATO; MANACERO, 2013b).

Referências Bibliográficas

BZOCH, P.; SAFARIK, J. State of the art in distributed file systems: Increasing performance. In: *Proceedings of the 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems*. Washington, DC, USA: IEEE Computer Society, 2011. (ECBS-EERC '11), p. 153–154. ISBN 978-0-7695-4418-2. Disponível em: <http://dx.doi.org/10.1109/ECBS-EERC.2011.34>.

COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2011. (International computer science series). ISBN 9780132143011. Disponível em: <http://books.google.com.br/books?id=GmYpKQEACAAJ>.

FERNANDES, S. E. N. *Sistema de Arquivos Distribuído Flexível e Adaptável*. Dissertação (Mestrado) — Universidade Estadual Paulista, 2012. Disponível em: <http://www.dcce.ibilce.unesp.br/spd/pubs/dissertacaoSilas.pdf>.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2003. (SOSP '03), p. 29–43. ISBN 1-58113-757-5. Disponível em: <http://doi.acm.org/10.1145/945445.945450>.

JHAVERI, H. J.; SHAH, S. A comparative analysis of election algorithm in distributed systems. *Special issues on IP Multimedia Communications*, n. 1, p. 84–87, October 2011. Published by Foundation of Computer Science, New York, USA. Disponível em: <http://www.ijcaonline.org/specialissues/ipmc/number1/3755-ipmc019>.

LISKOV, B. et al. A replicated unix file system. In: *Proceedings of the 4th Workshop on ACM SIGOPS European Workshop*. New York, NY, USA: ACM, 1990. (EW 4), p. 1–8. Disponível em: <http://doi.acm.org/10.1145/504136.504140>.

OKADA, T. K.; LOBATO, R. S.; MANACERO, A. Desenvolvimento de um sistema de arquivos distribuído adaptável - disponibilidade de servidores. In: UNIVERSIDADE ESTADUAL DE SÃO PAULO - CAMPUS DE SÃO JOSÉ DO RIO PRETO. *XXV Congresso de Iniciação Científica da UNESP*. [S.l.], 2013. p. 1.

OKADA, T. K.; LOBATO, R. S.; MANACERO, A. Disponibilidade de servidores em um sistema de arquivos distribuído e adaptável. In: UNIVERSIDADE DE SÃO PAULO - CAMPUS DE SÃO CARLOS. *XXI Simpósio Internacional de Iniciação Científica da USP*. [S.l.], 2013. p. 1.

OKADA, T. K. et al. Flexa - grupo de réplicas em um sistema de arquivos distribuídos. In: UNIVERSIDADE FEDERAL DE SÃO CARLOS. *IV Escola Regional de Alto Desempenho de São Paulo*. [S.l.], 2013. p. 1.

OSADZINSKI, A. The network file system (nfs). *Comput. Stand. Interfaces*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 8, n. 1, p. 45–48, jul. 1988. ISSN 0920-5489. Disponível em: <http://dx.doi.org/10.1016/0920-5489(88)90076-1>.

SATYANARAYANAN, M. et al. Coda: A highly available file system for a distributed workstation environment. *In IEEE Transactions on Computers*, p. 447–459, 1990.

SCHROEDER, B.; GIBSON, G. A. A large-scale study of failures in high-performance computing systems. In: *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006. (DSN '06), p. 249–258. ISBN 0-7695-2607-1. Disponível em: <http://dx.doi-.org/10.1109/DSN.2006.5>.

SHVACHKO, K. et al. The hadoop distributed file system. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Washington, DC, USA: IEEE Computer Society, 2010. (MSST '10), p. 1–10. ISBN 978-1-4244-7152-2. Disponível em: <http://dx.doi.org/10.1109/MSST.2010.5496972>.

TANENBAUM, A.; STEEN, M. van. *Distributed systems: principles and paradigms*. Pearson Prentice Hall, 2007. ISBN 9780132392273. Disponível em: <http://books-.google.com.br/books?id=DL8ZAQAAIAAJ>.

WANG, F. O. et al. *Understanding Lustre Internals*. [s.n.], 2009. Disponível em: <http://www.osti.gov/scitech/servlets/purl/951297>.

WEIL, S. A. et al. Ceph: A scalable, high-performance distributed file system. In: *In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*. [S.l.: s.n.], 2006. p. 307–320.

WILCOX-O'HEARN, Z.; WARNER, B. Tahoe: The least-authority filesystem. In: *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*. New York, NY, USA: ACM, 2008. (StorageSS '08), p. 21–26. ISBN 978-1-60558-299-3. Disponível em: <http://doi.acm.org/10.1145/1456469.1456474>.

Apêndice A

Resultados de Desempenho Individuais

A.1 Desempenho do FlexA após as modificações

Tabela A.1: Média dos tempos dos testes no FlexA com as modificações propostas

TESTES FLEXA						
Escrita (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,103	0,229	0,437	1,023	1,964	3,949
2	0,104	0,240	0,449	0,986	1,930	3,785
4	0,109	0,238	0,454	1,032	2,071	4,240
8	0,143	0,252	0,477	1,088	2,266	4,577
16	0,163	0,309	0,566	1,284	2,888	6,212
32	0,343	0,592	1,139	2,819	7,545	27,850
Leitura (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	1,113	1,159	1,284	1,471	2,596	3,769
2	1,113	1,153	1,298	1,452	2,508	3,651
4	1,113	1,161	1,286	1,458	2,558	3,740
8	1,110	1,153	1,277	1,452	2,506	3,617
16	1,142	1,162	1,301	1,474	2,633	4,407
32	1,848	1,257	1,606	1,762	3,067	5,460

Tabela A.2: Média da vazão no FlexA com as modificações propostas

TESTES FLEXA						
Escrita (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	9,67	21,86	22,89	24,45	25,46	25,32
2	9,60	20,82	22,25	25,34	25,90	26,42
4	9,18	20,97	22,03	24,22	24,14	23,58
8	7,02	19,81	20,97	22,98	22,06	21,85
16	6,15	16,16	17,68	19,46	17,31	16,10
32	2,91	8,45	8,78	8,87	6,63	3,59
Leitura (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,90	4,32	7,79	16,99	19,26	26,53
2	0,90	4,34	7,71	17,22	19,94	27,39
4	0,90	4,31	7,78	17,15	19,54	26,74
8	0,90	4,34	7,83	17,21	19,95	27,65
16	0,88	4,30	7,69	16,96	18,99	22,69
32	0,54	3,98	6,23	14,19	16,31	18,31

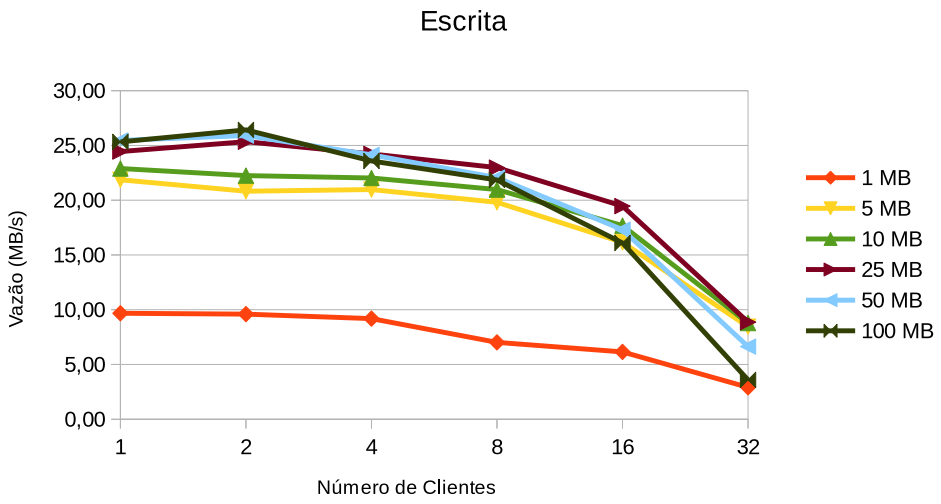

Figura A.1: Desempenho de escrita do FlexA

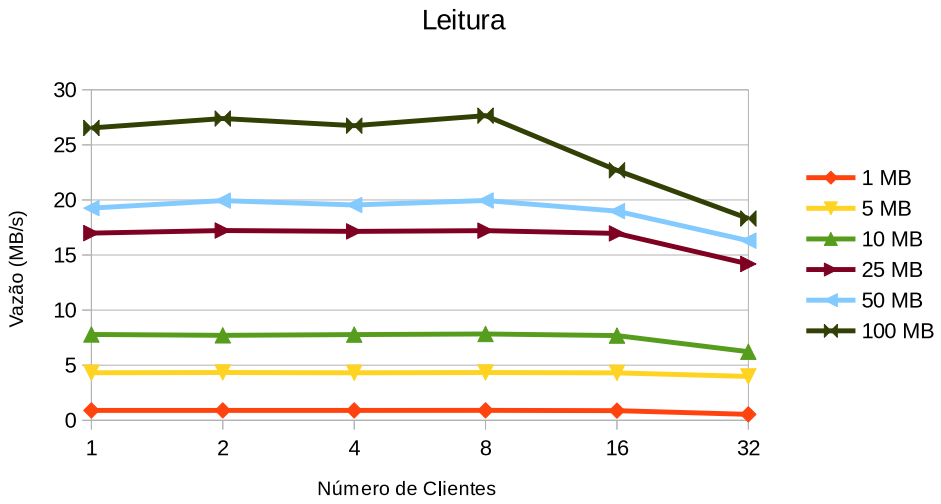

Figura A.2: Desempenho de leitura do FlexA

A.2 Desempenho do FlexA original

A versão original do sistema de arquivos FlexA não terminou o teste com 32 clientes, por isso os resultados não foram apresentados.

Tabela A.3: Média dos tempos dos testes do FlexA versão Fernandes (2012)

TESTES FLEXA – Versão inicial						
Escrita (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,103	0,229	0,437	1,023	1,964	3,949
2	0,388	0,641	0,878	1,637	3,000	5,609
4	0,384	0,640	0,960	1,783	3,373	6,400
8	0,391	0,691	1,002	1,947	3,657	8,386
16	0,485	0,801	1,268	2,733	6,652	16,603
32	-	-	-	-	-	-
Leitura (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	1,113	1,159	1,284	1,471	2,596	3,769
2	1,068	1,128	1,204	1,432	1,979	4,158
4	1,066	1,129	1,205	1,430	2,150	4,374
8	1,082	1,127	1,202	1,426	2,244	4,407
16	1,116	1,123	1,196	1,421	2,232	4,897
32	-	-	-	-	-	-

Tabela A.4: Média da vazão dos testes do FlexA versão Fernandes (2012)

TESTES FLEXA – Versão inicial						
Escrita (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	9,671	21,863	22,894	24,445	25,457	25,324
2	2,577	7,795	11,392	15,269	16,666	17,830
4	2,605	7,809	10,412	14,022	14,825	15,626
8	2,555	7,240	9,982	12,841	13,673	11,925
16	2,064	6,243	7,887	9,147	7,517	6,023
32	-	-	-	-	-	-
Leitura (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,899	4,315	7,786	16,993	19,257	26,534
2	0,937	4,433	8,305	17,457	25,261	24,047
4	0,938	4,427	8,300	17,477	23,253	22,862
8	0,924	4,436	8,322	17,532	22,283	22,692
16	0,896	4,451	8,361	17,591	22,399	20,419
32	-	-	-	-	-	-

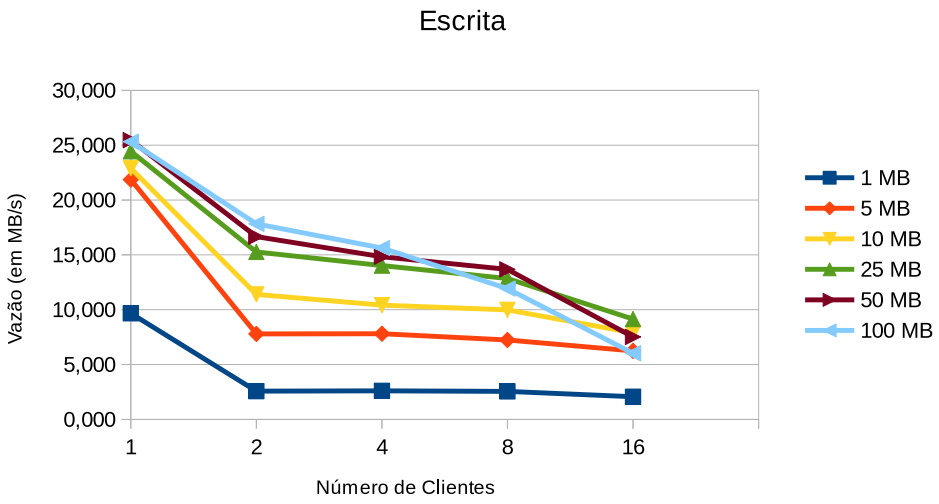

Figura A.3: Desempenho de escrita do FlexA versão Fernandes (2012)

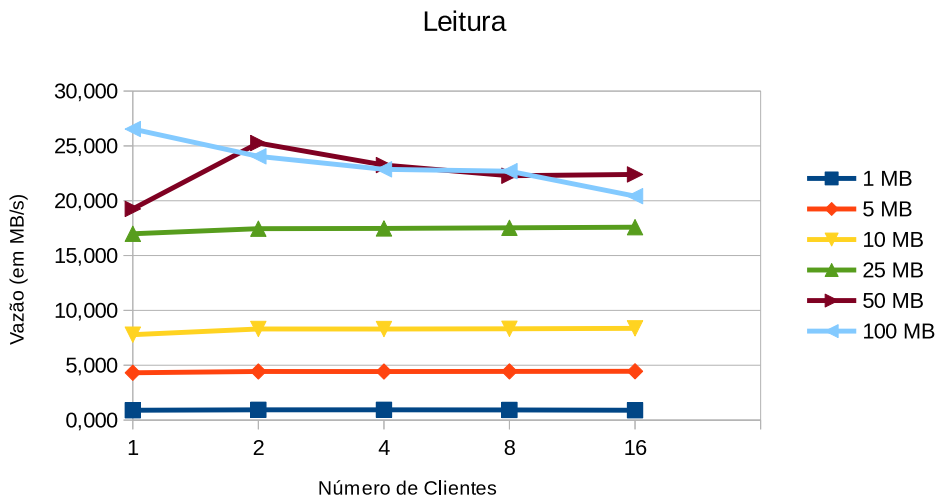

Figura A.4: Desempenho de leitura do FlexA versão Fernandes (2012)

A.3 Desempenho do NFS

Tabela A.5: Média dos tempos dos testes no NFS

TESTES NFS						
Escrita (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,116	0,240	0,440	1,078	2,102	4,241
2	0,121	0,251	0,439	1,072	2,295	4,600
4	0,114	0,253	0,498	1,205	2,546	5,389
8	0,134	0,303	0,570	1,808	7,199	13,427
16	0,128	0,319	0,656	3,493	8,775	29,817
32	0,125	0,311	0,682	8,591	29,369	72,334
Leitura (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,080	0,190	0,299	0,622	1,194	2,438
2	0,087	0,171	0,274	0,611	1,198	2,366
4	0,100	0,180	0,283	0,608	1,179	2,317
8	0,184	0,174	0,275	0,618	1,273	2,505
16	0,734	0,185	0,309	0,725	1,535	3,747
32	0,855	0,188	0,340	1,289	14,252	191,705

Tabela A.6: Média da vazão dos testes no NFS

TESTES NFS						
Escrita (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	8,621	20,833	22,727	23,191	23,787	23,579
2	8,264	19,920	22,779	23,321	21,786	21,739
4	8,772	19,763	20,080	20,747	19,639	18,556
8	7,463	16,502	17,544	13,827	6,945	7,448
16	7,813	15,674	15,244	7,157	5,698	3,354
32	8,000	16,077	14,663	2,910	1,702	1,382
Leitura (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	12,500	26,316	33,445	40,193	41,876	41,017
2	11,494	29,240	36,496	40,917	41,736	42,265
4	10,000	27,778	35,336	41,118	42,409	43,159
8	5,435	28,736	36,364	40,453	39,277	39,920
16	1,362	27,027	32,362	34,483	32,573	26,688
32	1,170	26,596	29,412	19,395	3,508	0,522

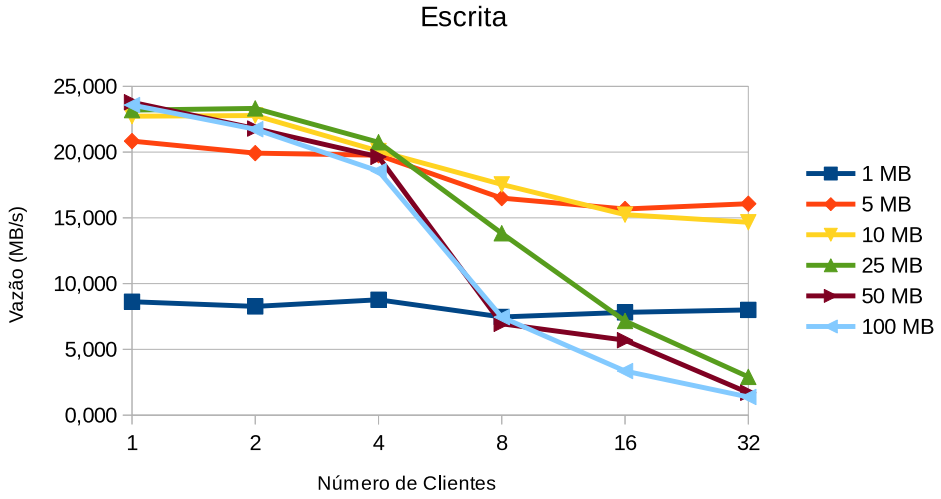

Figura A.5: Desempenho de escrita do NFS

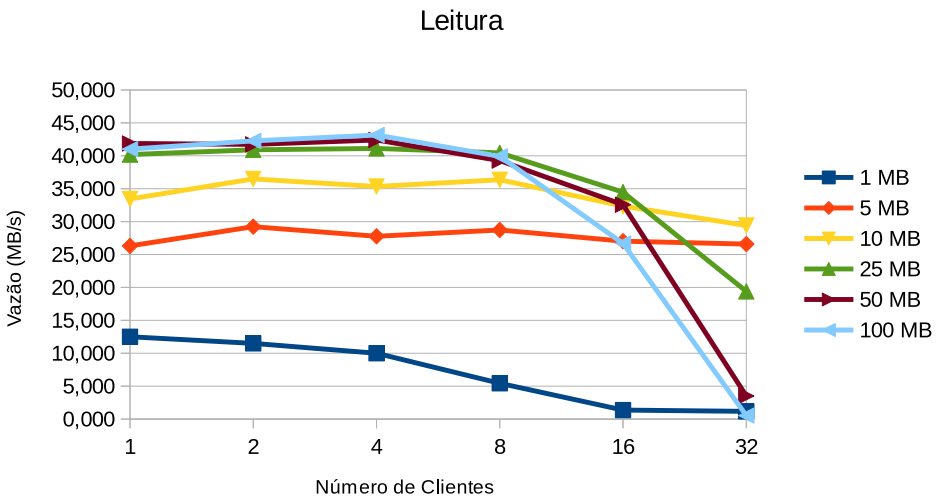

Figura A.6: Desempenho de leitura do NFS

A.4 Desempenho do Tahoe-LAFS

Tabela A.7: Média dos tempos dos testes no Tahoe-LAFS

TESTES TAHOE-LAFS						
Escrita (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,511	1,200	2,256	7,229	20,615	85,139
2	0,640	1,210	2,257	7,222	20,496	85,738
4	0,509	1,217	2,268	7,410	20,820	87,061
8	0,508	1,222	2,295	7,734	26,375	11,766
16	0,557	1,290	2,426	8,746	30,085	137,767
32	0,594	1,334	2,591	11,986	36,339	170,315
Leitura (em segundos)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	0,700	0,690	1,163	3,110	8,271	25,817
2	0,728	0,719	1,234	3,906	11,167	41,258
4	0,695	0,869	1,629	4,707	18,909	60,861
8	1,045	2,130	9,154	18,304	43,033	65,692
16	20,449	25,830	24,892	19,963	39,905	87,369
32	0,829	1,122	7,185	41,144	165,256	312,929

Tabela A.8: Média da vazão dos testes no Tahoe-LAFS

TESTES TAHOE-LAFS						
Escrita (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	1,96	4,17	4,43	3,46	2,43	1,17
2	1,56	4,13	4,43	3,46	2,44	1,17
4	1,96	4,11	4,41	3,37	2,40	1,15
8	1,97	4,09	4,36	3,23	1,90	8,50
16	1,80	3,87	4,12	2,86	1,66	0,73
32	1,68	3,75	3,86	2,09	1,38	0,59
Leitura (em MB/s)						
Qtd. Clientes	1 MB	5 MB	10 MB	25 MB	50 MB	100 MB
1	1,43	7,24	8,60	8,04	6,05	3,87
2	1,37	6,95	8,11	6,40	4,48	2,42
4	1,44	5,75	6,14	5,31	2,64	1,64
8	0,96	2,35	1,09	1,37	1,16	1,52
16	0,05	0,19	0,40	1,25	1,25	1,14
32	1,21	4,46	1,39	0,61	0,30	0,32

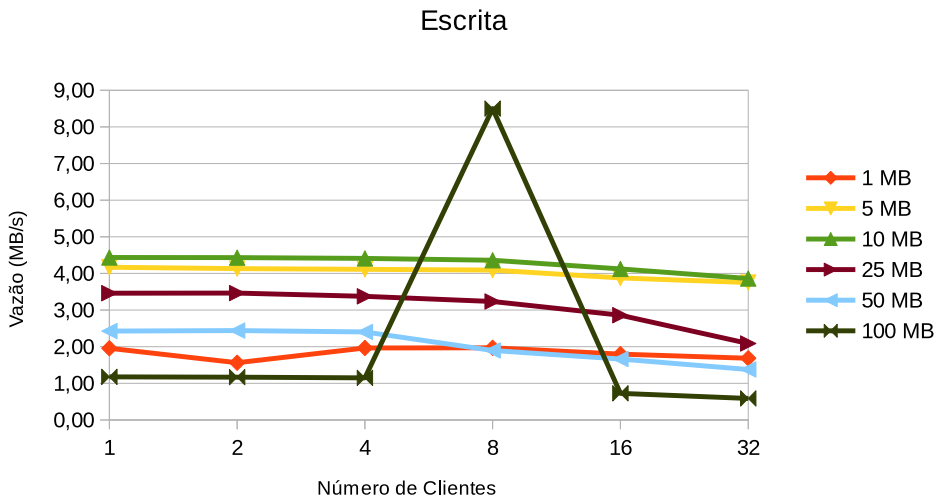

Figura A.7: Desempenho de escrita do Tahoe-LAFS

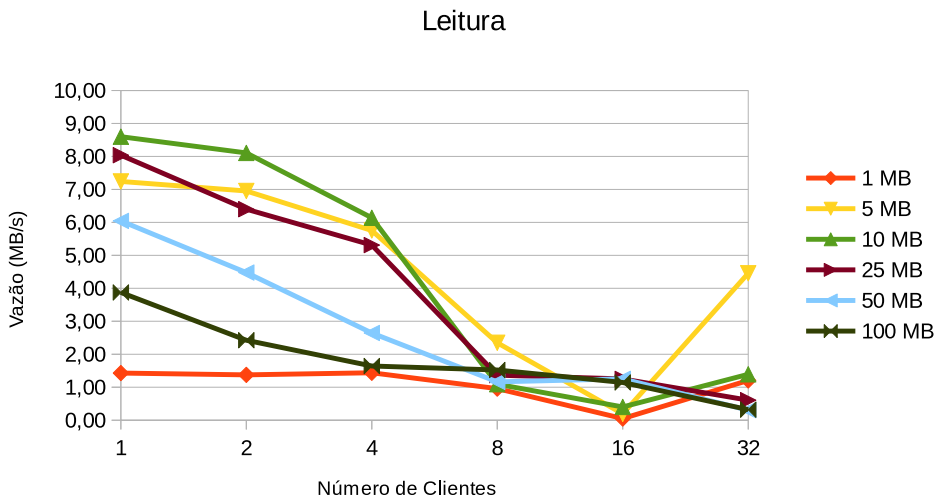

Figura A.8: Desempenho de leitura do Tahoe-LAFS

Apêndice B

Instalação do FlexA após as modificações

A instalação do FlexA não necessita de recursos especiais ou modificações no sistema operacional. Para a instalação é necessário um Sistema Operacional do tipo GNU/Linux com os pacotes *Python* 2.7, além das bibliotecas *netifaces* e *pycrypto*.

Após a instalação dos pacotes, é necessário definir pelo menos três máquinas como servidores primários e uma ou mais máquinas como servidores secundários. Antes de iniciar os módulos coletores correspondentes é necessário rodar o comando abaixo para gerar a configuração inicial:

```
$ python2 com.py -r
```

As máquinas definidas como servidores primários são criadas utilizando o comando abaixo:

```
$ python2 coletor_servidor.py
```

Enquanto as máquinas definidas como servidores secundários são criadas utilizando:

```
$ python2 coletor_replica.py
```

Para fazer os nós se localizarem automaticamente, é necessário rodar o seguinte comando:

```
$ python2 com.py -bs
```

Ao passar o parâmetro *-d* ao comando acima, a localização de nós é feitas por um *daemon* que procura novos nós após um determinado tempo.

No cliente é necessário executar o seu respectivo coletor caso se deseja receber

algum arquivo do sistema:

```
$ python2 coletor_cliente.py
```

Após isso, o sistema estará pronto para ser usado. É possível enviar um arquivo do cliente para o sistema usando o comando:

```
$ python2 flexa.py -p <NOME DO ARQUIVO>
```

E para baixar o arquivo depois de enviado:

```
$ python2 flexa.py -g <NOME DO ARQUIVO>
```

Apêndice C

Exemplo de funcionamento do sistema

Neste apêndice serão demonstradas as funcionalidades implementadas no capítulo 3 no sistema de arquivos FlexA.

Para este exemplo foram configurados três servidores primários, dois servidores secundários e um cliente como descrito no Apêndice B. Os servidores foram configurados dentro de uma máquina virtual no programa VirtualBox, enquanto o cliente foi executado diretamente na máquina *host* utilizada nos testes. Foram usadas as configurações abaixo:

- Máquina *host*:
  - Processador Intel®Core ™i7-3610QM CPU de 4 núcleos e 8 *thread* @ 2,30GHz;
  - Memória RAM de 8GB;
  - HDD de 1TB 7200RPM SATA II;
  - Rede Gigabit Ethernet (1000 Mbps);
  - Sistema Operacional Arch Linux e o Linux kernel versão 3.12.1-ARCH;
  - VirtualBox versão 4.3.2.
- Máquinas virtuais:
  - Um processador;
  - 1GB de RAM;
  - HDD de 10GB;
  - Rede Gigabit Ethernet (1000 Mbps);
  - Sistema Operacional Debian GNU/Linux 7.1.0 e Linux kernel versão 3.2.0-4-amd64.

Após a configuração e execução dos servidores primários, eles se rastrearam utilizando o código de sondagem, como mostrado na Figura C.1.

```
**********************************************
SONDAGEM de PRIMARIOS ativos
**********************************************
IP: 192.168.1.116

IP: 192.168.1.173

IP: 192.168.1.179

**********************************************
```

Figura C.1: Servidores primários se rastreando

O cliente então envia seu arquivo para os servidores primários (Figura C.2), que replica os dados para seus servidores secundários e realiza a sincronização dos respectivos metadados (Figura C.3). O servidor secundário então recebe a réplica e a armazena (Figura C.4).

```
Arquivo particionado em 3 parte(s)
tamanho das porcoes 10485768, 10485768, 10485768
Registrando no Banco de Dados
localporcao 1@192.168.1.173@5000$2@192.168.1.116@5000$3@192.168.1.179@5000$
servidores_porcao. ['192.168.1.173', '192.168.1.116', '192.168.1.179']
servidores_metadados []
Enviando o chunk arquivo_teste.enc-1 para o servidor 192.168.1.173:5000
 Enviando o chunk arquivo_teste.enc-2 para o servidor 192.168.1.116:5000
 Não há mais servidores que porções, sem necessidade de enviar metadados e arqui
vo de tamanho 0
wait_file on
Aguardando transferência...
Enviando o chunk arquivo_teste.enc-3 para o servidor 192.168.1.179:5000
Porção arquivoteste.enc1 transferida para Servidor 192.168.1.173
Porção arquivoteste.enc2 transferida para Servidor 192.168.1.116
Porção arquivoteste.enc3 transferida para Servidor 192.168.1.179
```

Figura C.2: Cliente enviando o arquivo para os servidores primários

```
Porção e metadados enviados para a RÉPLICA 192.168.1.76
Porção e metadados enviados para a RÉPLICA 192.168.1.162

Recebendo 231 bytes de cabeçalho
HEADER: [<div>servidor#sinc_metadados#5000#none#163eae9a56d666b8537b576ed469be32
5e5d4d1eb1a9ba751dc1bf7b1bec47d7dc6bb82aab3fbeac0e0854b0389c5fbc#3@192.168.1.162
@7500$3@192.168.1.76@7500$2@192.168.1.76@7500$2@192.168.1.162@7500$#1#False<div>
]
Tamanho header 231 bytes
Recebida mensagem de sincronização do ip 192.168.1.116
Novo metadado gerado: 3@192.168.1.162@7500$3@192.168.1.76@7500$2@192.168.1.76@75
00$2@192.168.1.162@7500$1@192.168.1.76@7500$1@192.168.1.162@7500$
Próximo IP da sincronização 192.168.1.179
```

Figura C.3: Servidor primário replicando os arquivos e sincronizando os metadados

```
Upload arquivo [arquivo_teste:10485768 bytes] pelo cliente [192.168.1.179:7000]
Transferindo arquivo...
Armazenando arquivo 163eae9a56d666b8537b576ed469be325e5d4d1eb1a9ba751dc1bf7b1bec
47d7dc6bb82aab3fbeac0e0854b0389c5fbc.enc-3
Registrando na tabela arquivos_sad
Registrando na arquivos_SAD, atualizando
Arquivo [arquivo_teste] transferido
```

Figura C.4: Servidor secundário armazenando a réplica

Um dos servidores primários então cai (o cabo foi desconectado virtualmente para simular uma falha). Os outros servidores primários detectam a falha (Figura C.5) e enviam uma mensagem para um servidor secundário iniciar a eleição (Figura C.6).

```
Servidor com IP 192.168.1.179
 avisado de que Primário caiu

Recebendo 122 bytes de cabeçalho
HEADER: [<div>none#servidoresinativos#0#none#none#0#none#none#192.168.1.179#none
#$192.168.1.173
#none#none#none#none#none#none<div>]
Tamanho header 122 bytes
Servidor(es) que cairam $192.168.1.173

IP da máquina que avisou 192.168.1.179

Recebendo 122 bytes de cabeçalho
HEADER: [<div>none#servidoresinativos#0#none#none#0#none#none#192.168.1.116#none
#$192.168.1.173
#none#none#none#none#none#none<div>]
Tamanho header 122 bytes
Servidor(es) que cairam $192.168.1.173

IP da máquina que avisou 192.168.1.116
```

Figura C.5: Um servidor primário detecta que outro servidor primário caiu

```
IP da máquina que avisou 192.168.1.116
Servidor 192.168.1.173
 realmente caiu. Tomando providências
Convocação da eleição será feita por outro primário
Servidor 192.168.1.173
 realmente caiu. Tomando providências
Host Server 192.168.1.179 irá convocar eleição
Réplica escolhida 192.168.1.162
Tempo de sondagem: 13.15158391

Recebendo 70 bytes de cabeçalho
HEADER: [<div>servidor#eleicao#7500#none#none#none#192.168.1.162#none#none<div>]
Tamanho header 70 bytes
```

Figura C.6: Servidor primário convoca a eleição

Os servidores secundários executam a eleição usando o algoritmo anel e elegem um dos nós como servidor primário. Este nó é informado que ganhou a eleição, então ele sincroniza os metadados com o servidor primário que iniciou a eleição e se torna um servidor primário (Figura C.7).

```
Tamanho header 93 bytes
Réplica eleita: 192.168.1.162
Métricas do eleito: 0.95186057937
Eleicao terminou, avisando 192.168.1.162 que ele foi o nó eleito
Tempo eleição: 0.00310707092285

Recebendo 68 bytes de cabeçalho
HEADER: [<div>replica#eleicao#5000#none#192.168.1.179#0#eleito#none#none<div>]
Tamanho header 68 bytes
Fui o nó eleito
Armazenando IP 0 do servidor caido

Recebendo 91 bytes de cabeçalho
HEADER: [<div>replica#sincronizacao#5000#none#servidor.db#5120#9b0b9147d94a926a
605778574ef3d91<div>]
Tamanho header 91 bytes
Transferindo arquivo...
Armazenando arquivo servidor.db
Sincronização do banco de dados feita com sucesso
Iniciando o serviço coletor_servidor
Tempo sincronização: 0.0136828422546

Sincronizador servidor [Ativo] IP: 192.168.1.162; Porta socket: 5000
```

Figura C.7: Servidor secundário se tornando primário

Finalmente o cliente tenta baixar o arquivo enviado para o sistema antes da falha ocorrer. Mesmo com um dos primários fora do ar, ele consegue baixar e enviar arquivos com sucesso (Figura C.8).

```
Porção /home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-1 recebida com suc
esso !
/home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-2
Esperando a porção: /home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-2
Porção /home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-2 recebida com suc
esso !
/home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-3
Esperando a porção: /home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-3
Porção /home/m45t3r/Source/flexa/cache/root/arquivo_teste.enc-3 recebida com suc
esso !
Agrupando porções...

Obtendo arquivo:  /home/m45t3r/Source/flexa/arquivo_teste

Arquivo a se desencriptar:  /home/m45t3r/Source/flexa/cache/root/arquivo_teste.e
nc
Number of chunks 3
```

Figura C.8: O cliente consegue executar suas funções normalmente, mesmo com um servidor primário fora do ar