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

Matheus Della Croce Oliveira

Sincronização, consistência e falhas no FlexA

São José do Rio Preto
2013

Matheus Della Croce Oliveira

Sincronização, consistência e falhas 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.

Orientador:

Prof. Dr. Fernando Ferrari

São José do Rio Preto
2013

Matheus Della Croce Oliveira

Sincronização, consistência e falhas 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. Fernando Ferrari

Matheus Della Croce Oliveira

Banca Avaliadora:

Prof. Dr. Aleardo Manacero Jr.

Prof. Dr. Norian Marranghello

São José do Rio Preto
2013

Aos meus avós Valentina e Euclydes

À minha mãe Cleunice

Aos meus tios Carlos, Paulo e Antônio

À Livia

"Perdoar as pessoas, ter esperança e acreditar em Deus."

- José Carlos Della Croce

Agradecimentos

Agradeço primeiramente a Deus por ter me concedido a vida e a minha maravilhosa família.

Agradeço à minha mãe Cleunice, minha avó Valentina e ao meu avô Euclydes por terem cuidado de mim com tanto amor durante todos esses anos. Agradeço também aos meus tios José Carlos, Paulo Cesar e Antônio e às minhas tias Bete, Renata e Charlene, pelos grandes valores ensinados, os quais carregarei durante toda a vida.

Agradeço à Livia Ferreira por ter sido minha grande companheira nesses anos de namoro.

Gostaria de agradecer também aos meus professores e orientadores Prof. Dr. Alexardo Manacero Jr. e Profa. Dra. Renata Spolon Lobato por terem concedido a oportunidade de participar do grupo de pesquisa GSPD e também pela paciência nas inúmeras vezes que fui tirar dúvidas referentes aos projetos e disciplinas. Agradeço também ao Prof. Dr. Fernando Ferrari por ter aceito ser meu orientador do projeto final/TCC.

Por fim, agradeço aos meus amigos de Rio Preto, da graduação e do GSPD, e ao Povo de Sempre, de Andradina. Vocês todos têm parte nesta conquista!

Matheus Della Croce Oliveira

Resumo

Os sistemas de arquivos distribuídos surgiram com a finalidade de prover armazenamento seguro e compartilhamento de dados. Neste tipo de sistema as informações são distribuídas por diversos computadores para se alcançar um maior nível de confiabilidade e desempenho. Dentre seus desafios de concepção estão elencados a escalabilidade, tolerância a falhas, transparência de acesso, entre outros. Este trabalho apresenta melhorias de desempenho, escalabilidade, confiabilidade no sistema de arquivos distribuído FlexA. Como forma de se alcançar tais objetivos, foram desenvolvidos mecanismos de sincronismo entre servidores primários e secundários, tratamento de consistências de dados replicados e detecção de falhas nos serviços deste sistema.

Palavras-chave: Sistemas distribuídos, sistemas de arquivos distribuídos, sincronização, replicação, consistência, detecção de falhas, FlexA, NFS, Tahoe-LAFS, HDFS, GFS, AFS, Ceph

Abstract

Distributed file systems came with the purpose of providing secure storage and data sharing. In this type of system, information is distributed among several computers to achieve a higher level of reliability and performance. Among its conception challenges one finds listed the scalability, fault tolerance, access transparency. This work presents improvements in performance, scalability, reliability of a distributed file system, called FlexA. In order to achieve these objectives, were developed mechanisms for synchronization between primary and secondary servers, management of replicated data's consistency and fault detection in this system's services.

Keywords: Distributed systems, distributed file systems, synchronization, replication, consistency, fault detection, FlexA Tahoe-LAFS, HDFS, GFS, AFS, Ceph

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AFS *Andrew File System*

CRUSH *Controlled Replication Under Scalable Hashing*

FlexA *Flexible and Adaptable Distributed File System*

GFS *Google File System*

HDFS *Hadoop Distributed File System*

IP *Internet Protocol*

KB *Kilobyte*

MB *Megabyte*

MDS *Metadata Storage*

NFS *Network File System*

OSDs *Object Storage Devices*

RPCs *Remote Procedure Calls*

SADs Sistemas de arquivos distribuídos

SDs Sistemas Distribuídos

Tahoe-LAFS *Tahoe Least Authority File System*

Capítulo 1

Introdução

1.1 Motivação

Os recursos de *hardware* e *software*, por mais que evoluam, estão sempre propensos a falhas e sobrecargas. Com o crescente aumento no volume de dados e a necessidade de compartilhá-los, optou-se por utilizar soluções descentralizadas com o intuito de diminuir a perda ocasionada por defeitos nos componentes físicos do sistema (FERNANDES, 2012).

Com base na premissa de armazenar os dados de forma segura e compartilhá-los com um bom desempenho, surgiram os sistemas de arquivos distribuídos. Na concepção de sistemas de arquivos distribuídos, características distintas como escalabilidade, confiabilidade, desempenho, tolerância a falhas, concorrência devem ser atendidas. Sendo assim, muitos sistemas de arquivos distribuídos focam em um conjunto de características, tornando-os mais específicos para determinados ambientes e situações (PATE, 2003; COULOURIS et al., 2011).

O sistema de arquivos distribuído FlexA (***Flex****ible and* ***A****daptable Distributed File System*) (FERNANDES, 2012), em desenvolvimento no laboratório do GSPD (**G**rupo de **S**istemas **P**aralelos e **D**istribuídos) da UNESP (GSPD, 2013) e foco deste trabalho, prioriza características de desempenho e tolerância a falhas. Sendo assim, este trabalho consiste em melhorá-las e agregar maior confiabilidade ao sistema em desenvolvimento.

1.2 Objetivo

O objetivo deste trabalho foi utilizar soluções como replicação de dados e detecção de falhas para melhorar a confiabilidade e desempenho do sistema de arquivos distribuído FlexA. Desta forma, este trabalho envolveu a sincronização de dados entre servidores primários e secundários para fornecer redundância de arquivos, além do desenvolvimento de mecanismos para o tratamento de inconsistências em dados replicados. Também foram construídos mecanismos para detectar falhas nos serviços deste sistema,

tanto em servidores primários como em secundários, sendo esta etapa necessária para obter um sistema tolerante a falhas.

1.3 Organização do texto

O capítulo 2 apresenta conceitos sobre sistemas distribuídos e sistemas de arquivos distribuídos, além de descrever os principais sistemas de arquivos distribuídos existentes. Desse estudo foram extraídas informações que auxiliaram o desenvolvimento deste trabalho. O capítulo 3 descreve a metodologia de desenvolvimento do projeto, apresentando os algoritmos e mecanismos utilizados para sincronizar os servidores primários e secundários, garantir a consistência entre os dados replicados e detectar falhas nos serviços do FlexA. Os testes e validação dos componentes desenvolvidos são apresentados no capítulo 4. Por fim, o capítulo 5 apresenta a conclusão, dificuldades encontradas e direções futuras.

Capítulo 2

Revisão bibliográfica

Este capítulo tem como objetivo apresentar os conceitos sobre sistemas distribuídos e sistemas de arquivos distribuídos necessários para o entendimento deste trabalho.

Inicialmente é feito um levantamento sobre as principais características dos sistemas distribuídos. A seguir o foco mantém-se no estudo dos diversos sistemas de arquivos distribuídos existentes, descrevendo-os sucintamente e extraindo-lhes informações sobre o tratamento de alguns problemas específicos, como sincronização, consistência e detecção de falhas. Tais informações serviram de base para a implantação de componentes no sistema de arquivos distribuído FlexA.

2.1 Sistemas distribuídos

A definição de (COULOURIS et al., 2011) especifica Sistemas Distribuídos (SDs) como sendo "Um sistema no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si".

A principal motivação para a construção de SDs é a necessidade de compartilhar recursos, sejam eles de *hardware*, tais como impressoras, ou *software*, como arquivos e bancos de dados. Estes recursos podem estar espacialmente separados por qualquer distância, o que gera alguns problemas para os SDs, como concorrência entre usuários e processos, falhas independentes, inexistência de um relógio global, entre outros (COULOURIS et al., 2011).

Na concepção de SDs, alguns desafios e metas de projeto são descritos por (COULOURIS et al., 2011), (TANENBAUM; STEEN, 2007) e (KSHEMKALYANI; SINGHAL, 2008):

- Transparência: esconde o fato de que processos ou recursos estão fisicamente distribuídos por vários computadores, promovendo o acesso como sendo um único sistema para o usuário ou aplicação.
- Comunicação: faz o uso de mecanismos apropriados para as trocas de mensagens entre os dispositivos da rede.

- Heterogeneidade: lida com computadores que possuem diversos tipos de sistemas operacionais, *hardware* e linguagens de programação.
- Escalabilidade: capacidade do sistema de suportar um aumento no número de usuários ou fornecimento de recursos.
- Controle de concorrência: administra o acesso simultâneo a um mesmo recurso por diversos usuários.
- Segurança: acesso seguro, através do uso de técnicas criptográficas, aos dados do sistema. Controla qual usuário possui autorização para acessar determinados recursos compartilhados.
- Sincronização: coordena os processos para que eles não acessem um recurso ao mesmo tempo ou para que se comportem de forma ordenada em relação a um determinado evento.
- Replicação: melhora o desempenho e a confiabilidade do SD utilizando-se técnicas que consistem em criar múltiplas cópias dos recursos em diferentes localizações do sistema.
- Consistência: garante que sejam satisfeitos os requisitos de consistência entre os dados.
- Tolerância a falhas: os componentes devem falhar de maneira independente, sendo que as falhas devem ser praticamente imperceptíveis para os usuários. Para isto, é necessário utilizar técnicas para detectar falhas e posteriormente tratá-las ou mascará-las, de modo que o sistema continue funcionando corretamente mesmo em sua ocorrência.

2.2 Sistemas de arquivos distribuídos

Sistemas de arquivos distribuídos (SADs) agregam a mesma motivação e desafios de construção dos SDs. Além disso, os SADs devem prover as mesmas funções dos sistemas de arquivos locais, como organização, proteção e armazenamento de dados (COULOURIS et al., 2011).

Dentre os desafios citados para a construção de SDs/SADs, particularmente a sincronização na replicação de dados, consistência, e detecção de falhas, são abordados neste trabalho. Sendo assim, as próximas subseções apresentam um estudo dos principais SADs existentes, inclusive o FlexA, sistema alvo deste trabalho.

2.2.1 Replicação de dados e consistência

O aumento no número de usuários acessando um mesmo recurso pode resultar em perda de desempenho ou gargalos no sistema. Para que isso não ocorra, a solução

mais utilizada é a replicação de dados. Dados replicados em diferentes locais garantem que a informação esteja disponível mesmo na presença de falhas ou sobrecarga de algum componente. Desta forma, usuários podem obter as cópias em outras localizações, aumentando-se o desempenho do SD. Além disso, a distribuição de várias cópias proporciona uma maior proteção contra possíveis dados corrompidos, acrescentado-se mais confiabilidade ao sistema.

Uma outra abordagem para evitar a degradação de desempenho num SD é diminuir o número de requisições dos usuários aos servidores. Isso pode ser feito com a utilização de um tipo especial de replicação: as *caches*. A *cache* geralmente é controlada pelos clientes (usuários que acessam os recursos), em vez dos servidores, e armazena cópias de dados. O conteúdo da *cache* varia de acordo com a aplicação e tem como objetivo reduzir o tempo de acesso aos dados do processo.

Em detrimento do aumento da confiabilidade e desempenho oriundos da replicação de dados, há o problema da manutenção da consistência. Uma vez que a cópia é modificada, todas as outras devem ser atualizadas para garantir que a informação mais atual esteja sempre disponível. A tarefa de sincronizar as réplicas de forma atômica, isto é, qualquer atualização nos dados devem ser repassadas imediatamente para todas as cópias, demanda um enorme uso de recursos computacionais, tornando-se ainda mais desafiadora quando há muitas cópias espacialmente separadas por grandes distâncias.

Para evitar a custosa operação de sincronia atômica das réplicas, pode-se relaxar o grau de consistência entre os dados replicados. Isto significa que, levando-se em consideração a tolerância a dados inconsistentes suportados pela aplicação, nem todas as cópias precisam ser atualizadas exatamente ao mesmo tempo. Desta forma, cada aplicação define qual "modelo de consistência" deve seguir, especificando para usuários e sistema as regras que serão seguidas nas operações de escrita e leitura de dados replicados, garantindo o funcionamento correto do sistema. A escrita é uma operação que altera os dados, enquanto a leitura não altera (TANENBAUM; STEEN, 2007).

Por fim, na construção de SDs que replicam seus dados é necessário definir alguns mecanismos para manter as réplicas consistentes. O chamado "gerenciamento de réplicas" (TANENBAUM; STEEN, 2007) envolve, inicialmente, duas situações:

- Encontrar a melhor localização para os servidores que podem hospedar os dados.
- Definir quando e qual componente do sistema irá realizar a replicação.

A definição de qual componente do sistema irá iniciar a replicação é muito importante. São descritos 3 tipos:

- Réplicas permanentes: utilizam-se técnicas como espelhamento de dados para conseguir que os recursos estejam disponíveis em áreas geográficamente separadas.
- Réplicas iniciadas por servidores: cópias são distribuídas entre servidores para servirem como apoio na distribuição e acesso aos dados, aumentando-se o desempenho do sistema.

- Réplicas iniciadas por clientes: conhecidas como *caches* de clientes. Usuários utilizam cópias temporárias locais para que processos possam acessá-las quase que instantâneamente.

O gerenciamento de réplicas também decide, por meio de protocolos de distribuição de conteúdo, quais as réplicas terão o conteúdo imediatamente atualizado ou não. Há basicamente três operações distintas a serem realizadas:

- Uso de protocolos de invalidação, cuja finalidade é de apenas propagar mensagens que invalidem dados desatualizados.
- Replicação passiva: transfere dados modificados de uma cópia para outra. Esta solução é útil para sistemas em que há muitas operações de leitura.
- Replicação ativa: o sistema informa a cada réplica, através da passagem de parâmetros, quais as operações que ela deve realizar na presença de uma atualização de dados.

2.2.2 Sincronização entre processos e algoritmos de eleição

A ausência de um relógio global nos SDs, ou seja, os diferentes valores de relógio nos nós (computadores) da rede, torna a sincronização entre processos bem mais complicada que em sistemas monoprocessados ou multiprocessados. Esta característica faz com que haja dificuldade em se determinar o estado de execução do sistema. Além disso, para garantir que não haja inconsistências e todos tenham acesso aos recursos compartilhados, o sistema deve controlar o acesso e a manipulação dos mesmos. Diante desses problemas, algoritmos de sincronização são desenvolvidos para que os processos comuniquem-se e tomem suas decisões no momento estabelecido pela aplicação (TANENBAUM; STEEN, 2007; COULOURIS et al., 2011).

Quando os processos do SD provêm do mesmo código fonte, ou seja, quando não há distinção entre eles, a sincronização pode requerer a utilização de um processo coordenador, que dá início à execução do sistema. Algoritmos que ajudam a definir quem será o processo coordenador são chamados de Algoritmos de Eleição, como por exemplo o Algoritmo de *Bullying* e o Algoritmo de Anel. Para este trabalho foi utilizado uma variação do Algoritmo de Anel.

No Algoritmo de Anel todos os processos são logicamente ou fisicamente ordenados na forma de um anel. Cada processo conhece todos os elementos do anel, mas as mensagens são transmitidas apenas para o próximo processo ativo na direção do anel. Quando um processo descobre que o coordenador não está ativo, ele envia para seu sucessor uma mensagem de eleição contendo sua identificação. Cada processo que recebe a mensagem também anexa sua identificação, sendo que o algoritmo pula os processos que não estão disponíveis. No fim, o processo que iniciou a eleição escolhe como coordenador o processo que tem a maior identificação e logo em seguida envia uma mensagem avisando cada sucessor até que se complete o ciclo do anel novamente (TANENBAUM; STEEN, 2007).

2.2.3 Tolerância e detecção de falhas

Segundo (KSHEMKALYANI; SINGHAL, 2008), um sistema distribuído tolerante a falhas continua provendo seus serviços corretamente mesmo com a presença de falhas nos nós, processos e enlaces de rede. Dessa forma, os SDs introduzem o conceito de falhas parciais pois, enquanto um componente apresenta defeitos, os outros podem continuar funcionando normalmente.

Falhas em sistemas computacionais são inevitáveis, podendo ocorrer no nível de *hardware*, como desligamentos e defeitos nos componentes físicos, ou de *software*, como problemas nos processos da aplicação. Para assegurar um maior nível de confiabilidade e disponibilidade ao serviço, as falhas devem ser detectadas no menor tempo possível.

A detecção de falhas, foco de uma parte deste trabalho, é uma das principais etapas na obtenção de um sistema tolerante a falhas. Sendo assim, é necessário conhecer quais os tipos de falhas podem ocorrer em SDs, descritos a seguir:

- Falha por queda: o servidor funciona normalmente, mas por algum motivo para de funcionar.
- Omissão: o servidor não responde e/ou não consegue enviar as respostas das requisições recebidas.
- Temporização: o servidor responde com atrasos.
- Resposta: a mensagem de resposta a alguma requisição possui valor incorreto.
- Arbitrária: também conhecidas como falhas bizantinas, são impossíveis de serem previstas por ocorrerem em momentos aleatórios durante a execução.

Em SDs, o principal mecanismo para verificar a presença de problemas com processos ou nós inativos é a utilização de outros nós para monitorá-los. Usualmente isso é feito com envios sucessivos de *pings* para os nós da rede, que verificam se o componente está respondendo às requisições. Caso a mensagem dê *timed out* (tempo esgotado), significa que o componente não responde e então há um problema que deve ser tratado. A literatura descreve soluções baseadas em trocas regulares de mensagens do tipo 'estou vivo' entre nós vizinhos da rede para identificar quem está em pleno funcionamento.

Um desafio para a detecção de falhas é diferenciar se as mesmas foram causadas por defeitos nos nós da rede ou na própria rede. Algoritmos que lidam com esse tipo de situação utilizam diversos nós para decidirem se algum vizinho caiu. Quando as mensagens de *ping* não são respondidas, o nó que detectou a falha pede para que seus vizinhos verifiquem se o nó requisitado realmente caiu. O cruzamento das informações de resposta podem determinar se o problema foi na rede ou não (COULOURIS et al., 2011; TANENBAUM; STEEN, 2007).

2.3 Exemplos de SADs

No restante do capítulo serão descritas algumas implementações de sistemas de arquivos distruídos. Eles foram escolhidos por serem representativos historicamente ou por aplicação.

2.3.1 *Network File System* (NFS)

Projetado e desenvolvido pela *Sun Microsystems*, o *Network File System* (NFS) é um SAD bastante utilizado em sistemas baseados em Unix. Tem como principal característica a transparência de acesso, sendo que para o usuário não há distinção entre operações em arquivos locais ou remotos. Tal fato é proveniente da utilização de um sistema de arquivos virtual (*Virtual File System* - VFS), implementado no núcleo dos sistemas operacionais, que redireciona as requisições feitas pelo usuário para o sistema de arquivos requisitado, seja ele local, o próprio NFS, ou qualquer outro sistema de arquivos (COULOURIS et al., 2011).

No modelo de acesso remoto há a divisão do sistema em duas partes: cliente e servidor. Toda a comunicação entre eles é feita por chamadas de procedimento remoto (*Remote Procedure Calls* - RPCs) (TANENBAUM; STEEN, 2007). A figura 2.1 ilustra sua arquitetura.

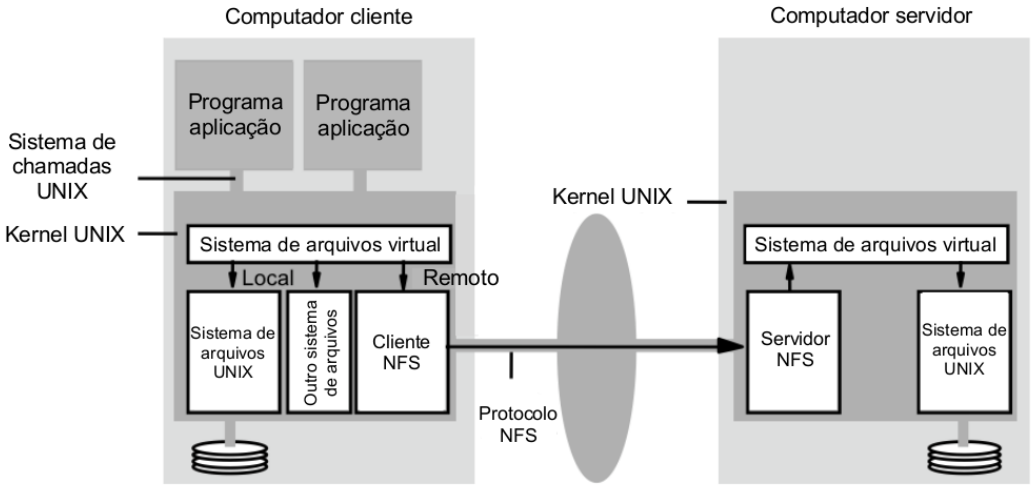

Figura 2.1: Arquitetura do NFS (FERNANDES, 2012), adaptado de (TANENBAUM; STEEN, 2007)

O NFS possui sistemas de *cache* no cliente e no servidor, cuja principal função é armazenar, através do sistema de arquivos local, arquivos lidos mais recentemente. O controle da consistência dos arquivos armazenados na *cache* do cliente é feito através do uso de *timestamps* que indicam a validade dos dados. A verificação da validade é feita de acordo com um intervalo de tempo entre as análises estabelecido pelo cliente

(COULOURIS et al., 2011).

O servidor foi projetado para não manter o estado do sistema (*stateless*), ou seja, cada requisição no sistema é independente das anteriores. Tal característica significa que no caso de quedas no serviço não há necessidade de recuperar as informações quando o sistema for reiniciado. Para o cliente, no evento de alguma falha durante as solicitações, basta apenas repetí-las até que o servidor forneça a resposta correta (COULOURIS et al., 2011; TANENBAUM; STEEN, 2007).

2.3.2 *Andrew File System* (AFS)

A principal característica do *Andrew File System* (AFS) é seu projeto pensado para suportar uma grande quantidade de usuários simultâneos no sistema. Segundo (COULOURIS et al., 2011), a alta escalabilidade é consequência do uso de *caches* que armazenam cópias inteiras de arquivos nos nós dos clientes. Dada essa característica, o AFS utiliza uma política bem severa de sincronização para manter a consistência de seus dados.

A arquitetura do AFS, assim como no NFS, é dividida em clientes e servidores. Os servidores executam um processo de nome *Vice*, e os clientes, *Venus*. O funcionamento do sistema é simples: o processo do cliente solicita os arquivos armazenados nos servidores do AFS e então os arquivos são enviados pela rede. O processo do cliente apresenta, para o usuário, o arquivo como sendo local à maquina. Qualquer modificação feita no espaço do AFS, bem como a criação ou atualização de arquivos, é enviada para os servidores. Sua arquitetura é mostrada na figura 2.2.

O AFS aumenta a eficiência de seu funcionamento com o uso da *cache* que armazena, nos clientes, cópias em porções de 64 *Kilobytes* (KB) de diretórios e arquivos lidos recentemente. Quando o usuário quer modificar qualquer arquivo, o cliente AFS pergunta para o servidor se o arquivo está na *cache* local. Em caso afirmativo, o cliente verifica se o arquivo está na sua última versão, e se estiver, ao invés de obter-se o arquivo do servidor, o arquivo da própria *cache* é utilizado (COULOURIS et al., 2011).

A consistência dos dados na *cache* do cliente é verificada através de RPCs do servidor para o cliente. O mecanismo, chamado de *callback*, informa o cliente caso haja alguma alteração em arquivos compartilhados. O AFS só considera o arquivo modificado quando for fechado pelo cliente, sendo que somente depois desta etapa o servidor envia um *token* de validade para os clientes detentores do arquivo. Se na verificação do *token*, quando o arquivo for aberto, informar que a versão é desatualizada, o cliente requisita uma atualização de seu arquivo (PATE, 2003; COULOURIS et al., 2011).

2.3.3 *Tahoe Least Authority File System* (Tahoe-LAFS)

Tahoe-LAFS é um SAD desenvolvido para prover armazenamento seguro de dados por longos perídos de tempo, assim como em operações de *backup*. Utiliza o princípio da menor autoridade (MILLER, 2006), em que usuários ou processos possuem o mínimo

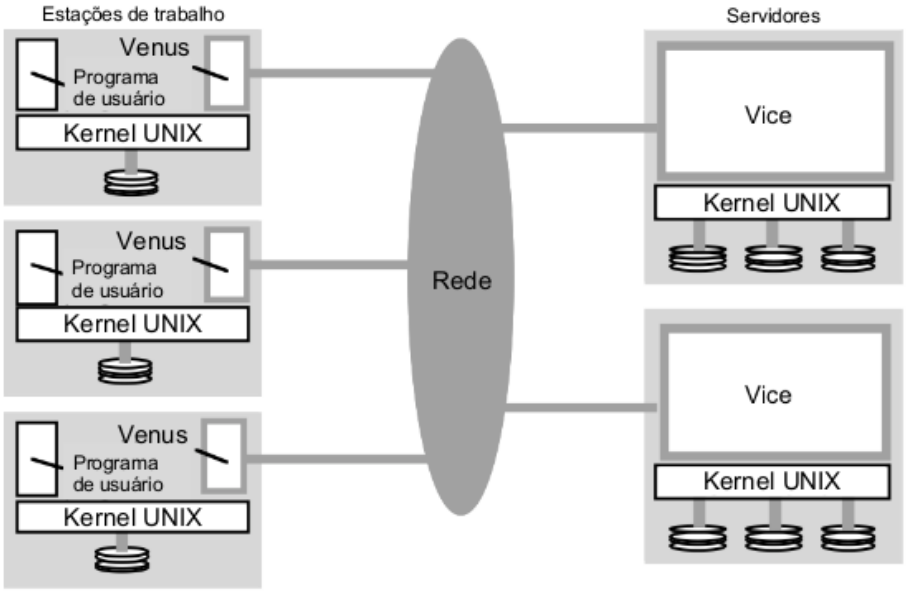

Figura 2.2: Arquitetura do AFS (FERNANDES, 2012), adaptado de (COULOURIS et al., 2011)

de privilégio necessário para completar suas requisições, tornando-se independentes de administradores. Os principais componentes do sistema são: clientes, servidores de armazenamento (*storage nodes*) e um componente central chamado *introducer*. Os *storage nodes* são os nós/servidores de armazenamento de dados. O *introducer* é o nó principal, em que as informações de conexão de todos os nós de armazenamento são mantidas. Para acessar o sistema de arquivos os clientes devem obter, através do *introducer*, as informações necessárias para se conectarem aos nós de armazenamento. Seu controle de acesso é realizado através de uma pequena *string* de *bits* que identifica arquivos ou diretórios e não há *cache* no sistema (WILCOX-O'HEARN; WARNER, 2008).

A tolerância a falhas é assegurada pelo algoritmo de *erasure coding* (FUJIMURA; OH; GERLA, 2008; PLANK et al., 2009), um método para correção de erros na transmissão de dados, e pela forma em que as partes do arquivos são distribuídas nos servidores. Na operação de escrita, os dados, após passarem por processos de criptografia e *hashing*, são codificados utilizando o algoritmo *Reed-Salomon*, em que o usuário define um parâmetro N, tal que N é o número de servidores de dados e metadados (informações do arquivo), e K, que é o número de servidores necessários para a leitura do arquivo. Os valores desses parâmetros devem seguir os intervalos $1 \leq N \leq 256$ e $1 \leq K \leq N$. Como padrão, Tahoe utiliza N=10 e K=3, ou seja, para cada arquivo escrito são necessários 10 nós de armazenamento, e para remontar o arquivo original são necessários apenas 3 deles.

Uma representação de sua arquitetura pode ser vista na figura 2.3.

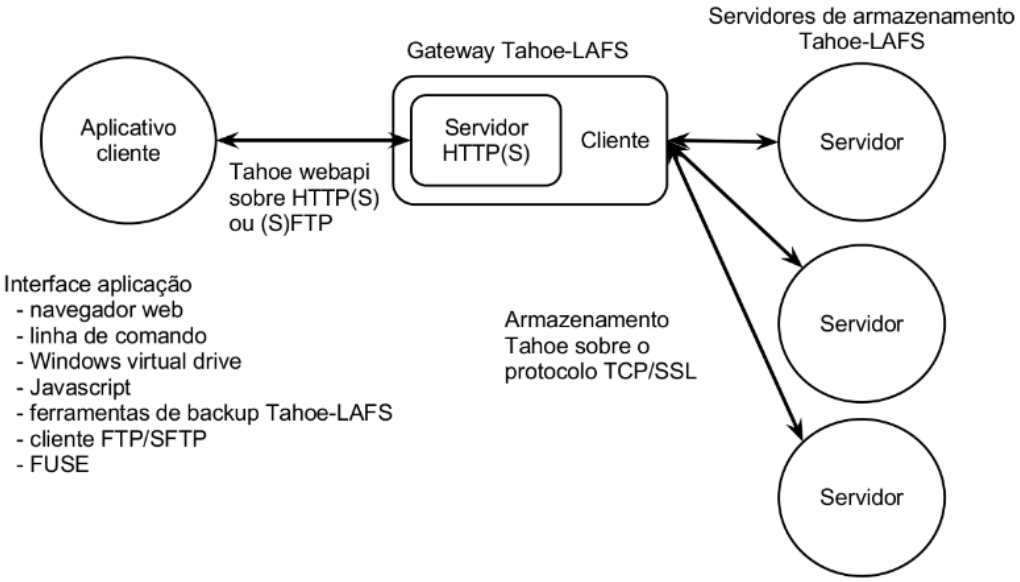

Figura 2.3: Arquitetura do Tahoe-LAFS (FERNANDES, 2012), adaptado de (WILCOX-O'HEARN; WARNER, 2008)

2.3.4 *Google File System* (GFS)

O *Google File System* (GFS) foi projetado de acordo com as necessidades específicas observadas durante o funcionamento dos sistemas do Google. É SAD apropriado para ambientes nos quais há a manipulação intensa de arquivos demasiadamente grandes. Sua arquitetura é baseada em *clusters* de servidores que possuem *hardware* de baixo-custo e, dada essa característica, o monitoramento de falhas, detecção de erros e recuperação automática do sistema devem ser constantes ((MCKUSICK; QUINLAN, 2010)).

Um *cluster* GFS é composto por três elementos essenciais: um servidor (GFS *master*), múltiplos servidores de porção (GFS *chunkserver*) e múltiplos clientes (GFS *client*). O GFS é executado no espaço de usuários do sistema operacional Linux. A ideia básica de funcionamento é que os arquivos sejam divididos em porções de tamanho fixo e armazenados nos servidores de porção como sendo arquivos do sistema Linux. A replicação é utilizada como meio principal de tolerância a falhas. As porções (*chunks*) armazenadas são replicadas para outros servidores de porção (a configuração padrão do número de réplicas é de 3 para cada porção). Os servidores de porção não necessitam de *cache* pois os arquivos mais frequentemente acessados são mantidos na memória pelo próprio sistema operacional.

A arquitetura do GFS é representada na Figura 2.4

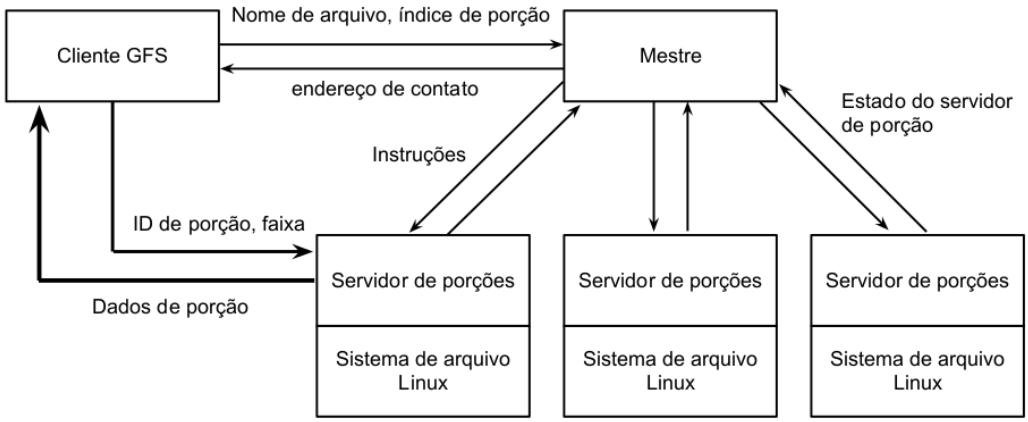

Figura 2.4: Arquitetura do GFS (FERNANDES, 2012), adaptado de (TANENBAUM; STEEN, 2007)

O servidor *master* centraliza as tomadas de decisão do sistema e controla o estado de cada *chunkserver*. É ele quem armazena os metadados: identificação (*namespaces*) de arquivos e porções, mapeamento dos arquivos para as porções e a localização das réplicas das porções. O mapeamento das porções e a localização nas réplicas são armazenados em um arquivo de *log* persistente no disco do servidor *master* e replicado para os servidores de porção. Se houver falhas no *master*, a localização nas réplicas pode ser restaurada, evitando-se problemas de inconsistência.

O modelo de consistência do GFS prevê que operações sob os *namespaces* de arquivos, como criação ou exclusão, sejam feitas de forma atômica, com o servidor *master* gerenciando as operações e utilizando travas e correções para garantir a consistência. Para suportar tal modelo de consistência, as aplicações do GFS realizam operações de *append* (anexar) para modificar os arquivos (GHEMAWAT; GOBIOFF; LEUNG, 2003).

A arquitetura inicial do GFS sofreu uma evolução, descrita por (MCKUSICK; QUINLAN, 2009). Conforme as análises das estimativas sobre a vazão de dados nos sistemas do Google, o GFS precisava aumentar ainda mais sua escalabilidade. Sendo assim, a versão inicial passou por reformulações que alteravam o tamanho dos blocos das porções e a forma como ocorria a E/S (entrada e saída) de dados. Além disso, devido à limitação de um único servidor *master* gerenciando todas as requisições, o novo GFS possui servidores para os metadados.

2.3.5 *Hadoop Distributed File System* (HDFS)

O *Hadoop Distributed File System* (HDFS) é o módulo responsável pelo armazenamento de arquivos do *framework* de nome *Hadoop*, desenvolvido pela *Apache* para gerenciar grandes porções de dados. Este SAD possui código aberto e utiliza *clusters* de servidores para garantir alta escalabilidade e desempenho. Assim como o GFS

(GHEMAWAT; GOBIOFF; LEUNG, 2003), os metadados e dados são armazenados em servidores distintos. Os metadados, constituídos de identificadores de arquivos, diretórios e mapeamento das porções (e réplicas) para os arquivos, permanecem em um servidor dedicado chamado *NameNode*. Os dados são distribuídos e replicados em *DataNodes*. Em cada *cluster* de *DataNodes* há apenas um *NameNode*.

O cliente, no ato do envio, serializa os arquivos em blocos de 128 *megabytes* e requisita ao *NameNode* a localização de três *DataNodes* para armazenar as réplicas. As porções de dados são enviadas aos servidores através de *pipelines*.

Na leitura, o cliente solicita ao *NameNode* as localizações das porções do arquivo, o servidor retorna a localização dos *DataNodes* mais próximos do cliente para que a operação possa ser concluída.

Como um mecanismo de detecção de falhas, os *DataNodes* enviam ao *NameNode* mensagens chamadas de *heartbeats* com a função de confirmar que o servidores estão em plena operação. Os *DataNodes* utilizam heartbeats para confirmar o recebimento das porções para *NameNode*. O intervalo padrão entre cada mensagem é de 3 segundos. Caso decorram 10 minutos sem confirmação, o *DataNode* é tido como inoperante e então o *NameNode* agenda a criação de novas réplicas para os blocos contidos no servidor caído. O *NameNode* utiliza *heartbeats* para funções, como requisitar a replicação de dados para outros servidores, registrar novos servidores, entre outros (SHVACHKO et al., 2010).

O funcionamento do envio no HDFS é representado na figura 2.5.

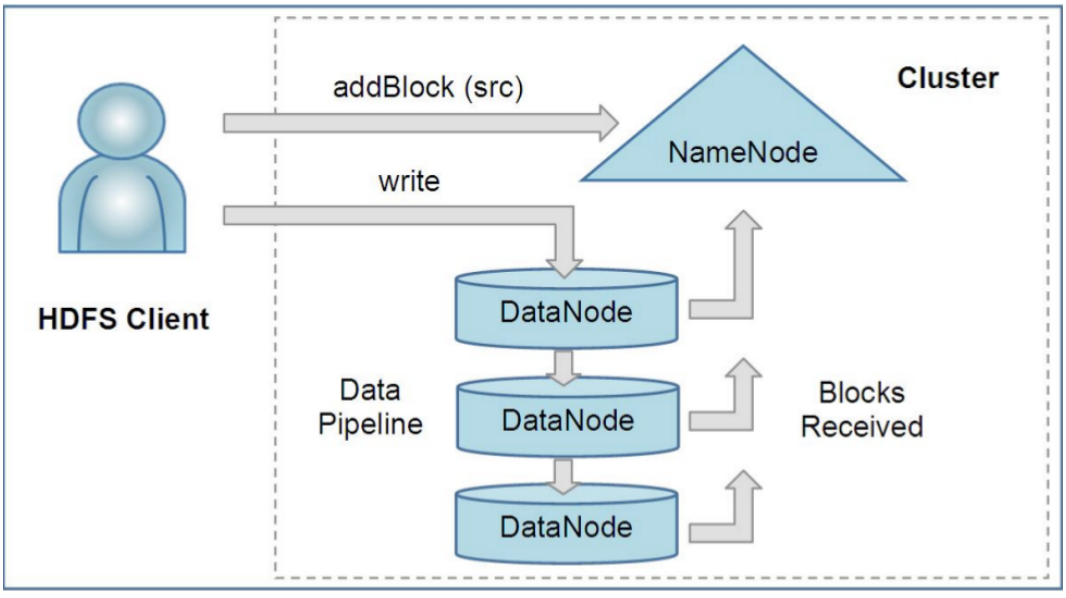

Figura 2.5: Mecanismo de envio no HDFS (SHVACHKO et al., 2010)

2.3.6 *Ceph Distributed File System*

Ceph é um SAD que utiliza um *cluster* dinâmico de metadados, replicação de dados e detecção e recuperação de falhas para prover alta escalabilidade, confiabilidade e desempenho. Os arquivos gerenciados são da ordem de *Petabytes*, exigindo algoritmos e estratégias para distribuir a carga de dados sobre os servidores.

Este SAD separa o gerenciamento de dados e metadados e sua arquitetura é composta por clientes, um *cluster* de OSDs (*Object Storage Devices*) e um *cluster* de MDS (*Metadata Storage*). Os OSDs são responsáveis pelo armazenamento de dados e metadados de arquivos, bem como operações que envolvem a entrada e saída de dados. O *cluster* de MDS gerencia os *namespaces* dos arquivos (nomes e diretórios) afim de manter a consistência e a segurança do sistema. A arquitetura do *Ceph* é representada na figura 2.6.

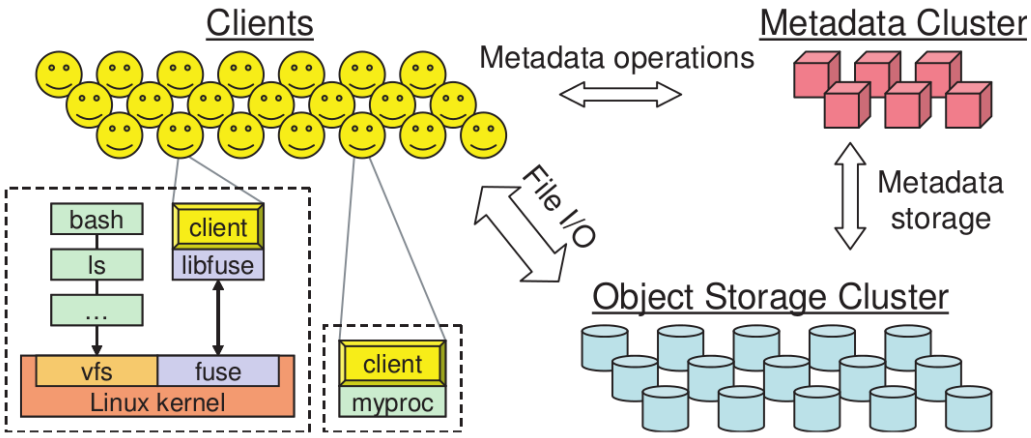

Figura 2.6: Arquitetura do *Ceph Distributed File System* (WEIL et al., 2006)

Dado o grande volume de acesso aos servidores de metadados, o algoritmo *Dynamic Subtree Partioning* distribui as operações de metadados entre os MDFs para utilizar melhor os recursos computacionais de todos os computadores do *cluster*.

Para evitar desbalanceamento na utilização de servidores de dados, *Ceph* usa um algoritmo que distribui novos dados aleatoriamente nos ODSs. Uma porção dos dados existentes também são re-distribuídos uniformemente para novos servidores.

Para a replicação, *Ceph* mapeia os objetos em *placement groups* (PGs). O algortimo CRUSH (*Controlled Replication Under Scalable Hashing*) faz o mapeamento dos PGs em uma lista ordenada de OSDs que armazenam os dados replicados. No envio de arquivos, o cliente escolhe o primeiro OSD funcional (o servidor primário) e faz o *upload*. O servidor, após receber os dados, gera outra versão para o objeto e envia os dados para os servidores OSDs escolhidos para armazenarem as réplicas. Ao terminar a replicação, os clientes são avisados que os dados foram completamente replicados.

Falhas como dados corrompidos e erros em disco são reportadas pelos próprios

OSDs. *Ceph* utiliza monitores para rastreiar a rede ativamente em busca de erros (servidores que não respondem), bem como realizar eleições de servidores de réplicas em primários. Essas características ajudam a manter a consistência e a disponibilidade dos dados no *cluster* (WEIL et al., 2006).

2.3.7 FlexA

Segundo (FERNANDES, 2012), o Sistema de Arquivos Distribuído Flexível e Adaptável (***Flex****ible and* ***A****daptable Distributed File System* - FlexA), foi concebido com base nas características herdadas dos sistemas de arquivos distribuídos *Network File System*, *Andrew File System*, *Tahoe-LAFS* e *Google File System*.

Este SAD possui uma arquitetura diferente do modelo cliente-servidor, em que há a presença de um servidor principal. Sua arquitetura é composta por estações que podem pertencer a três grandes grupos: Grupo dos Clientes, Grupo de Escrita (Servidores Primários) e Grupo de Réplicas (Servidores Secundários). O Grupo dos Clientes são os próprios usuários que fazem as requisições para o sistema. Os servidores primários armazenam dados e metadados, assim como servidores secundários armazenam dados e metadados replicados a partir dos primários. O sistema FlexA tem como objetivo utilizar o mínimo possível dos recursos de *hardware* dos servidores, deixando-se tarefas como serialização, criptografia e compressão de dados a cargo dos clientes. Com isso, FlexA se adapta às variações de *hardware* das estações dos clientes, reduzindo ou aumentando o desempenho do sistema. A figura 2.7 ilustra a arquitetura do sistema.

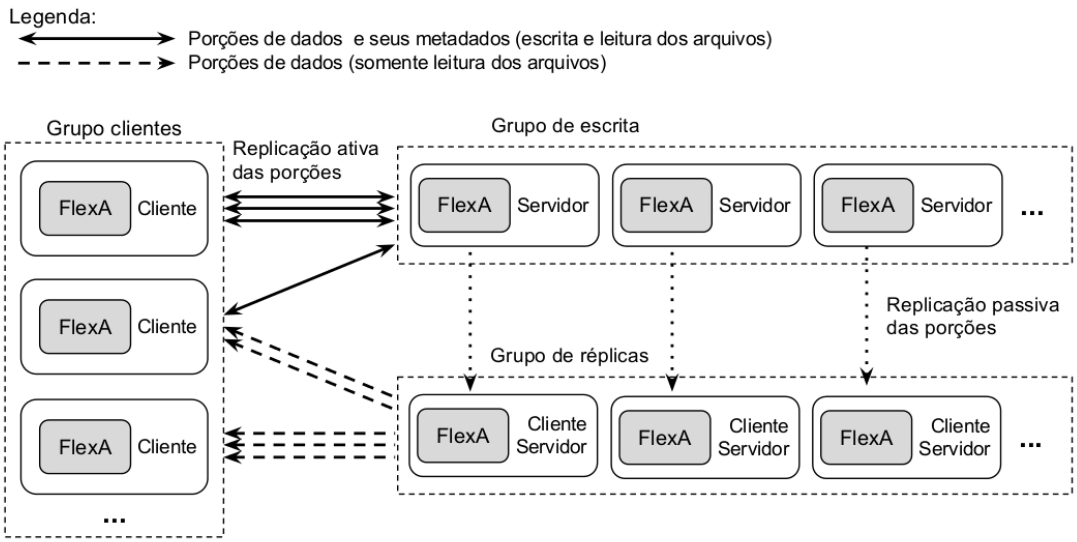

Figura 2.7: Arquitetura do FlexA (FERNANDES, 2012)

O sistema inicial, descrito por (FERNANDES, 2012), apenas mencionava o grupo de réplicas, sem implementá-lo. Para a redundância e a confiabilidade do sistema e dos

dados, os arquivos eram serializados em exatamente três porções de tamanhos iguais e duas delas eram enviadas para cada servidor primário. Desse modo, havia a garantia de que se um primário falhasse, o cliente poderia requisitar as porções para os outros dois pois cada servidor possuía 2/3 do arquivo. A representação da distribuição das porções consta na figura 2.8. A quantidade de servidores possíveis era no mínimo duas, para a leitura, e no máximo três, para a escrita.

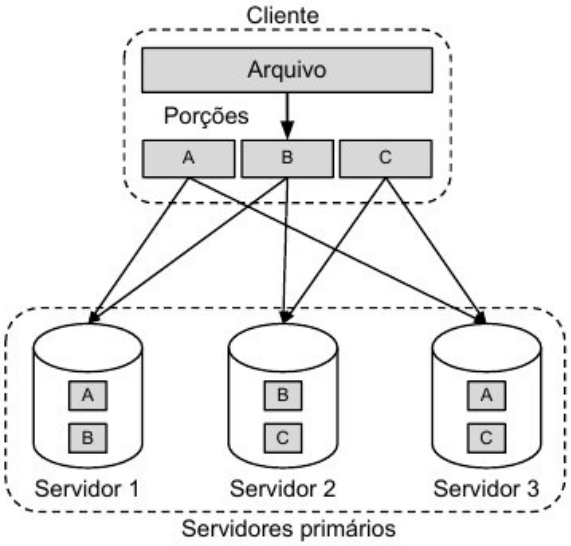

Figura 2.8: Distribuição das porções no FlexA (FERNANDES, 2012)

O FlexA é composto por 3 módulos: Coletor, Comunicador e Sincronizador.

- Coletor - É o modulo que representa o "serviço" do FlexA nos servidores. Tem o intuito de analisar as informações recebidas em clientes e/ou servidores.
- Comunicador - Cria o arquivo de configuração e rastreia ativamente a rede em busca de servidores, criando um arquivo com suas localizações.
- Sincronizador - É o módulo responsável pelo endereçamento e envio de mensagens.

FlexA utiliza um mecanismo de *cache* semelhante ao do AFS. Uma vez que um arquivo compartilhado é alterado, os clientes que possuem o arquivo são avisados sobre a validade dos dados e decidem se querem atualizar o arquivo (FERNANDES, 2012).

2.4 Considerações finais

Neste capítulo apresentou-se a fundamentação teórica sobre SDs e SADs, destacando-se como os principais SADs existentes lidam com os problemas de sincronia, consistência e falhas, afim de extrair informações que agreguem melhorias ao FlexA. O capítulo seguinte descreve o desenvolvimento das atividades efetuadas no FlexA.

Capítulo 3

Descrição e desenvolvimento do projeto

Este capítulo descreve as atividades desenvolvidas neste trabalho. A seção 3.1 apresenta as mudanças realizadas na arquitetura original do FlexA, que permitiram uma nova forma de disponibilidade, ocasionando a necessidade de sincronização das porções entre o grupo de escrita e réplicas. Com as réplicas em funcionamento, a seção seguinte aborda como são tratados os problemas de consistência entre versões de arquivos replicados. Ao fim, são descritos os mecanismos de detecção de falhas nos servidores primários e secundários.

3.1 Sincronização das porções entre o grupo de escrita e réplicas

A arquitetura original do FlexA não implementava grupo de réplicas e serializava arquivos de forma estática, dividindo-os em três porções e enviando a cada servidor do grupo de primários exatamente duas porções, como descrito na seção 2.3.7. O desenvolvimento do grupo de réplicas resultou em alterações nas operações de escrita, leitura e na quantidade de porções por arquivo e servidores primários. Tais mudanças são coerentes com o proprósito inicial do FlexA descrito por (FERNANDES, 2012): o sistema deve fornecer ajuste de suas funções de acordo com as necessidades do ambiente.

A nova estrutura faz com que cada cliente divida os arquivos em porções de forma dinâmica, dependendo do tamanho do arquivo original e da quantidade de servidores primários ativos. Também foi definido que cada servidor primário recebe apenas uma porção, deixando a parte de redundância de dados para os servidores secundários (ou servidores de réplicas).

A nova arquitetura do FlexA, bem como a descrição das mudanças ocorridas no sistema estão nas próximas subseções. A instalação e configuração deste novo sistema é apresentada no apêndice A.

3.1.1 A nova arquitetura do FlexA

A redundância dos dados, antes vinculada ao grupo de servidores primários, passou a ser do grupo de servidores secundários. A estratégia de replicação das porções recebidas pelos servidores primários, além de permitir uma maior durabilidade dos dados em caso de falhas, adiciona a possibilidade de se obter arquivos de servidores menos sobrecarregados e mais próximos ao cliente. O módulo desenvolvido que permite ativar o serviço referente a um servidor de secundário é chamado *coletor_réplica*.

A nova arquitetura, que dá suporte ao uso de réplicas, é representada na figura 3.1.

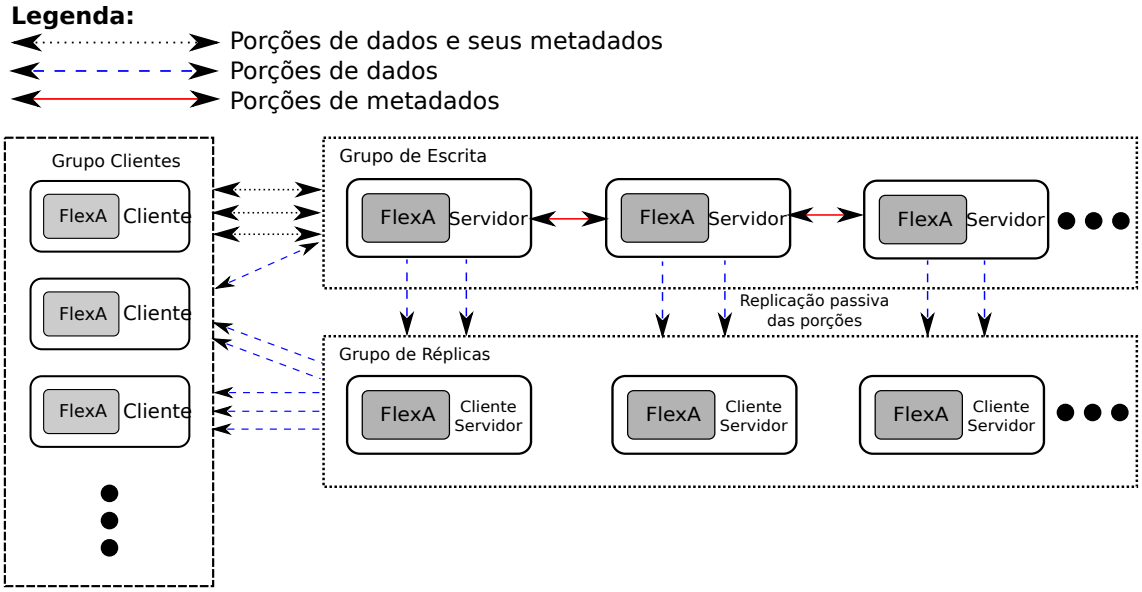

Figura 3.1: Nova arquitetura do FlexA

3.1.2 Alterações na operação de escrita

O sistema FlexA possui duas operação de escrita: *upload* e *update*. A operação de *upload* indica quando um arquivo é submetido aos servidores pela primeira vez. No *update* ocorre apenas a atualização de um arquivo já existente nos servidores.

A operação de *upload*, adaptada ao propósito das réplicas, pode ser descrita pelos seguintes passos:

1. O cliente serializa o arquivo, escolhe aleatóriamente os servidores primários que receberão os dados e envia apenas uma única porção para cada um.

2. Os servidores primários que receberam as porções dos arquivos, por sua vez, realizam a replicação para 2 servidores secundários, por padrão, e armazenam os metadados com a localização das porções replicadas. A escolha de quais servidores secundários receberão as porções é feita por um cálculo de métricas que envolve o espaço em disco disponível e o tempo de resposta da rede.

3. Para garantir que todos os servidores primários tenham a localização de todas as porções enviadas para os secundários, os metadados dos arquivos replicados são sincronizados entre os primários. Tal sincronismo garante que os clientes, na operação de leitura/*download*, obtenham de qualquer um dos primários os metadados atualizados e possam assim utilizar as réplicas para a leitura de dados.

Para coordenar o processo de sincronização de metadados entre os primários implementou-se uma adaptação do algoritmo de Eleição de Líder de Anel. A sincronização de metadados, com duas réplicas em funcionamento, é feita em duas fases. A primeira fase consiste na eleição de um líder e construção da mensagem que contém os metadados com a localização completa das porções replicadas. Sua execução é mostrada na figura 3.2, através dos seguintes passos:

1. Antes de enviar um arquivo, o cliente elege aleatóriamente um servidor primário líder e anexa, no cabeçalho da mensagem de *upload*/*update*, a indicação para o líder que dará início ao sincronismo.

2. O primário líder atribui valor falso à variável metadados_completos, que controla quando a mensagem com os metadados de localização das réplicas está completa.

3. Após ter enviado as porções para as réplicas e armazenado o local dos dados replicados, o líder envia os metadados para o primário sucessor no anel. Isto se repete até que a mensagem com os metadados completos retorne ao líder.

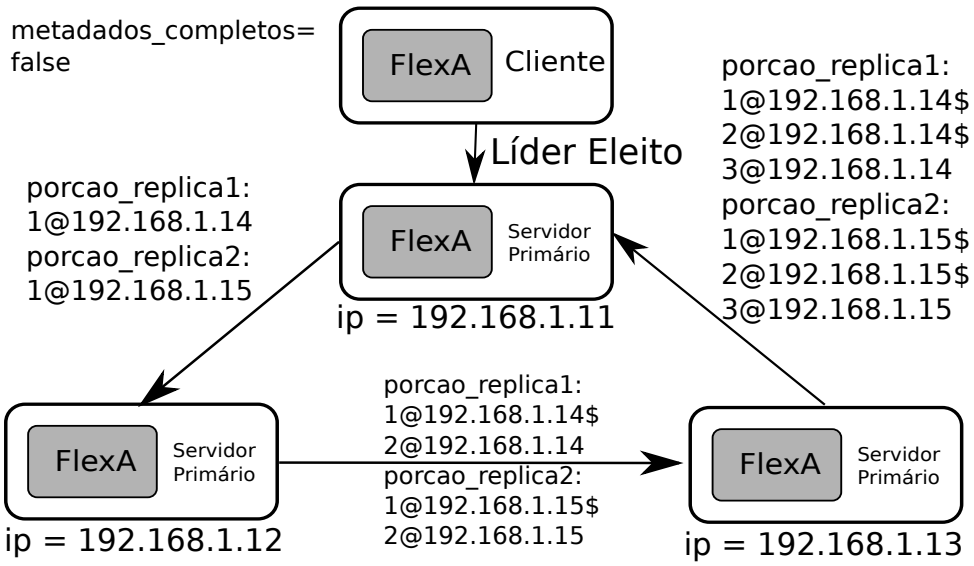

Figura 3.2: Primeira fase do algoritmo de sincronização de metadados nos primários

Na segunda fase, com a localização completa dos dados replicados, o servidor líder atribui valor verdadeiro à variável metadados_completos e envia os metadados até o nó anterior a ele no anel. A segunda fase é representada na figura 3.3.

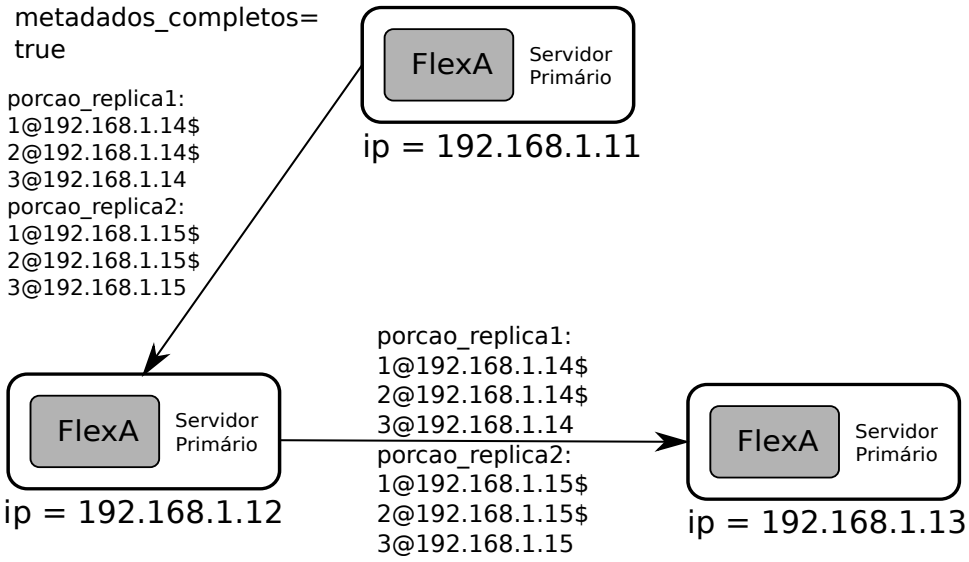

Figura 3.3: Segunda fase do algoritmo de sincronização de metadados nos primários

A atualização de arquivos (*update*) é realizada de forma semelhante ao *upload*, porém, durante a sincronização de metadados, caso haja algum primário com a versão desatualizada do arquivo, este primário faz uma requisição ao servidor que possui a versão mais atualizada. Antes de enviar as porções para réplicas, os servidores primários verificam a localização das porções replicadas e enviam suas novas versões para a mesma localização. Tais mecanismos evitam que os servidores primários e de réplicas contenham versões diferentes de arquivos.

3.1.3 Alterações na operação de leitura

A operação de *download* também foi modificada para dar suporte às replicas, já que inicialmente os arquivos eram lidos apenas dos servidores primários. Agora ela ocorre da seguinte forma:

1. O cliente solicita para algum dos servidores primários os metadados com a localização das porções do arquivo requisitado.

2. O servidor primário consulta seu banco de dados e retorna uma mensagem com a localização das porções armazenadas nos primários e nas réplicas. Além disso, o cliente calcula uma métrica com o tempo de resposta de cada servidor.

3. Para cada porção, o serviço FlexA no cliente escolhe a localização do servidor, seja ele primário ou réplica, para realizar a requisição de leitura. A escolha é dada pelo menor tempo de resposta.

4. Por fim, os servidores enviam as porções dos arquivos para o cliente.

A representação de funcionamento do *download* encontra-se na figura 3.4.

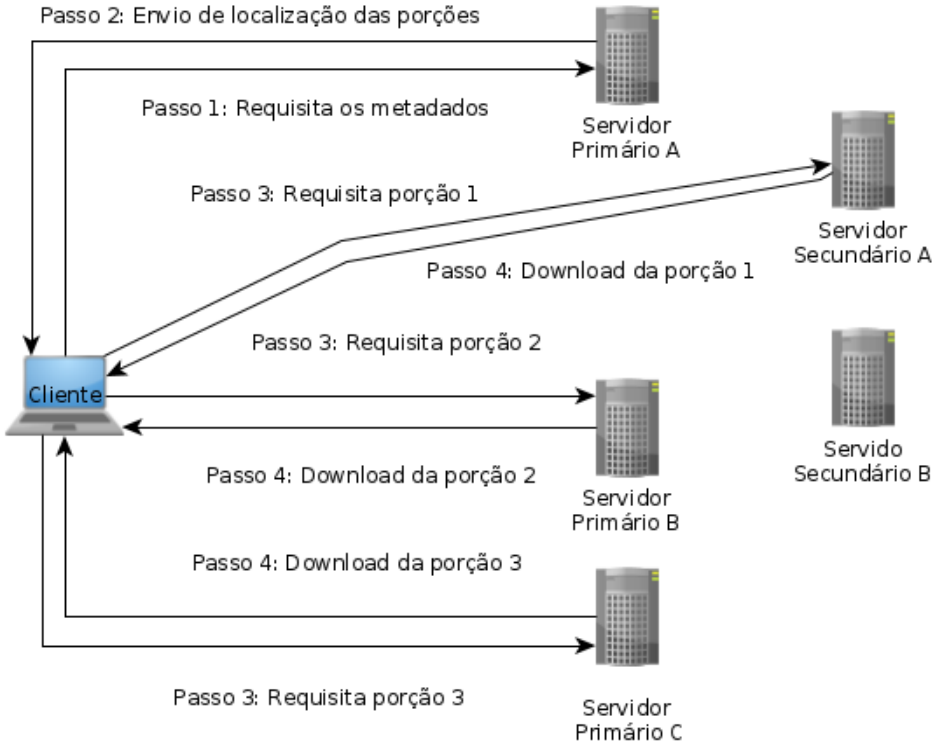

Figura 3.4: *Download* de um arquivo no FlexA

3.1.4 Ajuste da quantidade de computadores do grupo de escrita

Houve a flexibilização da quantidade de servidores primários utilizáveis no FlexA. A arquitetura original suportava no máximo três servidores no grupo de escrita. Com a estrutura nova, o sistema funciona com o mínimo de 3 servidores primários, mas não há mais limite para número de servidores possíveis no sistema. Caso a quantidade de servidores primários exceda o número de porções serializadas do arquivo, os servidores que não receberem as porções são atualizados apenas com os metadados dos arquivos.

Para incluir novos servidores no sistema basta rastrear a rede com o módulo Comunicador e os novos componentes serão automaticamente detectados e configurados para serem utilizados.

3.1.5 Ajuste do número de divisões por arquivo

Com as réplicas implementadas para redundância e apoio na distribuição de arquivos, não foi mais necessário o envio de duas porções para cada servidor. Além disso, a

divisão de arquivos pequenos em três porções nem sempre é eficiente. Assim, alguns critérios foram selecionados para ajuste no número de porções por arquivo. Dentre os estudados, os que mais se adequavam ao propósito deste projeto foram: tamanho do arquivo e quantidade de servidores disponíveis. Assim como no GFS, a ideia é trabalhar com arquivos grandes, sendo que não há a necessidade do cliente dividir arquivos pequenos demais. Na quantidade de divisões por arquivo foi definido um limite de 10MB para o tamanho de arquivo. Sendo assim, há duas situações possíveis:

- Arquivos menores ou iguais a 10MB não são divididos: o arquivo inteiro é enviado para o primeiro primário listado dentre os ativos, enquanto para os outros ocorre o envio de metadados. A representação do envio consta na figura 3.5.
- Arquivos maiores que 10MB: particionados em três porções e é enviada apenas uma única porção, em ordem, para cada um dos três primários listados. Este tipo de envio é representado na figura 3.6.

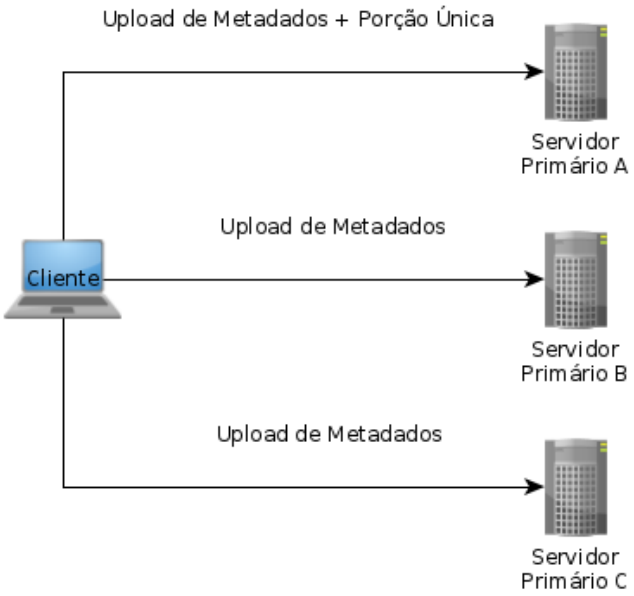

Figura 3.5: *Upload* de um arquivo menor ou igual a 10MB

Quando há um número maior de primários ativos do que a quantidade de porções, ocorre o envio de metadados para os servidores primários que não receberam as porções.

Na replicação, todos os primários que receberam as porções, enviam dados e metadados para duas réplicas, por padrão, escolhidas pela quantidade de espaço livre em disco e a latência de suas conexões. A figura 3.7 mostra a replicação de porções para arquivos menores que 10MB e a figura 3.8 para arquivos maiores que 10MB.

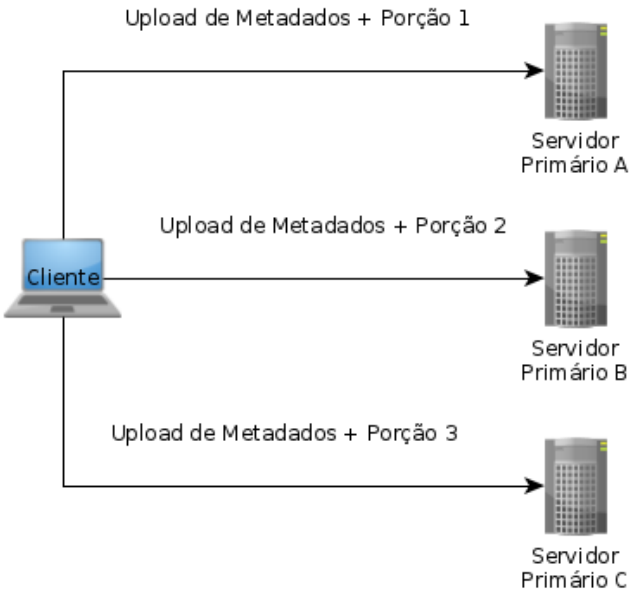

Figura 3.6: *Upload* de um arquivo maior que 10MB

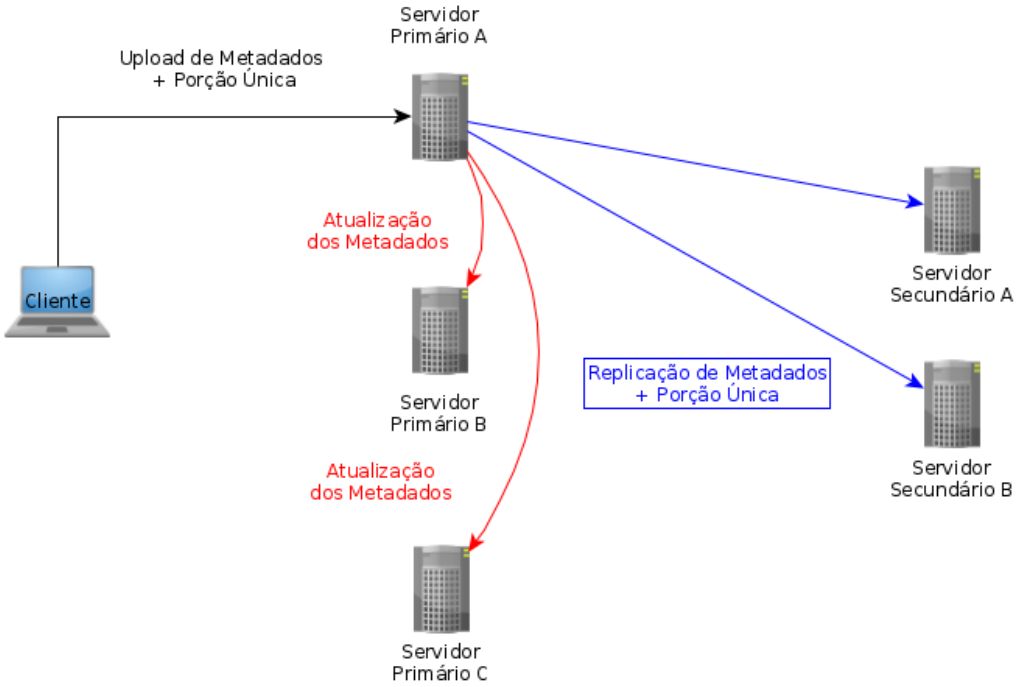

Figura 3.7: Replicação de arquivos menores que 10MB

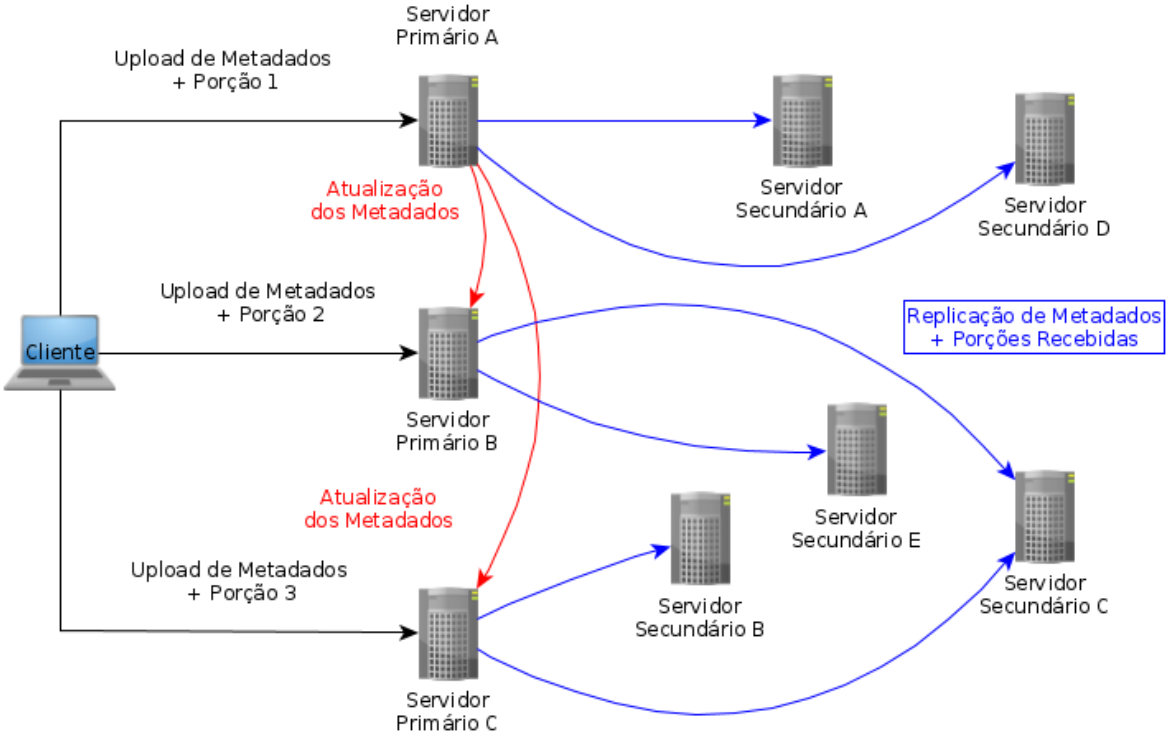

Figura 3.8: Replicação de arquivos maiores que 10MB

3.2 Manutenção da consistência entre as porções replicadas

Esta atividade possuía como objetivo a construção de mecanismos que garantissem a consistência entre as porções replicadas no sistema. O grupo de réplicas do FlexA foi projetado para adquirir um modelo de consistência que realiza operações atômicas sob os dados replicados, ou seja, a atualização das porções são sempre repassadas para todas as cópias existentes no sistema.

Para garantir que os dados replicados estivessem em suas últimas versões, inicialmente foi necessário definir o modo que a versão dos arquivos seria identificada e manipulada no sistema. Após isso, pensou-se na possibilidade de um servidor de réplica falhar e, durante o período de indisponibilidade do serviço, ocorrerem atualizações na versão dos dados contidos na réplica inativa. Tal possibilidade levaria a problemas de inconsistência, sendo necessário a sincronização da versão dos dados replicados antes do serviço Coletor dos servidores secundários voltar ao funcionamento.

A implementação da variável que controla as versões dos arquivos e o problema da sincronização de uma réplica desatualizada são descritos a seguir.

3.2.1 Identificação e atualização da versão dos arquivos

A primeira mudança realizada para possibilitar a identificação da versão dos arquivos foi a inserção de um campo *id_versao* no banco de dados dos servidores. Este campo se comporta como um contador de versões, tal que seu valor é inteiro e maior ou igual a zero. A primeira escrita de um arquivo (operação de *upload*), atribui valor zero ao campo. A cada atualização (*update*), os servidores primários incrementam a versão do arquivo em 1 e repassam os dados e metadados com a versão mais atual para as réplicas.

3.2.2 Sincronização ao iniciar o módulo Coletor dos servidores secundários

A manutenção da consistência no caso de um servidor de réplicas falhar e depois voltar à atividade é realizada da seguinte forma:

1. Ao iniciar o módulo Coletor da réplica, o processo verifica se já existe o arquivo de banco de dados. Caso o banco de dados não exista, significa que é a primeira execução da réplica, então ele é criado. Caso contrário, o processo da réplica já foi iniciado alguma vez e pode ser que haja inconsistência nos arquivos armazenados.

2. Quando o banco está presente é feita a varredura das informações dos arquivos e para cada um monta-se uma mensagem com os parâmetros de identificação do arquivo, porção armazenada no servidor primário e versão.

3. Cada cópia na réplica possui uma entrada no banco de dados com a localização referente à porção original no primário. Sendo assim, as mensagens com as in-

formações das porções são enviadas por *threads* para o primário que contém o arquivo original.

4. O primário recebe a mensagem e compara a versão da réplica com a que se encontra em seu banco de dados. São três as possibilidades:
   - Número da versão do arquivo na réplica menor que no primário (porção desatualizada).
   - Versão do arquivo na réplica igual a do primário (porção atualizada).
   - Não existe entrada no banco de dados (porção não existe pois o arquivo foi apagado).
5. Para as porções desatualizadas, o servidor primário envia dados e metadados da versão mais atual para as réplicas.

   Para as porções atualizadas é enviada uma mensagem apenas confirmando que a versão é a mais atual.

   E por fim, para porções que não existem mais nos primários, o servidor envia uma mensagem para a réplica apagar os dados e metadados.

3.3 Detecção de falhas nos servidores primários e secundários

Esta seção descreve os mecanismos utilizados para a detecção de falhas nos serviços do sistema, que icluem basicamente o módulo Coletor dos servidores primários e secundários. A detecção das falhas contemplou dois tipos de problemas: o processo do módulo Coletor parar sua atividade (ser fechado) e o caso de um servidor primário ser desligado ou estar com problemas na rede.

3.3.1 *Daemon* de vericação do módulo Coletor

Para verificar a atividade do módulo Coletor nos servidores primários ou secundários foi criado um *daemon* de nome *checa_processo*. Segundo (TANENBAUM, 2007), um *daemon* é um processo em *background* (segundo plano), que apresenta uma função específica. Neste caso, o processo *coletor_servidor* (referente ao servidor primário) ou *coletor_replica* (referente ao servidor secundário) é procurado na lista de processos ativos do sistema de acordo com um intervalo de 10 segundos entre as buscas. Caso o processo não tenha sido encontrado, o *daemon* encarrega-se de reativar o módulo automaticamente.

3.3.2 Sondagem entre servidores primários

A sondagem entre os servidores primários foi desenvolvida para detectar quando um primário está inativo por queda ou quando não consegue comunicar-se por problemas na rede.

Os primários, após rastrearem a rede com o módulo Comunicador e preencherem a localização dos servidores no arquivo *hosts.dat*, enviam mensagens de *ping* para a lista de servidores ativos encontrados, tal como representado na figura 3.9. Cada primário, ao enviar uma mensagem de *ping* aos servidores listados no *hosts.dat*, cria uma lista com o endereço de IP (*Internet Protocol*) dos nós sondados. Caso dê tempo esgotado (*timed out*) no *ping* de algum nó, o campo listado referente ao servidor é substituído por NULL. Ao fim de cada sondagem da lista de IPs no *hosts.dat* é comparada a lista de IPs criada com a lista esperada.

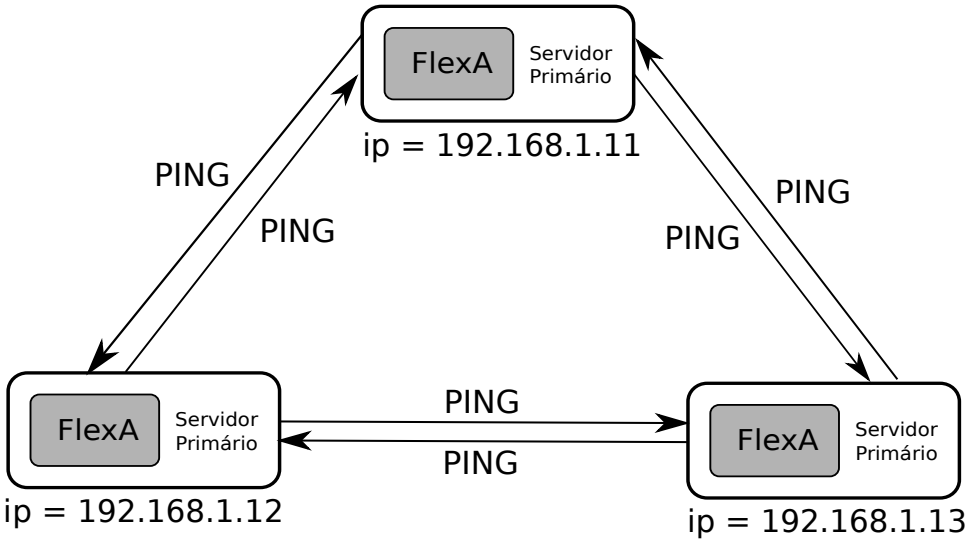

Figura 3.9: Sondagem entre primários

Quando há falhas e ocorre *timed out* no *ping* de algum servidor, o primário que identificou o problema para a sondagem e envia para os servidores remanescentes um pedido de verificação do servidor dado como inativo. Este pedido tem como finalidade detectar se realmente houve queda no servidor primário ou foi algum problema na rede ao qual eles estavam conectados. O pedido de sondagem é representado na figura 3.10.

Os primários ativos que receberam o pedido aguardam um intervalo de tempo de 2 segundos para que o primário inativo tenha a chance de reestabelecer sua atividade. Após isso é enviado um *ping* que verifica se de fato houve a queda. Caso tenha sido constatada a queda, é convocada uma eleição que escolhe um servidor secundário para tornar-se o novo primário. A queda de um dos primários é representada na figura 3.11.

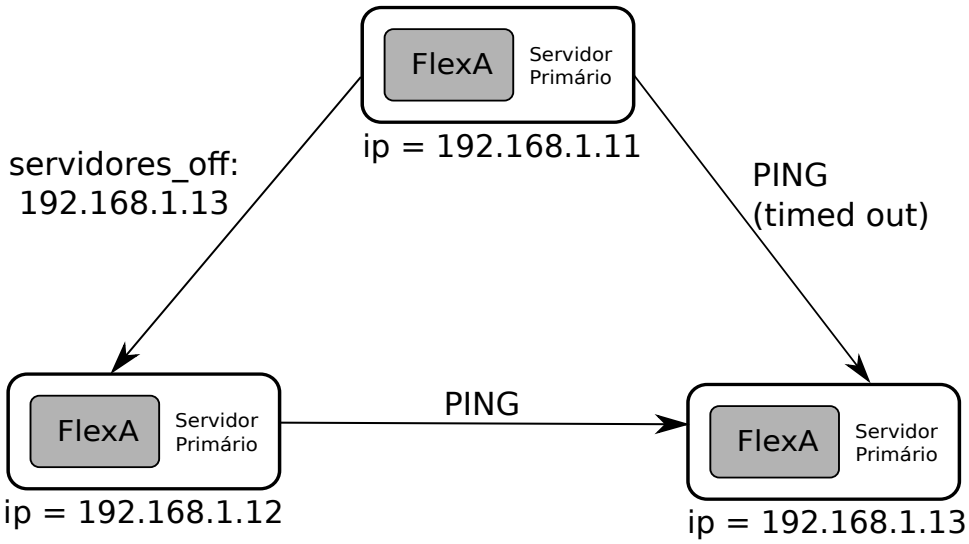

Figura 3.10: Pedido de verificação do primário inativo

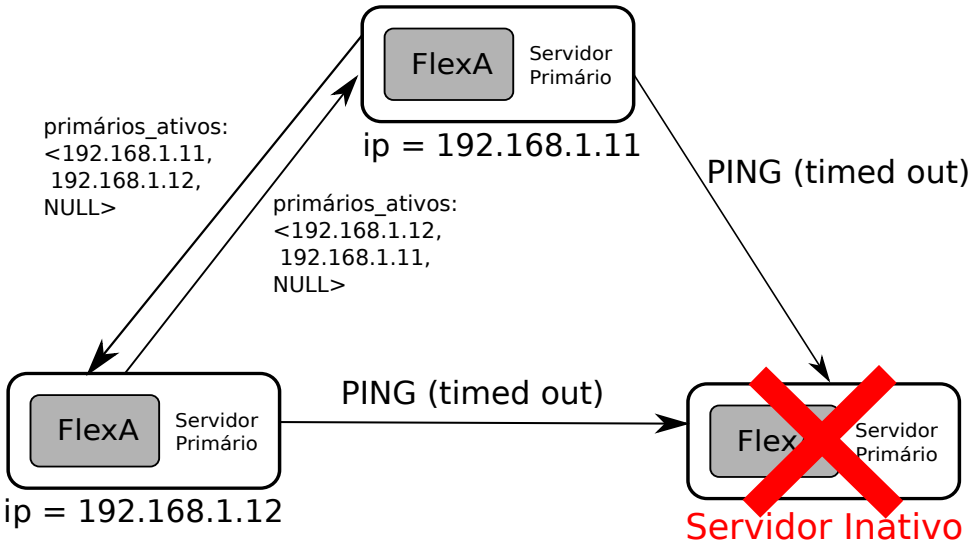

Figura 3.11: Confirmação de queda do primário

3.4 Considerações finais

Este capítulo teve como finalidade descrever a mudança na estrutura do FlexA e a construção dos mecanismos de sincronização entre grupo de escrita e réplicas, consistência nos dados replicados e detecção de falhas nos primários, propostos por este trabalho. O próximo capítulo mostra os testes feitos para validar essas implementações.

Capítulo 4

Testes e Resultados

O objetivo deste capítulo é apresentar os testes para validação das implementações desenvolvidas no FlexA. Em cada teste evidenciam-se os critérios de avaliação, as ferramentas utilizadas e os resultados coletados. Inicialmente serão apresentados os testes de desempenho, que consistiram em coletar e comparar as taxas de transferência nas operações de escrita e leitura da versão desenvolvida do FlexA e de outros SADs. Logo após, apresenta-se o teste de validação e desempenho para a identificação de falhas entre os primários.

4.1 Ambiente de testes

Os testes foram realizados no *cluster* do GSPD. O ambiente real é composto por 16 máquinas com duas configurações de *hardware* distintas. Oito nós possuem processadores Intel Pentium Dual E2180 - 1,8Ghz, 2GB de memória RAM e 40GB de HD. Os demais contam com processadores Intel Core i7-3770 - 3.4Ghz, 16GB de memória RAM e 500GB de HD. Nestas máquinas foram instaladas 4 máquinas virtuais em cada nó, totalizando 32 estações virtualizadas, que serviram para simular um ambiente com um maior número de clientes. As estações virtualizadas foram configuradas com a rede em modo *bridge*, de modo que cada uma obteve um endereço de IP real da rede.

Todos os computadores estão configurados com sistema operacional Linux Debian 7.1 de 64Bits e interligados em uma rede *gigabit*. O acesso ao *cluster* ocorre através de um servidor chamado Athena. A figura 4.1 ilustra o ambiente de testes.

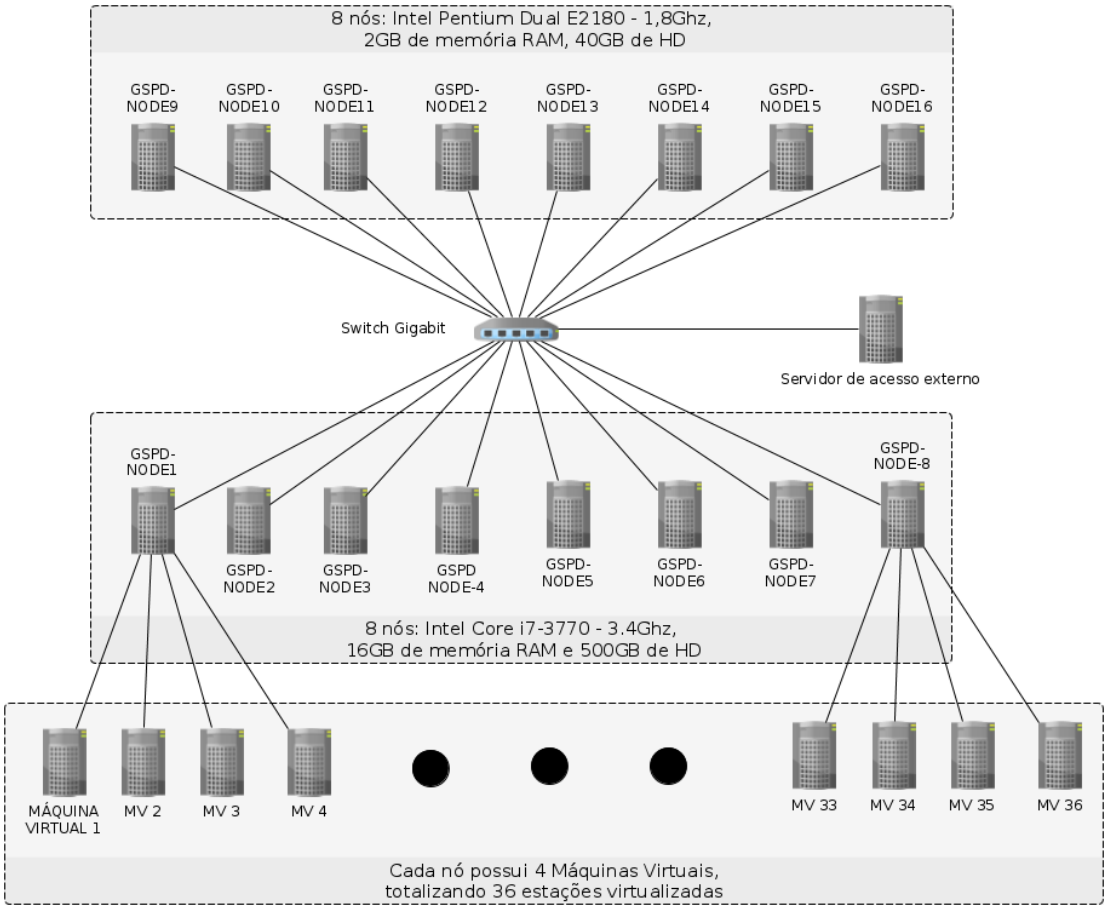

Figura 4.1: Ambiente de testes

4.2 Testes de desempenho

Os testes de desempenho tiveram como objetivo analisar a velocidade nas operações de escrita e leiutra do FlexA. Para isso, comparou-se a versão atual do FlexA, com sua versão inicial e outros SADs estudados, sendo o Tahoe-LAFS e o NFS escolhidos para a avaliação. As técnicas, ferramentas e critérios de avaliação são elencados a seguir. Ao fim, apresenta-se as tabelas e os gráficos comparativos entre os sistemas.

4.2.1 Técnicas, ferramentas e critérios de avaliação

Em (MARTINS, 2011) são descritos dois tipos de técnicas para avaliar o desempenho de um sistema computacional: técnicas de aferição e técnicas de modelagem. As técnicas de aferição podem ser aplicadas em sistemas totalmente construídos, utilizando-se ferramentas de *benchmarking* ou coleta de dados, ou em sistemas parcialmente construídos, utilizando-se prototipagem. Já as técnicas de modelagem envolvem a construção

de modelos que representam o sistema alvo e, posteriormente, utilizam-se simulações ou soluções analíticas para obter-se os resultados. Neste trabalho, optou-se pela técnica aferição por coleta de dados, sendo necessário instalar os sistemas de arquivos distribuídos, configurá-los e aprender como funcionavam para utilizá-los nos testes.

Foram pré-definidos tamanhos de 1MB, 5MB, 10MB, 25MB, 50MB e 100MB para os arquivos de teste. Como ferramenta de obtenção dos dados, utilizou-se *scripts* de avaliação escritos em linguagem *Python* (PYTHON, 2013) que realizavam repetidas operações de leitura e escrita para cada tamanho de arquivo. O *script* media o tempo real de transferência em cada requisição. Ao fim de cada sequência de testes, gera-se um arquivo com a média entre os tempos medidos. Este tipo de ferramenta permitiu concentrar os testes apenas nas operações de escrita e leitura, auxiliando a geração e compreensão dos resultados.

São feitas 20 iterações em cada arquivo. Este número foi obtido através de um teste de convergência que envolveu executar o sistema FlexA por no mínimo 100 vezes. No teste de convergência, para cada valor de tempo calculou-se a média e o desvio padrão. Quando o desvio padrão da iteração atual da sequência menos o desvio padrão da anterior diferiu, em módulo, em menos de 1% da média, significou que houve convergência para um determinado valor da sequência. As tabelas 4.1 e 4.2 mostram para quais valores na sequência de testes houve convergência na escrita e na leitura, respectivamente. Nota-se que para 1% o maior valor na sequência foi 18. Para os testes arredondou-se este número para 20.

Tabela 4.1: Valores das sequências que convergiram na escrita

Diferença em relação a media	1MB	5MB	10MB	25MB	50MB	100MB
0.5%	13	11	15	15	23	29
1%	11	11	11	12	16	18
2%	11	11	11	11	11	12

Tabela 4.2: Valores das sequências que convergiram na leitura

Diferença em relação a media	1MB	5MB	10MB	25MB	50MB	100MB
0.5%	11	11	11	11	13	22
1%	11	11	11	11	12	14
2%	11	11	11	11	11	11

Entre cada requisição por arquivo há uma pausa gerada por uma distribuição normal, com parâmetros $\mu = 15$ e $\sigma = 3$. Esta pausa tenta simular as requisições em um ambiente real.

Os testes foram feitos com 1, 2, 4, 8, 16 e 32 clientes fazendo requisições simultâneas, distribuídos entre as máquinas virtuais.

As quantidades de servidores e réplicas configuradas foram as seguintes:

- FlexA inicial: 3 servidores primários.

- FlexA atual: 3 servidores primários e 5 servidores secundários (réplicas).
- NFS: 1 servidor.
- Tahoe-LAFS: 1 *introducer* e 7 *storage nodes* (nós de armazenamento), com um *storage node* configurado como *helper*.

Na arquitetura original do FlexA, 3 clientes não conseguiram completar o teste com 32 clientes simultâneos. Isso se deve, provavelmente, à sobrecarga dos servidores primários, já que eles são os únicos a receberem as requisições. Dado esse motivo, não foram incluídos os testes com 32 clientes.

As *caches* de todos os sistemas foram desabilitadas para medir o tráfego de dados real na rede.

Os gráficos com o resultados indivudais podem ser visualizados no apêndice B. Na próxima subseção seguem as comparações entre os sistemas.

4.2.2 Comparação de Desempenho

A vazão, medida em *Megabytes* por segundo (MB/s), na escrita e leitura com 1 cliente no FlexA inicial e o desenvolvido, mostrados nas figuras 4.2 e 4.3, foram praticamente iguais.

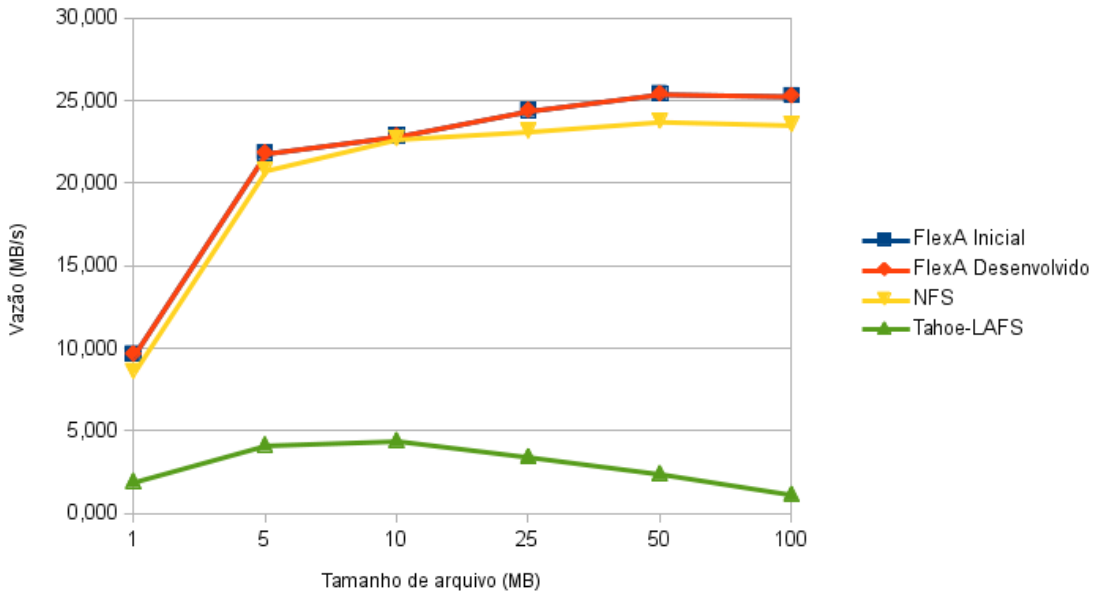

Figura 4.2: Comparativo da operação de escrita com 1 cliente

Com 2 clientes simultâneos (figuras 4.4 e 4.5), o FlexA desenvolvido obteve um desempenho superior em relação aos outros SADs na escrita de arquivo. Isto está

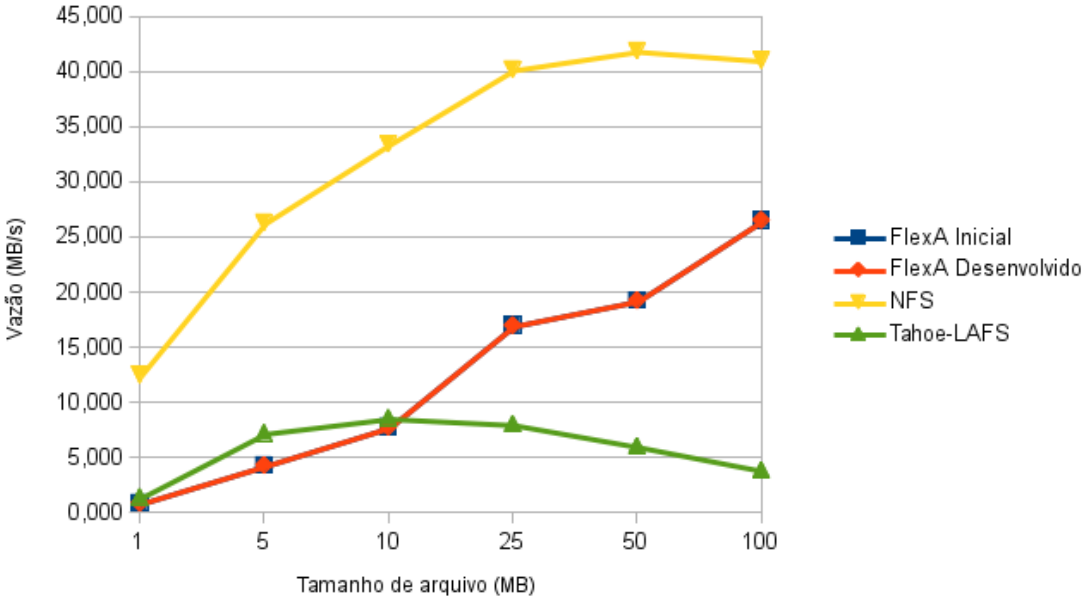

Figura 4.3: Comparativo da operação de leitura com 1 cliente

relacionado ao fato de ser enviado apenas uma única porção aos servidores primários. A escrita de arquivos se manteve praticamente a mesma.

Com 4, 8 e 16 clientes simultâneos (figuras 4.6 a 4.11), o desempenho do FlexA desenvolvido e NFS são muito parecidos para a escrita de arquivos de até 10MB. Para arquivos menores ou iguais a 10MB o FlexA não divide os arquivos em porções, distribuindo uma única porção a um único servidor, como no NFS. Para arquivos maiores que 10MB, o FlexA os divide em 3 porções e envia exatamente 1 porção para cada primário.

No teste com 32 clientes simultâneos (4.12 e 4.13) houve o problema com a escalabilidade do FlexA inicial. O NFS apresentou uma boa vazão na escrita de arquivos de até 10MB, mas para 25MB em diante houve uma queda brusca em seu desempenho, possívelmente pela sobrecarga do único servidor recebendo todas as requisições.

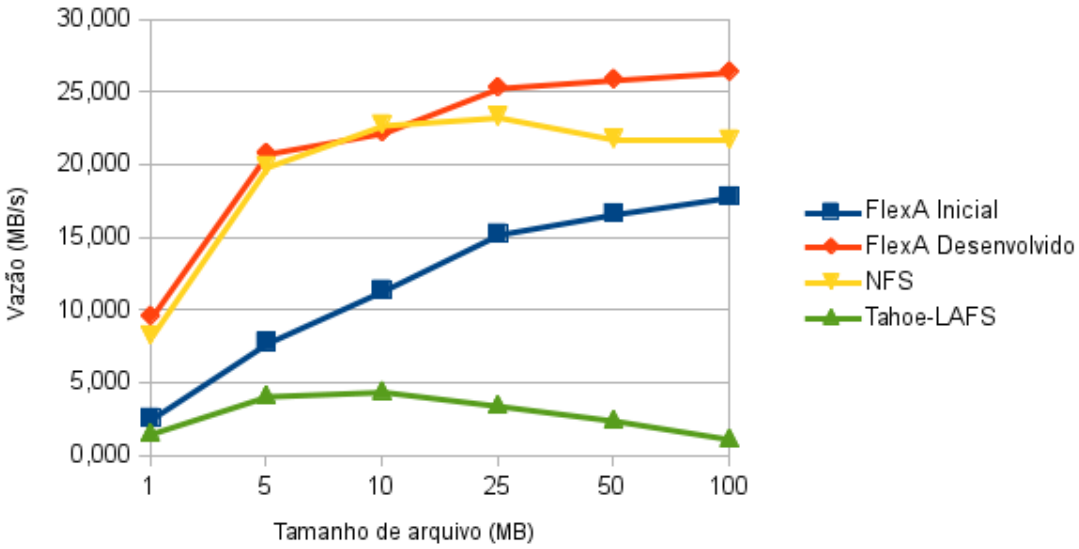

Figura 4.4: Comparativo da operação de escrita com 2 clientes

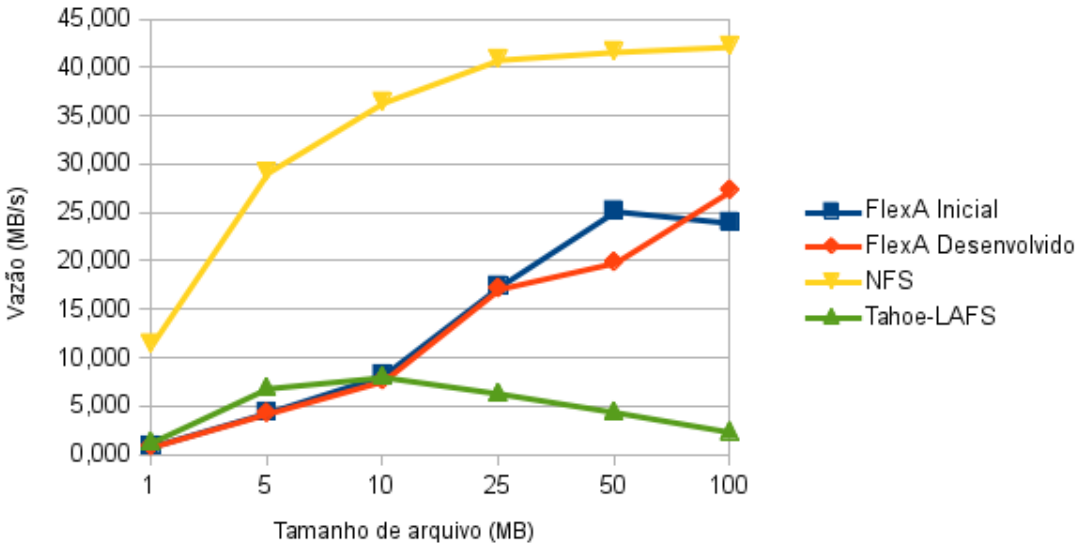

Figura 4.5: Comparativo da operação de leitura com 2 clientes

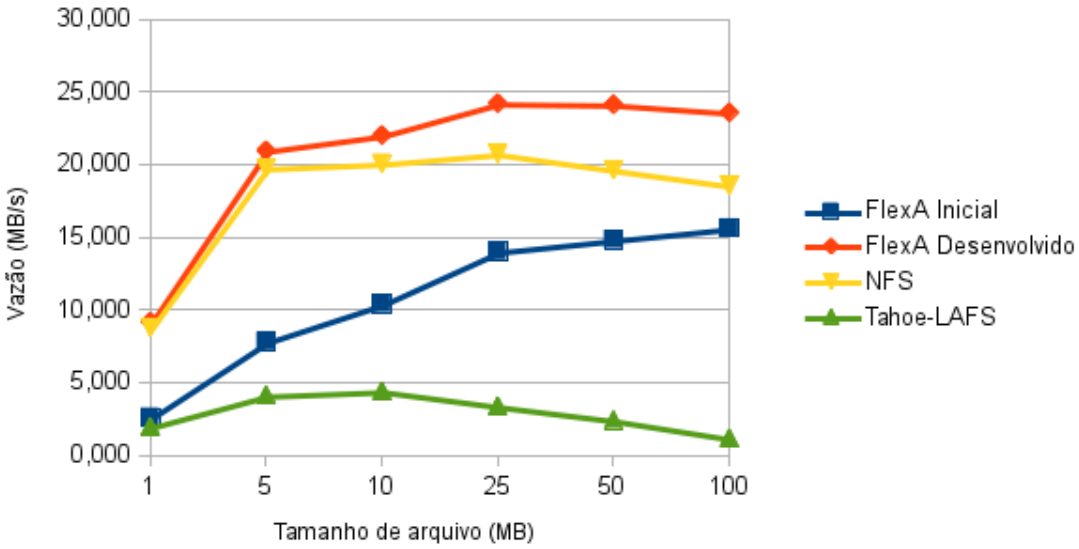

Figura 4.6: Comparativo da operação de escrita com 4 clientes

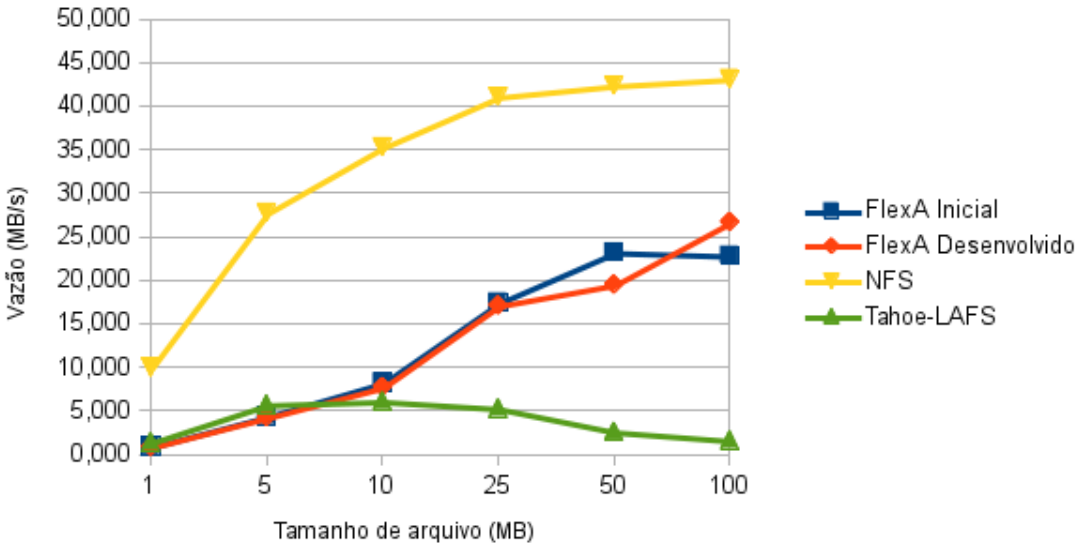

Figura 4.7: Comparativo da operação de leitura com 4 clientes

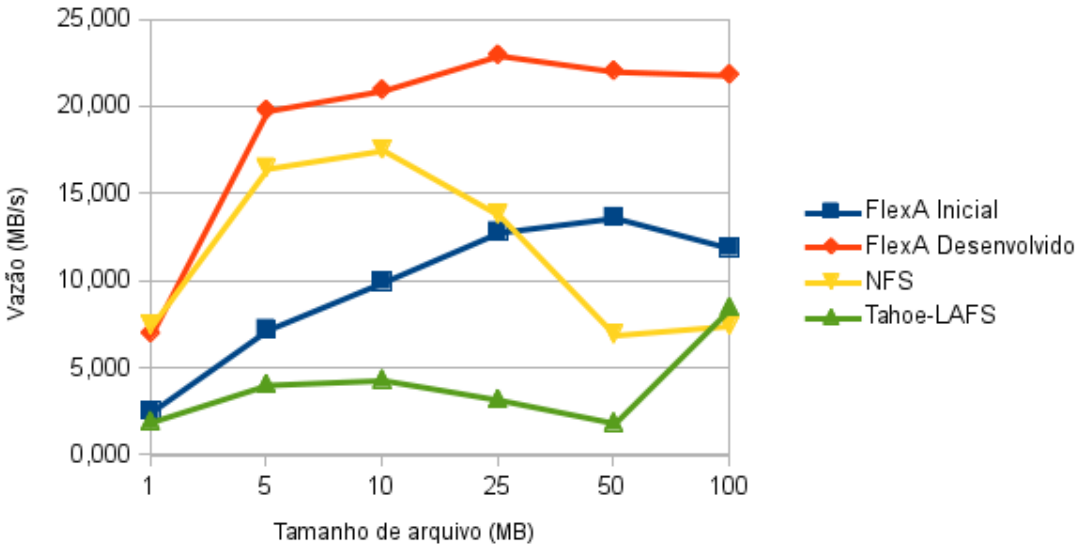

Figura 4.8: Comparativo da operação de escrita com 8 clientes

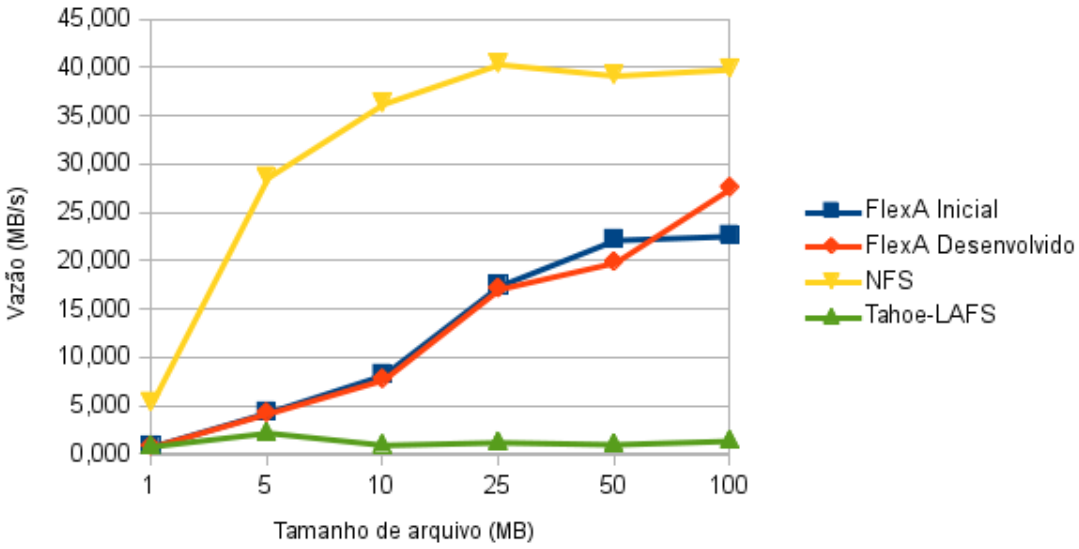

Figura 4.9: Comparativo da operação de leitura com 8 clientes

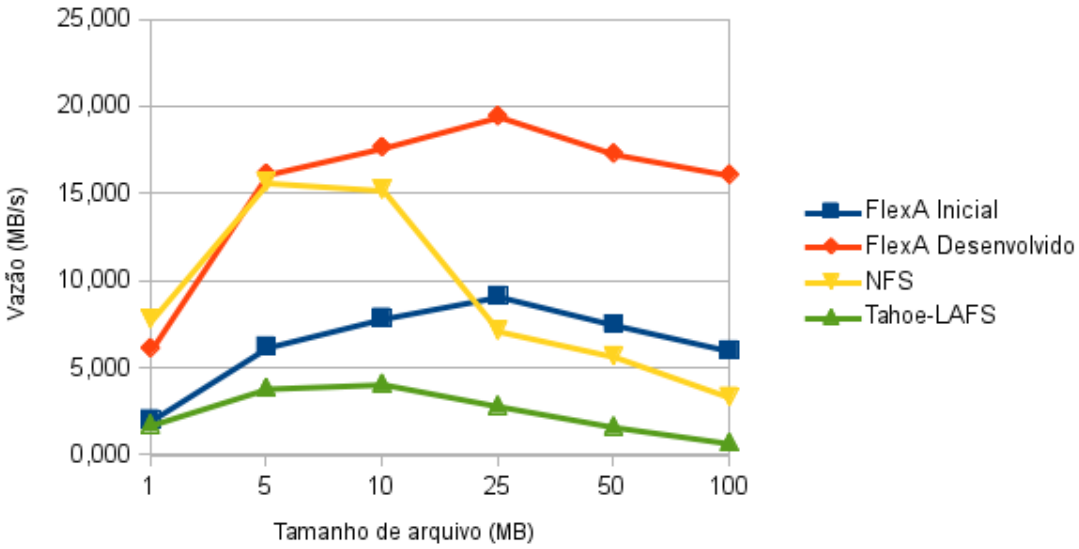

Figura 4.10: Comparativo da operação de escrita com 16 clientes

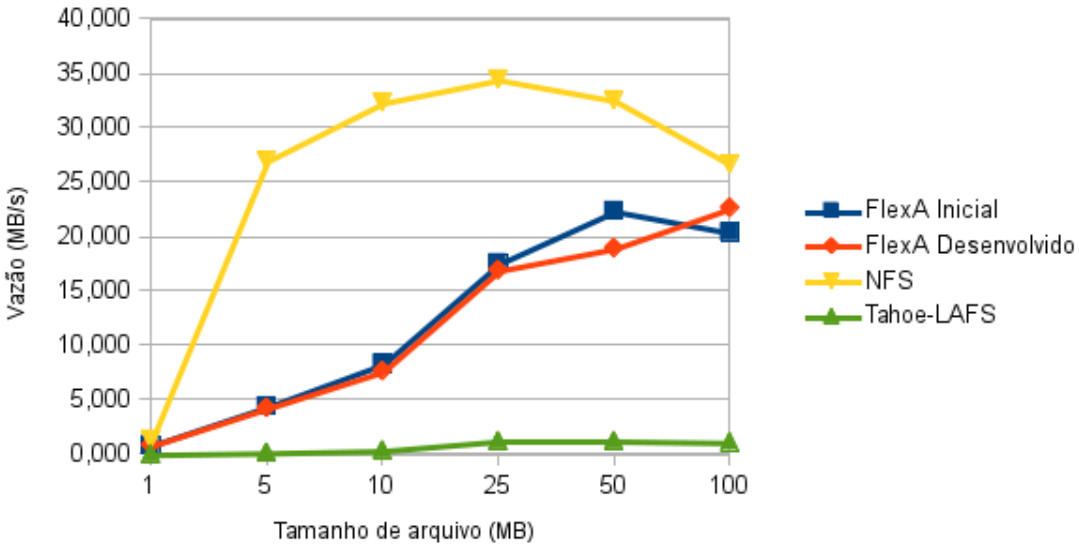

Figura 4.11: Comparativo da operação de leitura com 16 clientes

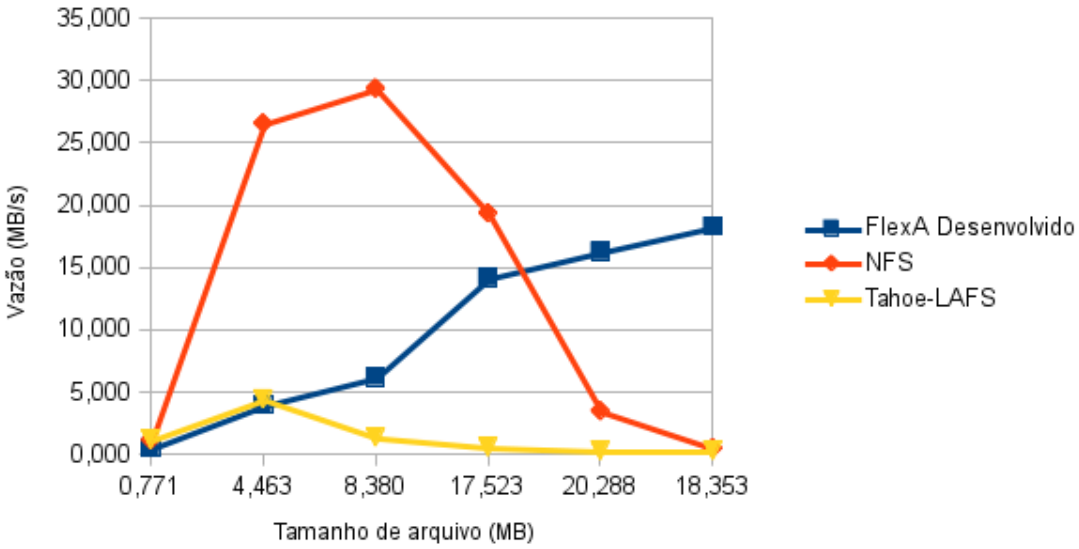

Figura 4.12: Comparativo da operação de escrita com 32 clientes

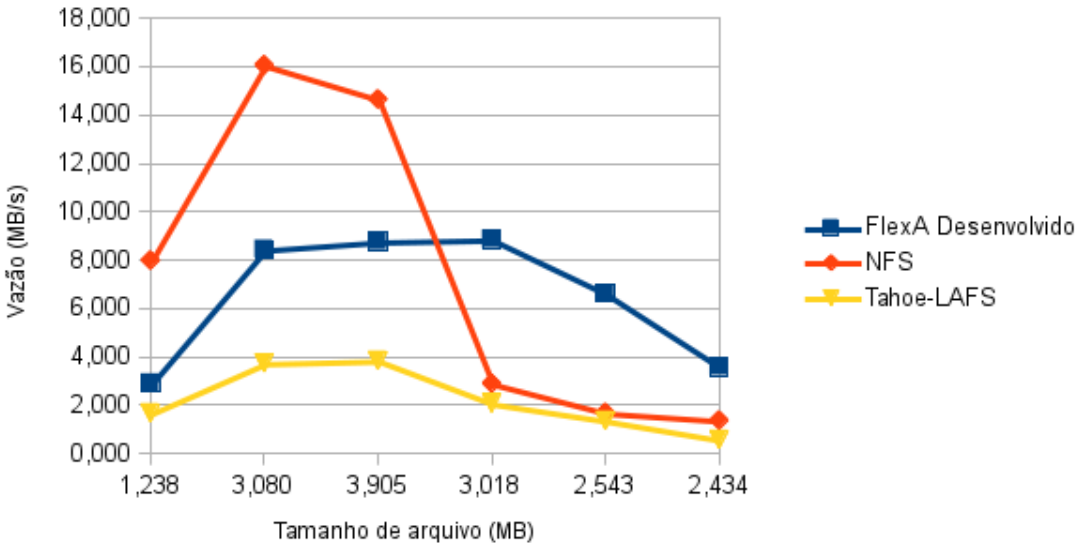

Figura 4.13: Comparativo da operação de leitura com 32 clientes

4.3 Desempenho na sincronização entre servidores primários e secundários

O objetivo deste teste foi medir o desempenho do FlexA na sincronização das porções recebidas pelos servidores primários e replicadas para os secundários. Para isto foram medidos os tempos, em segundos, e a vazão, em MB/s, da sincronização de arquivos de 1MB, 5MB, 10MB, 25MB, 50MB e 100MB com 1, 2, 4, 8, 16 e 32 clientes realizando operações de escrita. Foram utilizados 3 servidores primários e 5 secundários. Para cada tamanho de arquivo a operação de escrita foi repetida 20 vezes. No mesmo teste foi extraído o resultado para 1 servidor e para os 3 servidores. A figura 4.14 ilustra o gráfico com o tempo médio de sincronização em 1 servidor primário. O tempo médio de sincronização dos 3 servidores primários é apresentado no gráfico da figura 4.15.

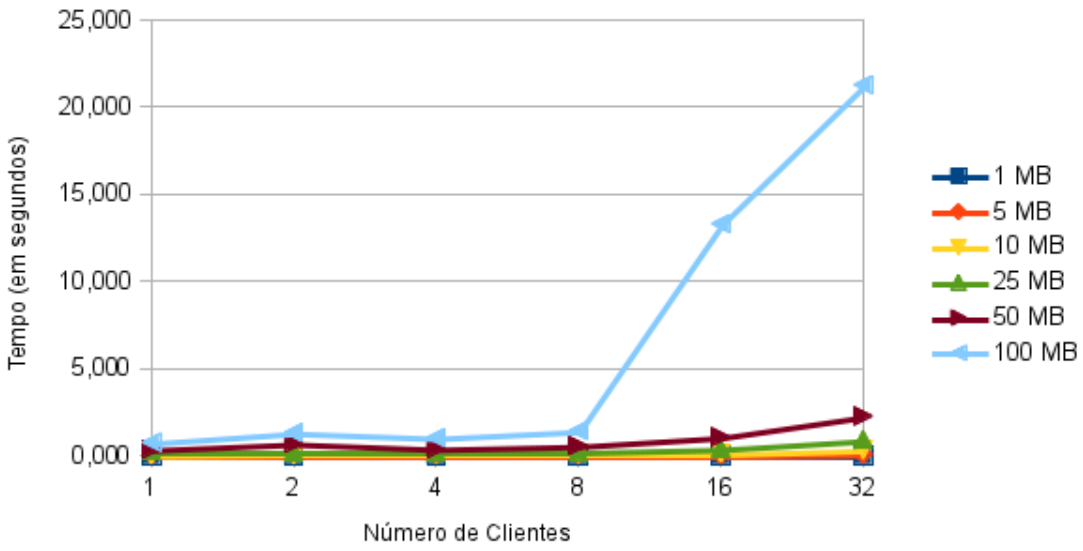

Figura 4.14: Tempo (s) médio na sincronização de 1 dos primários

A vazão na sincronização em 1 servidor primário e em 3 servidores estão nas figuras 4.16 e 4.17, respectivamente.

O tempo e a vazão na sincronização de um dos servidores do teste é parecido com o tempo e vazão média entre os três servidores, pois com excessão dos arquivos de 1MB, 5MB e 10MB os três servidores primários sincronizam a mesma quantidade de porções. O desempenho na sincronização começa a decair a partir de 16 clientes com arquivos de 25MB. Com 32 clientes e arquivos de 100MB a vazão por arquivo diminui muito, o que indica sobrecarga no servidor primário, provavelmente ocasionada pela quantidade excessiva de Entrada/Saída no disco rígido.

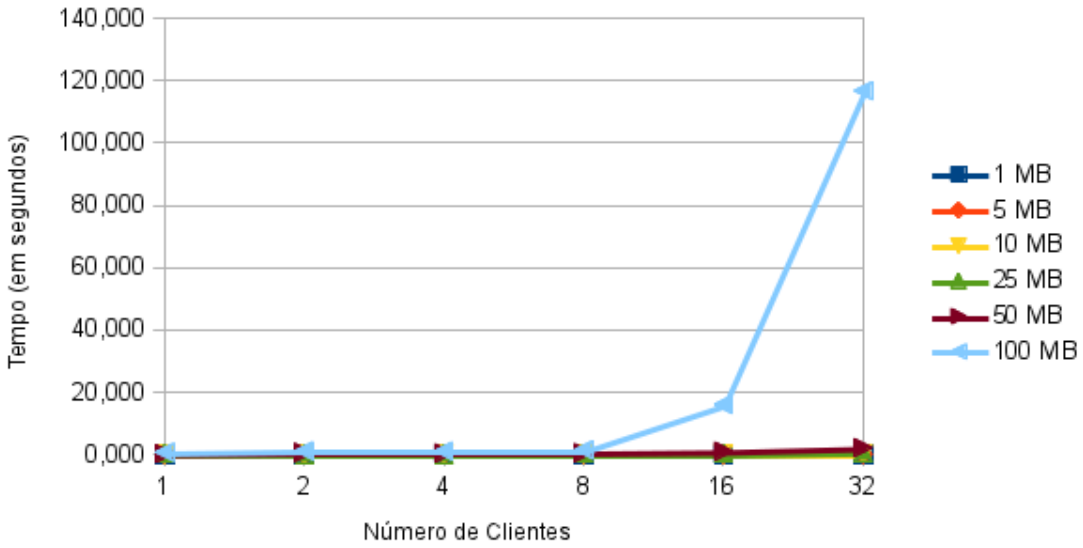

Figura 4.15: Tempo (s) médio na sincronização de 3 primários

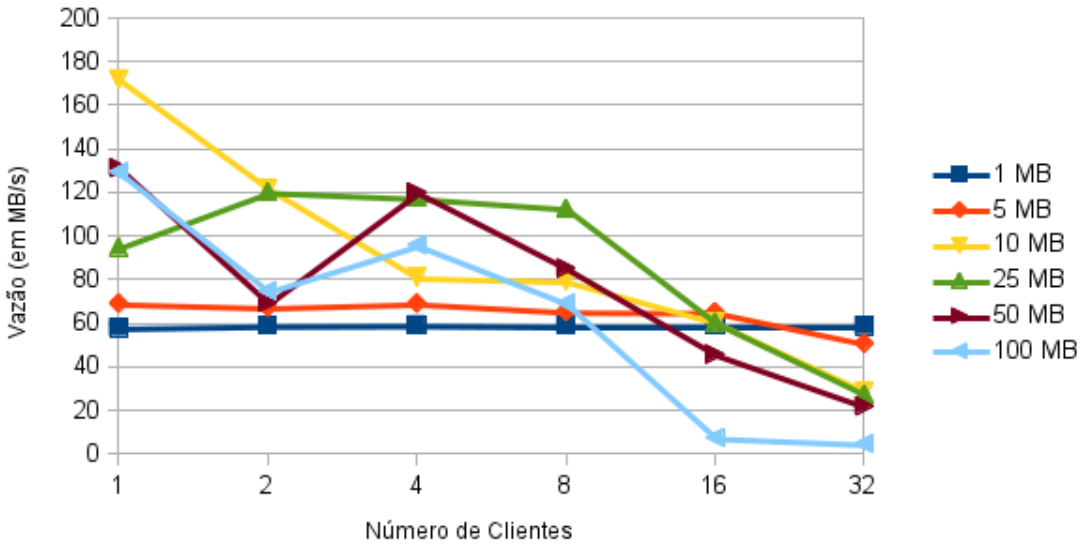

Figura 4.16: Vazão (MB/s) média na sincronização de 1 dos primários

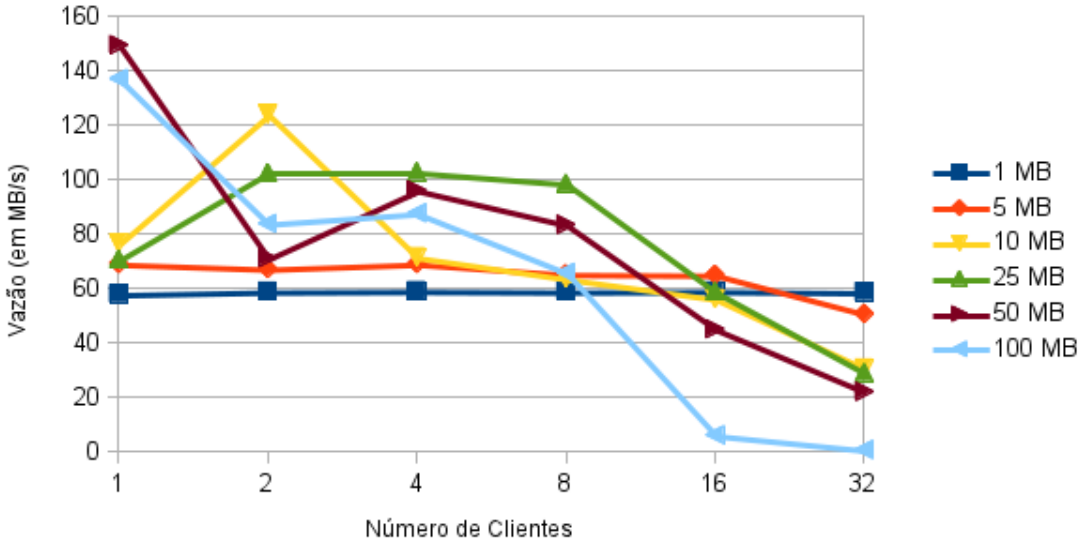

Figura 4.17: Vazão (MB/s) média na sincronização de 3 primários

4.4 Testes de detecção de falhas no processo do módulo Coletor

O intuito deste teste foi verificar o tempo de detecção de falhas no processo do módulo Coletor. Para isto, foi utilizado o módulo *checa_processo* com configuração padrão de 10 segundos entre as verificações do atividade do processo e um *script* que encerrava o processo do módulo Coletor de acordo com uma Distribuição de *Weibull* com parâmetros $\alpha = 0{,}78$ e $\beta = 20$. Segundo (SCHROEDER; GIBSON, 2006), o intervalo de tempo entre as falhas para nós individuais em sistemas distribuídos pode ser modelado pela distribuição de *Weibull* com parâmetro α variando entre 0.7 a 0.8. No teste, o módulo *checa_processo* foi iniciado simultâneamente com o *script* e ambos executaram durante 2 horas. A tabela 4.3 indica os tempos de detecção da falha e a tabela 4.4 mostra a média e o desvio padrão desses tempos.

A injecção das falhas, ou seja, o encerramento do processo pelo *script* foi muito mais intenso no início da simulação. Este fato é recorrente dos parâmetros utilizados na distribuição de *Weibull*. O módulo *checa_processo* comportou-se da forma esperada, detectando a queda no processo dentro de seu intervalo de verificação.

Tabela 4.3: Tempos de detecção de falhas no processo do Coletor

Início 10:58:56		
Reiniciou	Encerrou	Tempo de detecção (s)
10:59:05	10:58:56	8,8125002134
10:59:15	10:59:09	5,8749995267
10:59:25	10:59:17	7,8333333178
11:00:06	10:59:57	8,8125002134
11:01:16	11:01:11	4,8958332467
11:01:36	11:01:34	1,9583337911
11:12:57	11:12:55	1,9583331756
11:14:48	11:14:39	8,8125002134
11:21:38	11:21:30	7,8333333178
12:18:45	12:18:36	8,8124995978
Fim 13:58:56		

Tabela 4.4: Tempo médio, em segundos, e desvio padrão na detecção de falhas no processo do Coletor

Tempo Médio (s)	Desvio padrão
6,5604166614	2,7714241406

4.5 Testes de detecção de falhas por queda em primários

No teste de detecção de queda de um dos primários verificou-se, primeiramente, que o desligamento da interface de rede do servidor acarretava na detecção por queda. Após isto, foi feita uma avaliação do tempo de detecção de quedas entre servidores primários. Utilizou-se 8 máquinas virtuais instaladas no *VirtualBox*, seis delas eram servidores primários e as outras, réplicas. Para verificar os tempos de detecção, um *script* desativava, a cada 30 segundos, a interface de rede de um servidor primário escolhido aleatoriamente. Mediu-se os tempos para 3, 4, 5 e 6 servidores ativos. As médias e desvio padrão foram extraídas após 20 repetições.

Tabela 4.5: Tempo médio, em segundos, e desvio padrão na detecção de queda de primários

Qtd. Servidores	Tempo Médio (s)	Desvio padrão
3	13,50092	2,04734
4	13,41666	2,82893
5	13,57970	2,05700
6	14,01568	2,73178

Como pode ser observado, mesmo com o aumento da quantidade de servidores primários, o tempo de detecção não cresceu de forma brusca.

4.6 Considerações finais

Neste capítulo apresentaram-se as validações e testes de desempenho das implementações desenvolvidas no sistema FlexA. Devido às mudanças na distribuição das porções, que passou de duas por primário para apenas uma, houve um aumento no desempenho da operação de escrita do FlexA desenvolvido. A sondagem de primários mostrou-se eficaz para detectar problemas na rede e queda de servidores. O próximo capítulo conclui este trabalho, elencando trabalhos futuros e problemas encontrados.

Capítulo 5

Conclusões

Este trabalho apresentou uma revisão sobre sistemas de arquivos distribuídos, focando-se em características como replicação e sincronização, consistência e detecção de falhas. Além disso também apresentou o sistema de arquivos distribuído FlexA, elencando suas características e funcionalidades. Por último, este trabalho teve como maior contribuição o desenvolvimento de uma estrutura que permitiu ao FlexA continuar seguindo seu conceito inicial de flexibilidade e adaptabilidade, ajustando suas funções de acordo com as variações do ambiente, como de tamanho de arquivo e quantidade de servidores disponíveis.

5.1 Considerações finais

Com o desenvolvimento do grupo de réplicas no FlexA, o sistema obteve um ganho de desempenho considerável na operação de escrita. Os mecanismos de controle da versão dos arquivos garantem que o sistema não repasse para usuários arquivos desatualizados. Por fim, a detecção de falhas nos serviços do FlexA foi um importante passo para a implementação de mecanismos de tratamento de falhas, como a transformação de servidores secundários em primários.

5.2 Problemas encontrados

Durante o desenvolvimento deste trabalho foram encontrados alguns problemas, destacando-se principalmente:

- Código fonte original pouco modularizado.
- Dificuldade na instalação do sistema FlexA inicial.
- Problemas encontrados nos *sockets* de comunicação.

Para resolver estes problemas o código fonte original passou por uma reestruturação e o módulo Comunicador foi modificado para facilitar a instalação e configuração do FlexA.

5.3 Direções futuras

Em relação às melhorias das funções existentes no FlexA e das direções futuras, podem ser citadas as seguintes atividades:

- Definir novos limiares para a quantidade de porções geradas por arquivo.
- Minimizar o acesso ao banco de dados dos servidores.
- Gerar um histórico de versões de arquivos.

5.4 Publicações

Este trabalho é relacionado ao projeto de iniciação científica orientado pelo Prof. Dr. Aleardo Manacero Jr., pelo Departamento de Ciência da Computação e Estatística da UNESP, intitulado ”Sincronização e Réplicas em um Sistema de Arquivo Distribuído”. Ele foi executado por este autor anteriormente a este trabalho e permitiu todo embasamento para sua realização. O projeto rendeu a publicação de um resumo expandido e publicado nos anais da IV Escola Regional de Alto Desempenho, sob o título “FlexA - Grupo de Réplicas em um Sistema de Arquivos Distribuído” (OLIVEIRA et al., 2013).

Referências Bibliográficas

COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011, 9780132143011.

FERNANDES, S. E. N. *Sistema de Arquivos Distribuído Flexível e Adaptável*. Dissertação (Dissertação de Mestrado) — Universidade Estadual Paulista, 2012.

FUJIMURA, A.; OH, S.; GERLA, M. Network coding vs. erasure coding: Reliable multicast in ad hoc networks. In: *Military Communications Conference, 2008. MILCOM 2008. IEEE*. [S.l.: s.n.], 2008. p. 1 –7.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. [S.l.]: ACM, 2003. p. 29–43.

GSPD. Página do grupo de sistemas paralelos e distribuídos. Disponível em <http://www.dcce.ibilce.unesp.br/spd/>. 2013.

KSHEMKALYANI, A. D.; SINGHAL, M. *Distributed Computing: principles, algorithms and systems*. [S.l.]: Cambridge University Press, 2008.

MARTINS, R. D. *Avaliação de Desempenho em Sistemas de Arquivos Distribuídos*. Dissertação (Trabalho de Conclusão de Curso) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, São José do Rio Preto, 2011.

MCKUSICK, K.; QUINLAN, S. Gfs: evolution on fast-forward. *Commun. ACM*, v. 53, n. 3, p. 42–49, 2010.

MCKUSICK, M. K.; QUINLAN, S. Gfs: Evolution on fast-forward. *Queue*, ACM, New York, NY, USA, v. 7, n. 7, p. 10:10–10:20, ago. 2009. ISSN 1542-7730. Disponível em: <http://doi.acm.org/10.1145/1594204.1594206>.

MILLER, M. S. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. Tese (Doutorado) — Johns Hopkins University, 2006.

OLIVEIRA, M. D. C. et al. Flexa - grupo de réplicas em um sistema de arquivos distribuído. In: *IV Escola Regional de Alto Desempenho de São Paulo*. São Carlos: ERAD-SP, 2013.

PATE, S. D. *Unix Filesystems: Evolution, Design, and Implementation*. [S.l.]: Wiley Publishing, 2003.

PLANK, J. S. et al. A performance evaluation and examination of open-source erasure coding libraries for storage. In: *Proccedings of the 7th conference on File and storage technologies*. [S.l.]: USENIX Association, 2009.

PYTHON. Python home page. Disponível em <http://www.python.org.br>. 2013.

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. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633.

TANENBAUM, A. S.; STEEN, M. V. *Sistemas Distribuídos Princípios e Paradigmas*. 2nd. ed. [S.l.]: Ed. Pearson, 2007.

WEIL, S. A. et al. Ceph: A scalable, high-performance distributed file system. In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2006. (OSDI '06), p. 307–320. ISBN 1-931971-47-1. Disponível em: <http://dl.acm.org/citation.cfm?id=1298455.1298485>.

WILCOX-O'HEARN, Z.; WARNER, B. Tahoe: the least-authority filesystem. In: *Proceedings of the 4th ACM international workshop on Storage security and survivability*. [S.l.]: ACM, 2008. (StorageSS '08), p. 21–26.

Apêndice A

Instalação do FlexA

Para utilizar o sistema é necessário ter instalado o interpretador Python 2.7, o pacote netifaces para administração da interface de rede do computador e o pacote pycrypto que fornece um conjunto de algoritmos de criptografia.

Em cada componente do sistema deve-se criar o arquivo de configuração com o módulo Comunicador através do comando *python com.py -r*.

Em seguida deve-se iniciar o módulo coletor dos servidores primários, secundários e clientes com os comandos *python coletor_servidor.py*, *python coletor_replica.py* e *python coletor_cliente.py*, respectivamente. Após isso, deve-se executar o rastreio da rede em busca de servidores. No cliente e nos servidores secundários o rastramento é feito com o comando *python com.py -b* e nos servidores primários é *python com.py -bs*. Com isso, FlexA estará totalmente funcional. Suas requisições podem ser feitas no módulo FlexA do cliente com as opções -p e -g, para escrita e leitura de arquivos.

Apêndice B

Tabelas e gráficos com os resultados individuais

Neste apêndice serão apresentadas as tabelas e gráficos com os resultados das operações de escrita e leitura do FlexA na versão inicial e desenvolvida, NFS e Tahoe-LAFS.

B.1 Testes de desempenho no FlexA inicial

Os tempos de transferência, em segundos, do FlexA na implementação inicial nas operações de escrita e leitura são apresentados, respectivamente, nas tabelas B.1 e B.2. A vazão em MB/s para a escrita e leitura estão nas tabelas B.3 e B.3. Os gráficos da vazão em relação ao tamanho dos arquivos em MB e quantidade de clientes para a escrita e leitura são mostrados nas figuras B.1 e B.2.

Tabela B.1: Escrita de arquivos FlexA inicial (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.2: Leitura de arquivos FlexA inicial (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.3: Escrita de arquivos FlexA inicial (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.4: Leitura de arquivos FlexA inicial (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

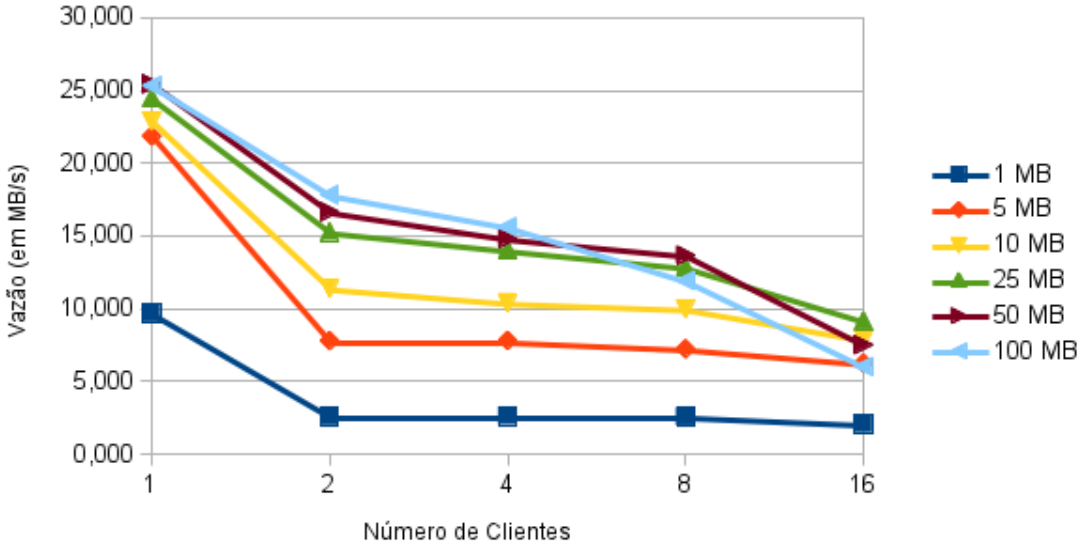

Figura B.1: Vazão (em MB/s) na escrita em relação ao número de clientes FlexA inicial

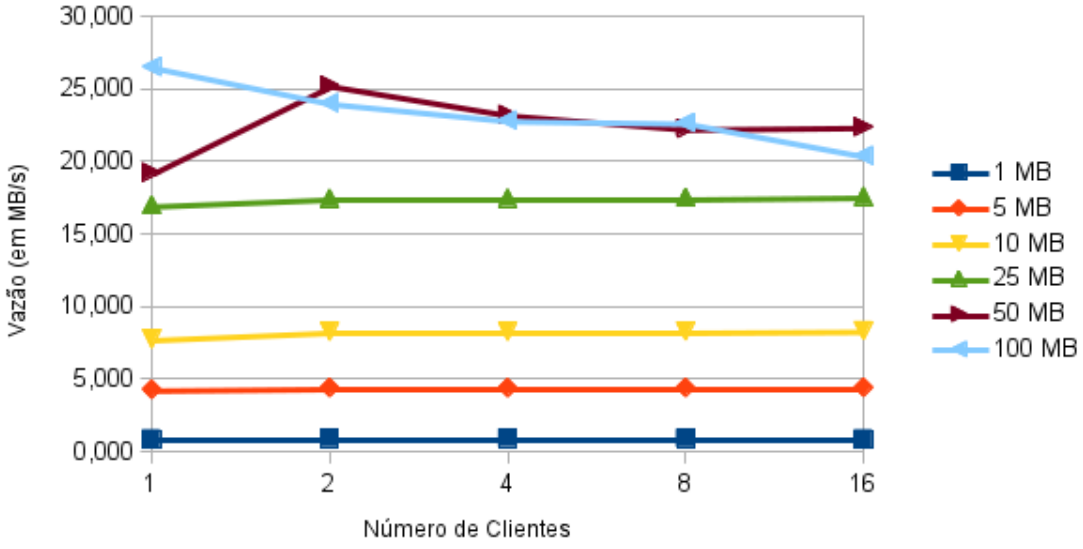

Figura B.2: Vazão (em MB/s) na leitura em relação ao número de clientes FlexA inicial

B.2 Testes de desempenho no FlexA desenvolvido

Os tempos de transferência, em segundos, do FlexA desenvolvido nas operações de escrita e leitura são apresentados, respectivamente, nas tabelas B.5 e B.6. A vazão em MB/s para a escrita e leitura estão nas tabelas B.7 e B.8. Os gráficos da vazão em relação ao tamanho dos arquivos em MB e quantidade de clientes para a escrita e leitura são mostrados nas figuras B.3 e B.4.

Tabela B.5: Escrita de arquivos FlexA desenvolvido (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.6: Leitura de arquivos FlexA desenvolvido (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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 B.7: Escrita de arquivos FlexA desenvolvido (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
1	9,671	21,863	22,894	24,445	25,457	25,324
2	9,597	20,820	22,249	25,343	25,904	26,421
4	9,183	20,971	22,026	24,221	24,137	23,583
8	7,016	19,811	20,972	22,980	22,062	21,850
16	6,149	16,156	17,683	19,464	17,312	16,099
32	2,912	8,447	8,783	8,867	6,627	3,591

Tabela B.8: Leitura de arquivos FlexA desenvolvido (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
1	0,899	4,315	7,786	16,993	19,257	26,534
2	0,898	4,336	7,707	17,224	19,938	27,391
4	0,898	4,307	7,778	17,149	19,544	26,740
8	0,901	4,337	7,831	17,214	19,949	27,651
16	0,876	4,304	7,689	16,961	18,987	22,689
32	0,541	3,978	6,226	14,189	16,305	18,314

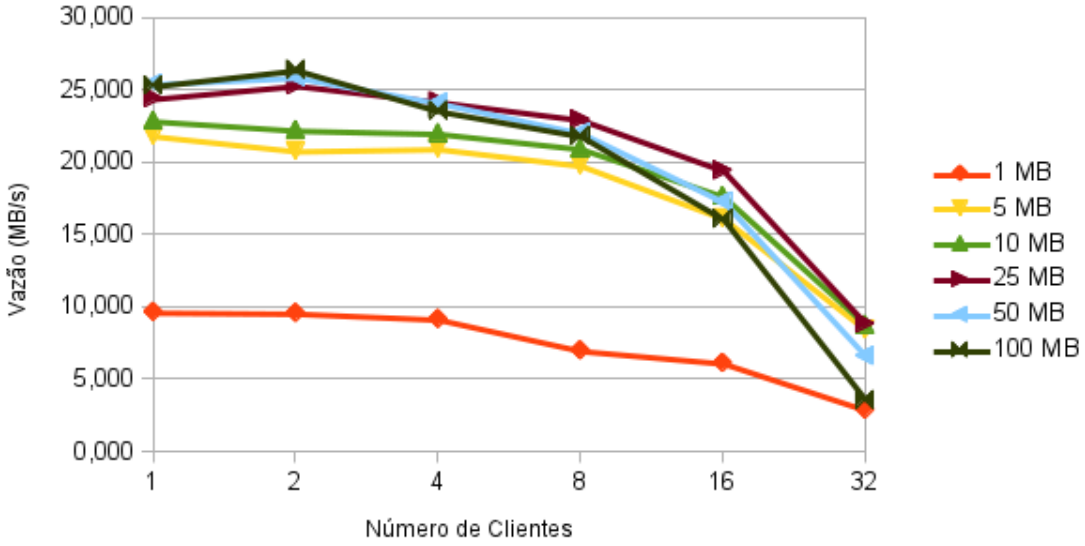

Figura B.3: Vazão (em MB/s) na escrita em relação ao número de clientes FlexA desenvolvido

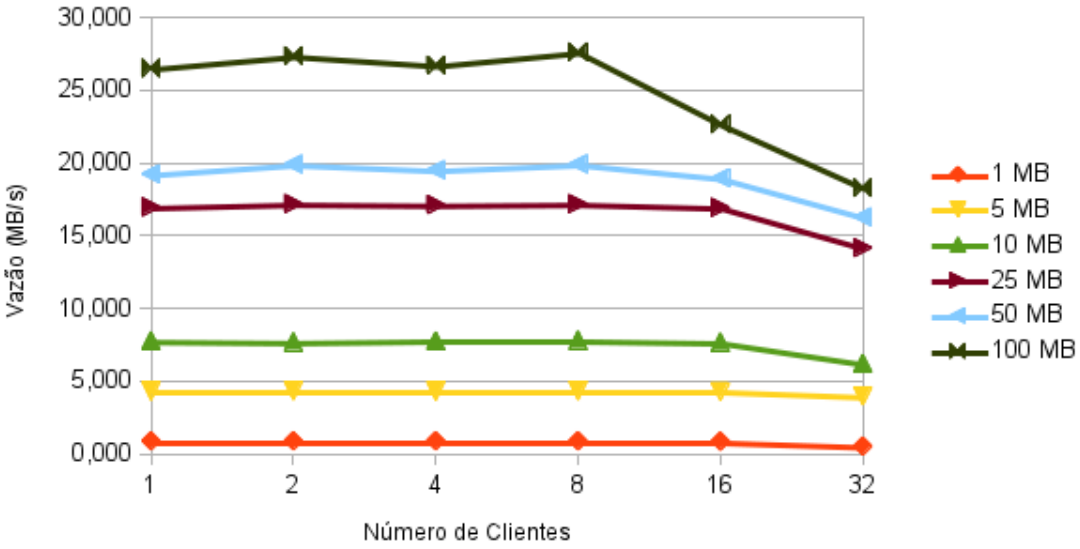

Figura B.4: Vazão (em MB/s) na leitura em relação ao número de clientes FlexA inicial

B.3 Testes de desempenho no NFS

Os tempos de transferência, em segundos, do NFS nas operações de escrita e leitura são apresentados, respectivamente, nas tabelas B.9 e B.10. A vazão em MB/s para a escrita e leitura estão nas tabelas B.11 e B.12. Os gráficos da vazão em relação ao tamanho dos arquivos em MB e quantidade de clientes para a escrita e leitura são mostrados nas figuras B.5 e B.6.

Tabela B.9: Escrita de arquivos NFS (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.10: Leitura de arquivos NFS (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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 B.11: Escrita de arquivos NFS (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.12: Leitura de arquivos NFS (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

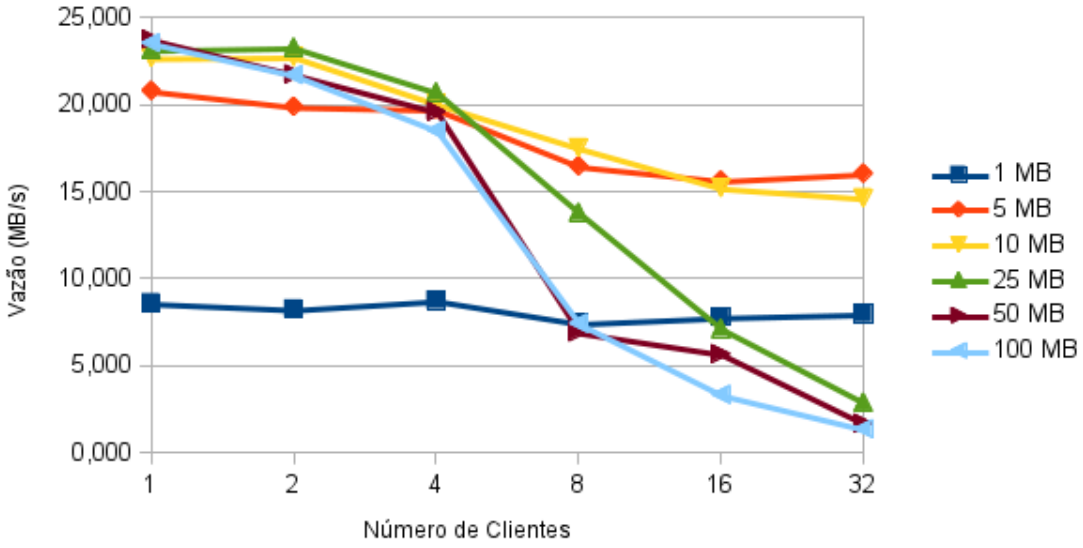

Figura B.5: Vazão (em MB/s) na escrita em relação ao número de clientes NFS

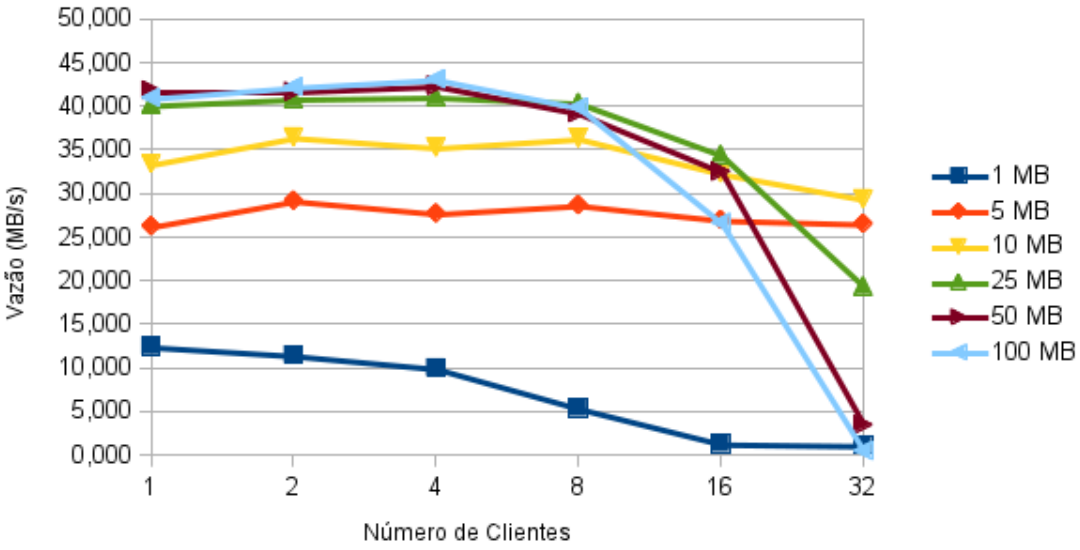

Figura B.6: Vazão (em MB/s) na leitura em relação ao número de clientes NFS

B.4 Testes de desempenho no Tahoe-LAFS

Os tempos de transferência, em segundos, do Tahoe-LAFS nas operações de escrita e leitura são apresentados, respectivamente, nas tabelas B.13 e B.14. A vazão em MB/s para a escrita e leitura estão nas tabelas B.15 e B.16. Os gráficos da vazão em relação ao tamanho dos arquivos em MB e quantidade de clientes para a escrita e leitura são mostrados nas figuras B.7 e B.8.

Tabela B.13: Escrita de arquivos Tahoe-LAFS (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
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

Tabela B.14: Leitura de arquivos Tahoe-LAFS (tempo, em segundos)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
1	0,700	0,690	1,163	3,1103	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 B.15: Escrita de arquivos Tahoe-LAFS (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
1	1,955	4,167	4,433	3,458	2,425	1,175
2	1,562	4,131	4,431	3,462	2,440	1,166
4	1,964	4,110	4,408	3,374	2,402	1,149
8	1,969	4,090	4,358	3,233	1,896	8,499
16	1,796	3,875	4,121	2,858	1,662	0,726
32	1,684	3,749	3,859	2,086	1,376	0,587

Tabela B.16: Leitura de arquivos Tahoe-LAFS (vazão, em MB/s)

Qtd. Clientes	1MB	5MB	10MB	25MB	50MB	100MB
1	1,428	7,242	8,598	8,038	6,045	3,873
2	1,375	6,952	8,106	6,401	4,477	2,424
4	1,438	5,754	6,140	5,312	2,644	1,643
8	0,957	2,348	1,092	1,366	1,162	1,522
16	0,049	0,194	0,402	1,252	1,253	1,145
32	1,206	4,456	1,392	0,608	0,303	0,320

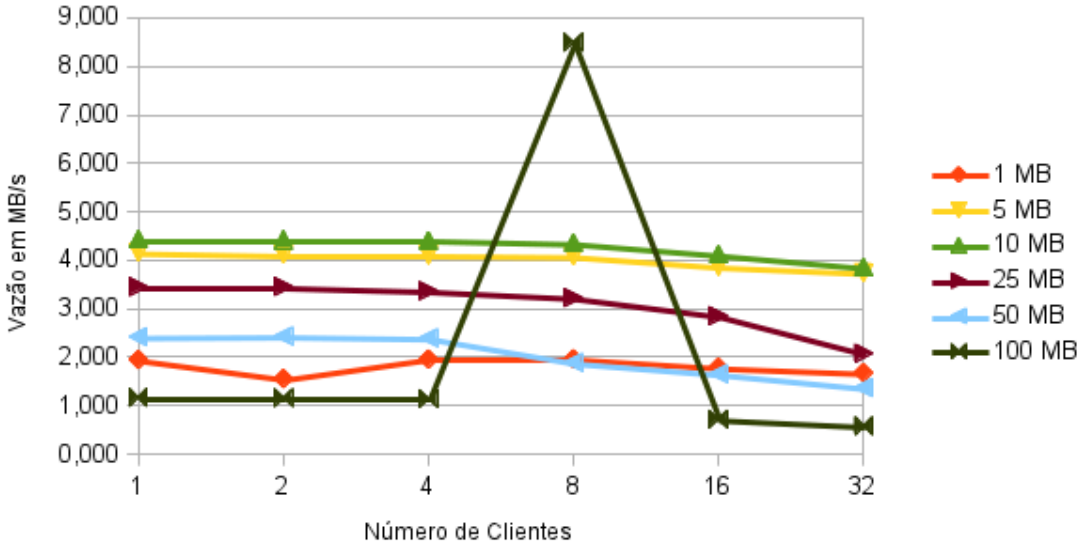

Figura B.7: Vazão (em MB/s) na escrita em relação ao número de clientes Tahoe-LAFS

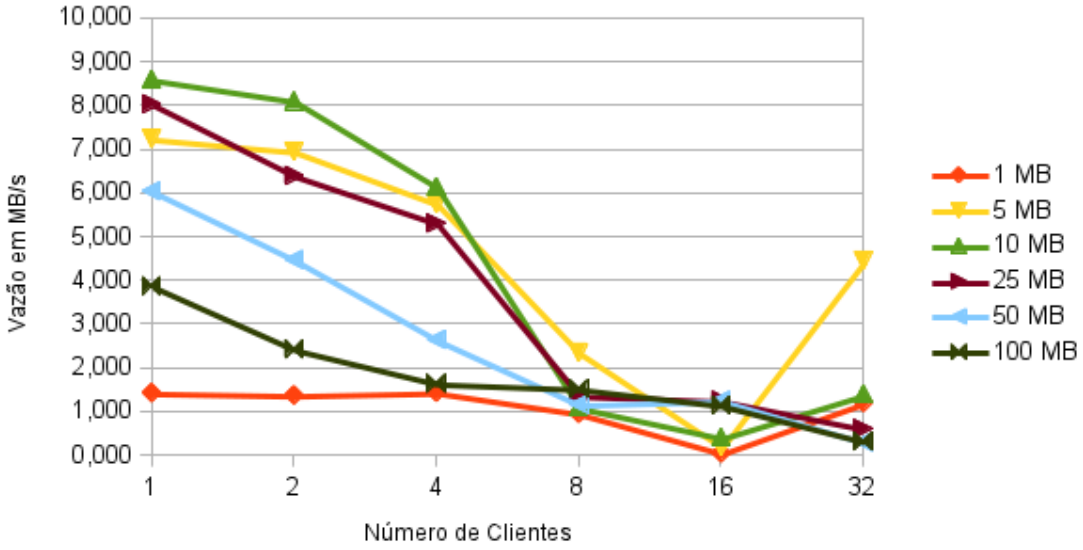

Figura B.8: Vazão (em MB/s) na leitura em relação ao número de clientes Tahoe-LAFS