Silas Evandro Nachif Fernandes

Sistema de Arquivos Distribuído Flexível e Adaptável

Dissertação submetida 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 obtenção do título de mestre no Programa de Pós-Graduação em Ciência da Computação da UNESP.

São José do Rio Preto
2012

Silas Evandro Nachif Fernandes

Sistema de Arquivos Distribuído Flexível e Adaptável

Dissertação submetida 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 obtenção do título de mestre no Programa de Pós-Graduação em Ciência da Computação da UNESP.

Orientadora: Dr.ª Renata Spolon Lobato

São José do Rio Preto
2012

Silas Evandro Nachif Fernandes

Sistema de Arquivos Distribuído Flexível e Adaptável

Dissertação submetida 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 obtenção do título de mestre no Programa de Pós-Graduação em Ciência da Computação da UNESP.

Prof.ª Dr.ª Renata Spolon Lobato

Silas Evandro Nachif Fernandes

Banca Examinadora:
Prof.ª Dr.ª Sarita Mazzini Bruschi
Prof. Dr. Norian Marranghello

São José do Rio Preto
2012

Agradecimentos

Agradeço primeiramente a Deus, a minha orientadora Prof.ª Dr.ª Renata Spolon Lobato, à seção administrativa, à Capes que financiou as minhas pesquisas, aos alunos Matheus Della Croce, Renato Martins e Gabriel Saraiva por ajudarem nos testes e avaliações desse projeto, ao grupo do GSPD, aos professores do DCCE e DCo, aos meus amigos, à minha família e ao meu grande amor.

Resumo

Com o aumento do volume de dados e a incerteza dos recursos de *hardware* e *software*, a descentralização dos dados em sistema de arquivos surgiu com a finalidade de diminuir a probabilidade de perda total desses dados. Com isso, este trabalho propõe um modelo de sistema de arquivos distribuído que incorpora características de transparência, escalabilidade, tolerância a falhas, criptografia, suporte a *hardware* de baixo custo, facilidade na implantação e manipulação dos arquivos.

Abstract

With the increasing volume of data and uncertainty of hardware and software resources, the decentralization of data in file systems came up with the aim of reducing the likelihood of total loss of such data. Thus, this paper proposes a model of distributed file systems that integrates features of transparency, scalability, fault tolerance, encryption, support for low cost hardware, easy management and handling of files.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

ACL:	Access Control List
AES:	Advanced Encryption Standard
AFS:	Andrew File System
CODA:	Constant Data Availability
CUDA:	Compute Unified Device Architecture
FEC:	Forward Error Correction
FlexA:	Flexible and Adaptable Distributed File System
FTP:	File Transfer Protocol
FUSE:	Filesystem in Userspace
GB:	Gigabyte
GFS:	Google File System
GHz:	Gigahertz
HDFS:	Hadoop Distributed File System
HTTP:	HyperText Transfer Protocol
IP:	Internet Protocol
KB:	Kbyte
LRU:	Least Recently Used
MB:	Megabyte
Mbps:	Megabit por segundo
NFS:	Network Filesystem
P2P:	peer-to-peer
RAM:	Random Access Memory
RFS:	Remote File Sharing
RPC:	Remote Procedure Call
RPM:	Rotações por minuto
RSA:	Rivest Shamir Adleman
SAD:	Sistema de Arquivos Distribuído

SSL:	Secure Sockets Layer
Tahoe-LAFS:	Tahoe-Least Authority File System
TCP:	Transmission Control Protocol
UCP:	Unidade Central de Processamento
UDP:	User Datagram Protocol
UUID:	Universally Unique IDentifier
VFS:	Virtual File System
WWW:	World Wide Web

1 Introdução

1.1 Motivação

O volume de dados cresce a uma velocidade surpreendente a cada ano, exigindo maiores condições de armazenamento e processamento compatível. Normalmente, esses dados estão centralizados em servidores ou estações simples, suscetíveis a falhas e sobrecargas, gerando interrupções no fornecimento do serviço.

Por mais que os recursos de *hardware* e *software* evoluam, há sempre a possibilidade de acontecer algum problema, como por exemplo falhas nos discos rígidos, sobrecargas do servidor, interrupções da rede de dados dentre outros. Isso levou a descentralização dos serviços com a finalidade de diminuir a perda total dos recursos, ou seja, falhas até podem ocorrer, mas, na maioria dos casos, afetarão somente uma porção dos serviços e não todo o seu conjunto. Além disso, com o aumento do volume de dados há também a necessidade de um compartilhamento maior das informações armazenadas, o que resulta no uso da técnica de sistemas distribuídos.

Há muitas maneiras de expressar o que é um sistemas distribuídos, dentre elas pode-se destacar que consiste de um conjunto de processos distintos e separados que se comunicam através da troca de mensagens (LAMPORT, 1978). Uma outra definição para sistemas distribuídos é de componentes localizados em computadores independentes e interligados que coordenam suas ações sobre troca de mensagens e que se apresentam a seus usuários como um sistema único e coerente (COULOURIS; DOLLIMORE; KINDBERG, 2005; TANENBAUM; STEEN, 2007).

A construção dos sistemas distribuídos vem da motivação em compartilhar recursos computacionais de *hardware*, *software* e dados. Essa motivação envolve alguns desafios como concorrência, inexistência de um relógio global, falhas independentes e outras consequências que serão estudados no decorrer deste trabalho (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Os sistemas distribuídos podem ser encontrados em diferentes modos apoiados sobre as redes de computadores como a internet, intranet e rede móvel.

A internet forma um grande sistema distribuído, em que usuários de qualquer lugar se conectam e usam serviços como transferência de arquivos, *e-mail* e *World Wide Web* (WWW), por exemplo. Por ser formado por um conjunto de protocolos abertos e as comunicações serem por troca de mensagens, novos computadores podem ser conectados e fornecerem novos serviços (COULOURIS; DOLLIMORE; KINDBERG, 2005).

A intranet representa a internet em um ambiente controlado, interligado por várias redes locais (*Local Area Networks* - LANs) e sendo administradas por uma empresa ou organização. A intranet pode oferecer recursos como servidores de arquivos, servidores de *e-mail*, servidor *web* e ainda estabelecer conexão com a internet (COULOURIS; DOLLIMORE; KINDBERG, 2005).

A computação móvel permite que recursos da internet/intranet possam ser agregados a dispositivos móveis, possibilitando maior amplitude do uso dos serviços oferecidos sem a necessidade de permanecer no seu ambiente atual (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Apoiado sobre a ideia de compartilhar recursos, a distribuição na camada do sistema de arquivos para redes traz o conceito de Sistema de Arquivos Distribuído (SAD), ou seja, um sistema de arquivos no qual os arquivos estão armazenados e distribuídos em computadores diferentes interligados através de uma rede (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Nesse contexto, um SAD deve promover uma solução eficiente para o gerenciamento dos arquivos distribuídos de modo que justifique a sua utilização e, na medida do possível, apresente características como transparência de acesso e localização, desempenho, escalabilidade, controle de concorrência, tolerância a falhas e segurança.

Esse conjunto de componentes interligados e agindo como um todo traz vantagens em relação a um crescimento direcionado dos recursos, uso mais eficiente dos ativos computacionais, disponibilidade mediante a replicação dos arquivos e componentes, e em alguns casos, maior mobilidade dos serviços. Contudo, algumas desvantagens são presentes como por exemplo a dificuldade do gerenciamento dos recursos distribuídos, a incerteza da rede de comunicação, a complexidade na cooperação entre os componentes e a segurança dos dados.

Desse modo, existem diversos SADs que procuram suprir as principais características de um sistema distribuído para arquivos. Entretanto, o agrupamento de todos esses atributos forma um conjunto complexo e de difícil administração. Isso porque quando se eleva o nível de complexidade de uma das características, ela pode afetar diretamente as outras funções, como por exemplo ao dar atenção para escalabilidade dos recursos computacionais, a transparência e a consistência dos arquivos podem

ser prejudicadas. Por causa dessa interferência entre os atributos, existe uma quantidade expressiva de SADs que busca priorizar uma determinada característica para um cenário específico (PATE, 2003; COULOURIS; DOLLIMORE; KINDBERG, 2005; HUANG; GRIMSHAW, 2011).

Com base nisso, esse trabalho procura extrair características de alguns SADs com a finalidade de desenvolver um modelo que possa atacar áreas como desempenho e tolerância a falhas. Mas para isso é importante conhecer os desafios que envolvem esses sistemas, o que busca-se esclarecer no Capítulo 2 através dos principais tópicos sobre os sistemas de arquivos distribuídos.

1.2 Objetivos

O objetivo deste trabalho consiste em aliar tecnologias de SADs existentes para construir um modelo flexível e adaptável, denominado de FlexA (***Flex****ible and* ***A****daptable Distributed File System*), o qual tem como propósito fornecer condições para ajuste de suas funções de acordo com as necessidades do ambiente. A sua construção parte da elaboração de um modelo de SAD de código aberto que forneça uma estrutura adaptável às novas implementações e modificações sobre as suas características, as quais são definidas pelo modelo de criptografia, pelo processo de divisão e replicação de arquivos e uma interface que pode ser agregada a outras aplicações. Através disso, busca-se um SAD apto a utilizar recursos computacionais de baixo custo com facilidade no gerenciamento de arquivos e diretórios, aliado ao uso de criptografia rápida e segura e mecanismos para tolerar as falhas.

1.3 Organização do texto

No capítulo 2 são apresentados os conceitos que envolvem os sistemas distribuídos com enfoque nos SADs, descrevendo alguns desafios com relação aos atributos desejáveis.

No capítulo 3 são apresentados os principais SADs e os recursos que cada um oferece, destacando-se as características mais relevantes para auxílio na construção do FlexA.

No capítulo 4 são apresentados o modelo FlexA e sua validação em um ambiente de teste.

No capítulo 5 são realizadas a avaliação e a comparação de desempenho do modelo construído em relação aos SADs estudados.

No capítulo 6 é apresentada a conclusão desse trabalho e são discutidos aspectos para continuidade do projeto.

2 Sistemas de arquivos distribuídos

2.1 Considerações Iniciais

Um SAD segue o mesmo conceito do sistema de arquivos local, porém com o diferencial do primeiro permitir que usuários ou programas remotos leiam ou escrevam dados que, aparentemente, estão armazenados na estação local do usuário, mas, na realidade, os dados estão distribuídos ao longo de uma rede. Essa característica do SAD possibilita que os usuários acessem arquivos de qualquer computador da rede de forma transparente para o usuário ou programa (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Neste capítulo, são apresentados alguns pontos importantes sobre sistemas distribuídos relacionados com os projetos de SADs, destacando-se os tipos de transparência que podem existir (seção 2.2.1), os problemas de desempenho que são comumente encontrados (seção 2.2.2), as dificuldades em adequar o projeto de SAD frente ao crescimento dos recursos computacionais (seção 2.2.3), a administração da concorrência dos arquivos (seção 2.2.4), as maneiras para contornar as falhas (seção 2.2.5), os modos para proporcionar confiabilidade dos dados (seção 2.2.6), a garantia da disponibilidade dos dados (seção 2.2.7) e os modelos arquiteturais (seção 2.3).

2.2 Características

Para o desenvolvimento de sistemas distribuídos, alguns tópicos importantes devem ser observados. Um sistema distribuído é caracterizado por fornecer fácil acesso aos seus recursos sem revelar que os seus componentes são distribuídos pela rede, permitindo que o conjunto possa ser expandido (TANENBAUM; STEEN, 2007).

Uma das principais características dos sistemas distribuídos é promover facilidade no acesso aos recursos distribuídos para os usuários, e às aplicações, de maneira controlada e inteligente.

Os recursos distribuídos podem ser de *hardware* como por exemplo uma impres-

sora ou *array* de discos, de *software* por meio do compartilhamento dos módulos de uma aplicação, ou de informação através do compartilhando dos dados.

Os principais atributos dos sistemas distribuídos são destacados pela transparência de acesso e localização, desempenho, escalabilidade, controle de concorrência, tolerância a falhas e segurança. Tais atributos são descritos a seguir.

2.2.1 Transparência

A transparência consiste em promover acesso a recursos distribuídos aparentando-se como um único sistema para usuário ou aplicação. Considerando os SADs, ela pode ser dividida entre alguns aspectos conforme Coulouris, Dollimore e Kindberg (2005) e Tanenbaum e Steen (2007):

- Transparência de acesso: não necessita fornecer a localização dos recursos, ou seja, os programas devem executar os processos de leitura e escrita de arquivos remotos da mesma maneira que operam sobre os arquivos locais, sem qualquer modificação no programa. O usuário não deve perceber se o recurso acessado é local ou remoto.

- Transparência de localização: os programas clientes devem ver um espaço de nomes de arquivos uniforme, sem a necessidade de fornecer a localização física dos arquivos para encontrá-los, mesmo que esses arquivos se desloquem entre os servidores.

- Transparência de mobilidade: independente dos arquivos se moverem entre servidores, os programas clientes não precisam ser alterados para a nova localidade do grupo de arquivos. Essa característica permite flexibilidade em mover arquivos sem comprometer toda a estrutura, ou ter que refazer *links* entre programas clientes e o local do arquivo.

- Transparência de desempenho: o desempenho da aplicação cliente não poderá ser comprometido enquanto ocorre uma variação dos processos sobre os recursos disponíveis pelos SADs, isto é, mesmo que haja concorrência no acesso pelos arquivos isso não deve afetar os usuários.

- Transparência de escalabilidade: os recursos computacionais podem sofrer alterações para abrigar maior poder computacional ou o ingresso de novos servidores sem prejudicar o serviço.

- Transparência a falhas: garantir a disponibilidade dos arquivos ininterruptamente e se ocorrerem falhas o programa cliente não deverá saber como elas serão tratadas.

- Transparência de replicação: várias cópias dos mesmos arquivos armazenados em locais diferentes para garantir a disponibilidade. A aplicação cliente deverá visualizar apenas uma cópia do mesmo, não necessitando saber a quantidade replicada e o local.

A transparência é altamente desejável em sistemas distribuídos, mas nem sempre é possível alcançá-la ou, em determinadas situações, não convém ocultá-la. Pode-se destacar uma situação que seja mais conveniente o usuário tomar uma decisão sobre alguma falha do que o sistema distribuído tentar resolver por si só. Isso pode ser observado quando um serviço, por repetidas vezes tenta, estabelecer uma comunicação com o servidor na internet, neste caso o melhor é informar ao usuário sobre a falha e que ele tente mais tarde, afirmam Tanenbaum e Steen (2007).

2.2.2 Desempenho

Nos SADs, os principais fatores observados que podem influenciar o nível de desempenho são comumente representados pela latência da rede e/ou latência dos servidores.

Uma forma de aliviar a latência é através do uso de técnicas de *cache*. Com o uso da *cache* é possível armazenar informações que são acessadas frequentemente, minimizando o custo de obtê-las novamente do local de origem. Esse processo evita que toda solicitação de arquivos viaje através da rede até o servidor onde o arquivo está armazenado.

A memória *cache* pode ser adotada em duas situações: a primeira ao lado do servidor, o que elimina o processo de buscas em discos rígidos; e a segunda ao lado do cliente, que supre as falhas/lentidão provocadas pela rede. Geralmente, devido as limitações da quantidade de memória principal dos computadores, a *cache* pode ser estendida para uma espaço reservado em disco.

Entretanto, se por um lado a *cache* melhora a latência, por outro ela causa problemas de sincronização. Por exemplo, se os dados presentes na *cache* do cliente são alterados é necessário sincronizar com o servidor, que, por conseguinte, deve avisar a todos os seus clientes que estão com uma versão mais antiga dos dados. Além disso, é importante a presença de um mecanismo que gerencie os níveis de uso da *cache*,

eliminando dados não utilizados e mantendo os mais atuais ou que poderão ser utilizados futuramente. Uma técnica que atende essa finalidade é o algoritmo *Least Recently Used* (LRU) (TANENBAUM; STEEN, 2007).

Outro aspecto importante, porém menos abordado, é o tempo envolvido na criptografia. A criptografia está relacionada nos casos em que o uso de recursos computacionais não são supervisionados ou que a exposição dos arquivos possam comprometer o conjunto.

Na maioria dos SADs, assume-se que os dados estão assegurados pelo servidor e, com isso, o foco da segurança é centralizado no canal de comunicação entre servidores e clientes. Ainda assim, os recursos criptográficos sobre os arquivos proporcionam um nível maior de segurança (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Em ambiente não controlado ou supervisionado, há a possibilidade dos arquivos armazenados no servidor serem acessados localmente por usuários não autorizados. É nesse cenário que a criptografia feita diretamente no arquivo impede que os dados sejam visíveis ou alterados.

A variação no desempenho com o uso de criptografia depende do algoritmo utilizado, do tamanho do arquivo e do *hardware* do computador.

2.2.3 Escalabilidade

A escalabilidade define a eficiência do sistema, conforme se elevam o número de recursos e o número de usuários. Alguns desafios sobre a escalabilidade em sistemas distribuídos são postulados por (COULOURIS; DOLLIMORE; KINDBERG, 2005; TANENBAUM; STEEN, 2007), os quais são descritos por fornecerem condições em agregar mais componentes conforme a demanda aumenta, por maximizar a eficiência das interações entre os recursos a fim de evitar perda no desempenho, por evitar projetos de sistemas com um número limitado de usuários ou recursos agregados e pela descentralização com o intuito de evitar gargalos.

Diante disso, algumas situações que podem acontecer com o uso de SADs são exploradas nos itens a seguir (COULOURIS; DOLLIMORE; KINDBERG, 2005):

- Sobrecarga do servidor: muitas requisições ao servidor geram um tempo maior de latência. O uso da *cache* aliviaria os frequentes acessos aos discos rígidos, porém os muitos tipos de acessos levariam a uma rápida atualização dos arquivos em *cache*, sendo rapidamente substituídos por outros, causando sobrecarga do sistema.

- Sobrecarga do cliente: o uso de *cache* cliente não resolveria completamente o problema do alto volume de requisições, pois em um cenário com muitos clientes utilizando o mesmo arquivo, qualquer alteração implicará no servidor informar aos demais clientes que a versão do arquivo em *cache* encontra-se desatualizada e uma nova requisição será necessária, provocando grande consumo dos recursos de rede.

- Sobrecarga na rede: se o SAD foi planejado para alta escalabilidade através de simples replicação dos arquivos pelos nós responsáveis em armazená-los, a latência aumentaria devido às buscas pelo arquivo em cada servidor da rede, pois não se sabe o local exato deste arquivo. Nesse caso, um servidor específico poderá ser utilizado para a resolução dos caminhos, agindo como centralizador, mas falhas ou sobrecarga poderão ocorrer sobre ele, comprometendo o sistema.

- Incompatibilidade de *hardware* e sistema operacional: se não definidas as interfaces do serviço de maneira que o *software* cliente e servidor possam ser implantadas por diferentes plataformas e computadores, o SAD poderá não ser compatível com as variações de *hardware* e as mudanças do sistemas operacional.

- Consistência dos arquivos: o acesso concorrente aos arquivos distribuídos nos quais há requisições de leitura e escrita pode ocasionar alguns problemas de consistência nos dados. Os sistemas de arquivos tradicionais oferecem a semântica de atualização de cópia única (*one-copy*) a qual, independente do acesso concorrente de diversas cópias replicadas, faz com que o retorno para o usuário seja da existência de apenas um arquivo.

2.2.4 Controle de concorrência

Quando vários usuários acessam o mesmo arquivo, em modo leitura, não existem problemas. Porém, a operação de escrita necessita ser controlada. O controle de concorrência gerencia todas as ações de leitura e escrita de um arquivo ou grupo deles. A ideia é que as alterações feitas por um único cliente não influenciem as operações de outros clientes que acessam naquele exato momento o mesmo arquivo. Existem técnicas e algoritmos responsáveis em administrar várias requisições de escritas sobre o mesmo arquivo. A implantação delas varia de acordo com a aplicação, mas a maior parte dos serviços de arquivos atuais seguem através de travas (*locks*) em nível de arquivo ou em nível de registro (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Com esse bloqueio é possível serializar as operações sobre o arquivo através de listas de ações a serem executadas. O resultado obtido da serialização é o mesmo que

se fossem executadas simultaneamente (COULOURIS; DOLLIMORE; KINDBERG, 2005). Porém, um problema em utilizar essa técnica é o *deadlock* que, em linhas gerais, significa um conjunto de processos em que um ou mais aguardam o desbloqueio de um determinado recurso que, por sua vez, aguarda o desbloqueio de outro recurso em uso pelo primeiro processo (KSHEMKALYANI; SINGHAL, 2008).

2.2.5 Tolerância a falhas

Um dos objetivos dos sistemas distribuídos, que por consequência estende-se aos SADs, é garantir a recuperação automática de falhas parciais, sem que isso interfira de forma significativa no desempenho do conjunto. Considerando que as falhas são inevitáveis, a maior preocupação concentra-se em detectá-las e como tratá-las. Entretanto, a detecção não é um processo exato, pois em um cenário não supervisionado, um problema consiste em como distinguir falhas reais de problemas de latência da rede. Nesse sentido, o tratamento das falhas são apoiadas em suposições de que algo pode estar errado e, através disso, ocultá-las do usuário e tratá-las. Ciente disso, algumas atitudes podem ser aplicadas como mascaramento de falhas, recuperação após falhas e redundância (COULOURIS; DOLLIMORE; KINDBERG, 2005).

O **mascaramento de falhas** condiz com situações de fácil resolução, em que problemas podem ser ocultados dos usuários enquanto o sistema realiza novas tentativas de acesso ou envio de novas mensagens, por exemplo. O próximo passo consiste na **recuperação após falhas**, o qual envolve o armazenamento do estado do atual processo, permitindo a sua rápida recuperação caso ocorram interrupções ou quedas do serviço. Por fim, a **redundância** dos componentes envolvidos traz a garantia de que haverá mais de um meio ou recurso disponível em caso de falha, permitindo a continuidade das operações.

A tolerância a falhas também compreende um conjunto de medidas que procuram antecipar e contornar possíveis problemas. Geralmente, esses problemas acontecem quando há uma sobrecarga do uso dos recursos de *software* e *hardware* do sistema, resultando em mensagens perdidas, mensagens corrompidas, travamento de sistemas e dados corrompidos (TANENBAUM; STEEN, 2007).

Das falhas pode-se destacar os seguintes problemas:

- **falha por queda**, problema físico ou lógico no servidor, causando o travamento do sistema operacional;
- **falha por omissão**, significa o não recebimento das mensagens, quer seja por causa de mensagens não aceitas ou por não enviar uma resposta depois da ação

concluída;

- **falha de temporização**, ocorre quando uma reposta está fora do intervalo de tempo adequado;
- **falha de resposta** consiste da resposta emitida estar incorreta, ou seja, o retorno não condiz com a solicitação;
- **falha arbitrária**, também conhecida como falha bizantina, ocorre quando um servidor envia mensagens inadequadas, mas que não são consideradas como incorretas. Também pode ser associada a um servidor que está atuando maliciosamente e, portanto, emitindo respostas erradas de forma proposital.

Em alguns casos, a saída mais adequada para tratar as falhas consiste da interação do usuário, como adiar as ações de um processo enquanto o serviço responsável por ele não estiver respondendo.

2.2.6 Segurança

Algumas medidas devem ser adotadas em favor da privacidade e da integridade dos arquivos. Nesse aspecto, os SADs utilizam recursos de criptografia, canais seguros e controle de acesso para evitar que dados sejam acessados por usuários sem permissões. Esses fatores levantam três aspectos de grande importância: confiabilidade (impedir acesso não autorizado), integridade (garantir a qualidade do arquivo mediante modificações não autorizadas ou falhas no sistema) e disponibilidade (garantir o estado de acesso aos recursos contra problemas provenientes do meio) (KSHEMKALYANI; SINGHAL, 2008).

O processo de confiabilidade pode ser alcançado pelo uso de técnicas de criptografia e autenticação, cifrando o conteúdo de um arquivo de tal modo que somente quem possuí a chave criptográfica poderá vê-lo. Já a autenticação verifica a identidade do solicitante por meio de chaves de acesso, ou outra técnica que o identifique. Além disso, após a autenticação, é analisado se determinada ação solicitada condiz com as permissões estabelecidas para aquela chave de identificação. Esse conjunto de criptografia, autenticação e autorização compõe o chamado mecanismo de segurança, o qual faz parte das políticas de segurança que é responsável em descrever quais ações são permitidas ou bloqueadas (TANENBAUM; STEEN, 2007).

Outra técnica importante para proteção do conteúdo transmitido emprega o uso de canais seguros, nos quais são utilizados recursos de criptografia, autenticação e conceitos de relógio lógico, ou físico, para promover um meio que garanta privacidade

e integridade dos dados transmitidos, impedindo que as mensagens sejam interceptadas e reproduzidas ou reordenadas (COULOURIS; DOLLIMORE; KINDBERG, 2005).

2.2.7 Replicação de arquivos

A garantia da disponibilidade dos arquivos em SADs é algo que não se pode prever, mas que é possível antecipar. A disponibilidade se faz com o uso da replicação de arquivos, técnica que pode ser empregada com ações simples como uso de *caches* no lado do cliente ou recursos mais avançados, os quais envolvem réplicas mantidas por gerenciadores de réplicas. Neste último caso, a replicação pode atuar em dois modos, passivo ou ativo.

A replicação passiva, igualmente conhecida como *backup* primário, atua em conjunto com um grupo de gerenciadores de réplicas, sendo que um é o gerenciador primário e o restante os secundários, ou *backups* escravos. A medida que as alterações são realizadas no gerenciador primário, as cópias atualizadas são enviadas aos *backups* escravos. Caso ocorra alguma falha com o servidor primário, um membro dos *backups* é eleito para assumir a posição de líder.

Na replicação ativa, os computadores do grupo de réplicas desempenham a mesma função, ou seja, a requisição é enviada a todos os gerenciadores que processam e respondem independentemente. Nesse modelo, a falha de algum gerenciador não implica no desempenho do serviço, visto que não é necessário reorganizar os membros do grupo com a finalidade de eleger um líder (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Um problema associado em manter várias cópias distribuídas é garantir a consistência delas. Operações de leitura não oferecem muitos problemas, mas a escrita precisa de maior atenção, pois após a atualização de algum arquivo, ele necessariamente deve ser replicado antes que um próximo evento exija a sua utilização. Com isso, a sincronização das réplicas precisam ser gerenciadas através de métodos que garantam consistência dos arquivos e um consumo reduzido da banda (TANENBAUM; STEEN, 2007).

A eficiência do método de sincronização varia entre tornar as cópias consistentes ou economizar na comunicação e transferência dos arquivos. Isso porque ao elevar a prioridade para consistência dos dados, conferências de validação são executadas em pouco espaço de tempo, aumentando o consumo da rede e tornando o processo pouco eficiente e muito custoso para o sistema. Por outro lado, um tempo maior na conferência das versões dos arquivos em benefício ao consumo da rede pode ocasionar a defasagem dos dados.

2.3 Modelos Arquiteturais

Discutidas algumas características que envolvem os SADs, é necessário descrever a forma como esse conjunto pode ser organizado dentre os modelos de arquiteturas de sistemas (TANENBAUM; STEEN, 2007): arquiteturas centralizadas, arquiteturas descentralizadas e arquiteturas híbridas.

As arquiteturas centralizadas expressam o conceito de que clientes solicitam serviços oferecidos por servidores. Esse tipo de arquitetura cliente-servidor é mais utilizada e menos complexa para implantação e administração comparado a outras arquiteturas. Em determinado momento o servidor também pode desempenhar papel de cliente em relação a outros servidores, como apresentado na Figura 2.1.

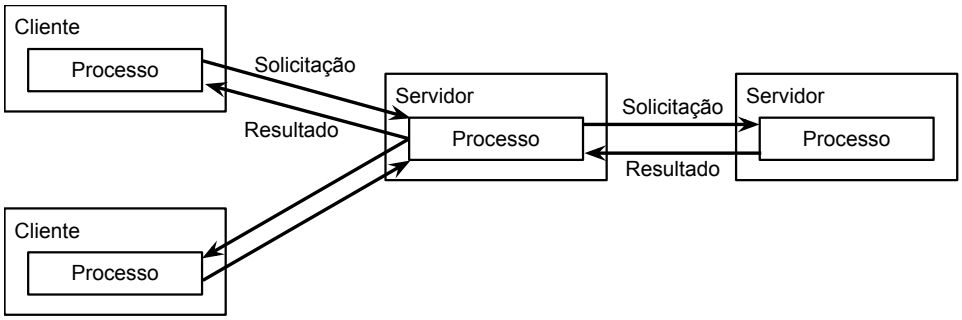


Figura 2.1: Arquitetura cliente-servidor (COULOURIS; DOLLIMORE; KINDBERG, 2005)

Há também a variação quanto a divisão de funções, onde o cliente pode desempenhar partes da interface de usuário e o servidor se responsabiliza pelo restante do conjunto, o qual pode incluir a aplicação e outros processos como o banco de dados, por exemplo.

No modelo descentralizado, o exemplo mais conhecido é o *peer-to-peer* (P2P). Nessa arquitetura não há distinções entre os processos como acontece no cliente-servidor, ou seja, todos os envolvidos possuem igual importância para o sistema, fornecendo um serviço uniforme, como mostrado na Figura 2.2 (COULOURIS; DOLLIMORE; KINDBERG, 2005).

As arquiteturas híbridas compreendem a união da arquitetura centralizada com descentralizada. Esse arquitetura é mais utilizada para dar início a um serviço, para que em seguida a aplicação consiga se estabelecer com outras partes distribuídas. Um exemplo são as redes *torrent*, que consistem em usuários solicitando informações a um servidor sobre quais clientes possuem um determinado arquivo. Ao final, a transferência do arquivo ocorre sobre a arquitetura P2P com os outros clientes, como ilustrado na Figura 2.3 (TANENBAUM; STEEN, 2007).

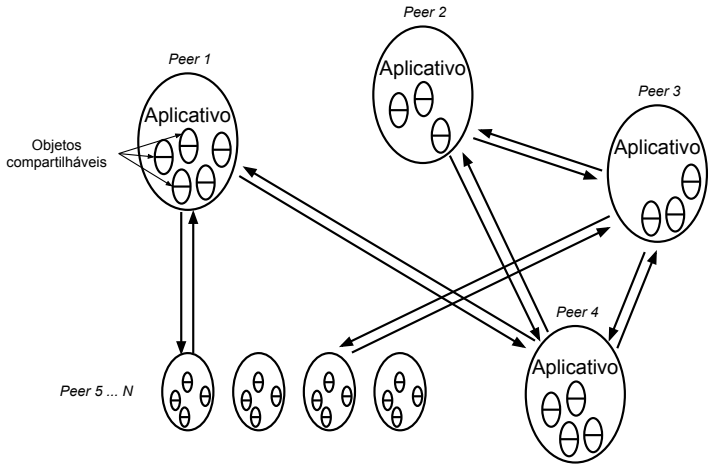


Figura 2.2: Arquitetura Peer-To-Peer (COULOURIS; DOLLIMORE; KINDBERG, 2005)

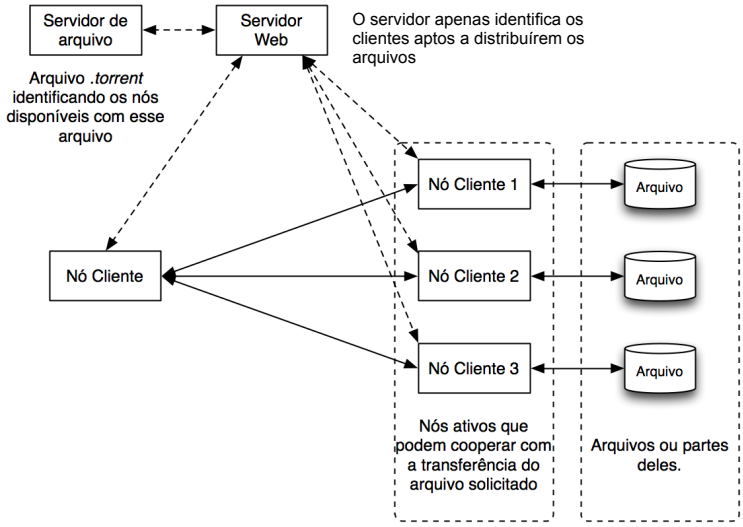


Figura 2.3: Funcionamento da estrutura *torrent* (TANENBAUM; STEEN, 2007)

2.4 Considerações Finais

O compartilhamento de recursos tornou-se necessário para as diversas interações entre os ativos computacionais, que vão desde periféricos como por exemplo impressoras dentro de uma empresa, até um ampla rede que proporcione transferência de arquivos e dados como a internet ou intranet. No início, os sistemas distribuídos podiam ser vistos como um método para economizar recursos de *hardware* e/ou *software*, mas o seu conceito evoluiu, propagando-se em toda parte como um meio eficiente para a distribuição dos recursos.

Nesse capítulo delimitou-se na relação com a distribuição dos arquivos, abordando as principais características de sistemas distribuídos que afetam os SADs e apresen-

tando alguns desafios e soluções.

Pode-se destacar como características importantes a transparência, o desempenho, a escalabilidade, o controle de concorrência, a tolerância a falhas e a segurança, que juntas formam um projeto de SAD desejável, porém complexo. Frente a essa quantidade de recursos, ao elevar o nível das interações de uma característica, as outras são afetadas diretamente, como é o caso da escalabilidade em relação a consistência dos arquivos. Com isso, sistemas são projetados com foco em determinados pontos, o que coopera para uma quantidade expressiva de SADs priorizando certas funções para um uso mais direcionado. É nesse sentido que no Capítulo 3 são estudados alguns SADs e suas principais características.

3 Estudos de casos

3.1 Considerações Iniciais

Dentre os diversos SADs existentes, nesse trabalho concentrou-se em explorar os recursos fundamentais de alguns modelos com base nos tradicionais projetos e alguns sistemas mais recentes. Dessa forma, neste capítulo é apresentado como esses SADs administram as suas funções, trazendo maior conhecimento para a construção do FlexA.

Na década de 1980, muitos SADs já chamavam a atenção na comunidade UNIX, destacando-se o Sun *Network Filesystem* (NFS), o *Remote File Sharing* (RFS) da AT&T e o *Andrew File System* (AFS) da Universidade de Carnegie Mellon. Esses SADs são descritos como modelos cliente-servidor nos quais arquivos de uma unidade de disco são servidos aos clientes pelo uso de um protocolo de transporte baseado na semântica UNIX, apresentando-se de modo transparente como se fosse um simples acesso local (PATE, 2003).

Dentre esses SADs, o NFS e o AFS foram os que receberam maior destaque na literatura. O NFS, desenvolvido inicialmente pela Sun Microsystem, é um dos mais conhecidos pois, embora existissem vários SADs operando em universidades e laboratórios, as interfaces e definições do NFS foram disponibilizadas em domínio público. Essa atitude permitiu que várias implementações fossem realizadas, tornando-o amplamente compatível entre os sistemas operacionais (PATE, 2003; COULOURIS; DOLLIMORE; KINDBERG, 2005; TANENBAUM; STEEN, 2007).

O AFS, desenvolvido na Universidade de Carnegie Mellon em conjunto com a IBM para servir de infraestrutura para base educacional, trouxe foco na alta escalabilidade com a utilização de milhares de clientes compartilhando arquivos de centenas de servidores. Destaca-se pela otimização das trocas de informações entre clientes e servidores e uso de políticas agressivas de *cache* no cliente (PATE, 2003; COULOURIS; DOLLIMORE; KINDBERG, 2005; TANENBAUM; STEEN, 2007).

Pode-se destacar outros SADs de grande importância como o SPRITE (NELSON;

WELCH; OUSTERHOUT, 1987), CODA (KISTLER; SATYANARAYANAN, 1992), *IBM General Parallel File System* (BARRIOS et al., 1998), Ceph (WEIL et al., 2006), XtreemFS (HUPFELD et al., 2008), HDFS (SHVACHKO et al., 2010), *Red Hat Global File System* (REDHAT, 2011) e GlusterFS (GLUSTER, 2011). Dentre estes SADs, a escolha do NFS, AFS e GFS para o desenvolvimento deste trabalho é justificada por extensa documentação disponível, permitindo explorar problemas comumente encontrados nos projetos de SADs. Quanto a escolha do Tahoe-LAFS, a sua importância para este projeto foi motivada pelo seu desenvolvimento em projeto de código aberto, fornecendo um modelo para o acesso de arquivos por *upload/download* (carga/atualização) e uso do princípio de menor autoridade (KARP, 2003; MILLER, 2006) para distribuir arquivos.

3.2 *Network File System*

O NFS foi projetado com o objetivo de atender algumas das características exploradas no capítulo 2, principalmente a transparência no acesso aos arquivos. Ele foi inicialmente projetado para ambientes UNIX, com a premissa de compartilhar remotamente os arquivos de modo transparente das estações da Sun Microsystem. Foi o primeiro SAD projetado como um produto final e que disponibilizou em domínio público as definições do seu protocolo, o qual usa uma interface de *Remote Procedure Call* (RPC). Foram feitas várias adaptações que o tornaram compatível com uma variedade de *software* e *hardware* (COULOURIS; DOLLIMORE; KINDBERG, 2005). A sua arquitetura é mostrada na Figura 3.1.

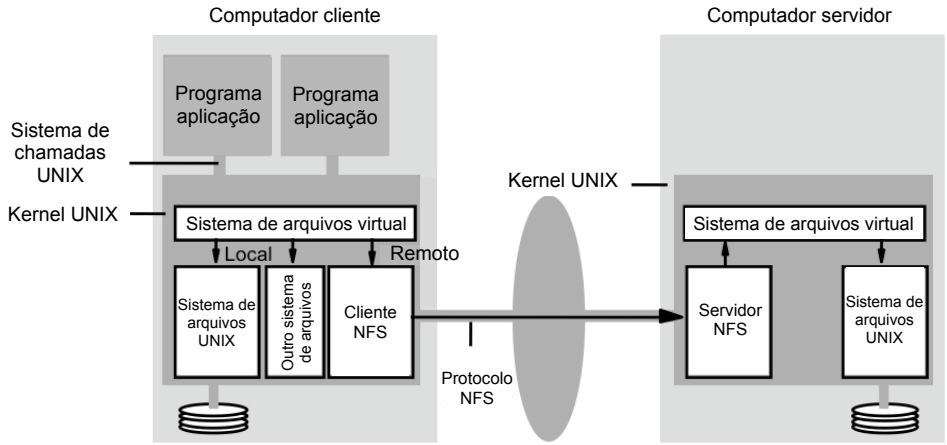


Figura 3.1: Arquitetura NFS (TANENBAUM; STEEN, 2007)

3.2.1 Arquitetura

Na interface cliente, a transparência do NFS ocorre através de um sistema de arquivos virtual (*Virtual File System* - VFS), responsável por encaminhar as requisições ao módulo de sistemas de arquivos apropriado, agindo como um intermediário para as aplicações do usuário. Isso possibilita trabalhar com o sistema de arquivos local, com o módulo do NFS ou outro sistema de arquivos (COULOURIS; DOLLIMORE; KINDBERG, 2005).

A interface cliente promove um meio conveniente para a aplicação do usuário, fornecendo um serviço de arquivo por **acesso remoto**, exemplificado na Figura 3.2a, que disponibiliza uma interface de operação sobre o arquivo na estação cliente, mas, na verdade, representa um conjunto de extensões que são encaminhas remotamente até o servidor e aplicadas diretamente aos arquivos.

Uma forma mais comum de realizar as operações sobre o arquivo é pelo modelo de **carga/atualização**, conforme Figura 3.2b, o qual consiste no cliente copiar todo o arquivo para a estação local e, em seguida, realizar as operações sobre ele. Ao final, o arquivo atualizado é submetido ao servidor, permitindo que outros usuários possam utilizá-lo (TANENBAUM; STEEN, 2007).

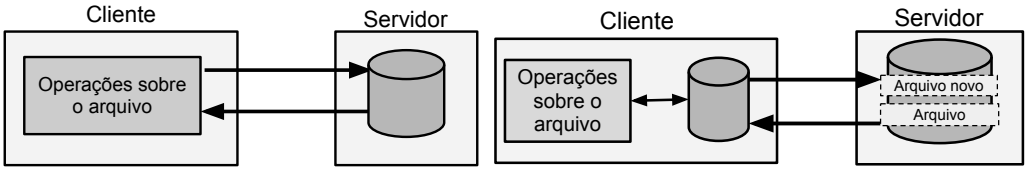


Figura 3.2: a) Modelo acesso remoto b) Modelo carga atualização (TANENBAUM; STEEN, 2007)

No lado do servidor, o NFS foi projetado inicialmente para ser sem estado, ou seja, o servidor não mantém os arquivos abertos pelos clientes. Isso simplifica todo o processo, pois, caso ocorra alguma queda do serviço, não é necessário recuperar as informações durante o reinício do sistema.

Para o cliente, a ausência de estados significa não conhecer a real situação do conjunto distribuído. Em caso de falha e reinício do sistema, o servidor não tem o conhecimento se alguma requisição foi interrompida e consequentemente não continua de onde parou antes do acontecido. Cabe ao cliente prosseguir com a solicitação, repetindo-a até que o servidor responda (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Posteriormente, o NFS foi restruturado para fornecer suporte às operações com estados em conjunto com o cliente, permitindo que ambos saibam sobre as operações realizadas em determinados arquivos. Isso auxilia no modo como a *cache* irá trabalhar,

além de possibilitar o uso de travas nativamente para o bloqueio de blocos de arquivos e a realização de operações atômicas, sem a implementação de soluções externas. Outra característica foi a substituição do protocolo de transporte *User Datagram Protocol* (UDP) pelo *Transmission Control Protocol* (TCP) na comunicação entre servidores e clientes, possibilitando confirmar o recebimento das mensagens (SHEPLER, 1999; SHEPLER et al., 2003; BATSAKIS; BURNS, 2005).

A organização do espaço de nomes procede com a utilização de uma hierarquia de diretórios UNIX através de um processo separado, chamado de serviço de montagem. Em cada servidor há uma lista de acesso contendo os nomes dos diretórios e quem poderá montá-los. No lado do cliente há um comando *mount* modificado para solicitar a montagem de sistemas de arquivos remotos. A comunicação através desse comando ocorre utilizando-se um protocolo de montagem apoiado em RPC, que recebe do cliente o nome do diretório e retorna o manipulador do arquivo do diretório específico, isso se as permissões de acesso do usuário forem suficientes (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Para facilitar o processo de montagem dos diretórios, foi acrescentado ao cliente NFS um recurso chamado de *automounter*, que armazena os nomes de caminho dos servidores. Quando o usuário busca resolver um determinado caminho, é encaminhado ao *automounter* que procura em sua lista o sistema de arquivos solicitado e envia um pedido de montagem do sistema de arquivos remoto. Depois disso, não será mais necessário chamar o *automounter*, pois os acessos ocorrerão utilizando diretamente a montagem realizada (COULOURIS; DOLLIMORE; KINDBERG, 2005).

3.2.2 *Cache* no servidor

A *cache* armazena blocos de arquivos, diretórios e atributos de arquivos que foram lidos do disco. Uma forma de agregar maior agilidade à *cache* é usando o método *read-ahead*, que antecipa quais serão os próximos blocos a serem lidos e já procura carregá-los na memória. Para não limitar a quantidade de dados, a *cache* assume um tamanho variável, permitindo maior flexibilidade para trabalhar com arquivos grandes.

Outra medida adotada para agilizar os processos da *cache* é a utilização da técnica *delayed-write*, o qual consiste em protelar a efetivação dos dados nos servidores em virtude das frequentes alterações que aquele arquivo possa sofrer. A efetivação ocorre somente após um período de tempo sem alterações pelo cliente, evitando constantes acessos de escrita ao mesmo arquivo a uma curta fração de segundos, o que ajuda a diminuir o consumo da banda e consequentemente a latência. Por fim, a operação *sync* que copia para o disco os blocos alterados a cada trinta segundos (KON, 1994;

COULOURIS; DOLLIMORE; KINDBERG, 2005).

Uma característica importante é a garantia de que os blocos dos arquivos foram realmente gravados em disco. As operações de escrita podem variar de duas formas: na primeira o arquivo é armazenado na *cache* e gravado em disco, somente após isso é enviada uma resposta ao cliente; a segunda maneira implica nas operações de escrita permanecerem somente na *cache* até que seja informada a efetivação (*commit*) daquele arquivo e, então, gravadas em disco. Na sequência, uma resposta da operação *commit* é encaminhada (COULOURIS; DOLLIMORE; KINDBERG, 2005).

3.2.3 *Cache* no cliente

A verificação dos dados em *cache* cliente procede através de um processo baseado em *timestamps*, que é o responsável pela validade dos dados armazenados.

A análise dos dados é executada em períodos de tempo, identificando se a última modificação dos arquivos no cliente corresponde com os dados do servidor. Se houver divergência, a *cache* cliente é atualizada.

A frequência da análise pode ser ajustada para adequar entre consistência e eficiência do conjunto, pois tempo muito curto para as atualizações apresentará forte consistência, garantindo que o arquivo na *cache* cliente seja o mais atual. Entretanto, esse ajuste aumenta o consumo dos dados trafegados pela rede, tornando ineficiente à medida que se diminui o tempo de verificação. Se o tempo for muito longo, poderá comprometer a consistência, abrindo margem para a utilização de arquivos desatualizados (COULOURIS; DOLLIMORE; KINDBERG, 2005).

O cliente pode gerenciar suas operações, permitindo que ele modifique os arquivos da sua *cache*, sem a necessidade de atualizar o servidor imediatamente após as alterações. As modificações são validadas no servidor somente após a requisição de outro cliente pelo arquivo ou após um tempo específico para sincronização, obtendo ganhos significativos sobre a latência caso o arquivo não seja utilizado concorrentemente (BATSAKIS; BURNS, 2005).

3.2.4 Segurança

Organizado sobre a arquitetura cliente-servidor e apoiado nas chamadas de procedimento remoto, o NFS concentra-se em garantir um canal de comunicação seguro seguido de um método efetivo para autenticação do usuário (TANENBAUM; STEEN, 2007).

A autenticação do NFS evoluiu do sistema fundamentado no ID do usuário, pas-

sando por troca de chaves Diffie-Hellman e terminando com o suporte ao protocolo RPCSEC_GSS em conjunto com Kerberos v5.

O protocolo RPCSEC_GSS expressa uma grande quantidade de mecanismos de segurança no auxílio à proteção dos canais de comunicação e suporte à integridade e confidencialidade das mensagens. O RPCSEC_GSS atua como uma camada sobre as interfaces de segurança que, entre elas, traz o suporte ao Kerberos v5 (KOHL; NEUMAN, 1993) e ao método de chaves públicas conhecido como Lipkey, descrito por Eisler (2000).

O desenvolvimento do NFS seguiu o critério de não utilizar mecanismos de segurança próprio, em vez disso optou-se por apenas padronizar o modo para a manipulação da segurança, permitindo que novos mecanismos de segurança sejam incorporados. Enquanto isso, o gerenciamento do acesso ocorre pelo uso da lista de controle de acesso (*Access Control List* - ACL) que especifica as permissões para o usuário ou grupo (TANENBAUM; STEEN, 2007).

3.3 *Andrew File System*

O AFS difere do NFS em seu projeto de implementação, atribuindo principalmente forte escalabilidade, característica para qual o NFS não oferecia uma solução otimizada.

O principal destaque desse SAD é o suporte a alta escalabilidade com muitos acessos simultâneos, condição possível devido à sua política de *caches* agressivas nos clientes e métodos de sincronização consistentes com melhor desempenho.

3.3.1 Arquitetura

O AFS é implementado sobre dois processos, Venus e Vice, que definem os módulos cliente e servidor, respectivamente (Figura 3.3).

No lado do cliente, a organização do espaço de nomes utiliza uma hierarquia de diretórios UNIX, sendo responsável pelos arquivos locais e compartilhados. A área responsável pelos arquivos compartilhados é definida como uma subárvore específica da hierarquia local (Figura 3.4), denominada de *cmu*.

Esses arquivos compartilhados pertencem aos servidores, mas cópias deles são mantidas na *cache* do cliente, enquanto os arquivos locais são administrados somente pelos processos locais da estação cliente (COULOURIS; DOLLIMORE; KINDBERG, 2005).

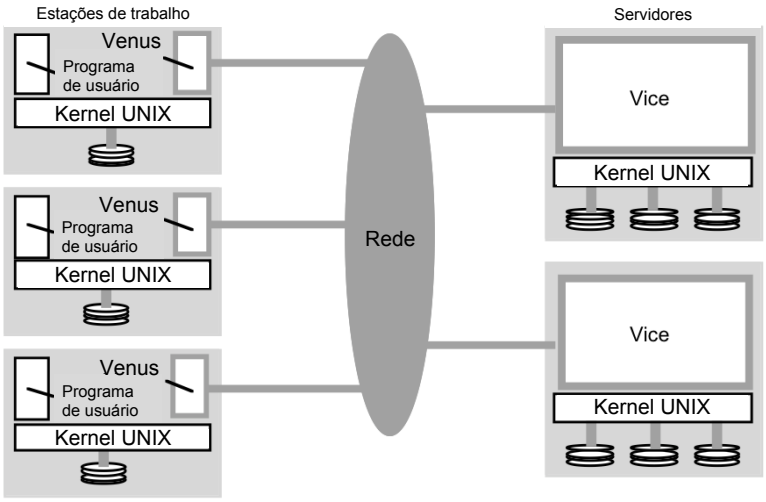


Figura 3.3: Arquitetura AFS (COULOURIS; DOLLIMORE; KINDBERG, 2005)

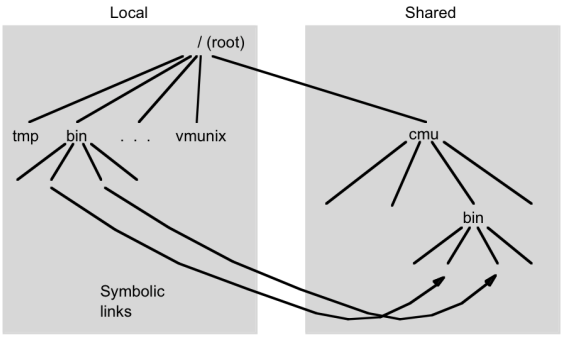


Figura 3.4: Espaço de nomes no AFS (COULOURIS; DOLLIMORE; KINDBERG, 2005)

Sob a ótica do usuário, as chamadas de sistema de arquivos como *open*, *close* entre outras, que forem destinadas a subárvore *cmu*, serão encaminhadas ao processo Venus que possui o papel de verificar as cópias locais com as dos servidores. Se não houver cópias em *cache*, é solicitado ao processo Vice uma nova cópia desses arquivos para estação local. Ao final, se existirem alterações no aquivo pelo cliente, essa cópia é enviada ao processo Vice, que se encarrega de atualizar os dados nos discos dos servidores (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Quanto à replicação, os arquivos compartilhados e raramente alterados pelos clientes podem ser administrados de forma diferente. A fim de evitar que problemas de consistência ocorram, somente uma cópia do arquivo é mantida com permissão leitura-escrita, as demais distribuídas pelos servidores Vice são apenas leitura. Se ocorrerem alterações no arquivo, a atualização é feita sobre a cópia leitura-escrita que é replicada para as outras cópias somente leitura (COULOURIS; DOLLIMORE; KINDBERG, 2005).

3.3.2 *Cache*

Para alcançar altos níveis de escalabilidade, o AFS usou técnicas agressivas de *cache* no cliente, que consistem em armazenar grandes porções dos arquivos na estação local do usuário. Nesse processo, a ação de servir dados ocorre com a cópia de diretórios e arquivos dos servidores para as estações clientes em porções de 64 Kbytes (KB), tamanho que ajuda a reduzir as trocas de mensagens e otimizar as interações. Os arquivos são armazenados na *cache*, sendo persistentes as reinicializações da estação local, ou seja, não há necessidade de copiá-los novamente do servidor para acessos futuros.

A permanência dos arquivos sobre condições de falhas ajuda a aumentar a escalabilidade do sistema. Neste caso, mesmo que o sistema reinicie inesperadamente e não consiga estabelecer uma conexão com o servidor, o cliente ainda terá uma cópia na *cache*. Esse conceito ajuda no desempenho das operações sobre os arquivos evitando muitos acessos aos servidores e diminuindo a latência do sistema (PATE, 2003; COULOURIS; DOLLIMORE; KINDBERG, 2005).

Contudo, o uso agressivo da *cache* pode gerar inconsistência dos dados e é nesse sentido que o AFS implementa o conceito de verificação dos dados por meio de RPC do servidor para o cliente, denominado de *callback*. Utilizando esse mecanismo, o servidor emite um *token* de garantia, informando o cliente sobre qualquer alteração realizada por outro cliente naquele arquivo copiado.

Conhecido em algumas literaturas como promessa de *callback*, eles são distribuídos junto com os arquivos transferidos. Neles há dois estados, válido ou cancelado, que determinam se há necessidade de obter novas cópias dos servidores.

A efetivação do arquivo só é computada no servidor quando o arquivo é fechado, desde que se tenha feito alterações no seu conteúdo. Por conseguinte, o servidor emite uma mensagem para todos os clientes que tenham a promessa de *callback* deste arquivo, alterando o seu estado para cancelado. Se no processo de abertura de um arquivo o seu *token* for verificado e seu estado estiver cancelado, o cliente busca uma nova atualização (PATE, 2003; COULOURIS; DOLLIMORE; KINDBERG, 2005).

3.4 *Google File System*

O GFS foi desenvolvido com o propósito de suprir algumas limitações da arquitetura tradicional cliente-servidor abordada até agora, buscando atender uma arquitetura composta de *clusters* de servidores para uso de aplicações paralelas.

A premissa dessa arquitetura está no processamento de várias partes que compõem um arquivo através de canais paralelos direto com os servidores. Essa técnica consiste em dividir um único arquivo em porções (*chunks*) menores e distribuí-los diretamente aos servidores, assim o cliente pode obter as porções distribuídas simultaneamente de vários canais independentes, que juntas formam o arquivo requisitado pela aplicação cliente (TANENBAUM; STEEN, 2007; HUANG; GRIMSHAW, 2011).

Para o desenvolvimento do sistema de arquivos da Google, foram propostas algumas características e alguns desafios, os quais são definidos a seguir (GHEMAWAT; GOBIOFF; LEUNG, 2003):

- Utilização de *hardware* de baixo custo com suporte a rápida recuperação de falhas.
- Manipulação eficiente de arquivos grandes, na ordem de gigabytes, além do suporte a arquivos pequenos.
- Suporte para atualizações por anexação de dados aos arquivos em vez sobrescrevê-los.
- Semântica eficiente para possibilitar vários usuários usando simultaneamente o mesmo arquivo, com uma sobrecarga mínima de sincronização.
- Preferência pela alta largura de banda em detrimento da baixa latência da rede, pois o objetivo é processamento em massa em vez de exigências rigorosas de tempo de resposta para um usuário ler ou escrever.

3.4.1 Arquitetura

A sua organização é representada por um servidor mestre (*master*) ligado a vários servidores de porção (*chunkservers*) que são acessados por diversos clientes. Cada componente consiste basicamente de computadores executando um sistema operacional convencional e processando em nível de usuário.

A ideia desse SAD é centralizar algumas funções em um único servidor, eliminando a complexidade do projeto e facilitando as decisões sobre o conjunto. Ciente de que a centralização pode gerar sobrecarga do sistema, conforme discutido no capítulo 2, os autores minimizaram o envolvimento do servidor mestre com atividades que causariam muito esforço, como ler e escrever os arquivos diretamente nos discos. Em vez disso, os clientes realizam as operações diretamente nos servidores de porção (GHEMAWAT; GOBIOFF; LEUNG, 2003).

3.4.2 Servidor Mestre

O mestre identifica o estado de cada servidor de porção juntamente com as porções dos arquivos armazenadas, formando um conjunto de metadados que ajudam a informar aonde a transferência dos arquivos deve ser feita.

A leitura e a gravação dos arquivos são realizadas diretamente entre os clientes e servidores de porção, eliminando uma parte do gargalo provocado pela centralização da comunicação e fornecendo uma solução simples para controle da alocação das porções dos arquivos com os servidores disponíveis. Esse esquema de organização, como mostrado na Figura 3.5, otimiza a comunicação sem que o mestre se torne um problema (GHEMAWAT; GOBIOFF; LEUNG, 2003; TANENBAUM; STEEN, 2007).

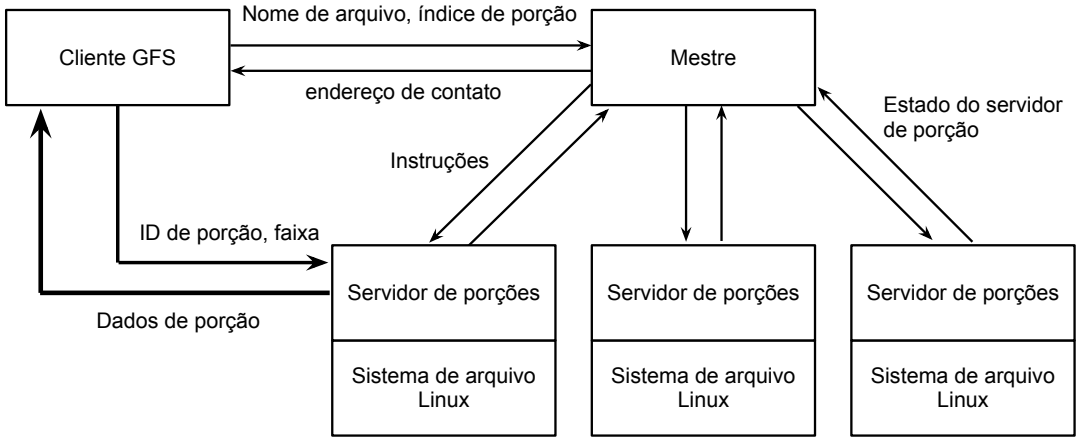


Figura 3.5: Arquitetura GFS (TANENBAUM; STEEN, 2007)

O mestre também é responsável por outras funções destinadas à administração e à organização das porções no conjunto de servidores:

- Comunicações periódicas com os servidores para saber quais porções eles armazenam e seus estados;
- Efetuar coleta de lixo para gerenciar a exclusão dos dados, pois, após a solicitação de remoção das porções, a mesma não é apagada fisicamente naquele instante. Ao invés disso, a porção é renomeada e marcada como oculta por até três dias (intervalo que pode ser alterado), sendo que durante esse tempo pode-se solicitar a sua leitura e até a sua restauração;
- Gerenciar e administrar o espaço de nomes, por meio de uma tabela de mapeamento dos caminhos para os metadados, enquanto as travas sobre as operações são impostas pelo mestre antes da execução de cada ação;

- Equilibrar o uso dos recursos computacionais disponíveis, adequando a melhor utilização do espaço em disco dos servidores de porção. Também procura espelhar as réplicas em diversos *racks* de servidores;
- Balancear a quantidade de réplicas que se encontram abaixo do número padrão ou especificado pelo usuário;
- Periodicamente o mestre examina a estrutura dos dados e organiza as réplicas para ocupar o melhor espaço em disco disponível;
- Algumas réplicas podem tornar-se obsoletas decorrente de falhas, e são removidas através do *garbage collection*. No entanto, há diferença entre réplicas desatualizadas e réplicas obsoletas que, neste caso, o mestre controla pelo número da versão de cada porção.

3.4.3 Arquivos

Quanto às porções, são de tamanho fixo de 64 Megabytes (MB) e identificadas por um manipulador (*chunck handle*) que é imutável e assinado pelo mestre no momento da criação, determinando se aquela parte do arquivo é do tipo somente leitura ou escrita.

O tamanho elevado dos blocos de arquivos proporcionam uma redução significativa no número de interações, principalmente na leitura ou escrita da mesma porção, eliminado várias solicitações para um acesso sequencial dos dados. Também elimina código extra na rede por manter um conexão TCP persistente com os servidores, considerando ser mais provável que um cliente realize muitas operações sobre uma mesma porção de dados (GHEMAWAT; GOBIOFF; LEUNG, 2003).

O lado negativo de se usar tamanhos elevados para cada porção ocorre quando o arquivo é menor em relação aos blocos de 64 MB, resultando em poucas porções e, consequentemente, sobrecarga nos servidores no caso de ocorrerem acessos simultâneos a esses poucos blocos. Neste caso, o GFS aplica um tratamento específico, utilizando um fator maior de replicação para esses arquivos nos servidores de porção (GHEMAWAT; GOBIOFF; LEUNG, 2003).

Por padrão, os arquivos são divididos em três porções que são replicadas entre os servidores do *cluster*. O seu critério de divisão é flexível e pode ser aplicado em diferentes níveis (GHEMAWAT; GOBIOFF; LEUNG, 2003).

3.4.4 Cliente

O cliente GFS comunica-se com o mestre através de metadados para saber quais porções são referentes a um determinado arquivo e onde se localizam, enquanto as operações sobre os arquivos são realizadas diretamente pelo cliente GFS nos servidores de porção.

Não há *caches* das porções nos clientes, pois com o seu tamanho elevado a sua prática traria pouco benefício. Apenas os metadados são colocados na *cache* por um tempo limitado para que o cliente possa fazer operações subsequentes diretamente com os servidores de porção, sem a necessidade de novas interações com o mestre.

Do lado do servidor de porção, o próprio sistema operacional se encarrega de colocar os blocos mais acessados na memória principal (GHEMAWAT; GOBIOFF; LEUNG, 2003).

3.4.5 Tolerância a Falhas

O GFS busca métodos para contornar as falhas provocadas por danos nos equipamentos ou dados corrompidos. O seu processo para tolerância especializa-se na recuperação e replicação, caracterizados por:

- **Rápido retorno**: permite que os servidores de porção e o mestre restabeleçam seus processos em segundos, não importando como eles tenham parado;
- **Replicação de porção**: é responsável em replicar as porções em diferentes níveis para diferentes partes do espaço de nomes;
- **Replicação do mestre**: faz réplicas do estado do servidor mestre em vários computadores, permitindo que um novo servidor possa ser estabelecido em caso de falhas com o anterior.

A integridade dos dados é realizada individualmente por cada servidor de porção, mantendo os *checksums* dos dados na tentativa de identificar arquivos corrompidos e evitar que eles se propaguem.

Para auxílio no processo de identificação das falhas, o GFS gera um *log* detalhado dos eventos, permitindo que administradores isolem e identifiquem possíveis problemas (GHEMAWAT; GOBIOFF; LEUNG, 2003).

3.5 Tahoe-LAFS

Tahoe-LAFS é um SAD que oferece confidencialidade, integridade e disponibilidade seguindo o princípio de menor autoridade (MILLER, 2006), em que o usuário ou processo que precisa realizar uma tarefa deve ser capaz de realizá-la sem ter ou manipular mais autoridade do que é necessário (WILCOX-O'HEARN; WARNER, 2008).

De código aberto e desenvolvido em Python, o Tahoe-LAFS iniciou-se no ano de 2006 pela *allmydata.com*[1], com foco em fornecer um SAD sobre o espaço do usuário compatível com *hardware* de baixo custo sem deixar de oferecer armazenamento seguro, escalável e barato. A sua aplicabilidade varia desde a utilização de um simples serviço de *backup* até um amplo sistema de armazenamento na nuvem.

O Tahoe-LAFS pode ser representado por um conjunto composto por clientes, servidores de armazenamento e um componente central, denominado de *introducer*.

O servidor *introducer* procura apenas conhecer a localização dos nós, delegando toda a transferência de dados para interações diretas entre cliente e servidores de armazenamento, como pode ser observado na Figura 3.6.

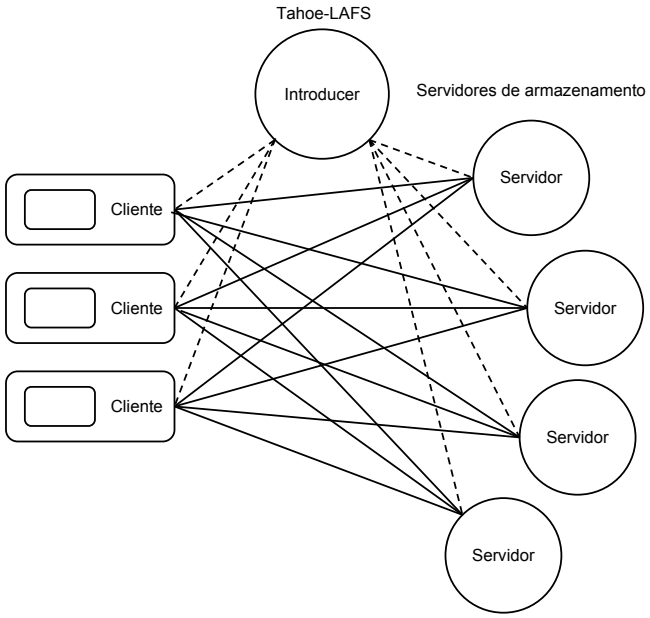


Figura 3.6: Interação processos Tahoe-LAFS (WILCOX-O'HEARN; WARNER, 2008)

O cliente do Tahoe-LAFS é composto por um *Gateway*, o qual é responsável pela criptografia e integridade dos arquivos entre o usuário e servidores de armazenamento.

A interação do usuário com o *Gateway* ocorre sobre o protocolo HTTP (*HyperText Transfer Protocol*) ou FTP (*File Transfer Protocol*), a qual pode ser realizada através de

[1] Atualmente disponibilizado pela página do projeto http://tahoe-lafs.org/

alguns meios como navegador *web*, linha de comando, clientes FTP, *Windows virtual drive*, FUSE (*Filesystem in Userspace*) dentre outros.

Após a codificação dos arquivos pelo *Gateway*, os dados são distribuídos entre os servidores de armazenamento sobre o protocolo TCP em conjunto com o SSL (*Secure Sockets Layer*), conforme mostrado na Figura 3.7.

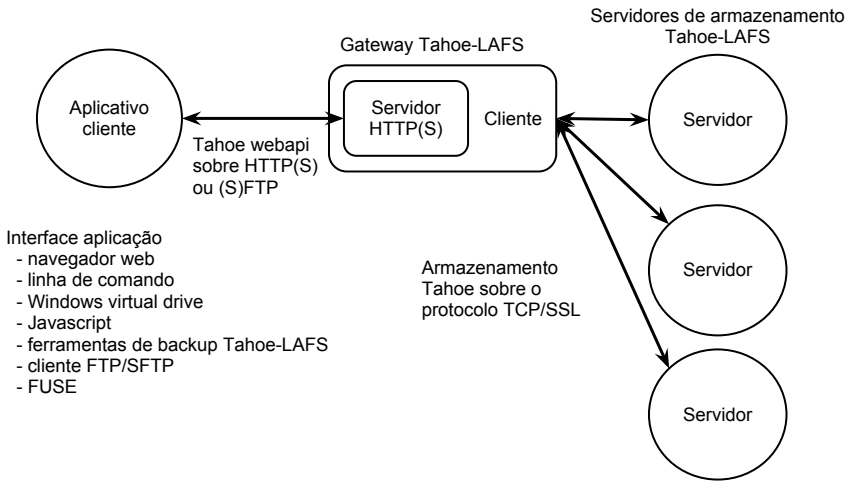


Figura 3.7: Visão Geral Tahoe-LAFS (WILCOX-O'HEARN; WARNER, 2008)

O processo de distribuição segue um ajuste de $1 <= K <= N$, em que N representa a quantidade de servidores que serão utilizados para armazenar os arquivos e K o mínimo necessário de servidores para disponibilizar o arquivo. Por padrão, o Tahoe-LAFS usa uma configuração de *K=3* e *N=10*, totalizando 10 servidores para armazenar os dados e no mínimo 3 para recuperar o arquivo completo. Se o número de computadores ativos for menor que K, os arquivos que foram submetidos anteriormente em condições normais (nós ativos igual ou maior que K) não são recuperados, isso devido à quantidade inferior de nós de armazenamento necessários para recompor o arquivo original. Entretanto, a escrita de novos arquivos é permitida mesmo com nós ativos abaixo de K, porém esses arquivos não serão replicados para os demais nós N quando estivem ativos, limitando-se o acesso a esses arquivos somente aos nós que receberam.

A confiabilidade, a integridade e a disponibilidade dos arquivos são obtidas respectivamente pelo uso de criptografia, *hashing* e *erasure coding* dos dados. O resultado final é um conjunto de partes criptografadas do arquivo original e distribuídas por N servidores de armazenamento. Essa sequência de passos impede que os arquivos sejam acessados indevidamente ou corrompidos caso alguns dos servidores falhem.

3.5.1 Gerenciamento de Segurança

A estrutura do Tahoe-LAFS é representada por uma hierarquia de arquivos e diretórios que são gerenciadas por controles individuais. A diferença em relação às estruturas tradicionais de acesso utilizadas em sistemas de arquivos está na substituição das caixas de diálogo dos controles de cada arquivo/diretório por sequências de caracteres, que são suficientes para identificar e acessar os dados. No Tahoe-LAFS o controle de acesso é administrado por dois manipuladores denominados de *file-capability* (*file-caps*) e *dir-capability* (*dircaps*) que definem o gerenciamento do arquivo e o gerenciamento dos diretórios, respectivamente.

Os *filecaps* e *dircaps* carregam informações como identificação, localização, chave de acesso e validação do arquivo ou diretório. Todo esse conjunto forma uma sequência de caracteres com 96 bits, de fácil manipulação para o cliente utilizar diretamente pelo navegador web ou outro método de acesso.

Através do *filecap* e *dircap* é possível compartilhar um arquivo ou diretório específico em vez de caminhar pela hierarquia compartilhando vários arquivos ou vários diretórios. Essa abordagem simplifica a maneira como ocorre a interação entre cliente e o espaço de nomes do Tahoe-LAFS, sem a necessidade de super-usuários ou administradores com poderes de acesso mais elevado.

As permissões são definidas diretamente em cada manipulador, variando entre leitura-escrita e somente-leitura. Também é possível delegar essas permissões para outros manipuladores, de modo que permissões de leitura-escrita possam gerar outras permissões de leitura-escrita ou somente-leitura, enquanto os manipuladores com somente-leitura possam oferecer apenas permissões de somente-leitura. Além dessas duas categorias de acesso, há também a permissão de verificação que permite analisar a integridade dos dados. Ela é incorporada nos dois modos de acesso e pode ser obtida separadamente com o intuito de verificar as partes dos arquivos sem necessariamente acessá-los.

3.5.2 Confidencialidade, Integridade e Disponibilidade

A confidencialidade dos dados armazenados no Tahoe-LAFS é proporcionada pela criptografia simétrica *Advanced Encryption Standard* (AES) de 128 bits em conjunto com assimétrica RSA (Rivest-Shamir-Adleman), descrito por (RIVEST; SHAMIR; ADLEMAN, 1978).

O processo consiste em assinar as permissões do arquivo com a chave RSA seguido de sequências *hash* para formar a chave de criptografia AES-128. Por fim, o cliente

recebe os manipuladores do tipo escrita (*write-cap*), somente-leitura (*read-only-cap*) e verificação (*verify-cap*), totalizando o conjunto de permissões sobre o arquivo, os *filecaps*.

A *verify-cap* permite apenas verificar as propriedades do arquivo sem que o cliente ou a aplicação necessite abri-lo.

O *read-only-cap* dá-lhe acesso à chave de descriptografia, oferecendo apenas a permissão de somente-leitura, sem possuir recursos suficientes para alteração do arquivo no SAD. Com essa permissão é possível derivar um nível abaixo, o qual corresponde ao *verify-cap*, mas não um nível acima.

O *write-cap* lhe dá acesso à chave privada, utilizada para controle total sobre o arquivo e para assinar todas as suas ações. Com ela é possível gerar *read-only-cap* e *verify-cap* do arquivo.

Diferente dos modelos tradicionais de SADs, os servidores do Tahoe-LAFS não conhecem o conteúdo dos arquivos armazenados, eles apenas gerenciam os dados através de um índice de armazenamento. Independente de falhas ou falta de segurança dos servidores, a confiabilidade dos dados será preservada, pois somente o usuário ou a aplicação cliente possuem as chaves de criptografia.

Depois de garantir a confiabilidade, o passo seguinte consiste em certificar a integridade, processo que é alcançado com o uso de *hashing* sobre o arquivo cifrado antes do envio aos servidores de armazenamento. A saída gerada pelo *hashing* é agregada às chaves de criptografia.

Por último, a disponibilidade do arquivo é obtida com o uso de *erasure coding* em conjunto com a divisão e distribuição de partes do arquivo para um conjunto de servidores. O *erasure coding*, também conhecido como *Forward Error Correction* (FEC) é um método para correção de erros na transmissão dos dados (FUJIMURA; OH; GERLA, 2008; PLANK et al., 2009).

3.5.3 Diretórios

No Tahoe-LAFS os diretórios não seguem o modelo tradicional em árvore, em vez disso são representados por tabelas de mapeamento contendo os nomes dos filhos com as suas *filecaps*. Essa tabela é serializada formando os *dircaps*, responsáveis pelas instruções de como interpretar o conteúdo de um determinado diretório. Igualmente aos *filecaps*, os *dircaps* são pequenas cadeias formadas em ASCII que podem ser distribuídas facilmente entre os clientes.

Sem o conceito de diretório raiz e subdiretórios, ele simplesmente toma como início o local do *dircap* e percorre os outros diretórios através de ligações associadas com aquele *dircap*. Esta estrutura pode ser exemplificada pela Figura 3.8.

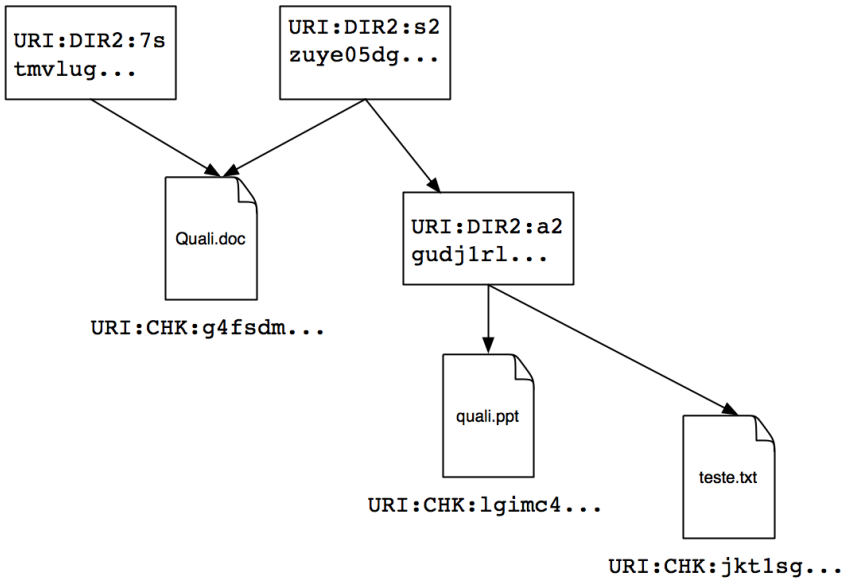


Figura 3.8: Diretório Tahoe-LAFS.

O uso de *dircaps* permite tratar individualmente cada diretório de modo que um arquivo possa ter vários pais. Além disso, eles também suportam variações de *read-write-cap* e o *read-only-cap*.

3.6 Considerações Finais

Cada SAD estudado neste capítulo traz alguma particularidade que pode ser agregada ao modelo proposto. Na Tabela 3.1 é mostrada a comparação entre as características desses SADs.

No **NFS** a principal característica está na transparência, a qual ocorre através de um sistema de arquivos virtual. O **AFS** representa um projeto com foco na escalabilidade oferecendo suporte a vários usuários e uso de políticas agressivas de *cache* com método eficiente de consistência. O **GFS** segue com o propósito de busca paralela de partes do arquivo em vários servidores. O **Tahoe-LAFS** é um SAD disponibilizado em código aberto que atua sobre o espaço do usuário, trazendo compartilhamento de arquivos e diretórios através da URL e utilizando o conceito de controle de permissões descentralizadas.

Destacadas as principais características dos SADs, no capítulo 4 será descrito o modelo FlexA detalhando as suas especificações e destacando os atributos herdados

dos SADs aqui estudados.

Tabela 3.1: Comparação entre NFS, AFS, GFS e Tahoe-LAFS

Característica	NFS	AFS	GFS	Tahoe-LAFS
Arquitetura	Centralizada cliente-servidor.	Centralizada cliente-servidor.	Híbrida baseada em *cluster* com dados e metadados armazenados separadamente.	Descentralizada baseada em múltiplos servidores.
Escalabilidade	Limitada ao único fluxo do servidor, permitindo apenas um número limitado de clientes.	Alta escalabilidade através do uso de políticas agressivas de *cache* nos clientes e da possibilidade de agregar conjuntos de servidores.	Alta escalabilidade utilizando *cluster* de servidores.	Alta escalabilidade com distribuição descentralizada dos dados.
Cache	Blocos de arquivos e diretórios lidos mais recentemente são mantidos no cliente e nos servidores através do próprio sistema local.	No cliente através do armazenamento de arquivos inteiros.	Armazenamento de metadados no cliente.	Ausente.
Confiabilidade	De responsabilidade do sistema local.	De responsabilidade do sistema local.	De responsabilidade do sistema local.	Método independente do sistema local com criptografia direta sobre o arquivo utilizando chave assimétrica RSA-2048 bits e simétrica AES-128 bits.
Interface	Transparente com a utilização de uma camada virtual (VFI).	Baseado nas chamadas UNIX.	Independente, mas com chamadas similares ao ambiente UNIX.	Independente, fornecendo vários meios de acesso.
Tolerância a falhas	Sem replicação de arquivos.	Uso de réplicas em diferentes servidores.	Replicação de porções dos arquivos entre servidores de armazenamento.	Distribuição das partes do arquivo entre os nós de armazenamento.
Segurança	Através de RPCs seguras com diversos métodos de segurança.	RPCs seguras.	Autenticação apoiada pelo sistema local.	Manipuladores individuais para identificação e transferência de dados utilizando canais seguros.

4 Descrição e validação do modelo

Neste capítulo é apresentado o desenvolvimento do sistema de arquivos distribuído FlexA, destacando a sua confiabilidade bem como a integridade e a disponibilidade dos dados através de características herdadas de outros SADs.

4.1 Características

Algumas características do NFS, AFS, GFS e Tahoe-LAFS foram selecionadas como base para o desenvolvimento do FlexA. Tais características buscam abstrair o melhor conceito de cada SAD agregando-as ao modelo FlexA. Elas seguem com o propósito de estender o uso do modelo para as exigências comumente encontradas em um cenário, como por exemplo o uso de computadores com *hardware* simples, falta de equipamento para proteção contra falha de energia, falta de um supervisor ou administrador presente em tempo integral e carência de uma rede de comunicação rápida. Considerando isso, buscou-se agregar as características apresentadas a seguir ao modelo FlexA.

- Habilidade da estação cliente trabalhar como servidora, permitindo ao SAD adequar a sua utilização para aquele nó em específico. Isso significa que os clientes poderão atuar como servidores secundários, auxiliando no fornecimento de porções e, por conseguinte, aliviando os servidores primários em caso de sobrecarga. Essa configuração pode ser definida manualmente pelo usuário ou através de um processo automatizado que analisa os dados estatísticos dos servidores e clientes (tempo ativo na rede, espaço disponível para armazenamento, largura da banda, dentre outros fatores), oferecendo flexibilidade no uso de mais servidores (GHEMAWAT; GOBIOFF; LEUNG, 2003).

- Facilidade no gerenciamento dos recursos criptográficos através da utilização do conceito de controle de permissões, descrito por Wilcox-O'Hearn e Warner (2008).

- Suporte a *hardware* de baixo custo juntamente com a possibilidade de atuar no espaço do usuário. Essa característica permite a adaptação do modelo proposto aos computadores e sistemas operacionais, além de proporcionar fácil implantação, administração e manipulação. Característica descrita por Ghemawat, Gobioff e Leung (2003) e por Wilcox-O'Hearn e Warner (2008).

- A tolerância a falhas será apoiada na distribuição de porções dos arquivos entre os nós de armazenamento, descrita por Ghemawat, Gobioff e Leung (2003) e por Wilcox-O'Hearn e Warner (2008).

- Uso de *cache* agressivo no cliente, eliminando os acessos subsequentes sobre arquivos já utilizados, o que pode levar ganhos sobre a latência da rede e dos recursos computacionais presentes. Característica implantada no AFS e que pode ser estudada no livro de Coulouris, Dollimore e Kindberg (2005).

- Suporte à camada de abstração para simplificação das ações dos usuários com o módulo servidor, possibilitando compatibilidade com outros recursos e tecnologias. Essa técnica está presente no NFS e é descrita por Coulouris, Dollimore e Kindberg (2005).

4.2 Visão geral do FlexA

Elencadas as principais características para o desenvolvimento do modelo de SAD, nesta seção descreve-se como esses principais pontos e outros ajustes foram agregados para formar um modelo de SAD funcional.

4.2.1 Arquitetura

Diferente do modelo tradicional cliente-servidor, o FlexA, como mostrado na Figura 4.1, eliminou o conceito de servidor principal, o qual seria responsável por mediar as comunicações com os arquivos. Em seu lugar, um módulo presente em cada estação se encarrega de identificar e administrar as comunicações.

Nesse modelo, os computadores pertencentes ao SAD são administrados em grupos específicos, o qual podem ser de três tipos: grupo de escrita, réplicas e clientes.

O primeiro, denominado de grupo de escrita ou servidores primários, compreende as estações com o processo servidor ativo, sendo responsáveis em administrar e armazenar os arquivos junto com os seus metadados. Composto de três computadores (quantidade necessária para auxiliar na distribuição paralela e garantir disponibilidade

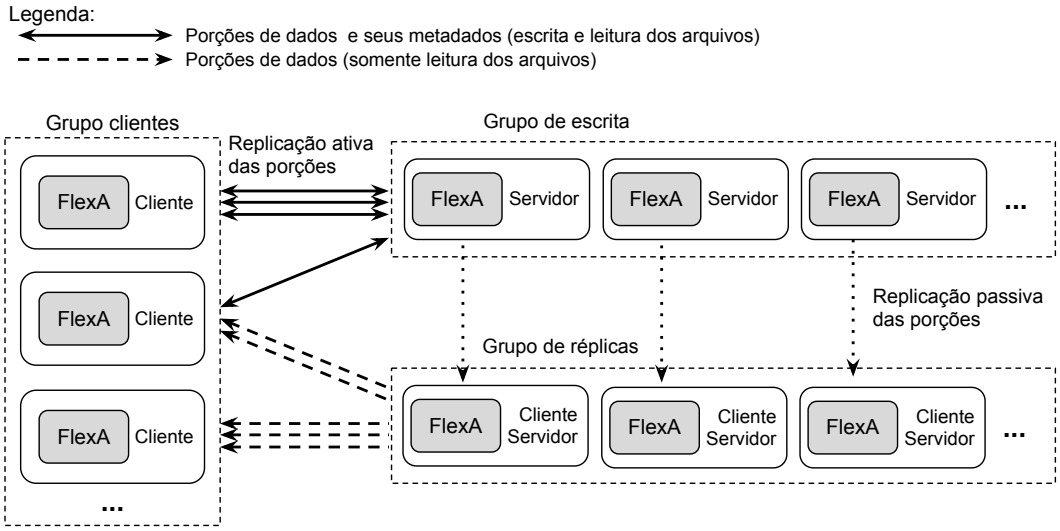


Figura 4.1: Arquitetura FlexA

em caso de falha de algum deles), esse grupo é a fonte principal de armazenamento, projetado para atuar paralelamente no fornecimento de pequenas partes do arquivo, chamadas de porções.

Essas porções são formadas através da divisão do arquivo criptografado em três partes iguais, permitindo que os servidores desse grupo possam completar as requisições dos clientes em menor tempo e simultaneamente nos três servidores. Com isso, a largura de banda e os recursos computacionais não ficam limitados apenas a uma estação, totalizando um conjunto que pode disponibilizar um canal de acesso três vezes maior que um modelo tradicional.

Além disso, essa configuração privilegia situações em caso de falha de um dos computadores, pois tais porções são distribuídas a uma proporção de dois terços do arquivo para cada servidor, conforme é mostrado na Figura 4.2. Essa organização permite que o cliente possa interagir normalmente com o grupo mesmo que um dos servidores esteja *offline*.

O segundo grupo, denominado de réplicas ou servidores secundários, é formado por computadores que podem ser clientes e servidores ao mesmo tempo, formando um conjunto de *backup* caso alguma estação servidora falhe. Esse conjunto oferece suporte na distribuição dos arquivos dentro do FlexA, auxiliando os servidores primários em caso de sobrecarga, a qual é detectada pelo tempo médio de transferência das porções.

O terceiro grupo é formando pelas estações clientes, os quais são responsáveis pela criptografia e distribuição das porções do arquivo aos servidores primários.

Com o modelo apresentado, a arquitetura FlexA aproxima-se do padrão *peer-to-*

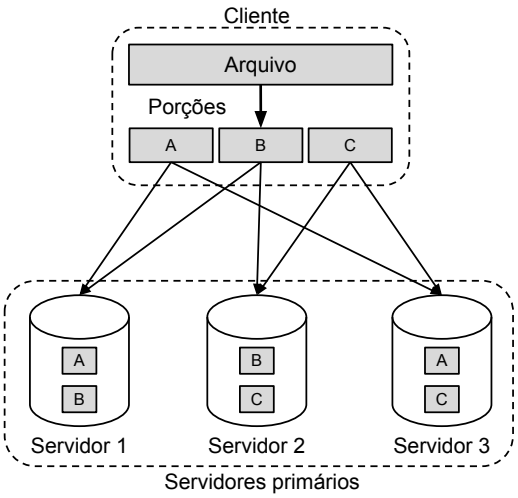


Figura 4.2: Distribuição das porções

peer, possibilitando que os três grupos comuniquem-se entre si para descoberta de novas estações e administração das suas funções, sem a necessidade de um centralizador. Um fator que privilegiou essa característica foi a ausência de uma unidade certificadora, pois, com o seu modelo de controle de acesso descentralizado, cada cliente é responsável pelas permissões de seus arquivos, sem a intervenção de administradores ou superusuários.

A eliminação do servidor intermediário e a adição do acesso simultâneo aos diferentes conjuntos minimizou os efeitos da latência da rede e preveniu possíveis gargalos na transferência.

Outra característica que trouxe ganhos sobre o conjunto de armazenamento foi a migração dos processos mais custosos (criptografia e divisão do arquivo) para o cliente, aliviando o consumo dos recursos de *hardware* dos nós de armazenamento.

O processo de escrita/leitura dos arquivos no FlexA consiste da utilização do método de *upload*/*download* (TANENBAUM; STEEN, 2007), no qual o cliente faz cópia do arquivo completo para a estação local e, em seguida, realiza as operações sobre ele. Após isso, o arquivo atualizado é submetido aos servidores de escrita.

O FlexA pode ser resumido em três principais módulos: Coletor, Sincronizador e Comunicador, conforme Figura 4.3. Esses três módulos representam o núcleo do FlexA e estão presentes nas estações clientes e servidoras, de modo que toda a comunicação entre eles ocorra diretamente utilizando *sockets* sobre protocolo TCP/IP com conexões persistentes para transferência de conjuntos de porções.

O ***módulo Coletor*** é o principal componente para que haja interação entre clientes e servidores. A sua função é analisar e identificar toda comunicação de entrada

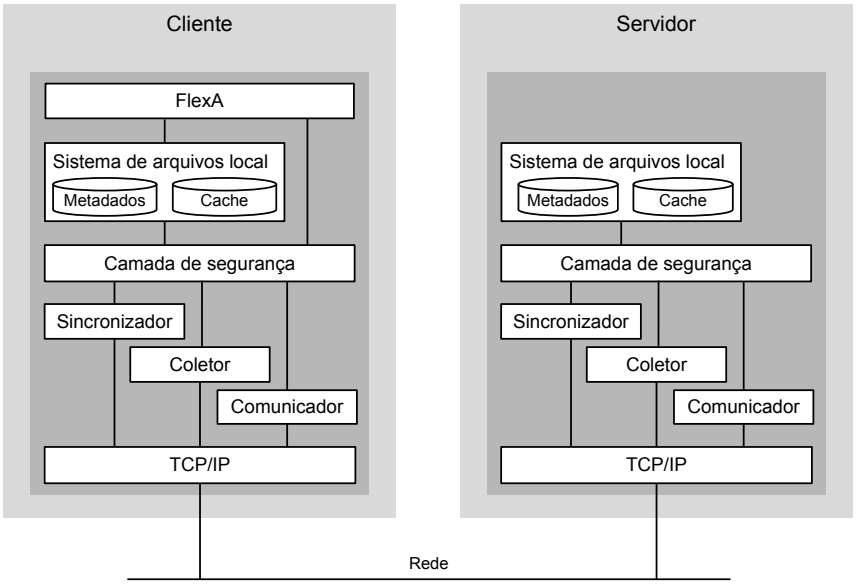


Figura 4.3: Disposição dos módulos do FlexA

do processo FlexA, sendo o responsável pela distinção entre servidores primários, secundários e clientes. A ele são atribuídas todas as requisições que após analisadas e certificadas são encaminhas para a sua execução.

O ***módulo Sincronizador*** é responsável pelo endereçamento de todas as mensagens. Ele consiste em informar aos demais servidores e clientes sobre qualquer evento que resulte em modificação nos dados armazenados no grupo. A sua utilização é por demanda de solicitação, não necessitando que o processo permaneça ativo como é preciso com o *módulo Coletor*.

O ***módulo Comunicador*** assume a função de reconhecimento e busca dos componentes da rede, identificando quais computadores estão com o *módulo Coletor* ativo. Por não possuir um servidor central que administre toda a comunicação, o *módulo Comunicador* torna-se responsável por toda essa etapa. O seu funcionamento é baseado na procura de estações pertencentes a algum dos grupos do FlexA, registrando cada computador através do seu UUID (*Universally Unique IDentifier*), seu IP e o seu tipo de grupo dentro de uma tabela de roteamento, presente em cada estação. No final desse processo, que é executado periodicamente (por padrão a cada trinta segundos, mas pode ser ajustado pelo usuário) por cada computador que tenha o serviço FlexA instalado, a estação local saberá quais computadores ativos estão com o SAD habilitado e a que grupo pertencem.

No cliente, há também o *módulo flexa* que é responsável por intermediar as ações do usuário e do restante dos módulos. Através dele, utilizando a linha de comando, é possível realizar as operações sobre o arquivo. Diferente da interface tradicional de sis-

tema, esse modelo disponibiliza as operações de *put*, *get*, *list*, *delete* e *new permission*, que, respectivamente, desempenham a função de distribuir o arquivo, obter as porções do conjunto, listar os arquivos disponíveis no SAD, apagar os arquivos e refazer as permissões do arquivo.

Todas as informações sobre o arquivo, mais a chave de validação e verificação de escrita, são armazenadas em banco de dados nas estações servidoras. Parte dessa informação, como por exemplo nome, diretório, data de criação, *hash* e os IPs onde estão essas porções, é distribuída pela rede nas estações clientes. Com isso, cada cliente tem o conhecimento desses arquivos, dependendo apenas do proprietário enviar os manipuladores do arquivo para que se tenha o devido acesso.

A sincronização das porções ocorre automaticamente, mediante as interações do cliente no grupo de escrita. Essas interações são propagadas no grupo de réplicas e os clientes ativos são informados sobre novas alterações.

4.2.2 Flexibilidade e Adaptabilidade

A adaptabilidade permite que clientes possam fazer parte do grupo de servidores, auxiliando no fornecimento dos arquivos distribuídos.

Através dessa característica, é possível compartilhar recurso da estação cliente, como por exemplo espaço em disco, ou ainda tornar o cliente uma estação servidora por completo.

O conceito de flexibilidade vem da possibilidade de realizar alterações no FlexA para adequar ao cenário utilizado, ou seja, fornecer meios para modificar as suas funcionalidades pela substituição ou pelo ajuste dos componentes do modelo. Os componentes que podem ser modificados foram projetados para serem independentes do conjunto, dentre eles pode-se destacar a modificação ou a substituição do algoritmo de criptografia, a alteração dos níveis de replicação e a adaptação da interface para outras aplicações como, por exemplo, o uso através de um navegador web.

Também é delegada maior autoridade para o usuário, sendo de sua escolha qual processo será ativo naquele computador, cliente ou servidor. Porém, ao menos três computadores com o processo servidor devem ser iniciados de modo independente, garantindo um número necessário para formar o conjunto de servidores primários. Quanto à escolha determinada pelo usuário, ela varia entre estações clientes ou estações servidoras que formarão o grupo de réplicas futuramente.

Outro fator que trouxe flexibilidade para modelo é o seu desenvolvimento em lin-

guagem Python[1] e de código aberto.

4.2.3 Controle de acesso

O controle de acesso no FlexA é apoiado sobre o modelo de permissões do Tahoe-LAFS, o qual oferece maior flexibilidade e segurança com o seu tratamento individual aos arquivos e diretórios. Estruturadas na forma de um grafo, as permissões são manipuladas diretamente pelo usuário, sem a necessidade de um administrador ou superusuário. Através disso, os campos tradicionais de usuário e senha são substituídos por manipuladores que definem o tipo de acesso ao arquivo ou diretório (WILCOX-O'HEARN; WARNER, 2008).

Com a finalidade de integrar o modelo de segurança descentralizada do Tahoe-LAFS, algumas modificações foram realizadas no seu conceito para adequar ao FlexA. O modelo desenvolvido utiliza chaves simétricas AES no modo CBC (*Cipher Block Chaining*) (DWORKIN, 2001) em conjunto com sequências *hash*, fornecendo dois níveis de acesso, escrita-leitura e somente-leitura. A ausência das chaves assimétricas RSA, utilizadas no Tahoe-LAFS, trouxe flexibilidade para a implantação do modelo desenvolvido, pois menos passos são necessários para completar a segurança do arquivo, tornando o princípio de menor autoridade menos complexo.

Como resultado desse modelo de controle de acesso, o cliente obtém dois tipos de manipuladores (somente-leitura e escrita-leitura) para a administração dos arquivos no FlexA. Cada manipulador possui uma chave de criptografia e validação.

O funcionamento do mecanismo de controle de acesso pode ser descrito através de sequências *hash* sobre uma determinada chave, como mostrado na Figura 4.4.

A chave de criptografia utilizada no manipulador escrita-leitura *Write Key - WK*, é computada através de *WK = sha256Truncate(Key)*, sendo *Key* uma sequência de 16 bytes gerada aleatoriamente e *sha256Truncate* correspondendo ao *hash* de 256 bits (32 bytes) codificado para *base64* (JOSEFSSON, 2006) com saída truncada em 32 bytes.

A codificação em *Base64* facilita o armazenamento da chave em modo texto, tornando menos complexa a transferência entre os meios convencionais, como por exemplo o correio eletrônico.

Em seguida, o manipulador somente-leitura (*Read Key - RK*) é gerado através do mesmo método utilizado no manipulador escrita-leitura, porém sobre a chave *WK*, sendo *RK = sha256Truncate(WK)*, com saída em 32 bytes.

[1] Linguagem de programação de alto nível que oferece produtividade, legibilidade, suporte a vários paradigmas de programação, multiplataforma e de livre utilização.

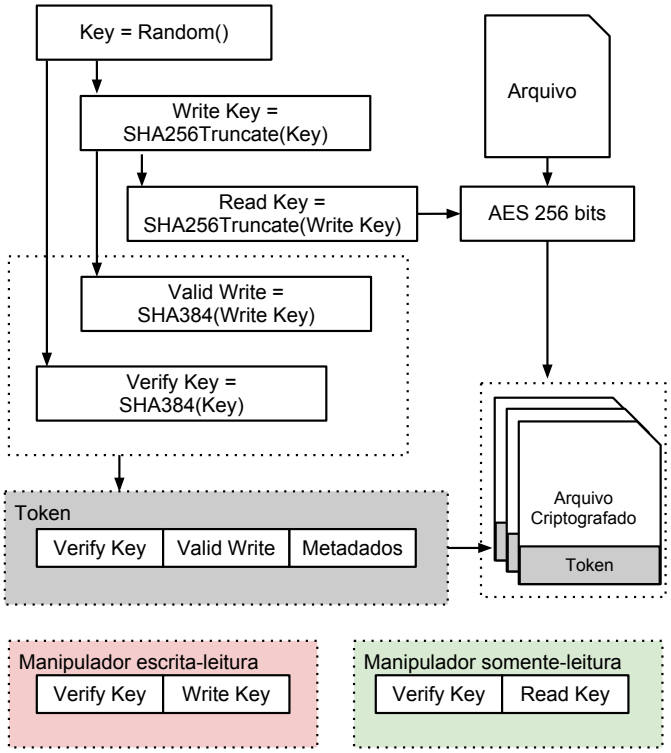


Figura 4.4: Processo criptográfico do FlexA (adaptado de Wilcox-O'Hearn e Warner (2008))

Após isso, o arquivo é codificado utilizando a chave *RK* sobre a criptografia simétrica AES de 256 bits (32 bytes), de modo que seja possível transformar uma *WK* em *RK*, porém não o contrário.

O próximo passo é gerar a chave de validação de escrita *Valid Write (VW)* e a chave de verificação *Verify Key (VK)*, ambas de 48 bytes. A *VK*, determinada por *VK = sha384(Key)* encontra-se nos manipuladores do cliente e no ID enviado aos servidores. Ela é utilizada para certificar a comunicação entre os nós e só é gerada uma única vez pois, após esse processo, a chave inicial *Key* é descartada. A *VW*, obtida por *VW = sha384(WK)*, também é gerada pelo cliente, mas só os servidores possuem a cópia. A sua função é determinar se a *WK* do cliente é legítima, tarefa essa que pode ser executada exclusivamente por quem possui o manipulador escrita-leitura. Se o resultado gerado pela *VW* do cliente for igual a *VW* da porção já armazenada, a alteração do arquivo é efetivada nos servidores. Essa etapa é mostrada na Figura 4.5.

Em situações de uso concorrente, o arquivo é substituído pelo último submetido, porém, a cada nova alteração, uma mensagem é enviada a todos clientes com o arquivo em *cache*, informando sobre a modificação. Por conseguinte, é solicitado ao cliente se ele deseja atualizar a sua versão, mantê-la ou criar uma nova versão daquele arquivo.

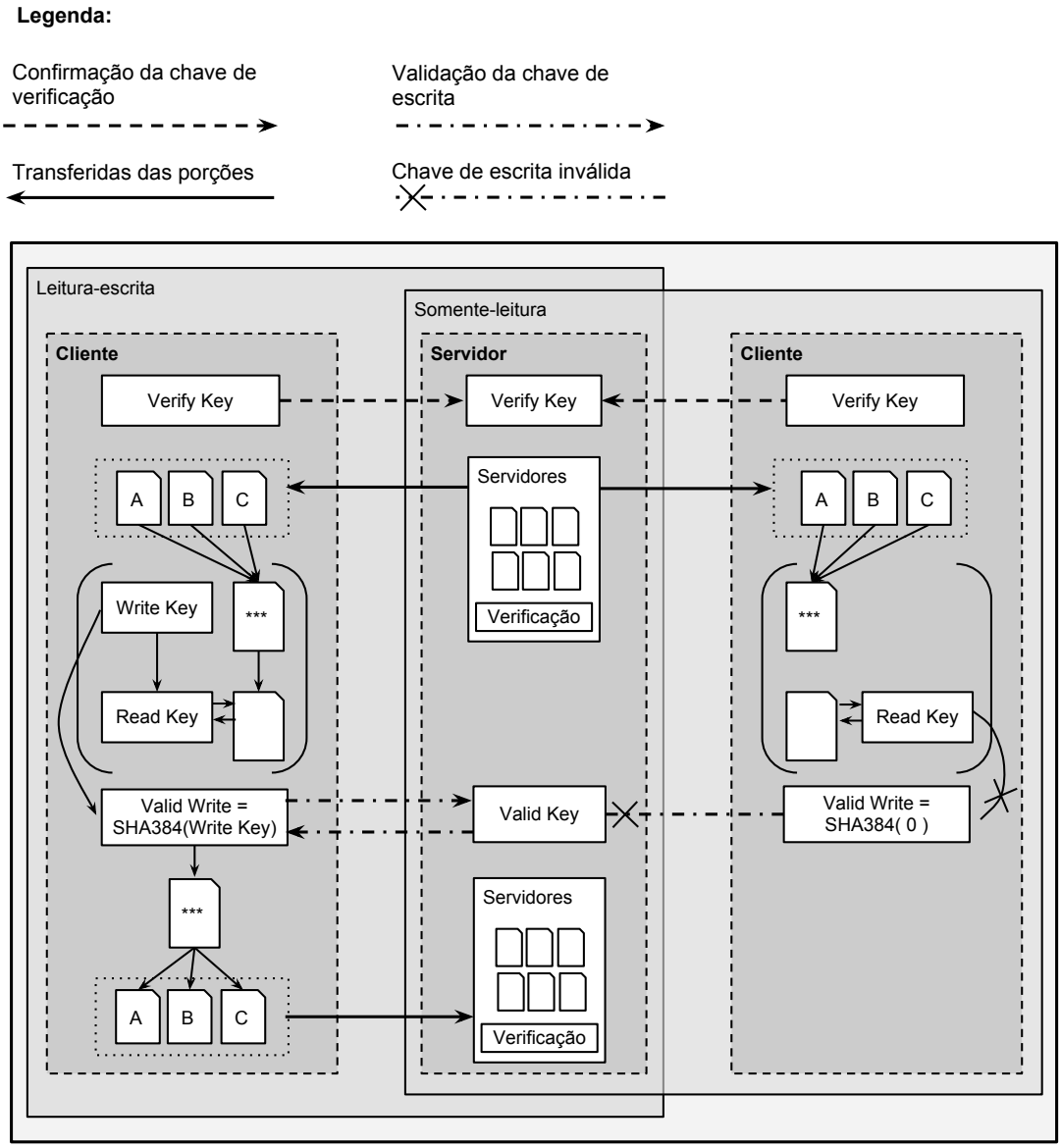


Figura 4.5: Controle de acesso do FlexA

4.2.4 *Hardware* de baixo custo

Os processos no FlexA ocorrem de maneira inversa ao encontrado em arquiteturas tradicionais de SAD, em que o servidor processa as requisições e distribui pela rede. No FlexA, foi utilizado um método de construção no qual o cliente é o responsável pela maioria da carga de processamento para tornar o arquivo disponível. Essa inversão na utilização dos recursos computacionais torna o servidor menos propenso a falhas por sobrecarga.

Para o cliente, o consumo do *hardware* é causado pela criptografia do arquivo e pela sua divisão em porções para a distribuição entre os servidores. Esses dois eventos

executados no cliente provocam pouco impacto no consumo geral da UCP, comparado a um cenário em que o servidor seria responsável pela criptografia e a divisão/distribuição das requisições de todos os clientes. Nesse último caso, os servidores necessitam de um poder computacional muito elevado, tornando-se indispensável o uso de *hardware* específico, o que pode encarecer a implantação de um SAD.

4.2.5 Tolerância a falhas

A tolerância a falhas no FlexA ocorre através da divisão do arquivo em blocos menores, os quais são transferidos para um conjunto de servidores. Esse processo permite que o cliente trabalhe com porções distribuídas, evitando que problemas isolados em servidores prejudiquem a integridade dos dados.

As porções dos arquivos são enviadas diretamente aos servidores primários, os quais são responsáveis por armazenar os dados e distribuí-los aos servidores secundários, se esses estiverem ativos. Os servidores primários fornecem todo o suporte a leitura e escrita dos arquivos, *download* e *upload* das porções, mesmo em caso de falha de uma das três estações. Em caso de falhas em mais de um servidor, se não houver outros servidores secundários habilitados para assumir uma posição no grupo de escrita, o cliente será informado sobre a indisponibilidade do serviço.

Quanto aos servidores secundários, eles representam uma etapa significativa para auxiliar nas exigências sobre os servidores primários. Com esse grupo, como pode ser mostrado na Figura 4.6, o cliente pode utilizar porções distribuídas em outras estações clientes, as quais formam computadores com aspecto de servidores, mas sem a opção de escrita, somente leitura dos dados. Essa técnica viabiliza o acesso a determinadas porções quando os servidores primários estiverem apresentando maior latência nas requisições. A limitação para a escrita de arquivos no servidores secundários previne que problemas futuros de consistência ocorram entre diferentes versões de arquivos e também simplifica a construção do projeto, eliminando código extra para o tratamento de diferentes versões do mesmo arquivo.

O uso do grupo de réplicas é opcional, possibilitando controlar a disponibilidade das porções de acordo com a demanda de solicitações. Para o grupo de escrita é necessário que dois de seus três servidores estejam ativos.

No caso de falhas ou queda do serviço, os processos do FlexA podem ser iniciados rapidamente, pois os computadores não precisam realizar, imediatamente, uma nova busca pela rede na procura do módulo comunicador. Basta apenas os clientes continuarem enviando as suas requisições até que os servidores retornem.

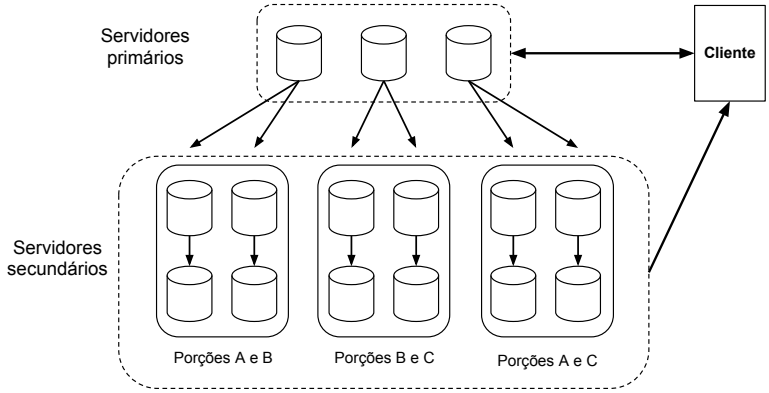


Figura 4.6: Replicação de porções

Em situação de comprometimento de algum dos servidores primários, há a possibilidade de remanejar as estações do grupo de réplicas através do processo de eleição. Tal processo é apoiado sobre o algoritmo de *bully*, o qual determina um novo candidato aos servidores primários através da prioridade dos membros do grupo de réplicas. Determinada a nova estação, todos os processos ativos do SAD são comunicados sobre o novo membro (TANENBAUM; STEEN, 2007).

4.2.6 Desempenho

Um dos propósitos do FlexA é trazer algumas das características importantes de outros SADs, sem comprometer o desempenho do conjunto. Com esse intuito, algumas melhorias serão exploradas a seguir.

Cache

Assim como no AFS, o uso da *cache* de arquivos inteiros nas estações clientes evita transferências subsequentes sobre arquivos já utilizados. Para versões defasadas, o usuário é informado sobre a disponibilidade de um novo arquivo no grupo de servidores.

Operações desconectadas

Cada estação armazena informações sobre o estado dos servidores e suas porções, eliminado a necessidade de buscas pela rede a cada consulta. Tais informações são obtidas através das comunicações periódicas pelo módulo comunicador. Essas informações junto com o arquivo na *cache* cliente, permite que operações possam ser realizadas diretamente no arquivo local sem a necessidade de uma conexão com o servidor.

Conjunto criptográfico

Um conjunto criptográfico rápido e descentralizado permite que os servidores primários concentrem-se apenas em administrar as porções e distribuir para os servidores

secundários. Além disso, as porções são transferidas paralelamente entre diferentes servidores, utilizando blocos de 4096 bytes.

Escalabilidade

No FlexA, o suporte a escalabilidade é feito através da utilização dos servidores secundários, os quais fornecem mais camadas de replicação sobre as porções, eliminando possíveis gargalos no grupo de escrita.

O ganho sobre essa característica será válido somente em situações de sobrecargas frequentes nos servidores, pois o FlexA é projetado apenas para trabalhar com três acessos paralelos às porções, limitando seu desempenho a um fator de três nós. Isso quer dizer que o escalonamento de estações servidoras secundárias não excederá o desempenho fornecido pela largura de rede dos três servidores primários, o qual normalmente é 33,6 MB/s, considerando três canais simultâneos de 11,2 MB/s para cada um dos servidores primários em uma rede Ethernet de 100 Mbps. Com isso, independente da quantidade de estações do grupo de leitura, cada cliente poderá se beneficiar de no máximo três estações servidoras. Em outras palavras, o grupo de leitura não melhora o desempenho, mas mantém o fornecimento dos arquivos dentro das condições de uso normal caso ocorra uma sobrecarga.

Esse conjunto de servidores secundários foi planejado apenas para a operação de leitura e não de escrita dos arquivos, visto que a função de escrita poderia afetar a consistência das porções, pois mais mensagens de sincronização seriam necessárias para validar e replicar as porções. Além disso, a divisão e distribuição para mais servidores da rede gera mais conexões de saída do nó cliente, o que pode comprometer o desempeno da escrita devido ao número de recursos computacionais exigidos do computador cliente para manter mais de três conexão simultâneas abertas, considerando que seu canal de saída é limitado a uma taxa constante (geralmente 100 Mbps). Entretanto, o processo inverso com vários canais servidores enviando pequenas partes do arquivo para o cliente, mantém o nível do fluxo de dados dos servidores baixo o suficiente (porções menores significa que mais rápidas são transferidas) para gerenciar mais requisições de leitura.

O processo de replicação pode ser descrito por $(((f/k)*c)*g)$ onde f representa o arquivo a ser replicado, k o número de porções distintas de um arquivo, c a quantidade de porções por unidade servidora e g a quantidade de servidores que será propagada.

4.3 Validação

A validação, conforme descrita nessa seção, mostra que o FlexA cumpre as principais funcionalidades de um SAD, que é permitir o armazenamento e a recuperação dos dados como se fosse em um sistema de arquivos tradicional. Para a versão atual do FlexA utilizada para os testes de validação e avaliação de desempenho não foram implantadas as seguintes funções:

- Uso do grupo de réplicas para apoio da distribuição dos arquivos e suporte na sobrecarga ou falhas dos servidores primários;
- Possibilidade de ajuste do número de conexões simultâneas para os clientes e ajuste do número de divisões por arquivo, permitindo trabalhar com mais servidores simultâneos;
- Automatização do processo de busca e verificação de novos servidores.

Após a instalação do FlexA, descrita no Apêndice A, é gerado um arquivo de configuração (*config.dat*) que contém informações da interface de rede, a faixa de varredura para busca de servidores e o local da *cache*, como mostrado na Figura 4.7.

```
interface:en1
ip: 192.168.0.103
netmask:255.255.255.0
faixa varredura ip:[0, 120]
local_cache:/Volumes/Dados/FlexA/cliente/cache/
```

Figura 4.7: Arquivo de configuração do FlexA

Com esse arquivo de configuração é possível definir o local da *cache* em disco do cliente ou onde os arquivos do servidor serão gravados. Também é possível delimitar a faixa de busca na rede para a procura de servidores.

Toda interação com o FlexA é realizada através da linha de comando, conforme comandos apresentados na Tabela 4.1.

Para a operação de escrita (*flexa.py -p file-5MB*), o módulo *flexa* do cliente e módulo *coletor* dos servidores são utilizados, gerando para o usuário da estação cliente informações sobre a existência do arquivo na *cache*, a validade dos manipuladores de permissões, a divisão do arquivo, o envio do conjunto de porções aos servidores e o retorno da efetivação de transferência dos dados, conforme mostrado na Figura 4.8.

Tabela 4.1: Comandos FlexA

Comando	Opção	Descrição
com.py	-r	Identifica a rede e gera novo arquivo de configuração
com.py	-b	Busca servidores na rede
coletor.py		Inicia o *módulo coletor*
flexa.py	-p <arquivo>	Envia arquivo para os servidores
flexa.py	-g <arquivo>	Recupera arquivo dos servidores
flexa.py	-l <arquivo, *>	Lista arquivo específico ou todos os disponíveis
flexa.py	-d <arquivo>	Apaga um arquivo específico dos servidores
flexa.py	-np <arquivo>	Gera novas permissões do arquivo

```
x:cliente silas$ python flexa.py -p file-5MB
Arquivo em cache
Permissao de escrita-leitura encontrada para o arquivo [file-5MB]
Serializando arquivo...
Aguardando transferência...
Aguardando transferência...
Enviando conjunto [ enc-2,enc-3 ] para o servidor 192.168.2.107:5000...
Enviando conjunto [ enc-3,enc-1 ] para o servidor 192.168.2.108:5000...
Enviando conjunto [ enc-1,enc-2 ] para o servidor 192.168.2.106:5000...
Conjutno [enc3, enc1] transferido para Servidor 192.168.2.108
Conjutno [enc2, enc3] transferido para Servidor 192.168.2.107
Conjutno [enc1, enc2] transferido para Servidor 192.168.2.106
x:cliente silas$
```

Figura 4.8: Saída no terminal cliente para operação de escrita

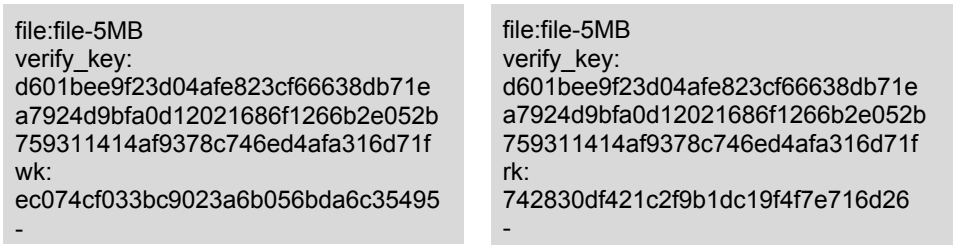
No lado dos servidores, a saída do módulo coletor exibe o tipo de operação sobre o arquivo e o solicitante da operação. Essas informações são geradas em cada servidor, diferenciando apenas no tipo de porção, como pode ser mostrado na Figura 4.9.

```
user@server01:~/FlexA/servidor$ python coletor_servidor.py

Sincronizador Servidor        [Ativo] IP:192.168.2.107; Porta socket:5000


Recebendo 513 bytes de cabeçalho
Upload arquivo [file-5MB.enc-2:1747640 bytes] pelo cliente [192.168.2.101:7000]
Transferindo arquivo...
Adicionando verificador...
Arquivo [file-5MB.enc-2] transferido


Recebendo 91 bytes de cabeçalho
Upload arquivo [file-5MB.enc-3:1747640 bytes] pelo cliente [192.168.2.101:7000]
Transferindo arquivo...
Arquivo [file-5MB.enc-3] transferido
```

Figura 4.9: Saída no terminal do *módulo coletor* servidor para a operação de escrita

Ao enviar o arquivo são gerados no cliente os manipuladores de escrita-leitura e somente-leitura (Figura 4.10 (a) e (b)) que são responsáveis pela recuperação e a alteração do arquivo nos servidores. Cada manipulador contém o nome do arquivo, a chave de verificação que é utilizada para certificar a comunicação entre os nós e uma chave específica para escrita-leitura ou somente-leitura.

Para a operação de leitura (*flexa.py -g file-5MB*), é utilizado no cliente o módulo *flexa* e o seu módulo *coletor*, enquanto no servidor é usado apenas o módulo *coletor*.



(a) Escrita-leitura (b) Somente-leitura

Figura 4.10: Manipulador arquivo

Para o usuário, a saída em tela da operação de leitura exibe o tipo de manipulador presente no computador e faz uma busca nos servidores de escrita pelas porções necessárias para compor o arquivo solicitado, mostrado na Figura 4.11. Enquanto isso, o módulo *coletor* cliente aguarda dos servidores tais porções solicitadas pelo módulo *flexa*, mostrado na Figura 4.12.

```
x:cliente silas$ python flexa.py -g file-5MB
Permissao de escrita-leitura encontrada para o arquivo [file-5MB]
Buscando arquivo...
Aguardando transferência...
Aguardando porções...
Requisição de porção [ file-5MB(enc-2) ] encaminhada para o servidor 192.168.2.107
Requisição de porção [ file-5MB(enc-3) ] encaminhada para o servidor 192.168.2.108
Requisição de porção [ file-5MB(enc-1) ] encaminhada para o servidor 192.168.2.106
Arquivos transferidos
Agrupando porções...
Decodificando...
x:cliente silas$
```

Figura 4.11: Saída terminal cliente para operação de leitura

```
x:cliente silas$ python coletor_cliente.py

Sincronizador Servidor        [Ativo] IP:192.168.2.101; Porta socket:7000


Recebendo 43 bytes de cabeçalho
Transferindo arquivo [file-5MB.enc-3:1747640 bytes] pelo cliente [192.168.2.108]...

Recebendo 43 bytes de cabeçalho
Transferindo arquivo [file-5MB.enc-2:1747640 bytes] pelo cliente [192.168.2.107]...

Recebendo 43 bytes de cabeçalho
Transferindo arquivo [file-5MB.enc-1:1747640 bytes] pelo cliente [192.168.2.106]...
Arquivo [file-5MB.enc-3] transferido
Arquivo [file-5MB.enc-1] transferido
Arquivo [file-5MB.enc-2] transferido
```

Figura 4.12: Saída terminal *módulo coletor* cliente para operação de leitura

Para os servidores, a saída no terminal para a tarefa de leitura compreende a validação dos manipuladores do arquivo e o endereço do solicitante, conforme Figura 4.13.

Os testes de validação foram aplicados em ambiente virtual com o intuito de aferir a confiabilidade, integridade e disponibilidade dos arquivos sobre as condições limitadas proporcionadas pelo ambiente de testes.

O cenário de validação foi constituído por três servidores virtuais com sistema operacional Ubuntu Linux 32-Bit, 128 MB de memória RAM, 4 GB de disco virtual

```
user@server01:~/FlexA/servidor$ python coletor_servidor.py

Sincronizador Servidor          [Ativo] IP:192.168.2.107; Porta socket:5000


Recebendo 180 bytes de cabeçalho
Download arquivo [file-5MB.enc-2] para o cliente [192.168.2.101:7000]
Enviando arquivo...
Arquivo [ file-5MB.enc-2 ] enviado para cliente 192.168.2.101:7000
```

Figura 4.13: Saída terminal *módulo coletor* servidor para operação de leitura

e adaptador de rede em modo *Bridge* com 1000 Mbps. O cliente foi executado diretamente sobre o *host* do computador, um Intel Core2Duo de 2,4 Ghz, 4 GB de memória RAM, disco rígido de 160 GB a 5400 RPM, com sistema operacional Mac OS X.

Sobre esse cenário avaliou-se o tempo médio de escrita e de leitura (*upload* e *download*) para arquivos de 1 MB, 5 MB, 10 MB, 25 MB e 50 MB, com uma frequência de três vezes para cada operação. Através desse processo, permitiu-se identificar a variação do consumo do tempo em relação ao tamanho do arquivo, conforme mostrado na Figura 4.14.

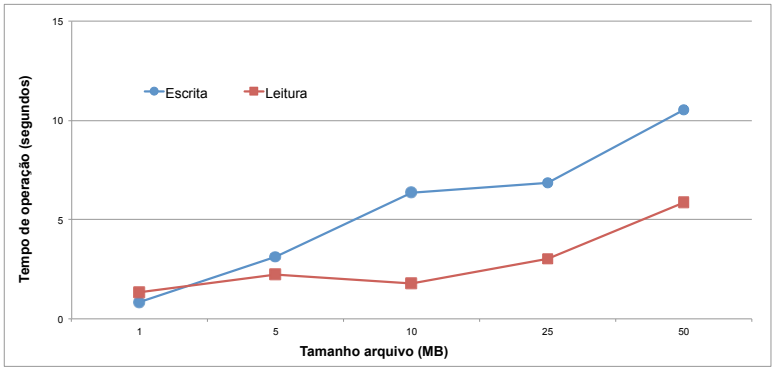


Figura 4.14: Tempo de operação

Com testes anteriores, foi possível identificar e tratar alguns problemas encontrados no início do desenvolvimento do projeto. Dentre eles pode-se destacar o baixo desempenho no uso do protocolo HTTP, perceptível quando em transferência simultânea das porções entre os servidores. Isso acontece porque o protocolo HTTP ao oferecer um método simples de fácil implantação e administração da comunicação na rede consome mais etapas, o que pode gerar latência.

Em virtude disso, todo o processo de comunicação foi refeito utilizando *sockets* sobre TCP/IP, mostrando-se eficiente para esse projeto. Com isso, trouxe a possibilidade de ajustes na comunicação quanto a troca de dados, permitindo trabalhar com blocos de 4096 bytes.

4.4 Considerações Finais

O modelo desenvolvido é uma forma de expressar a possibilidade de agregar as características de outros SADs e torná-lo funcional. Na Tabela 4.2 é mostrada um resumo das principais características do FlexA.

Tabela 4.2: Características FlexA

Característica	FlexA
Arquitetura	Descentralizada baseada em grupos de servidores.
Escalabilidade	Alta escalabilidade com distribuição descentralizada dos dados.
Cache	No cliente através do armazenamento de arquivos inteiros e seus metadados.
Confiabilidade	Método independente do sistema local com criptografia direta sobre o arquivo no computador cliente utilizando chave simétrica AES-256 bits.
Interface	Independente.
Tolerância a falhas	Distribuição das porções do arquivo entre os grupos de servidores.
Segurança	Manipuladores individuais para identificação e transferência de dados utilizando canais seguros.

5 Avaliação e comparação de desempenho

5.1 Considerações Iniciais

A avaliação e a comparação de desempenho segue com um propósito de estabelecer e aplicar o modelo FlexA dentro de um cenário analisando seu comportamento frente as requisições em comparação com outros SADs. Para essa finalidade, existem diversas ferramentas de *benchmarks* que podem ser aplicadas, mas o seu resultado pode ser inconclusivo quando sistemas de arquiteturas diferentes são comparados.

Por isso, nesse capítulo são exploradas técnicas para avaliar e comparar o desempenho dos SADs estudados e do modelo desenvolvido, possibilitando afirmar com clareza a variação de desempenho em um conjunto distinto de sistemas. Para a avaliação, optou-se pela técnica de aferição por coleta de dados, utilizando um conjunto de testes os quais focam principalmente no desempenho da distribuição dos arquivos. Nas seções seguintes serão mostrados os critérios de avaliação em um ambiente real.

5.2 Cenário, critérios de avaliação de desempenho e ferramentas

A avaliação foi efetuada em um cenário de pequena escala composto por 9 computadores variando entre servidores, servidores específicos para armazenamento e clientes, conforme mostrado na Tabela 5.1

Tabela 5.1: Disposição da arquitetura de sistemas de arquivos distribuídos

SAD	Servidores	Nós de armazenamento	Clientes
FlexA	0	3	6
NFS	1	0	6
Tahoe-LAFS	1	3	5

A avaliação ocorreu através de operações de escrita e leitura de arquivos com tamanhos pré-definidos de 1 MB, 5 MB, 10 MB, 25 MB e 50 MB. Esses arquivos procuram incluir a variação de dados encontrados em ambiente acadêmico, que podem ser arquivos de textos, músicas, fotos, vídeos e aplicativos.

A aferição foi calculada pelo tempo real da operação, em segundos, gerando uma estimativa da taxa de transferência, dada em Megabytes por segundo (MB/s). Dessa forma, avaliou-se o comportamento do sistema quanto à velocidade na transferência de arquivos. O uso da *cache* nos sistemas avaliados foi desconsiderado com a finalidade de analisar o tempo envolvido apenas no processo de escrita e leitura entre clientes e servidores, e também evitar uma comparação desigual ao Tahoe-LAFS que não utiliza *cache*.

O cenário de pequena escala, como mostrado na Figura 5.1, é composto por estações com processadores Intel Pentium Dual E2160 - 1,8 GHz, 2GB de memória RAM, disco rígido de 40 GB a 7200 RPM, com sistema operacional Ubuntu Linux 64 bits e interligados com uma rede Ethernet 100 Mbps *full-duplex* por um *Switch* 3Com.

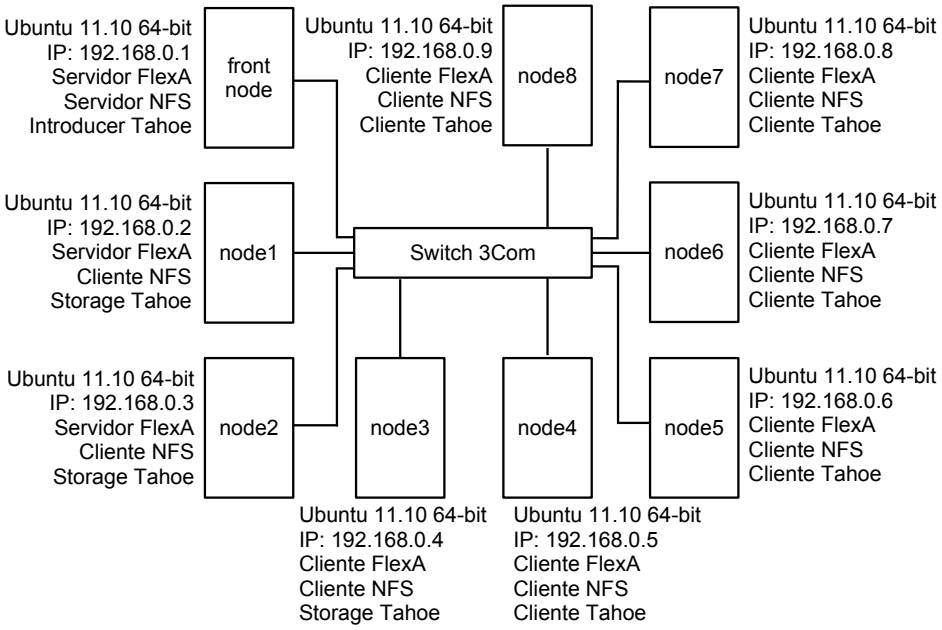


Figura 5.1: Cenário de testes

Para o cenário de avaliação, os testes foram aplicados e analisados através de um *script* escrito em linguagem Python, que executou repetidas vezes operações de *uploads* e *downloads* para os cinco tipos de arquivos simultaneamente entre os clientes.

A avaliação foi aplicada cliente após cliente, aferindo simultaneamente cada novo computador com os anteriores, conforme Figura 5.2.

Ao final, foram executadas doze interações de leitura e escrita para cada um dos cinco arquivos, totalizando sessenta operações de escrita e sessenta de leitura em cada cliente.

Para análise da largura de banda do cenário de testes, utilizou-se a ferramenta *iperf*,

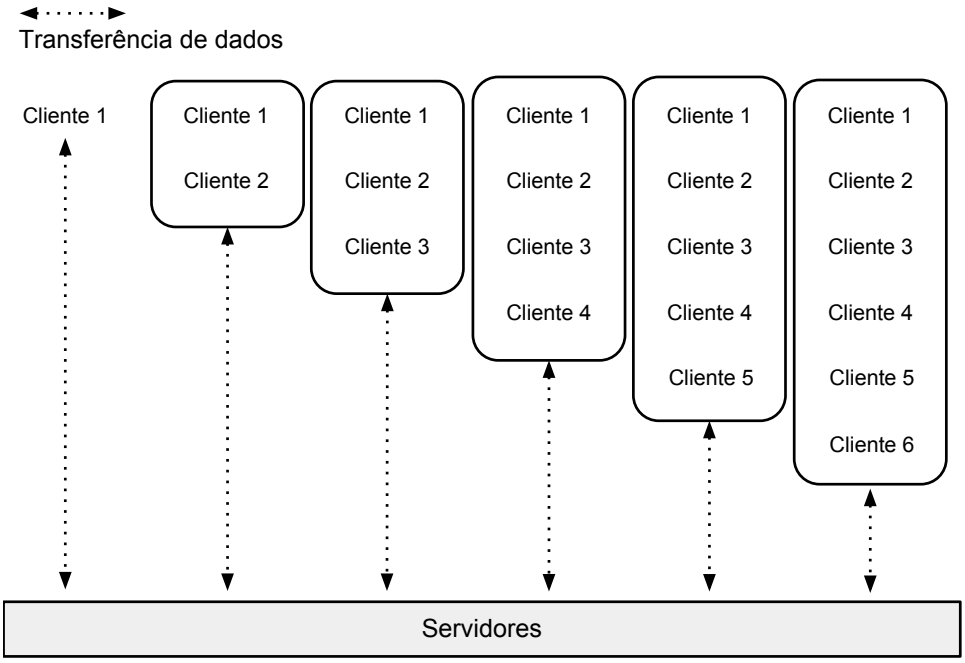


Figura 5.2: Sequência das interações entre o conjunto cliente com os servidores

disponível na maioria das distribuições Linux. Com ela, no cenário avaliado, a média para transferências de pacotes atingiu 11,2 MB/s de uma rede de 100 Mbps.

Para os SADs estudados que utilizam conjunto de servidores paralelos (FlexA, GFS e Tahoe-LAFS), a sua largura de banda total (grupo de servidores) é multiplicada pela quantidade de estações de armazenamento. Nesse caso, a taxa de transferência para esse conjunto foi atribuída como global, envolvendo o consumo de rede de todos os servidores.

5.3 Resultados

Para o propósito desse trabalho foram consideradas as avaliações do Tahoe-LAFS e do NFS, dentre os SADs estudados no capítulo 3. Essa escolha justifica-se pelo fato de que os principais recursos do Tahoe-LAFS são agregados no FlexA e da comparação com um SAD cliente-servidor sem camadas adicionais, como o NFS.

O cenário de avaliação foi composto de seis casos para o FlexA (seis clientes e três servidores), cinco casos para o Tahoe-LAFS (cinco clientes, três servidores de armazenamento e um servidor de administração) e seis casos para o NFS (seis clientes e um servidor). O número de casos formados para cada avaliação representa a quantidade de clientes simultâneos disponíveis para os testes no cenário utilizado, totalizando nove computadores. Para o NFS, limitou-se o número de clientes em seis, mesmo com mais duas estações sem uso, com o objetivo de obedecer a quantidade de acessos simultâ-

neos do FlexA em virtude do número de computadores disponíveis.

5.3.1 FlexA

Com uma taxa de transferência global teórica de 33,6 MB/s, na avaliação do FlexA não foi considerado o uso do grupo de réplicas, pois, independente da quantidade de servidores excedentes, o acesso paralelo do cliente é limitado a três conexões simultâneas. Nesse caso, o grupo de réplicas não traria ganho no desempenho, mas sim condições para aliviar a sobrecarga do acesso aos servidores de escrita e tolerância a falhas.

Desse modo, sem o grupo de réplicas permitiu-se obter resultados de leituras em várias condições com o mínimo necessário de estações servidoras.

Na Figura 5.3 é mostrada a comparação do processo de escrita para os seis casos.

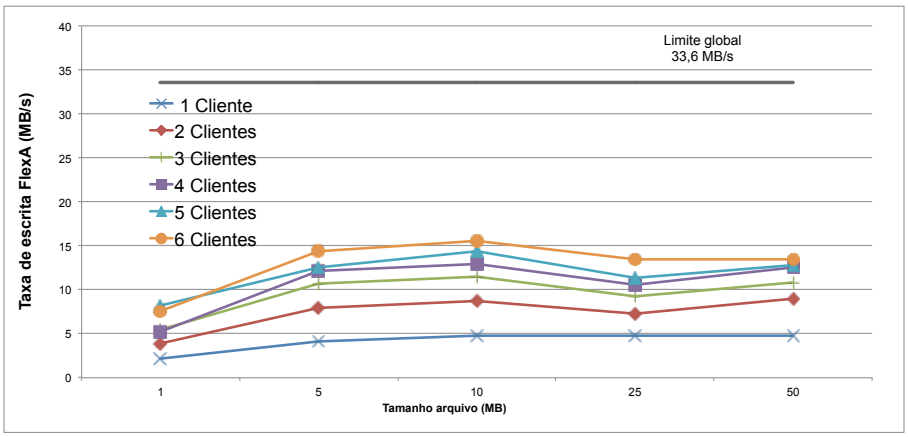


Figura 5.3: Taxa de escrita FlexA

A análise do processo de escrita por um único cliente expõe uma uniformidade na taxa de transferência a partir de arquivos com 10 MB, o qual deve-se ao limite imposto pela largura de banda da sua rede, aproximadamente 11,2 MB/s. Entretanto, nesse mesmo ponto, a vazão dos dados limita-se próxima aos 5 MB/s, ou seja, 44,6% da sua capacidade total. Isso acontece por causa do processo de replicação realizado pelo cliente, o que gera um consumo adicional para a distribuição das porções entre os três servidores do grupo de escrita.

Desse modo, aliado ao fator replicação (o qual compreende o dobro de dados enviado pelo cliente), o consumo real da rede para cada cliente é aproximadamente 50% maior que o consumo nominal da aplicação.

Expandindo essa situação para um cenário com 6 clientes de acesso simultâneo e considerando o seu consumo real, como mostrado na Figura 5.4, tem-se um processo

de transferência próximo ao limite teórico da rede.

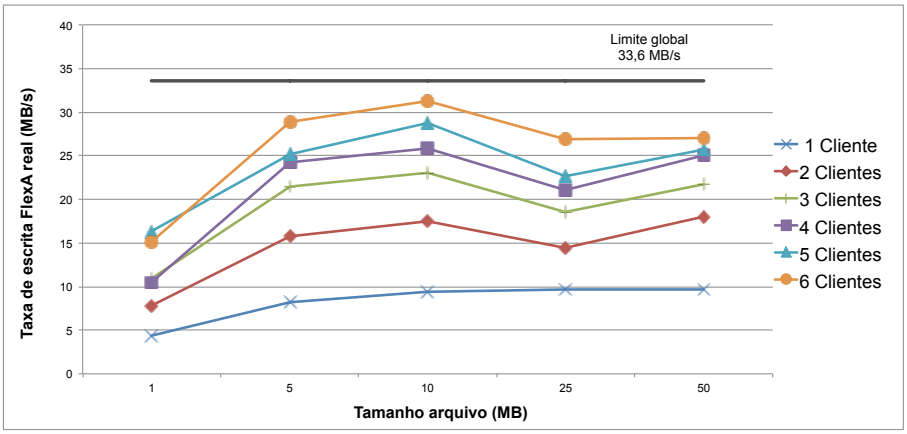


Figura 5.4: Taxa de escrita FlexA real

Para o processo de leitura, mostrado na Figura 5.5, a taxa de transferência segue uma proporção maior, próxima ao limite teórico para seis clientes. Nesse caso, o cliente obtém uma cópia das porções simultaneamente, ao contrário do processo de escrita que, somando todas as porções envias pelo processo de sincronização, compreendem a duas vezes ao tamanho do arquivo original.

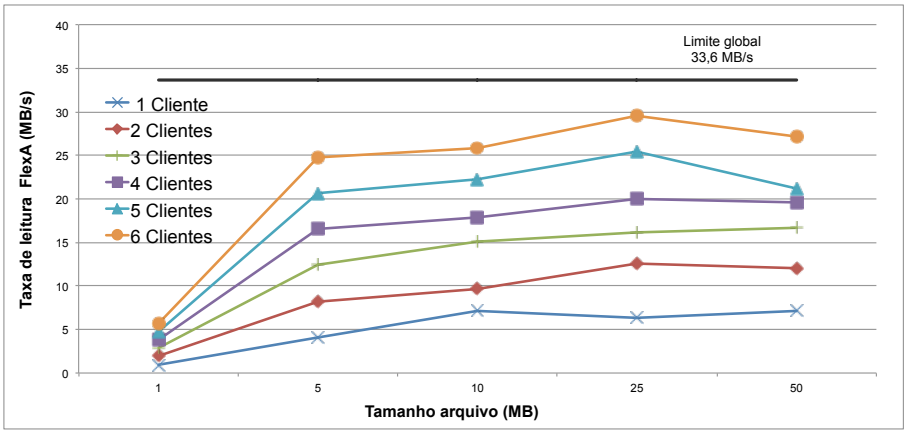


Figura 5.5: Taxa de leitura FlexA

Com a sincronização em dois terços (66%) do arquivo para cada servidor, o volume global de dados armazenados no FlexA segue o dobro para cada arquivo (aproximadamente 200% do arquivo original), como mostrado na Figura 5.6

Esse arranjo possibilita ao cliente obter a mesma porção em até dois servidores diferentes, e no caso de algum dos três falhar, dois serão suficientes para recuperar as porções e montar o arquivo.

Para o cliente, as tarefas mais custosas (criptografia, divisão e distribuição), são responsáveis por uma pequena parcela no consumo dos recursos de *hardware*. Na Figura 5.7 é mostrado o consumo da UCP para uma operação de escrita utilizando um

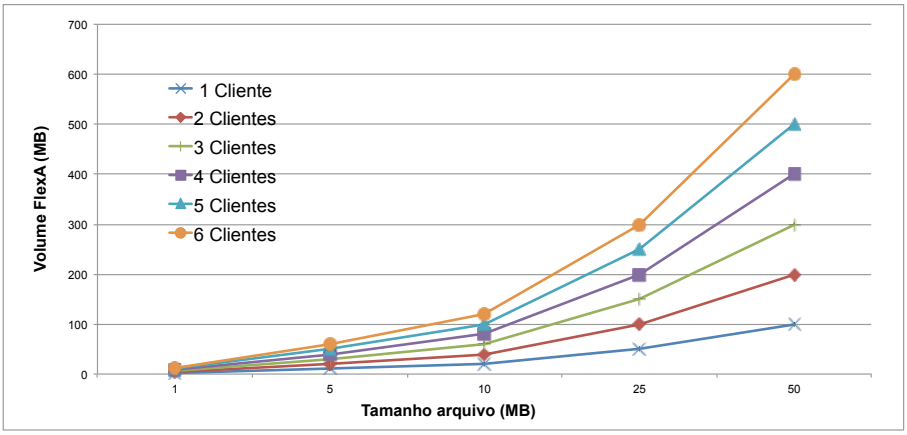


Figura 5.6: Volume de dados processado pelo FlexA

arquivo de 50 MB, o qual atinge um máximo de aproximadamente 50% para o cliente e menos de 10% para os servidores. Para a leitura do mesmo arquivo, conforme apresentado na Figura 5.8, os servidores permanecem abaixo dos 5% enquanto o cliente sofre uma variação maior, mas sem extrapolar a faixa dos 50%.

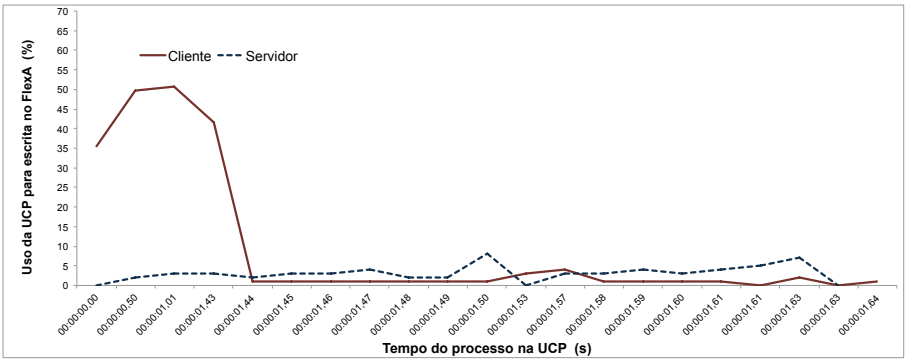


Figura 5.7: Porcentagem de uso de UCP para escrita no FlexA

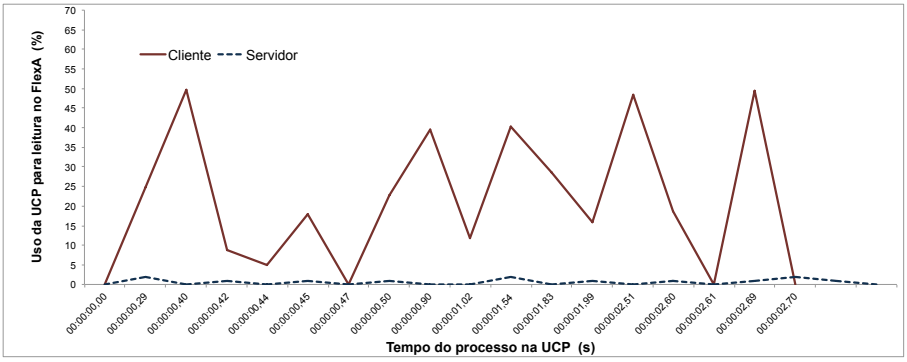


Figura 5.8: Porcentagem de uso de UCP para leitura no FlexA

Tal variação no cliente é resultado do *módulo coletor* trabalhando em conjunto com o *módulo flexa*, que gera um consumo maior da UCP para obter as porções dos

servidores, agregá-las e descriptografá-las. Diferente do processo de envio, no qual é usado somente o *módulo flexa* em conjunto com as camadas *sockets* de comunicação.

Quanto ao consumo de memória RAM, nas Figuras 5.9 e 5.10 são apresentadas as variações do uso em relação a 50 MB de arquivos para escrita e leitura, respectivamente.

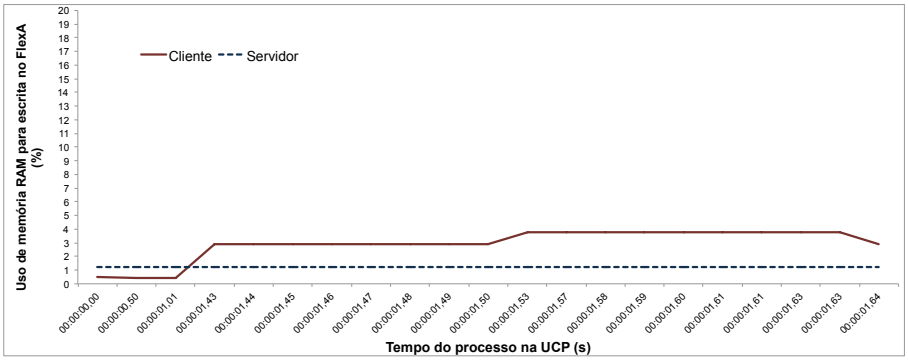


Figura 5.9: Porcentagem do uso de memória RAM para escrita no FlexA

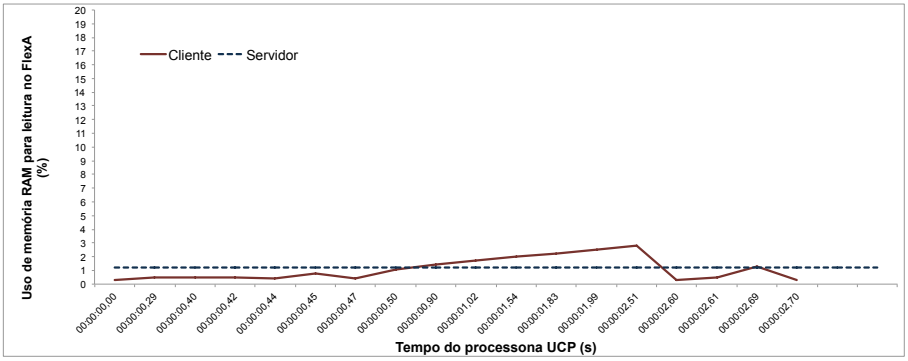


Figura 5.10: Porcentagem do uso de memória RAM para leitura no FlexA

O consumo de memória RAM para escrita indica o acumulo dos eventos internos dos módulos, com destaque para a criptografia, divisão e distribuição do arquivo para os clientes. Para a leitura, o pico no consumo de memória é destacado pela requisição das porções, junção das partes e descriptografia do arquivo.

Analisando o tempo ao longo do crescimento dos arquivos, os servidores apresentam um baixo consumo de processamento, limitando-se apenas a disponibilidade da largura de banda e o desempenho dos discos rígidos. Quanto ao cliente, o impacto desse consumo em relação à configuração do cenário avaliado e à carga de arquivos testados gera um tempo a mais para o término do processo, como mostrado na Figura 5.11.

Esse consumo do *hardware* independe da quantidade de clientes ou acessos simultâneos, variando apenas em relação à configuração do computador. O processamento

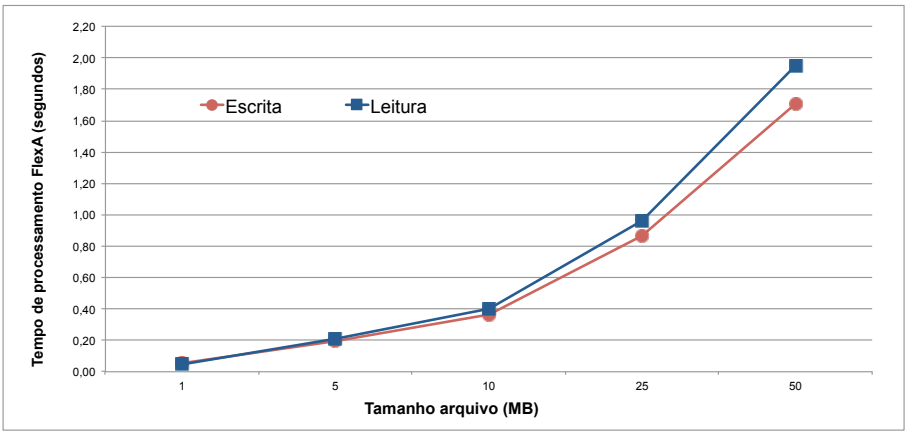


Figura 5.11: Variação do consumo de hardware no cliente

local viabiliza ganhos sobre a rápida evolução do *hardware* convencional, de modo a aproveitar efetivamente esses recursos.

Nesse sentido, um outro cenário com outra configuração para as estações clientes foi utilizado, com o intuito de aferir um conjunto diferente de *hardware*. Nesse novo cenário, as estações clientes são configuradas principalmente por computadores com processador Intel Core i7 870 - 2,93GHz, 8 GB de memória RAM, disco rígido de 500 GB a 7200 RPM e com sistema operacional Linux 64 bits. O impacto desse novo conjunto para o consumo da aplicação cliente é mostrado na Figura 5.12, alcançando uma economia no processo criptográfico de 47,17% para a tarefa de leitura e 46,47% para a tarefa de escrita sobre arquivos de 50 MB em relação ao primeiro caso de teste.

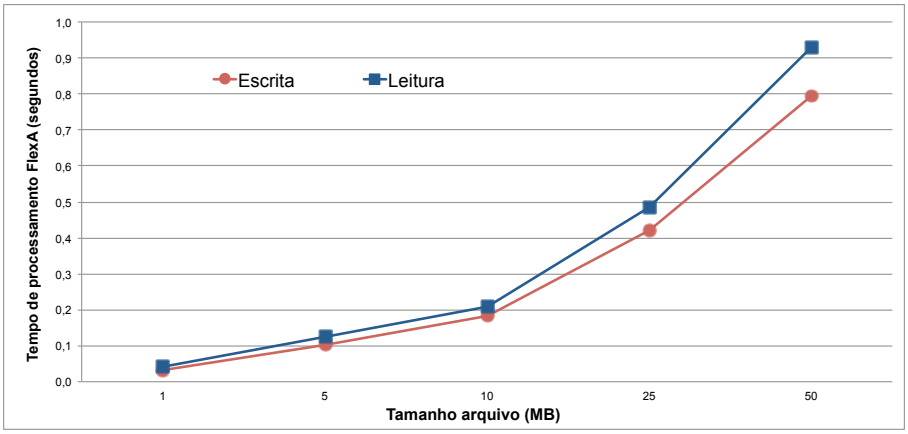


Figura 5.12: Variação do consumo de hardware para um segundo cenário

5.3.2 Tahoe-LAFS

Por padrão o Tahoe-LAFS vem configurado para utilizar dez servidores de armazenamento, sendo que somente três desses servidores serão necessários para reconstruir

o arquivo. Nesse caso, pelo número limitado de computadores disponíveis para avaliação foram considerados apenas três servidores, sendo que dois serão necessários para reconstruir o arquivo, igualando ao cenário utilizado para avaliação do FlexA.

A sua avaliação partiu do mesmo procedimento aplicado no caso anterior, através de interações sequenciais em cada cliente e simultâneas para os arquivos nos cinco casos.

O resultado da avaliação para a taxa de escrita e leitura global são mostrados nas Figuras 5.13 e 5.14, respectivamente.

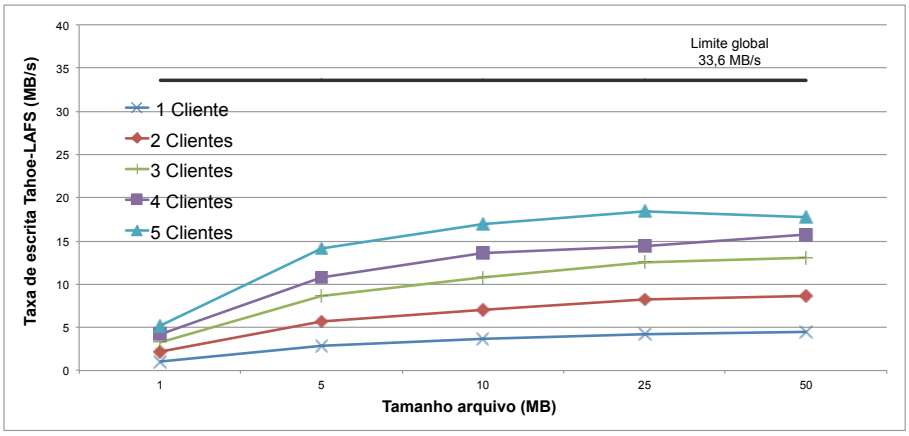


Figura 5.13: Taxa de escrita global Tahoe-LAFS

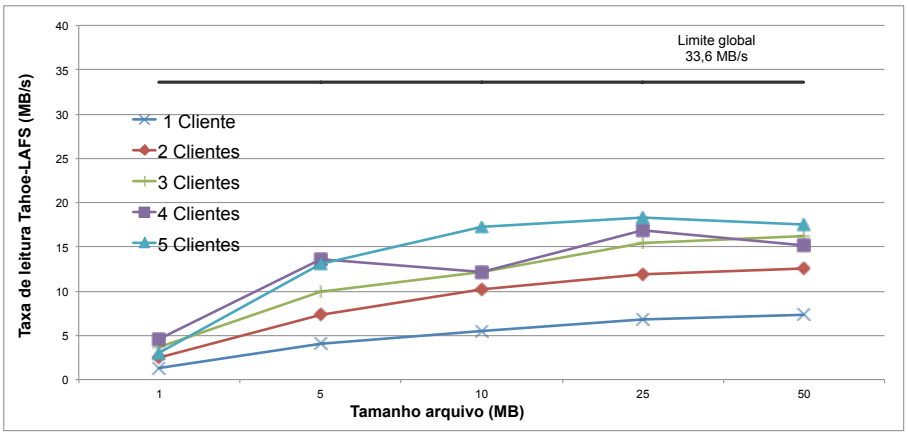


Figura 5.14: Taxa de leitura global Tahoe-LAFS

Para avaliação do uso da UCP e memória no Tahoe-LAFS, utilizou-se do mesmo critério usado no FlexA através do registro da variação do uso dos recursos computacionais em relação a *upload* e *download* de arquivos de 50 MB, como é mostrado nas Figuras 5.15 e 5.16.

Por mais que a porcentagem do consumo da UCP não exceda os 50%, como no FlexA, o seu uso permanece constante durante a execução da tarefa. Isso pode se

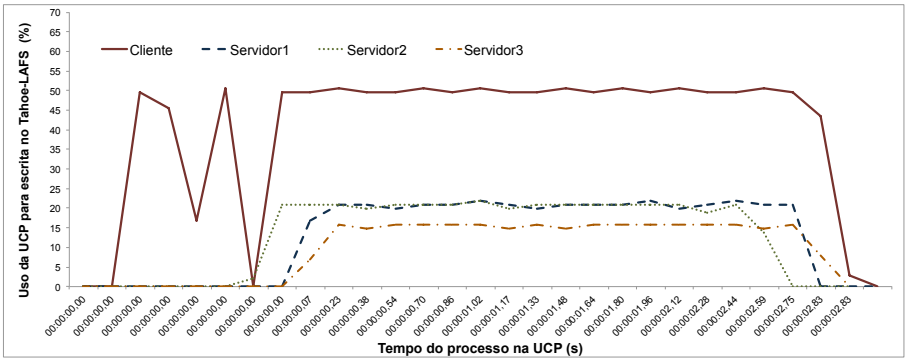


Figura 5.15: Porcentagem de uso de UCP para escrita no Tahoe-LAFS

explicado através do seu processo de segurança e distribuição dos arquivos, causando maior consumo do *hardware* em todos os nós envolvidos.

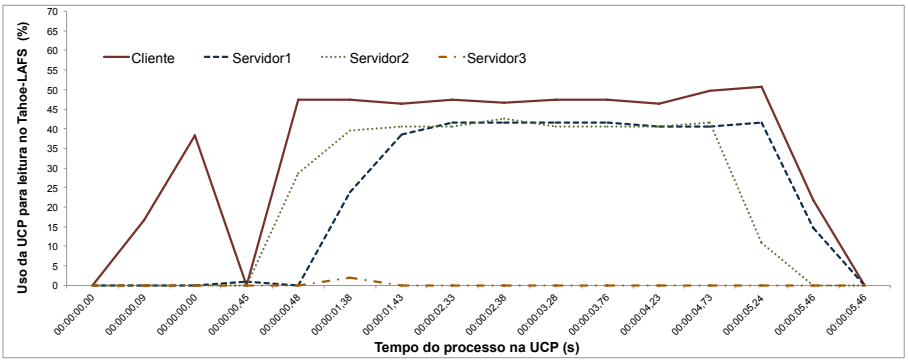


Figura 5.16: Porcentagem de uso de UCP para leitura no Tahoe-LAFS

Nesse contexto, o consumo de memória RAM segue uniforme com uma pequena elevação para o cliente, conforme Figura 5.17 e Figura 5.18. Não foi considerado o consumo do servidor *introducer* por não apresentar qualquer variação quanto ao consumo da UCP e da memória RAM.

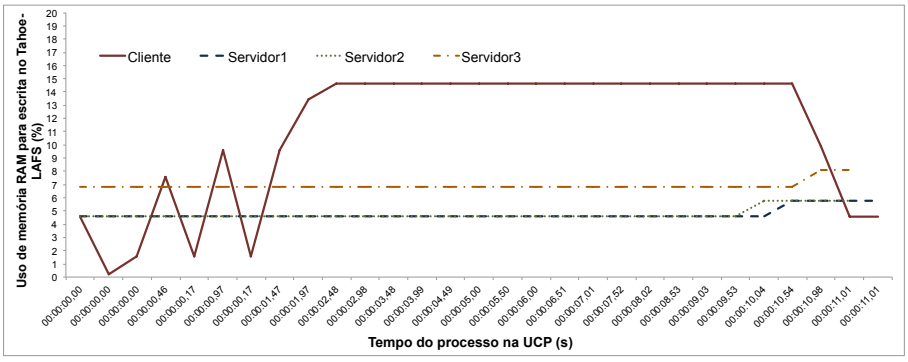


Figura 5.17: Porcentagem do uso de memória RAM para escrita no Tahoe-LAFS

Configurado para fornecer arquivos com apenas dois servidores ativos de três disponíveis, o Tahoe-LAFS distribui 50% do arquivo para cada um deles, gerando um

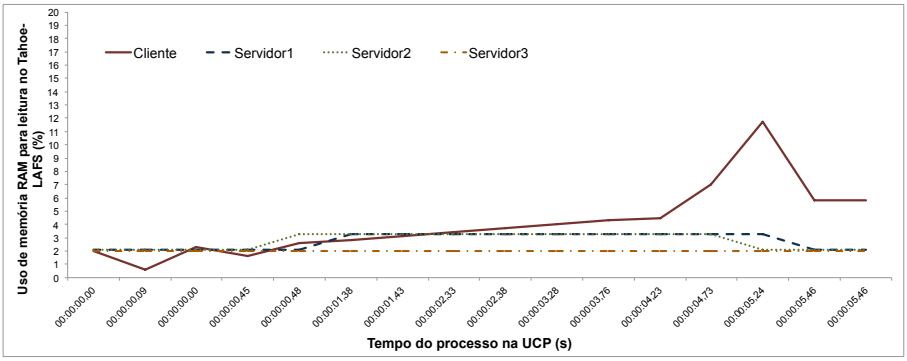


Figura 5.18: Porcentagem do uso de memória RAM para leitura no Tahoe-LAFS

volume de dados de 50% a mais do arquivo original, como pode ser observado na Figura 5.19.

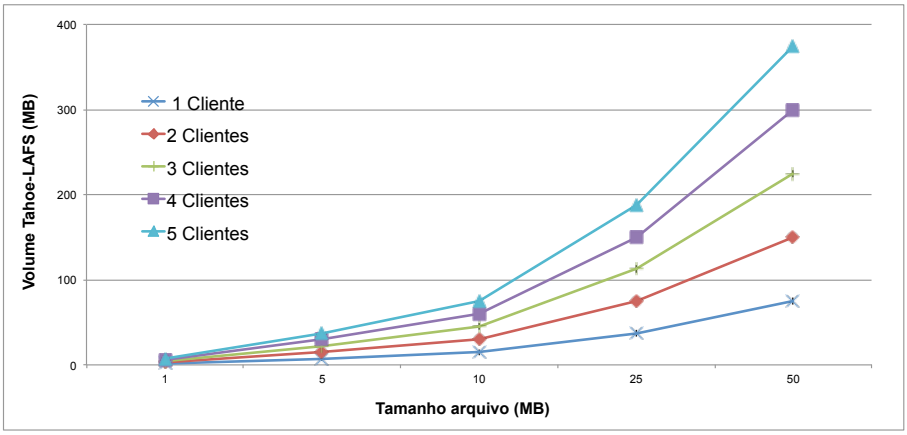


Figura 5.19: Volume de dados processado pelo Tahoe-LAFS

5.3.3 NFS

Com a avaliação do NFS busca-se compreender a relação entre uma arquitetura cliente-servidor sem camadas adicionais em comparação com a arquitetura do Tahoe-LAFS e FlexA, os quais dispõem de conjunto de servidores para suprir a demanda.

Assim como no FlexA e Tahoe-LAFS, a avaliação do NFS seguiu o mesmo conceito sobre um conjunto formado por seis clientes e um servidor.

Nas Figuras 5.20 e 5.21 são mostradas, respectivamente, as saídas do *script* para a operação de escrita e para a operação de leitura dos arquivos.

Por não ser formado com um conjunto de servidores, a taxa de transferência limitou-se à largura do canal de comunicação de apenas uma estação, o servidor NFS. Nesse aspecto, o limite teórico do servidor NFS tem um teto de 11,2 MB/s para um conjunto de seis clientes com condição de fluxo simultâneo de 67,2 MB/s. Com isso,

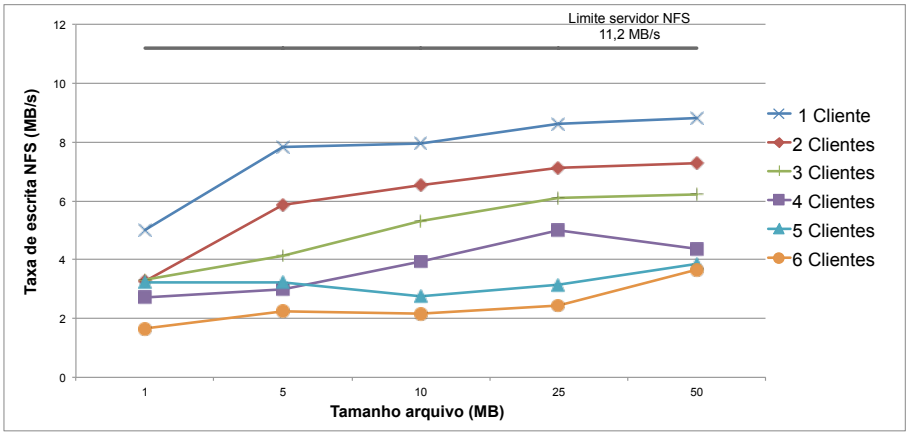


Figura 5.20: Taxa de escrita NFS

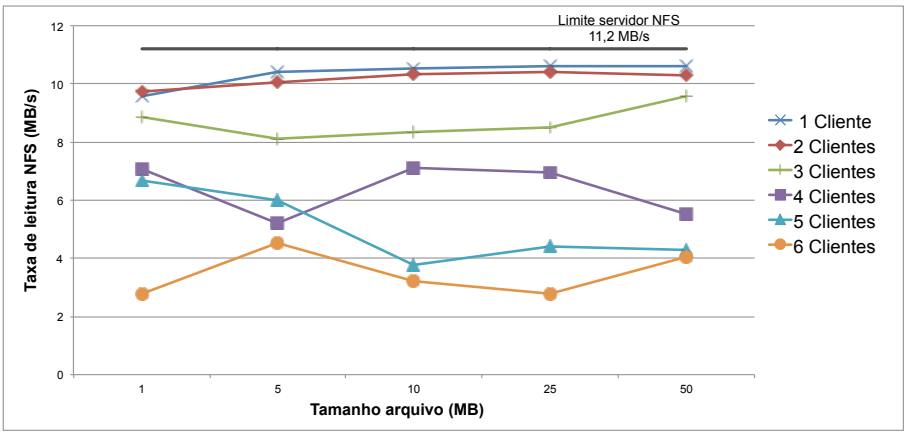


Figura 5.21: Taxa de leitura NFS

considerando o critério de avaliação por grandes volumes de arquivos em paralelo, é esperada a rápida saturação da comunicação desse SAD, mostrando um declínio da taxa de transferência em relação ao seu limite teórico.

Por não depender de nenhum método de criptografia sobre o arquivo e divisão em porções, o uso da UCP no NFS apresenta valores abaixo dos 10%, conforme mostrado nas Figuras 5.22 e 5.23.

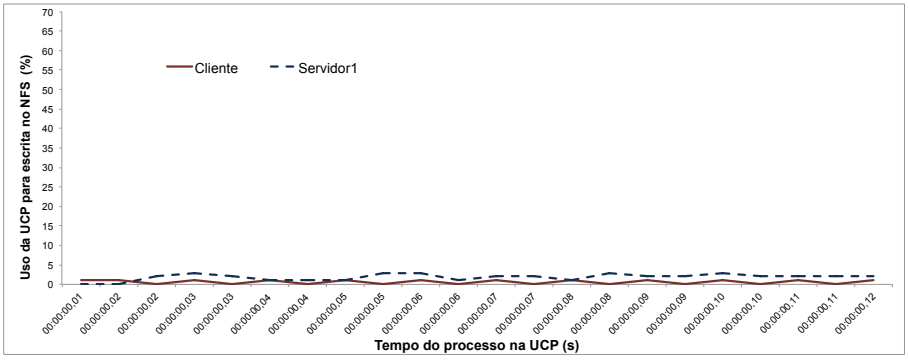


Figura 5.22: Porcentagem de uso de UCP para leitura no NFS

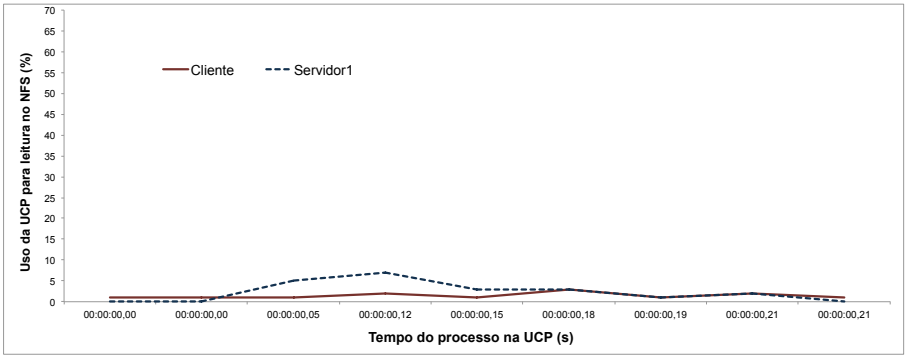


Figura 5.23: Porcentagem de uso de UCP para escrita no NFS

Informações sobre o consumo de memória RAM foram descartadas por não representarem mais que 0,5% dos recursos disponíveis.

O volume de dados dados gravados no servidor é o mesmo que o enviado pelo cliente, pois trata-se exatamente do mesmo tamanho enviado, não oferecendo mecanismos de replicação.

5.4 Comparação de desempenho

Analisados os principais SADs que fizeram parte do estudo e desenvolvimento deste trabalho, cabe agora comparar o desempenho geral entre eles, destacando os ganhos e o quanto custoso em termos de uso da UCP é para o sistema.

Para comparação entre os SADs, foram considerados os testes com cinco clientes simultâneos sobre os cinco tipos de arquivos para operação de escrita e leitura.

Para a operação de escrita, Figura 5.24, é possível analisar que embora o NFS seja um dos SADs mais rápidos sem concorrência de acesso, como analisado na Figura 5.20, o seu uso em ambiente com várias requisições torna-se mais lento, prejudicando o desempenho geral do sistema. Através do NFS pode-se entender que mesmo em um ambiente de pequena escala, a centralização da arquitetura pode representar um fator de degradação do desempenho, além dos riscos de falhas.

A operação de escrita para o FlexA e o Tahoe-LAFS apresenta similaridade, isso porque a lógica do funcionamento (criptografia, divisão e distribuição) é parecida, diferenciando-se em alguns aspectos como, por exemplo, no modelo de criptografia sem o uso de chaves assimétricas RSA para o FlexA, na divisão do arquivo em 66% contra 50% para o Tahoe-LAFS e na distribuição utilizando *sockets* por TCP/IP contra HTTP do Tahoe-LAFS. Entretanto, a operação de escrita do Tahoe-LAFS apresenta-se mais rápida ao longo da evolução dos arquivos, sendo menos otimizada apenas na

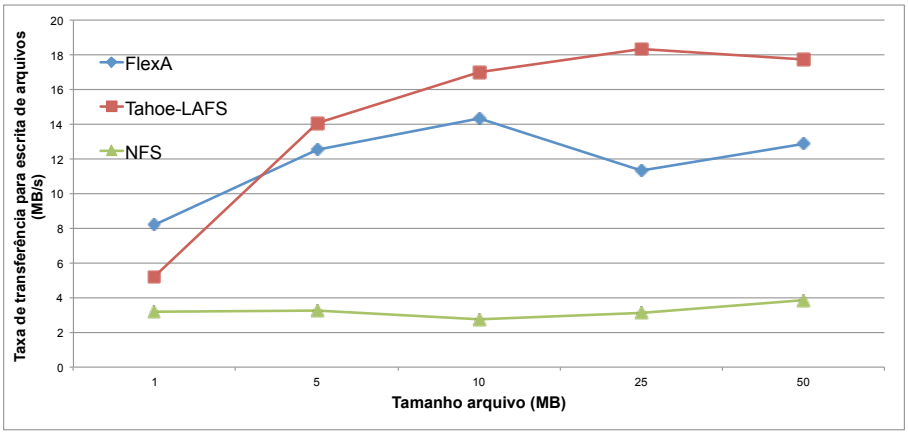


Figura 5.24: Comparação entre as taxas de transferências para escrita de arquivos

escrita de arquivos abaixo de 4 MB, para a qual o FlexA é ligeiramente mais rápido com aproximadamente 8 MB/s comparado com o 5 MB/s do Tahoe-LAFS. O motivo do FlexA ser significativamente mais lento que o Tahoe-LAFS para escrita de arquivos acima de 5 MB é justificado pelo número maior de divisões e distribuições que o FlexA faz por conjunto de três servidores, totalizando 66% do arquivo para cada estação contra 50% de um arquivo para o Tahoe-LAFS.

Para a operação de leitura, Figura 5.25, arquivos próximos a 1 MB trouxe vantagem para o NFS, pois, conforme explorado nesse trabalho, ele não possui qualquer recurso de segurança além do oferecido pelo sistema operacional local, tornando-o mais rápido para operações com arquivos menores.

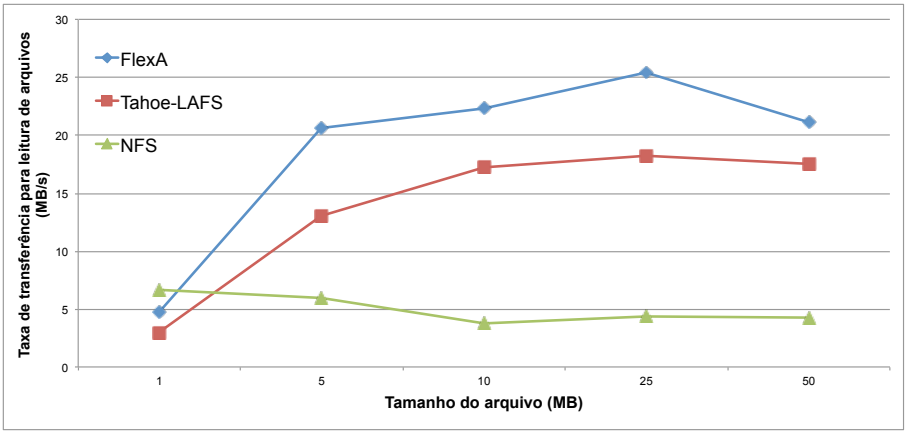


Figura 5.25: Comparação entre as taxas de transferências para leitura de arquivos

Nas situações em que o tamanho dos arquivos é superior a 1 MB o cenário muda, priorizando SADs formados por conjunto de servidores paralelos. Nessa avaliação, destaca-se o FlexA, que apresentou a maior evolução para a taxa de transferência. Por causa do seu fator de divisão e processo de escolha para obter as porções distribuídas, a leitura do Tahoe-LAFS limita-se apenas em dois dos três canais de comunicação.

Ao contrário, no FlexA são utilizados todos os servidores disponíveis em todas as operações de leitura.

A avaliação do uso dos recursos de *hardware*, Figura 5.26, compreende o quanto servidores e clientes consomem para operações de leitura e escrita. Nessa análise, o Tahoe-LAFS destaca-se pelo seu elevado consumo da UCP e memória RAM no conjunto de servidores e no cliente, tornando-se um solução mais custosa quando comparado aos outros SADs.

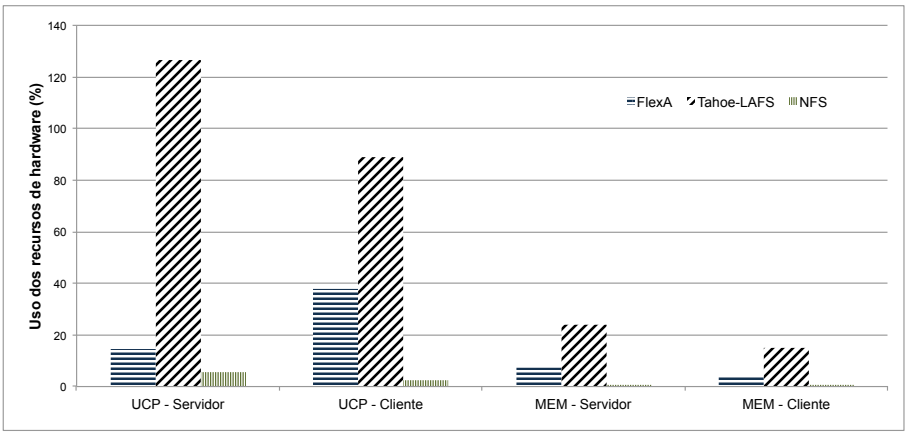


Figura 5.26: Comparação entre o uso dos recursos de *hardware*

Em seguida vem o FlexA com um consumo intermediário abaixo dos 50% no cliente e menos de 20% na soma do uso das UCPs dos três servidores. Essa economia do uso dos recursos computacionais é alcançada através da simplificação do modelo de permissões descentralizadas herdada do Tahoe-LAFS.

Por último e mais econômico o NFS com menos de 10% para uso da UCP nos servidores, menos de 5% para o cliente e menos de 1% no uso de memória RAM em ambos os casos.

5.5 Considerações finais

A análise em um ambiente de pequena escala permitiu mostrar que o modelo FlexA está inserido no contexto de SADs aptos a uso dentro um de cenário real.

Essa comparação também permitiu avaliar o desempenho do FlexA quanto a facilidade de implantação e uso de dependências para o seu correto funcionamento.

Comparado a taxa de transferência geral do FlexA com o Tahoe-LAFS, pode-se afirmar que ambos obtiveram resultados próximos, com a diferença do Tahoe-LAFS ser 7,8% mais rápido nas operações de escrita e o FlexA ser 15,03% mais rápido nas operações de leitura. Além disso, o consumo de *hardware* no FlexA se saiu melhor

com menos uso da UCP e memória RAM para as operações de leitura e escrita. Nesse caso, além de ser menos exigente na configuração das estações, ele também é mais suscetível a melhorias quanto à evolução do *hardware*, isto é, ele consegue tirar melhor proveito do poder de processamento local cliente para agilizar algumas das suas tarefas (criptografia e divisão).

Outro ponto positivo para o FlexA é a sua estrutura menos complexa do que o Tahoe-LAFS, possibilitando condições para ajustes e adaptações de suas características de modo a alcançar melhor suporte às exigências do ambiente.

6 Conclusão

Neste trabalho foi apresentada a definição e a implantação de um modelo de SAD apoiado sobre a metodologia do Tahoe-LAFS, com foco na utilização de recursos computacionais de baixo custo, no uso de criptografia rápida e segura, na facilidade do gerenciamento de arquivos e mecanismos para tolerar as falhas.

Projetado para ser menos complexo do que o Tahoe-LAFS com o propósito de facilitar ajustes de suas funções em seu código fonte, o qual é desenvolvido em linguagem Python e de código aberto. O FlexA é um modelo desenvolvido que expressa a possibilidade de agregar características de outros SADs tornando-se funcional, ou seja, é possível armazenar e recuperar os dados como se fosse um sistema tradicional.

A simplificação da sua construção traz flexibilidade na utilização dos recursos computacionais com um uso mais eficiente do poder de processamento do cliente e menores condições de sobrecarga em servidores. Também proporciona maior independência através das operações desconectadas com uso agressivo da *cache*.

A migração das tarefas mais custosas para o cliente (criptografia, divisão e distribuição) traz a possibilidade de um uso eficiente dos seus recursos computacionais, de modo que o custo de uso da UCP e da memória RAM sejam diluídos nas estações clientes. Além disso, o FlexA é mais suscetível a melhorias quanto a evolução do *hardware* cliente, comparado a outros SADs que priorizam somente o consumo dos recursos computacionais dos servidores.

A adaptação do modelo de permissões descentralizadas do Tahoe-LAFS permite maior independência de administradores e mais agilidade sobre o processo de criptografia, pois com a ausência da chave assimétrica RSA e com a utilização de mais sequências *hash* sobre a chave de codificação, o FlexA consegue realizar o processo criptográfico com menos passos, sem deixar de oferecer um nível de segurança aceitável.

O FlexA fornece confiabilidade dos dados com o uso de criptografia direta sobre os arquivos, impedindo casos em que a falha ou a falta da segurança do conjunto de armazenamento comprometa a integridade dos mesmos. Também garante disponibili-

dade dos arquivos com a utilização de grupos de armazenamento e dispõe de condições à tolerância a falhas com a distribuição do conteúdo entre diversas estações.

O modelo desenvolvido para a avaliação foi construído para trazer um ambiente favorável para o cliente, fornecendo um SAD menos complexo para a implantação e utilização em condições normais, sem deixar de oferecer algumas das características encontradas na maioria dos sistemas.

O seu funcionamento foi validado e seu desempenho foi comparado com outros dois SADs. Um deles foi o Tahoe-LAFS, considerado o precursor desse desenvolvimento e o outro o NFS, com o intuito de obter uma comparação com sistema exclusivamente cliente-servidor. Outros SADs estudados, como por exemplo o AFS e GFS não foram aferidos por serem similares, em alguns aspectos, aos outros dois.

Com a avaliação e a comparação entre eles, foi possível analisar o desempenho do FlexA e o custo para o sistema suportar tal SAD. Nessa avaliação, com foco apenas na técnica de carga/atualização dos arquivos, destaca-se a proximidade entre FlexA e Tahoe-LAFS, sendo o primeiro com melhor taxa de leitura, e o segundo sistema com melhor desempenho na escrita.

Para o FlexA o melhor desempenho na operação de leitura é explicado pelo fator de divisão dos arquivos de modo a usar três servidores simultaneamente o que, por conseguinte, gera maior consumo no seu processo de escrita em relação ao Tahoe-LAFS. Além dessa diferença, há um destaque no consumo dos recursos computacionais, em que o FlexA pôde oferecer um sistema mais eficiente para distribuição de arquivos com uso simultâneo entre clientes.

O principal destaque do NFS está em oferecer um SAD extremamente econômico quanto ao uso do *hardware*. Por não fornecer camadas de segurança individuais fora aquelas que o sistema operacional já aplica e por não fornecer um sistema integrado de replicação dos dados, o NFS disponibiliza um sistema mais eficiente para situações onde o uso concorrente não é intensivo.

Quanto à instalação, nenhum dos SADs avaliados apresentou necessidades especiais para as suas operações. Por padrão, o NFS e Tahoe-LAFS já fazem parte dos repositórios de algumas distribuições Linux, enquanto o FlexA precisa apenas do interpretador Python e mais duas dependências, uma para administrar a interface de rede e outra para trabalhar com criptografia, conforme descrito no apêndice A.

Também, conseguiu-se avaliar o desempenho dos servidores FlexA em ambiente virtual, que mostrou-se efetivo diante das requisições. Tal situação foi possível justamente por ser projetado a executar as suas tarefas mais custosas na estação cliente,

liberando os servidores para administrar somente as suas cópias.

Com esse baixo consumo nos servidores e garantias de confiabilidade, integridade e disponibilidade dos arquivos, o uso em ambiente virtual fornece uma saída interessante, técnica que vem crescendo nos últimos anos com o objetivo de consolidar um maior número de servidores lógicos, usando de modo eficiente todos os recursos de servidores físicos.

6.1 Trabalhos futuros

O desenvolvimento do FlexA segue em busca de diversas melhorias, como por exemplo no suporte ao espaço do usuário com o uso do FUSE, no uso de algoritmo mais eficiente para a criptografia dos arquivos, na disponibilidade de novas interfaces de usuário, nas técnicas de compressão de dados e na compatibilidade com dispositivos móveis.

Há também algumas funcionalidades descritas no projeto do FlexA que não foram implantadas:

- Uso do grupo de réplicas para apoio da distribuição dos arquivos e suporte na sobrecarga ou falhas dos servidores primários;
- Condições para ampliar o número de computadores do grupo de escrita, permitindo maior escalabilidade;
- Possibilidade de ajuste do número de conexões simultâneas para os clientes e ajuste do número de divisões por arquivo, permitindo trabalhar com mais servidores simultâneos;
- Automatização do processo de busca e verificação de novos servidores;
- Uso de diretórios e subdiretórios;
- Sincronização das porções entre grupo de escrita e réplicas.

Atualmente o FlexA está envolvido em outros projetos que exploram recursos de processamento paralelo em CUDA (*Compute Unified Device Architecture*). Tais projetos abordam a compressão e criptografia dos arquivos, possibilitando ganhos no processamento dos dados e melhor desempenho no processo de segurança.

Referências

BARRIOS, M. R. et al. *GPFS: A Parallel File System.* [S.l.], 1998. Disponível em: <http://www.redbooks.ibm.com/>.

BATSAKIS, Alexandros.; BURNS, Randal. Cluster delegation: high-performance, fault-tolerant data sharing in nfs. In: *Proceedings of the High Performance Distributed Computing, HPDC-14*. [S.l.]: 14th IEEE International Symposium, 2005. p. 100–109.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. *Distributed systems: concepts and design*. [S.l.]: Addison-Wesley, 2005.

DWORKIN, Morris. *Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. [S.l.], 2001.

EISLER, M. *LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM, RFC 2847*. [S.l.], 2000.

FUJIMURA, A.; OH, S.Y.; 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, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The google file system. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. [S.l.]: ACM, 2003. p. 29–43.

GLUSTER. *GlusterFS*. 2011. Disponível em: <http://www.gluster.org>.

HUANG, H. Howie; GRIMSHAW, Andrew S. Design, implementation and evaluation of a virtual storage system. *Concurr. Comput. : Pract. Exper.*, v. 23, p. 311–331, March 2011.

HUPFELD, F. et al. Xtreemfs - a case for object-based file systems in grids. *Concurrency and Computation: Practice and Experience*, v. 20, 2008.

JOSEFSSON, S. *The Base16, Base32, and Base64 Data Encodings, RFC 4648*. [S.l.], 2006.

KARP, A. H. Enforce pola on processes to control viruses. *Commun. ACM*, v. 46, n. 12, p. 27–29, 2003.

KISTLER, James J.; SATYANARAYANAN, M. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst.*, ACM, v. 10, p. 3–25, 1992.

KOHL, J.; NEUMAN, C. *The Kerberos Network Authentication Service (V5), RFC 4120*. [S.l.], 1993.

KON, Fabio. *Sistemas de arquivos distribuídos*. Dissertação de Mestrado — Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 1994.

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

LAMPORT, Leslie. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, v. 21, p. 558–565, 1978.

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

NELSON, M.; WELCH, B.; OUSTERHOUT, J. Caching in the sprite network file system. *SIGOPS Oper. Syst. Rev.*, ACM, v. 21, p. 3–4, 1987.

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

PLANK, James 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.

REDHAT. *Red Hat Global File System*. 2011. Disponível em: <http://www.redhat.com/software/rha/gfs/>.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, ACM, v. 21, 1978.

SHEPLER, S. *NFS Version 4 Design Considerations, RFC 2624*. [S.l.], 1999.

SHEPLER, S. et al. *Network File System (NFS) version 4 Protocol, RFC 3530*. [S.l.], 2003.

SHVACHKO, K. et al. The hadoop distributed file system. In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. [S.l.: s.n.], 2010.

TANENBAUM, Andrew S.; STEEN, Maarten Van. *Distributed systems: principles and paradigms*. [S.l.]: Pearson Prentice Hall, 2007.

WEIL, Sage A. et al. Ceph: a scalable, high-performance distributed file system. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. [S.l.: s.n.], 2006. (OSDI '06), p. 307–320.

WILCOX-O'HEARN, Zooko; WARNER, Brian. 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 FlexA

Quanto a instalação, o FlexA não necessita de recursos especiais, ou qualquer alteração no sistema operacional. Para a implantação é necessário ter instalado o interpretador Python 2.7, o qual vem por padrão nas distribuições Linux, o pacote *netifaces* para administração da interface de rede do computador e o pacote *pycrypto* que é o responsável em fornecer um conjunto de algoritmos de criptografia.

Após a instalação das dependências, necessita-se definir no *módulo comunicador* a pasta que será utilizada como *cache* das porções ou através do arquivo de configuração (*config.dat*) que é criado utilizando o comando *python com.py -r* do *módulo comunicador*.

Em seguida deve-se iniciar o *módulo coletor* dos servidores e clientes através do *python coletor.py*. Com o *módulo coletor* ativo em todas as estações os novos usuários precisam identificar os servidores através da busca pelo *módulo comunicador* com o comando *python com.py -b*. Esse comando identifica a rede e a faixa de IPs que o FlexA poderá trabalhar.

Feito isso, o FlexA já estará pronto para as requisições dos usuários, as quais são realizadas através do módulo *flexa.py [opção] <arquivo>* seguido da opção que se deseja (*put*, *get*, *list*, *delete* ou *new permission*) sobre um arquivo.