RENATO DULIZIO MARTINS

AVALIAÇÃO DE DESEMPENHO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

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

São José do Rio Preto
2011

RENATO DULIZIO MARTINS

AVALIAÇÃO DE DESEMPENHO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

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

Orientadora:
Profa. Dra. Renata Spolon Lobato

São José do Rio Preto
2011

RENATO DULIZIO MARTINS

AVALIAÇÃO DE DESEMPENHO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

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

Profa. Dra. Renata Spolon Lobato

Renato Dulizio Martins

Banca Examinadora:
Prof. Dr. Aledir Silveira Pereira
Prof. Dr. Norian Marranghello

São José do Rio Preto
2011

Em memória do meu querido pai Mário Martins.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela vida que me deu e por todas as oportunidades que me tem dado.

À minha mãe Jesuína, pela forte guerreira que sempre foi, enfrentando tanta dificuldade nessa vida e sempre fez de tudo para que não faltasse nada a mim e aos meus irmãos.

Ao meu irmão Rodrigo, fonte maior de inspiração, que me ajudou muito ao longo de todo o curso, e à minha irmã Rafaela, um anjo de Deus enviado

À minha orientadora Profa. Dra. Renata Spolon Lobato, pelo respaldo fornecido ao longo dos dois anos e meio de orientação e muito aprendizado.

À minha namorada Amanda, pela compreensão, apoio, força e companheirismo dedicados a mim ao longo desse ano difícil que se passa.

A todos meus amigos e familiares que sempre me deram apoio para estudar e crescer na vida.

A todos os professores do DCCE que contribuíram para a minha formação acadêmica.

E finalmente, a todo pessoal do GSPD, em especial aos orientadores Prof. Dr. Aleardo Manacero Jr. e novamente à Profa. Dra. Renata Spolon Lobato.

RESUMO

Este trabalho tem por finalidade estudar formas para avaliar o desempenho de sistemas de arquivos distribuídos, e a montagem de um cenário para a avaliação. É feita uma análise comparativa entre vários, dentre eles o NFS (*Network File System*), GFS (*Google File System*), o Tahoe-LAFS (Tahoe – *Least Authority File System*) e um sistema de arquivos distribuídos em desenvolvimento no Grupo de Sistemas Paralelos e Distribuídos (GSPD). Como resultado, todas as etapas da análise, desenvolvimento e testes, bem como gráficos e tabelas ilustrativas que servirão como base para a análise comparativa entre os mesmos, serão apresentados na monografia.

ABSTRACT

This work has the objective to evaluate the performance of distributed file systems, developing an evaluation scenario. It made a comparative analysis between the NFS (Network File System), GFS (Google File System), Tahoe-LAFS (Tahoe – Least Authority File System) and a distributed file system under development at the Grupo de Sistemas Paralelos e Distribuidos (GSPD). As results, every element of analysis, development and testing, as well illustrative graphics and tables will serve as a basis for comparative analysis between them, will be presented in this thesis.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATUAS E SIGLAS

ACL: *Access Control List*

AES: *Advanced Encryption Standard*

AFS: *Andrew File System*

CPU: *Central Processing Unit*

DCCE: Departamento de Ciências de Computação e Estatística

DNS: *Domain Name Service*

E/S: Entrada/Saída

FEC: *Forward Error Correction*

FUSE: *Filesystem Userspace*

GB: *Gigabyte*

GFS: *Google File System*

GHz: *Gigahertz*

GSPD: Grupo de Sistemas Paralelos e Distribuídos

HDFS: *Hadoop Distributed File System*

HTTP: *Hyper Text Transfer Protocol*

IBILCE: Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto

IP: *Internet Protocol*

MB: *Megabyte*

NFS: *Network File System*

P2P: *peer-to-peer*

PC: *Personal Computer*

POLA: *Principle of Least Authority*

POSIX: *Portable Operating System Interface for Unix*

RAM: *Random Access Memory*

RFC: *Request For Comments*

RPC: *Remote Procedure Call*

RPM: Rotações por minuto

RSA: *Rivest, Shamir and Adleman*

Tahoe-LAFS: *Tahoe - Least Authority File System*

TCP: *Transfer Control Protocol*

UDP: *User Datagram Protocol*

URL: *Uniform Resource Locator*

VFS: *Virtual File System*

WAN: *Wide Area Network*

Capítulo 1 – Introdução

1.1 Motivação

Embora existam vários sistemas de arquivos distribuídos que satisfaçam os requisitos de um ambiente distribuído, tais como transparência, desempenho, escalabilidade, controle de concorrência, tolerância a falhas e segurança, o resultado final do cumprimento destas características é um conjunto bastante complexo, mas bastante desejável de se ter em um sistema de arquivos distribuído. Porém, conforme se eleva a complexidade de uma dessas características, as outras são afetadas negativamente.

Devido a este fato, existem diversos sistemas de arquivos distribuídos, sendo que cada um deles procura priorizar uma determinada característica para sua otimização em um cenário específico. Neste contexto está inserida a necessidade da avaliação de desempenho, fazendo-se uma análise de resultados obtidos relacionada às suas características.

1.2 Objetivos

Este trabalho tem como objetivo investigar maneiras para se avaliar sistemas de arquivos distribuídos, adotando a utilização de métodos de aferição para efetuar tal tarefa. Será feita a implantação e avaliação de diversos sistemas de arquivos distribuídos, tais como o NFS (*Network File System*) [Shepler et al. 2000] [Shepler et al. 2003], Tahoe-LAFS (*Tahoe – Least Authority File System*) [Wilcox-O'Hearn 2008], o GFS (*Google File System*) [Ghemwat 2003] através de uma implementação de código livre da sua descrição, o HDFS (*Hadoop Distributed File System*)

[Shvachko et al. 2010], e ainda o sistema de arquivos distribuídos descrito por [Fernandes 2011], que está em fase de desenvolvimento no Grupo de Sistemas Paralelos e Distribuídos (GSPD).

Em seguida, utilizar os resultados obtidos durante a avaliação para fazer uma análise comparativa entre as características de cada sistema de arquivo distribuído avaliado, embasado nos conceitos estudados na revisão bibliográfica.

1.3 Organização do texto

O capítulo dois apresenta uma revisão bibliográfica, introduzindo conceitos de sistemas distribuídos e sistemas de arquivos distribuídos, bem como um estudo sobre alguns sistemas de arquivos distribuídos específicos e maneiras para se efetuar a avaliação de desempenho.

O capítulo três descreve detalhadamente os cenários de avaliação montados para a presente avaliação, bem como o ambiente computacional onde estes foram inseridos.

O capítulo quatro traz os resultados detalhados obtidos na avaliação, e também uma análise individual e comparativa entre os mesmos.

Após a análise dos resultados e apoiado nos conceitos e características estudadas, referidas no capítulo dois, serão apresentadas as devidas conclusões, contribuições, dificuldades encontradas durante o estudo e direcionamento para possíveis trabalhos futuros formando assim o capitulo cinco.

Capítulo 2 – Revisão bibliográfica

2.1 Sistemas distribuídos

Existem diversas maneiras diferentes de se expressar o que é um sistema distribuído, dentre elas podemos destacar três mais relevantes.

A definição de [Lamport 1978] diz que: “Um Sistema Distribuído consiste de uma coleção de processos distintos que são especialmente separados e que se comunicam por troca de mensagens”.

Já em [Coulouris et al. 2005], é especificado como: “Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas por troca de mensagens”

E por fim, [Tanembaum and Steen 2007] define como: “Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente”.

A construção de sistemas distribuídos vem da motivação em compartilhar recursos computacionais de *hardware, software* e dados, o que envolve alguns problemas como concorrência, inexistência de um relógio global, falhas independentes, dentre outras mais [Coulouris et al. 2005].

Vários conceitos importantes devem ser observados para a construção de um sistema distribuído, como a transparência na distribuição em seus diferentes níveis, fornecer acesso fácil aos recursos do sistema, o tratamento eficiente das falhas no sistema quando estas acontecerem, e permitir que o sistema possa ser expandido [Tanembaum and Steen 2007], [Coulouris et al. 2005].

Quanto às arquiteturas de sistemas distribuídos, elas podem ser divididas em três categorias: arquiteturas centralizadas, descentralizadas e híbridas.

Arquitetura centralizada: Expressa o conceito de que clientes solicitam serviços oferecidos por servidores, também chamada de arquitetura cliente-servidor. Esse tipo de arquitetura é menos complexa e uma das mais utilizadas para compor um cenário de sistema distribuído. Nesse modelo, um servidor pode desempenhar papel de cliente em relação a outros servidores, conforme ilustrado na Figura 2.1.

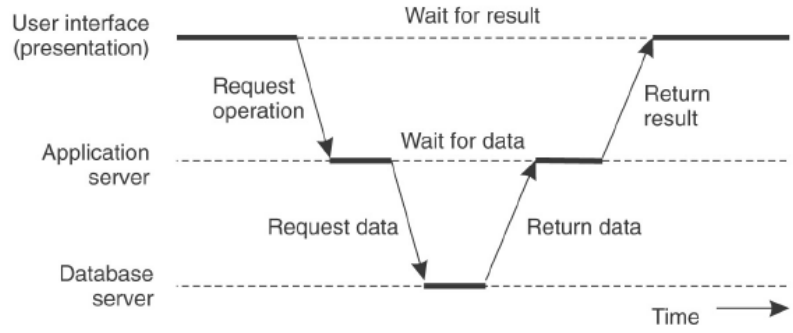

Figura 2.1: Arquitetura cliente-servidor [Tanenbaum and Steen 2007]

Arquiteturas descentralizadas: Neste modelo, o exemplo mais conhecido é o *peer-to-peer* (P2P). No modelo descentralizado, não há distinção entre os processos como ocorre no modelo cliente-servidor, ou seja, todos os envolvidos possuem igual importância para o sistema, fornecendo um serviço uniforme [Kshemkalyani and Singhal 2008].

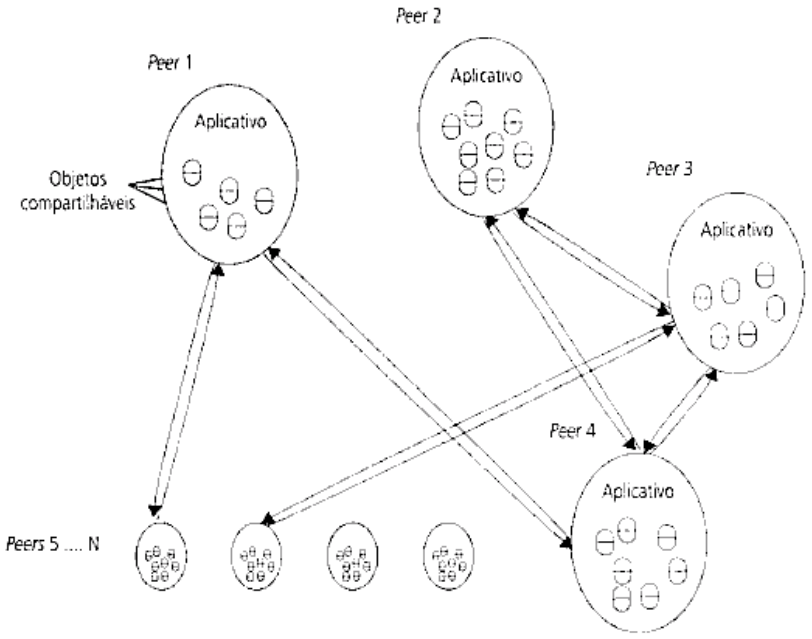

Figura 2.2: Arquitetura P2P [Coulouris et al. 2005]

Arquiteturas híbridas: É uma mescla das arquiteturas centralizada e descentralizada. Essa arquitetura é mais utilizada para dar início a um processo, para que em seguida a aplicação consiga se estabelecer com outras partes daquela aplicação, como por exemplo nas redes *torrent.* Tais redes consistem de usuários solicitando informações a um servidor sobre quais clientes possuem um determinado arquivo. Ao final, a transferência desse arquivo ocorre sobre a arquitetura P2P com os outros clientes, como ilustrado na figura 2.3 [Tanenbaum and Steen 2007].

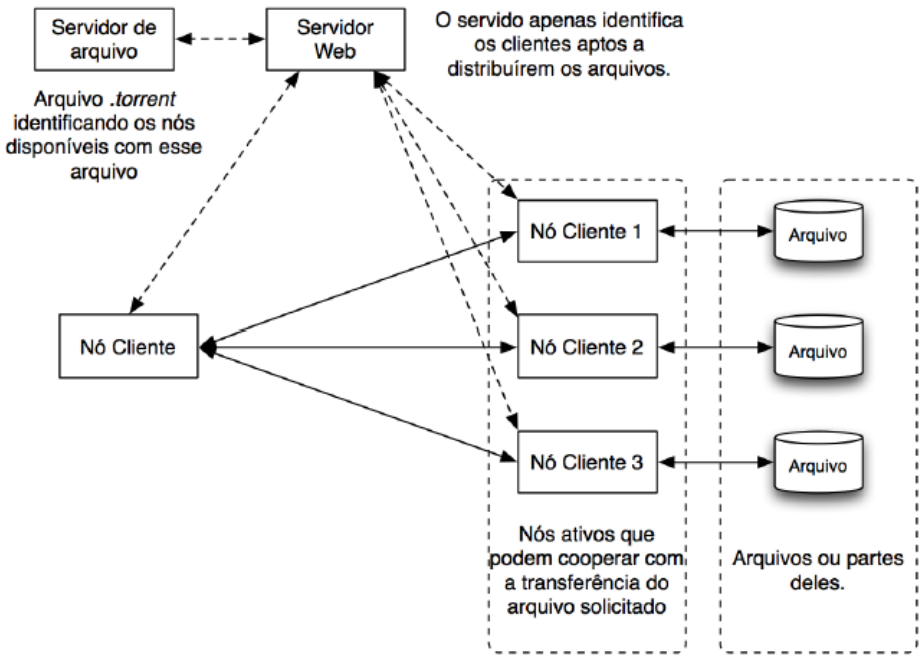

Figura 2.3: Funcionamento da estrutura *torrent*

Partindo da necessidade de compartilhamento de recursos, pode-se citar os sistemas de arquivos distribuídos, que tem como finalidade compartilhar arquivos em um sistema distribuído de maneira eficiente e transparente, de forma que para o usuário seja semelhante ao uso de um sistema de arquivos local.

2.2 Sistemas de arquivos distribuídos

Um sistema de arquivos distribuído deve atender a vários requisitos básicos de transparência existente em um sistema distribuído, como transparências de acesso, localização, mobilidade, desempenho, escalabilidade. Além disso, requisitos como controle de atualizações concorrente de arquivos, replicação de arquivos, tolerância a

falhas, manter consistência e segurança dos dados, atender aos diferentes tipos de *hardware* e *software* presentes no sistema distribuído, e ainda obter eficiência equivalente à de um sistema de arquivos local [Coulouris et al. 2005].

Os problemas encontrados para se projetar sistemas de arquivos distribuídos são de diversas naturezas, e conforme visto em [Tanenbaum and Steen 2007] pode-se destacar como as principais dificuldades:

- No cliente, como fazer um uso eficiente de *cache* para se conseguir resultados equiparáveis aos obtidos em um sistema local, além da manutenção da consistência dessas cópias mantidas em *cache* por mais de um cliente, quando ocorre uma atualização;
- A recuperação de falhas tanto nos clientes como nos servidores;
- Necessidade de alto desempenho na rede, seja para operações tanto de leitura ou escrita nos servidores e clientes;
- Dificuldades na mudança de escala.

Embora existam vários sistemas de arquivos distribuídos que agregam essas necessidades básicas de um ambiente, a união de todas essas características torna o projeto bastante complexo, pois a melhoria no nível de atendimento de alguma característica afeta negativamente nas outras. Nesse sentido, existem quantidades expressivas de sistemas de arquivos distribuídos atualmente, cada uma otimizando um requisito para um tipo de atividade mais específica.

Nesse contexto, serão apresentados neste trabalho quatro sistemas de arquivos distribuídos: o *Network File System* (NFS), o *Tahoe Least Authority File System* (Tahoe-LAFS), o *Google File System* (GFS), usando o *Hadoop Distributed File System* (HDFS) como uma implementação da especificação feita pelo GFS, e ainda um sistema de arquivos distribuídos em desenvolvimento no GSPD.

2.3 *Network File System* (NFS)

Considerado uns dos principais sistemas de arquivos distribuídos, o NFS foi projetado em 1984 pela Sun Microsystems, com o objetivo de atender algumas das características citadas na seção 2.2, principalmente no que diz respeito à transparência no acesso aos arquivos. A sua arquitetura está ilustrada na Figura 2.4 [Coulouris et al 2005].

2.3.1 Arquitetura

Na interface cliente, a transparência do NFS ocorre através de um sistema de arquivos virtual (*Virtual File System* - VFS), que consiste em atribuir as requisições ao módulo de sistemas de arquivos apropriado, agindo como um *middleware* – camada de *software* que faz intermédio na comunicação entre outras aplicações – para as aplicações do usuário. Isso possibilita trabalhar com o sistema de arquivo local, com o módulo do NFS ou outro sistema de arquivos [Coulouris et al. 2005].

O serviço de arquivo remoto do NFS emprega o uso de um modelo de acesso remoto, ilustrado na Figura 2.4, que realiza as operações sobre uma interface que contém as várias operações disponíveis para a aplicação local. Contudo, o servidor é o responsável em aplicar tais operações [Tanenbaum and Steen 2007].

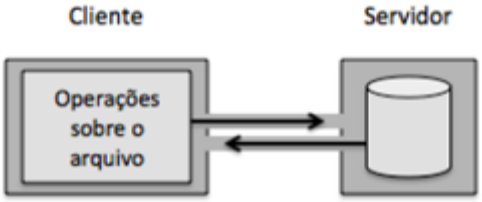

Figura 2.4: Modelo de acesso remoto

No lado do servidor, o NFS foi projetado para funcionar sem manter o estado do sistema, ou seja, o servidor não mantém quais são os arquivos abertos pelos clientes. Isso simplifica todo o processo, pois, caso ocorra uma queda do servidor, não será necessário recuperar as informações. No lado do cliente não será necessário conhecer a atual situação do servidor, apenas que ele continue enviando as solicitações dos dados até que o servidor responda [Coulouris et al. 2005].

Na sua quarta versão, além do suporte para trabalhar sem informações de estado, existe a alternativa de trabalhar com operações que mantêm 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 (*locks*) nativamente para o bloqueio de blocos de arquivos e a realização de operações atômicas. Outra característica foi a substituição do protocolo de transporte UDP (*User Datagram Protocol*) pelo TCP (*Transfer Control Protocol*) na comunicação entre servidores e clientes, possibilitando confirmar o recebimento das mensagens [Shepler et al. 2000],[Shepler et al. 2003],[Batsakis and Burns 2005].

A organização do espaço de nomes procede da utilização da hierarquia de diretórios UNIX através do 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 eles são montados utilizando o comando *mount*. [Coulouris et al 2005].

Para facilitar o processo de montagem dos diretórios, foi acrescentado ao cliente NFS o recurso *automounter*, que armazena os caminhos dos servidores. Quando o usuário busca resolver um determinado caminho de acesso a um arquivo no servidor, 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 [Coulouris et al 2005].

Uma visão geral da arquitetura do NFS está ilustrada na Figura 2.5.

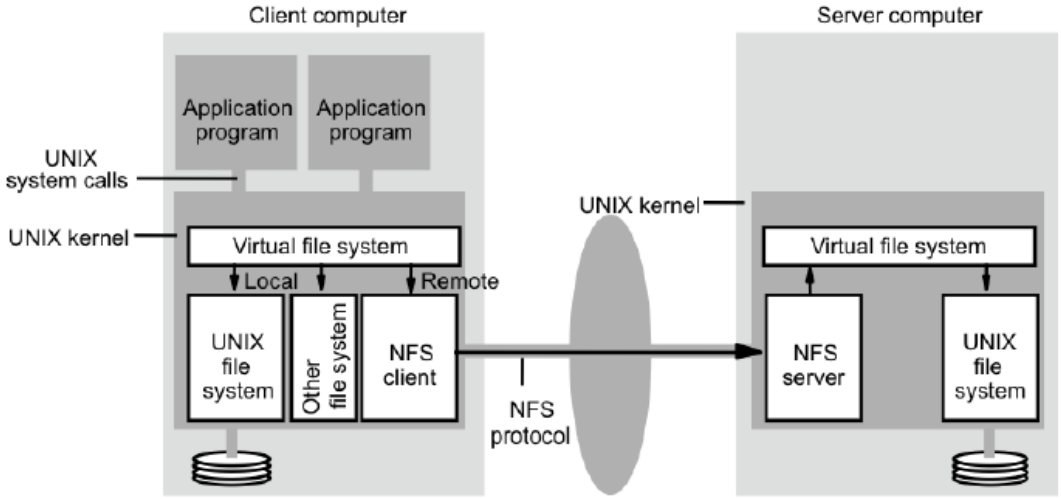

Figura 2.5: Arquitetura do NFS [Tanenbaum and Steen 2007]

2.3.2 *Cache*

No lado do servidor, *cache* armazena blocos de arquivos, diretórios e atributos de arquivos que foram lidos do disco. Uma forma de agregar maior agilidade à *cache* é através do método *read-ahead*, que antecipa quais serão os próximos blocos a serem lidos e já procura carregá-los na memória.

Há também a utilização da técnica *delayed-write*, o qual consiste em adiar a escrita dos dados nos servidores. A gravação ocorre somente após um período de tempo sem alterações pelo cliente, evitando constantes acessos de escrita a uma curta fração de segundos ao mesmo arquivo. E por fim, a operação *sync* que copia para o disco os blocos alterados a cada 30 segundos [Kon 1994] [Coulouris et al 2005].

No cliente, a *cache* é usada para armazenar suas operações e com isso

diminuir os acessos aos servidores. Entretanto, isso pode causar problemas de consistência nas *caches* dos demais arquivos distribuídos por outros clientes. Para isso, antes dele utilizar os dados que estão em sua *cache*, ele busca no servidor, através de um processo baseado em verificação de *timestamps*, que atualiza os arquivos da sua *cache* para a versão mais atual.

2.3.3 Segurança

Os esforços para proporcionar segurança ocorreram através de um canal de comunicação seguro entre cliente e servidor. Para isso, ele fornece suporte ao RPCSEC_GSS [Eisler et al. 1997] que expressa uma grande quantidade de mecanismos de segurança no auxílio à proteção dos canais de comunicação e suporte a integridade e confidencialidade das mensagens. O RPCSEC_GSS atua como uma camada sobre as interfaces de segurança que, entre elas, trás o suporte ao Kerberos v5 [Kohl 1993] e ao método de chaves públicas conhecido como Lipkey [Eisler 2000].

O desenvolvimento do NFS seguiu o critério de não utilizar mecanismos de segurança próprio, em vez disso, optou em apenas padronizar o modo para a manipulação da segurança. Com isso novos mecanismos de segurança podem ser incorporados ao NFS. Enquanto isso, o controle de acesso ocorre através da lista de controle de acesso (*Access Control List* - ACL) que especifica as permissões para o usuário ou grupo [Tanenbaum and Steen 2007]. A Figura 2.6 exibe uma visão geral da segurança no NFS.

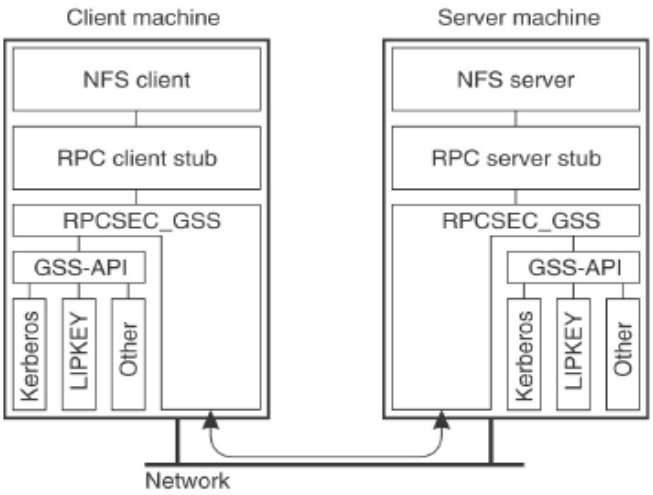

Figura 2.6: Segurança NFS

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

O Tahoe-LAFS, ou simplesmente Tahoe, é um sistema de arquivos distribuído utilizado inicialmente para serviços de *backup*, que executa no espaço de processos do usuário [Wilcox-O'Hearn 2008].

Um sistema de arquivos Tahoe é constituído de três elementos básicos: servidores de armazenamento (*storage nodes*), um componente central chamado *introducer*, e os clientes. Os *storage nodes* são os nós de armazenamento de dados do sistema. O *introducer* é uma espécie de nó mestre, onde as informações de conexão de todos os *storage nodes* são mantidas. Dessa forma, os clientes que desejam acessar o sistema de arquivos se conectam aos nós de armazenamento através das informações obtidas junto ao *introducer*.

Uma visão geral da arquitetura de processos de um sistema de arquivos Tahoe se comunicando é ilustrada na Figura 2.7:

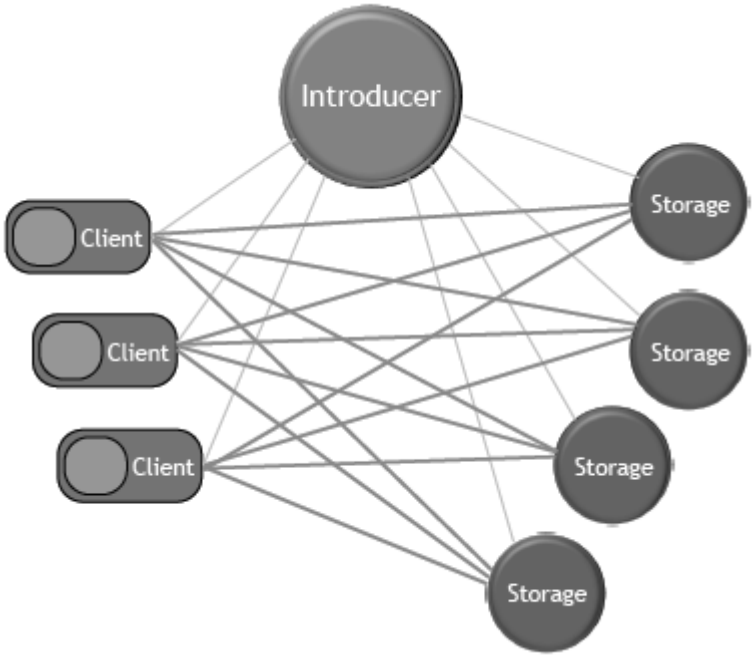

Figura 2.7: Interação entre processos Tahoe-LAFS

2.4.1 Controle de acesso

O Tahoe utiliza controle de acesso baseado em capacidades para gerenciar o acesso a arquivos e diretórios. Um *capability* (capacidade) é uma pequena *string* de *bits* que identifica unicamente um arquivo ou diretório. Essa *string* deve ser pequena o suficiente para ser facilmente armazenada e transmitida, porém deve ser grande o

bastante para que ela seja única.

Existem dois tipos de arquivos: mutáveis e imutáveis. Os arquivos mutáveis podem ter seu conteúdo alterado, enquanto os imutáveis são escritos uma vez e não podem mais ser alterados, apenas acessados para leitura.

Cada arquivo imutável possui duas capacidades associadas: o *read capability*, (*read-cap*), que garante o acesso de leitura do arquivo, e o *verify capability* (*verify-cap*), que é utilizado para verificar a integridade do conteúdo do arquivo.

Para arquivos mutáveis, existem três tipos de capacidades: o *read-write-cap*, o *read-only-cap*, e também o *verify-cap*. Analogamente ao caso anterior, o *read-write-cap* garante acesso de leitura e escrita do arquivo, o *read-only-cap* garante apenas acesso de leitura, enquanto o *verify-cap* é utilizado para verificar a sua integridade.

O *capability* associado a um arquivo é chamado *filecap*, enquanto o associado a um diretório é dito *dircap.* Então, para garantir o acesso a um determinado arquivo ou diretório, basta que o usuário possua o seu respectivo *filecap* (ou *dircap*), pois dessa forma ele garante a capacidade de realizar tal acesso.

2.4.2 Criptografia

No Tahoe, são adotadas diferentes técnicas de criptografia para garantir a integridade e consistência dos arquivos armazenados. Para arquivos imutáveis, a criptografia é aplicada com a utilização de chaves simétricas AES (*Advanced Encryption Standard*)

Cada arquivo mutável está associado com um par de chaves assimétricas RSA (*Rivest, Shamir and Adleman*) único, onde usuários que possuem permissão de escrita utilizam a chave privada para poder assinar digitalmente novas versões daquele arquivo. Como uma chave de criptografia RSA é consideravelmente grande para ser manipulada (2048 *bits*), são utilizadas técnicas para criar uma chave reduzida única a partir da chave RSA, que relacione as duas de maneira única.

Diretórios são representados por tabelas que são formadas por uma coluna com o nome dos arquivos, e outras duas colunas, uma para os *read-only-cap* e outra para *read-write-cap*. Aqui é aplicada uma propriedade chamada *transitivy read-only*, onde usuários que possuem o acesso de leitura-escrita do diretório podem obter o *read-write-cap* dos arquivos presentes no diretório, mas quem possuir acesso

somente de leitura deve poder conseguir apenas acesso aos *read-only-cap* dos arquivos contidos no diretório.

Todo o conteúdo da tabela do diretório é então criptografado da mesma forma que um arquivo mutável, mas além disso a coluna dos *read-write-cap* é criptografada separadamente, antes de ser armazenada.

2.4.3 Codificação

No Tahoe, todos os dados, após serem criptografados, são codificados utilizando um algoritmo de *erasue coding*, (ou FEC – *Forward Error Correction*), que é um método para correção de erros na transmissão de dados [Fujimura et al. 2008]. O algoritmo utilizado é o *Reed-Salomon* [Plank et al. 2009], que utiliza dois parâmetros: N, que é o número total de *shares* – porção de dados dos arquivos criada durante a codificação – a serem gerados, e K, o número de *shares* que serão necessários para a leitura do arquivo. Os valores desses parâmetros devem pertencer aos intervalos 1<=N<=256, e 1<=K<=N.

Como padrão, o Tahoe utiliza os valores de N=10 e K=3, ou seja, para cada arquivo a ser escrito, são criadas dez porções de dados do arquivo que são armazenados em diferentes servidores de armazenamento, sendo que quaisquer três porções juntas são capazes de remontar o arquivo original.

2.5 *Google File System* (GFS)

O GFS é um sistema que foi projetado e implementado para uso interno no Google, para ambientes que lidam com milhares de arquivos muito extensos – que podem variar de algumas centenas de *megabytes* a vários *gigabytes* – e onde operações de leitura ou de adição (*append*) de arquivos são predominantes [Ghemwat 2003].

Foram considerados certos aspectos mais proeminentes durante a análise da operação dos servidores pertencentes à Google Inc. São esses:

Falhas no sistema: Considerando que o *hardware* utilizado pela empresa é composto em sua maioria de sistemas de baixo custo (i.e.: computadores comuns, como PCs), falhas em componentes individuais do sistema acontecem em uma taxa relativamente alta e um tanto quanto regular, porém são considerados naturais, e de forma alguma inesperados. Com isso em mente, o sistema deve ser largamente capaz

de identificar qualquer falha (auto-monitoramento) e se recuperar de forma automática, a qualquer instante.

Tamanho dos arquivos: A maioria dos arquivos no sistema tem um tamanho que varia de algumas centenas de *megabytes* a alguns *gigabytes*, sendo mais comuns arquivos com vários *gigabytes*. Arquivos pequenos são suportados, porém, o GFS não é otimizado para esses casos de predominância de arquivos de tamanho reduzido.

Leitura de arquivos: A leitura no GFS pode ser dividida em dois tipos: o primeiro tipo consiste na leitura de fluxos de dados maiores, geralmente múltiplos *megabytes*, sendo que se o mesmo cliente solicita mais de uma leitura sucessiva, os dados usualmente estão armazenados de forma contínua no disco. O segundo tipo engloba leituras menores (alguns poucos *kilobytes*) e em regiões aleatórias. Pressupõe-se que aplicações levem esses aspectos em consideração e organizem suas leituras menores para que sejam efetuadas em seqüência e em lote.

Escrita de arquivos: As escritas geralmente consistem em grandes fluxos de dados (aproximadamente do mesmo tamanho das leituras do primeiro tipo) que são adicionadas a arquivos (*appends*). Após isso, os arquivos raramente são modificados. Também pode haver escritas de pequenos trechos arbitrários em um determinado arquivo.

Concorrência: O sistema deve ser capaz de fornecer uma semântica bem definida quanto às operações de escrita realizadas de forma concorrente por múltiplos clientes no mesmo arquivo. Isso implica na essencialidade de se fornecerem operações de escrita atômicas, com o mínimo possível de *overhead*.

Utilização da rede: Largura de banda e tempo de disponibilidade da conexão são preferidos a baixas latências. A maioria das aplicações que executam sobre o sistema costumam priorizar o processamento de grandes blocos de dados a altas taxas ao invés do tempo de resposta reduzido.

2.5.1 Arquitetura

O GFS oferece uma *interface* considerada familiar, apesar de não exportar uma API padrão (como POSIX, por exemplo): os arquivos são armazenados de forma hierárquica, são acessados pelo uso de caminhos (*pathnames*), e podem ser objetos de operações bem conhecidas como criação, exclusão, abertura, fechamento, leitura e escrita.

***Chunks*:** São blocos de dados com 64MB de tamanho que armazenam o conteúdo dos arquivos existentes no sistema. Cada *chunk* é armazenado no *chunkserver* como um arquivo comum em um sistema de arquivos *UNIX*. Cada *chunk* é identificado por um valor único de 64 *bits* denominado *chunk handle*. Os *chunks* são replicados em vários *chunkservers*: o comportamento padrão é manter três cópias ativas de cada *chunk* a qualquer momento, porém o nível de replicação de um *chunk* em particular pode ser alterado por um cliente.

Metadados: São informações sobre os arquivos e os *chunks*, e ficam armazenados permanentemente na memória do mestre. São três tipos de metadados: a estrutura de arquivos e de *chunks*, o mapeamento de arquivos para *chunks* e a localização de todas as cópias de cada *chunk*. Os dois primeiros tipos são armazenados também de forma persistente – em um registro de operações no disco do mestre e de espelhos remotos. As informações sobre localização de *chunks* são obtidas pelo mestre que as solicita de cada *chunkserver* sempre que eles se juntam ao sistema ou o mestre é reiniciado.

Armazenar todos os metadados na memória do mestre aumenta seu desempenho ao responder requisições de clientes. Além disso, permite que o mestre possa efetuar uma checagem de seu estado em segundo plano, fundamental para as atividades de *garbage collection*, replicação de *chunks* e balanceamento de carga e uso de disco entre os *chunkservers*. O consumo de memória não é fator preocupante, pois cada *chunk* pode ser representado usando menos de 64 *bytes* de dados.

O mestre não pode manter todo seu estado apenas na memória, pois uma falha do mestre destruiria todo o sistema de arquivos de forma irrecuperável. Por este motivo, o mestre mantém um registro local de todas as mudanças efetuadas nos metadados, que é chamado de registro de operações (*operation log*). Além disso é possível utilizar espelhos para armazenar uma cópia do registro de operações localmente, para prover a redundância necessária em caso de falha do mestre.

Um tipo de metadado que nunca é armazenado de forma persistente consiste na localização dos *chunks* nos vários *chunkservers*. Para isso, o mestre se comunica periodicamente com os *chunkservers* através de mensagens de controle *HeartBeat*, e cada *chunkserver* envia uma lista dos *chunks* que detém ao conectar-se ao sistema.

Um sistema GFS é composto de um único mestre (*master*), vários *chunkservers* e vários clientes, como ilustrado na Figura 2.8.

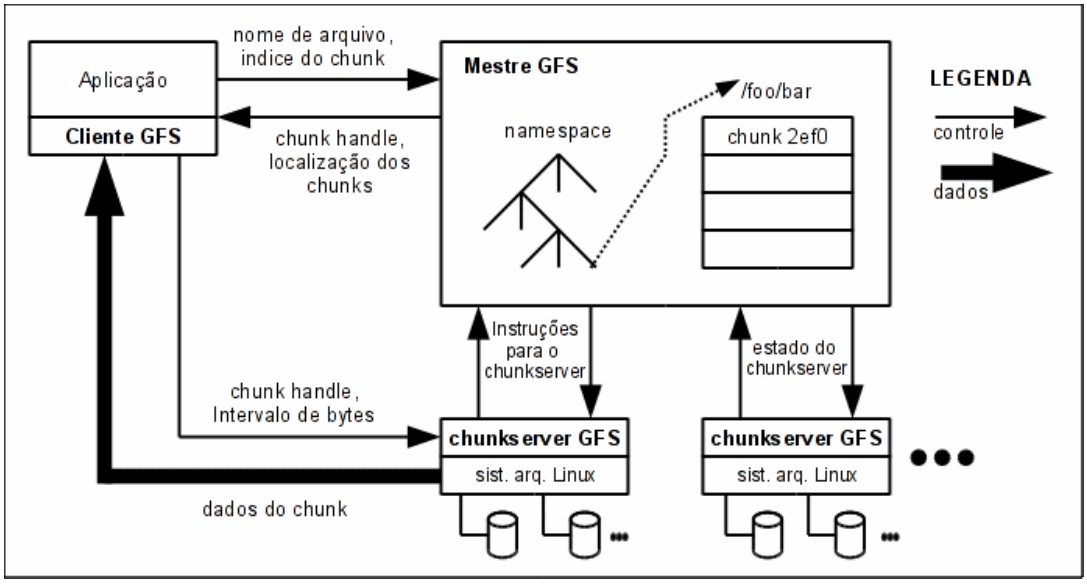

Figura 2.8: Visão geral do GFS [Ghemwat 2003]

Arquivos são divididos em blocos de tamanho fixo, denominados *chunks*. O mestre coordena as atividades de todo o sistema: armazena os metadados dos arquivos, controla a distribuição de concessões de escrita, solicitações de leitura, efetua *garbage collection*, monitora e controla os *chunkservers* por meio de mensagens *HeartBeat*. Um *chunkserver* é responsável por armazenar os *chunks*, além de efetuar as leituras e escritas requeridas pelos clientes. Um mesmo *chunk* é armazenado de forma redundante em múltiplos *chunkservers*.

Modelo de consistência: Garante que as mudanças no sistema de arquivos sejam efetuadas de forma atômica e que a consistência seja mantida de forma geral. Todas as mudanças efetuadas no sistema de arquivos são tratadas exclusivamente pelo mestre, que implementa um sistema de bloqueio na estrutura do sistema de arquivos para garantir que todas as operações efetuadas sejam atômicas e, conseqüentemente, o sistema permaneça sempre consistente.

Os tipos de modificações possíveis no sistema de arquivos são: *writes* e *record appends*. Os *record appends* possuem ainda a variação *automatic record append, que é* um tipo de mutação especial otimizada para escritas concorrentes. Maiores detalhes sobre essas operações podem ser encontrados em [Ghemwat 2003].

Integridade dos dados: Cada *chunkserver* armazena um *checksum* de seus *chunks*, e o utiliza para detectar a presença de *chunks* corrompidos em seus discos. Isso é feito dividindo-se cada *chunk* em blocos de 64 *kilobytes*, com cada bloco associado a um *checksum* de 32 *bits*. Sempre que o *chunkserver* atende uma

requisição de leitura, este irá verificar o *checksum* de todos os blocos solicitados. Caso seja detectada alguma corrupção, o solicitante será informado do problema, e será instruído a repetir a requisição a outro *chunkserver.* O mestre então irá enviar uma cópia válida dos blocos corrompidos para o *chunkserver* que encontrou o problema, que depois de receber e armazenar a nova cópia, irá excluir a cópia com problema.

2.5.2 Funcionamento do GFS

A implementação do GFS visa envolver o mestre o mínimo em qualquer operação efetuada pelos clientes. Um dos conceitos fundamentais na operação do sistema é chamado concessão de acesso (*chunk lease*), e faz com que os clientes e os *chunkservers* possam se comunicar sem a necessidade de interferências constantes do mestre.

O processo de concessão acontece da seguinte forma, ilustrado pela Figura 2.9.

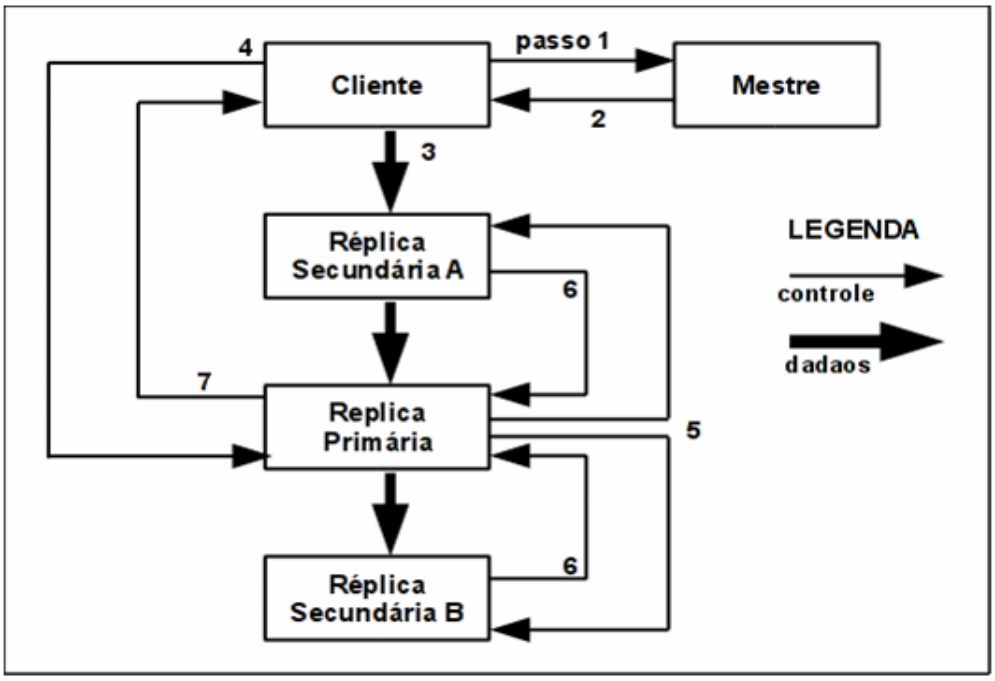

Figura 2.9: Sistema de concessão do GFS [Ghemwat 2003]

1) O cliente envia uma requisição ao mestre para descobrir qual o atual primário para um *chunk* determinado, e a localização de suas outras cópias.

2) O mestre responde com o nome do primário e dos outros *chunkservers* que possuem as cópias do *chunk* solicitado.

3) O cliente envia os dados da mutação para todos os *chunkservers* envolvidos, numa ordem arbitrária. Os *chunkservers* irão armazenar os dados em um *cache* até que sejam usados ou percam a validade.

4) Após receber a confirmação de recebimento de todos os *chunkservers*, o cliente envia um pedido de escrita para o primário, contendo também a identificação dos dados enviados anteriormente. O primário ordena e numera as mutações para as quais recebe requisição (inclusive de clientes diferentes), e então aplica as mudanças em sua cópia local dos dados, na ordem determinada.

5) O primário encaminha o pedido de escrita para todos os outros *chunkservers* que contêm as cópias do *chunk* a ser modificado (*chunkservers* secundários). Os secundários aplicam as mudanças obedecendo à ordem definida pelo primário.

6) Os secundários, ao terminarem as modificações, respondem ao primário sinalizando que acabaram.

7) O primário responde ao cliente. Quaisquer eventuais erros ocorridos durante as modificações nos secundários são comunicados ao cliente. Caso haja um erro no primário, a requisição não é enviada aos secundários (nem ordenada e numerada). Em qualquer caso, a região é marcada como inconsistente, e o cliente tenta novamente as etapas de 3) a 7) por algumas vezes, e caso ainda encontre erros, recomeça todo o processo, partindo de 1).

2.5.3 *Hadoop Distributed File System* (HDFS)

O *Hadoop Distributed File System* (HDFS) [Shvachko et al. 2010] é um sistema de arquivos distribuídos de código aberto, parte do projeto Apache Hadoop [Hadoop 2011], implementado com base nas especificações do GFS.

Algumas modificações na nomenclatura dos servidores podem ser verificadas: o servidor mestre e os *chunkservers* do GFS são nomeados no HDFS como *NameNode* e *DataNodes*, respectivamente, porém eles possuem as mesmas funcionalidades.

2.6 Sistema de Arquivos Distribuído no Espaço do Usuário

Trata-se de um sistema de arquivos distribuídos em fase de desenvolvimento [Fernandes 2011] no Grupo de Sistemas Paralelos e Distribuídos (GSPD), que busca

aliar tecnologias de sistemas de arquivos distribuídos existentes para formar um modelo funcional compatível com um ambiente controlado. O modelo proposto toma como base características do Tahoe-LAFS (controle de capacidades), do GFS (administração centralizada das informações), do AFS (uso eficiente de *cache* no cliente) e ainda do NFS (apoio da camada de abstração).

A arquitetura do modelo proposto pode ser vista na Figura 2.10.

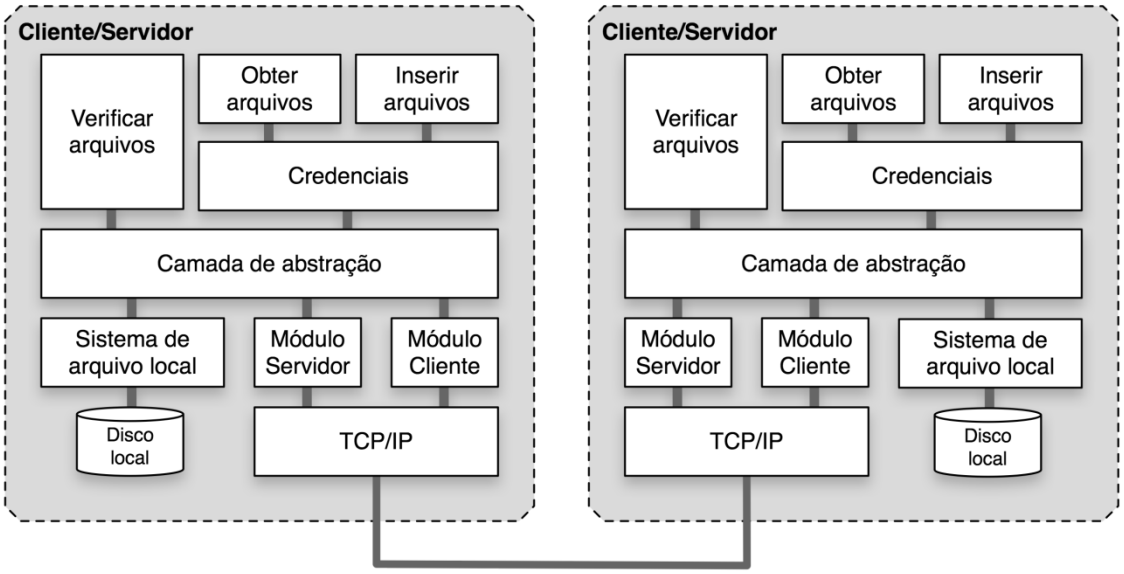

Figura 2.10: Arquitetura do sistema de arquivos em desenvolvimento. Fonte: [Fernandes 2011]

O protótipo funcional do sistema proposto possibilita a execução de operações de leitura e escrita de arquivos, com os conceitos de criptografia e controle de capacidade. As demais funcionalidades estão em desenvolvimento.

2.7 Avaliação de desempenho

A análise de desempenho de sistemas computacionais exerce um papel de grande importância, tanto em sistemas já existentes como em sistemas em desenvolvimento. O objetivo principal é determinar qual o dimensionamento mais adequado dos componentes do sistema de forma que as metas pré-estabelecidas sejam alcançadas. Dentre as técnicas de análise de desempenho existentes, destacam-se as técnicas de aferição e técnicas de modelagem [Machado 2008].

Técnicas de aferição: aplicadas sobre o próprio sistema objeto da avaliação. Podem ser aplicadas para analisar o desempenho de sistemas em desenvolvimento

que estão parcialmente construídos (prototipação) ou em sistemas que já estão totalmente construídos (*benchmarking* e coleta de dados).

Técnicas de modelagem: baseadas na construção e análise de modelos que representem o sistema em estudo. Podem ser aplicadas tanto para sistemas disponíveis quanto para sistemas em fase de estudo para construção, por meio de técnicas de solução analíticas ou técnicas de simulação.

2.8 Avaliação de desempenho de sistemas de arquivos distribuídos

Na literatura, encontra-se avaliação de desempenho de sistemas de arquivos distribuídos efetuadas de diversas maneiras. De acordo com [Boito 2009], uma das formas que podemos utilizar é de acordo com um padrão de acesso específico, como no caso de aplicações científicas onde se quer otimizar a utilização da *cache* e a distribuição de dados [Kassick 2008].

Na abordagem de [Huang and Grimshaw 2010], esta avaliação pode ser feita em diversas etapas: avaliando o desempenho de leitura/escrita de arquivos de tamanhos diferentes; avaliando o acesso paralelo de diversos clientes para a leitura e re-leitura de um mesmo arquivo; avaliando o rendimento do sistema de arquivos distribuídos em relação ao número de operações realizadas por segundo com a criação de diretórios e vários arquivos de 1 *kilobyte*; e ainda, avaliação sobre a transferência dos dados com e sem criptografia, e com a utilização ou não de outros algoritmos especiais utilizados no sistema de arquivos distribuído avaliado em questão, mostrando a discrepância existente entre as situações.

Em [Vieira 2010], a avaliação de desempenho é feita com a análise dos eventos ocorridos na rede, medindo o uso de CPU, uso de memória, e ainda o tráfego de rede gerado pela escrita de arquivos nos sistemas de arquivos avaliados em questão. Neste trabalho, a coleta de alguns dados para a avaliação é feita por meio de *scripts* na linguagem PHP.

No trabalho realizado em [Barbosa Jr. et al. 2007], a avaliação é feita com a medição da vazão dos acessos de leitura/escrita nos sistemas de arquivos distribuídos avaliados, que são feitos de duas maneiras distintas. Primeiro, fixando o tamanho dos blocos de dados e variando o tamanho do arquivo lido/escrito no sistema, e em seguida fixando o tamanho do arquivo e variando dessa vez o tamanho dos blocos de dados. A métrica utilizada é a taxa de transferência, dada em *kilobytes* por segundo.

Além disso, foi feita uma análise qualitativa, considerando fatores como instalação, configuração e administração dos sistemas de arquivos distribuídos.

2.9 Ferramentas para avaliação de desempenho

Algumas ferramentas podem auxiliar durante a avaliação de desempenho, dentre elas:

- **IOzone**: Ferramenta para *benchmark* de sistemas de arquivos locais e sistemas de arquivos distribuídos baseados no modelo do NFS, que podem ser montados no sistema de arquivos local. Realiza diversas operações de entrada/saída de dados, como leitura, escrita, re-leitura, re-escrita, entre outras, num total de 13 padrões diferentes. Gera dados tabulados como saída, o que facilita na geração de gráficos a partir dos resultados obtidos [IOzone 2011]. Esta ferramenta foi utilizada para avaliação em grande parte dos trabalhos citados no tópico anterior, sempre que houve compatibilidade.
- **Cacti**: Ferramenta para gerenciamento de redes de computadores, de código aberto, que monitora o estado de elementos de rede e programas, medindo a largura de banda utilizada, uso de CPU e memória [Costa 2008]. Possui uma interface Web, que auxilia nas tarefas de monitoramento e geração de gráficos com os dados gerados na análise. Esta ferramenta foi utilizada na avaliação feita por [Vieira 2010].
- **htop**: Ferramenta para gerenciamento do consumo de recursos do computador, como memória RAM e uso de CPU, medindo ainda a quantidade de processos em execução, o tempo de uso do sistema, dentre outros [htop 2011].
- ***Scripts* de avaliação**: Em alguns casos, escrever *scripts* em alguma linguagem de programação que forneça suporte a realização de operações sobre os sistemas objeto da análise pode ser uma maneira mais viável de se coletar dados para realizar tal tarefa, como acontece em [Vieira 2010].

Capítulo 3 – Método de avaliação, cenários e métricas

A utilização de técnicas de modelagem para executar a avaliação de desempenho de sistemas de arquivos distribuídos implica na definição de diversos parâmetros entrada necessários para gerar o modelo, tanto no lado do cliente quanto no servidor, além da rede de comunicação.

Dessa forma, para efetuar a avaliação de desempenho no presente trabalho, optou-se pela técnica de aferição por coleta de dados. Dessa maneira, existe um contato direto com os sistemas de arquivos distribuídos, quanto à facilidade de uso e administração, englobando tarefas como a instalação, configuração, manutenção, e utilização de suas funcionalidades.

Além disso, os resultados obtidos com a coleta dos dados possuem maior precisão, uma vez que mostram o comportamento real do sistema que está sendo avaliado. Dessa forma, foram montados alguns cenários reais com situações encontradas no dia a dia.

3.1 Cenários de avaliação de desempenho

Conforme feito em [Boito 2009], onde foi feita a avaliação de acordo com um padrão de acesso específico, no presente trabalho desejou-se avaliar os sistemas de arquivos distribuídos de acordo com o padrão de um ambiente acadêmico de pequena escala. Sendo assim, o cenário de avaliação aplicado foi planejado e dividido em duas etapas.

Na primeira, foram executadas operações de escrita e leitura de arquivos de tamanhos pré-definidos, de 1MB, 5MB, 10MB, 25MB e 50MB. As métricas utilizadas para esta avaliação foram os tempos reais e de processamento, em segundos, e uma estimativa da taxa de transferência, dada em *kilobytes* por segundo. Dessa forma, avalia-se o comportamento dos sistemas quanto à velocidade na transferência de arquivos [Huang and Grimshaw 2010], [Barbosa Jr. et al. 2007].

Na etapa seguinte, foram escritos 10 diretórios e 10 arquivos de 1 *kilobyte* no interior de cada um. Além disso, foram escritos mais 10 diretórios da mesma forma, dentro dos criados anteriormente, cada um contendo 10 arquivos de 1 *kilobyte*. No final, foramcriados 110 diretórios e 1100 arquivos. Assim, deseja-se testar a eficiência do sistema de arquivos distribuídos em relação à quantidade de operações realizadas por segundo, métrica utilizada nesta etapa juntamente com o tempo real e de processamento [Huang and Grimshaw 2010].

Para os cenários de avaliação descritos, foi aplicado um ambiente com variação quanto à concorrência de acesso. Foram executados em um ambiente com apenas um cliente realizando acessos, e para vários clientes simultâneos [Huang and Grimshaw 2010].

Além da concorrência de acesso dos clientes, foi montado um cenário com concorrência de processamento, tanto nas máquinas clientes quando no servidor, de quatro maneiras diferentes: sem concorrência, concorrência no servidor, concorrência no cliente e concorrência em ambos. Essa concorrência foi caracterizada pela execução de um processo de compactação de arquivo em segundo plano.

Em paralelo às etapas de leitura/escrita e de operações, nos cenários sem concorrência de processamento, foi feita a medição das taxas de consumo de CPU durante a execução dos testes, tanto no lado cliente como no servidor [Vieira 2010].

Dessa forma, pode-se abranger os diversos tipos de situações encontradas no dia-a-dia de um ambiente acadêmico de pequena escala desejado. Uma visão do cenário de avaliação pode ser vista na Tabela 3.1, que contempla os tipos de avaliação que foram aplicados com suas métricas, e na Tabela 3.2, que descreve o cenário de avaliação submetido a situações de concorrência. Para cada uma das situações descritas na Tabela 3.1, foram aplicadas as situações de concorrência descritas na Tabela 3.2.

Tabela 3.1: Descrição do cenário de avaliação e métricas utilizadas

Cenários de avaliação segundo a operação realizada		
Tipo	**Descrição**	**Métricas**
Leitura/Escrita	Utilizando arquivos de tamanhos pré-definidos: 1MB, 5MB, 10MB, 25MB e 50MB	Tempo de execução, tempo de processamento e taxa de transferência
Operações	Criação de 10 diretórios contendo 10 arquivos de 1KB, e ainda em cada diretório cria-se mais 10 diretórios e 10 arquivos de 1KB	Operações por segundo

Tabela 3.2: Situações de concorrência

Cenários de avaliação em situações de concorrência	
Concorrência de processamento	**Concorrência de acesso**
Um cliente	Sem concorrência de processamento; Concorrência de processamento no servidor; Concorrência de processamento no cliente; Concorrência de processamento em ambos.
Vários clientes	Sem concorrência de processamento; Concorrência de processamento no servidor; Concorrência de processamento no cliente; Concorrência de processamento em ambos.

3.2 Ferramentas e *scripts* utilizados

Os cenários de avaliação estabelecidos na seção 3.1 foram implementados através da execução de *scripts* escritos na linguagem Python, pois esta é uma linguagem livre, interpretada, multiplataforma e com suporte a vários paradigmas de programação [Python 2011].

Na seqüência, são exibidos o código-fonte do *script* de avaliação de leitura do Tahoe e o código-fonte do *script* de avaliação de operações do NFS. Uma relação

completa de códigos-fonte dos *scripts* utilizados para as demais avaliações podem ser encontrados no Apêndice A.

```
#>script para avaliacao de leitura Tahoe
#>diretorio utilizado: passado no arquivo ./root.cap
#>arquivos e diretorios necessarios:
#./root.cap>arquivo com o dircap do diretorio de escrita
#./server-URLs>arquivo com IP:porta dos storage nodes
#>executar como ./python read_tahoe.py <arq_saida>
import os, sys, httplib, urllib, random, time, urlparse
import json as simplejson

stats_out = sys.argv[1]
server_urls = []
for url in open("server-URLs", "r").readlines():
    url = url.strip()
    if url:
        server_urls.append(url)
root = open("root.cap", "r").read().strip()
filenames = [1024,5120,10240,25600,51200]

def read_and_discard(nodeurl, root, pathname):
    if nodeurl[-1] != "/":
        nodeurl += "/"
    url = nodeurl + "uri/%s/" % urllib.quote(root)
    if pathname:
        url += urllib.quote(pathname)
    f = urllib.urlopen(url)
    while True:
        data = f.read(4096)
        if not data:
            break
f = open(stats_out+".tmp", "w")
for size in filenames:
    pathname = str(size)+'kb'
    server = random.choice(server_urls)
    print "reading", pathname
    start_clock, start_time = time.clock(), time.time()
    read_and_discard(server, root, pathname)
    t_down_clock = time.clock() - start_clock
    t_down_time = time.time() - start_time
    kbps = float(size/t_down_time)
    f.write("file: %s\n" % pathname )
    f.write("time elapsed: %f\n" % t_down_time)
    f.write("clock elapsed: %f\n" % t_down_clock)
    f.write("transfer rate: %f\n\n" % kbps)
f.write("time & clock in seconds, transfer rate in
kbps\n")
f.close()
os.rename(stats_out+".tmp", stats_out)
```

```
#>script para avaliacao de operacoes por segundo NFS
#>diretorio utilizada: /shared/nfs/
#>arquivos e diretorios necessarios:
#/tmp/files/1kb> arquivo de 1kb
#/tmp/saidas/> diretorio para a escrita dos resultados
#>executar como ./$python operations_nfs.py <arq_saida>

import os, sys, shutil, time
stats_out = sys.argv[1]
lld = ['dirA','dirB','dirC','dirD','dirE',
       'dirF','dirG','dirH','dirI','dirJ']
tld = ['dir1','dir2','dir3','dir4','dir5',
       'dir6','dir7','dir8','dir9','dir0']
os.chdir('/shared/nfs/')
clock_start = time.clock()
time_start = time.time()
for lowlevel in lld:
    os.mkdir(lowlevel)
    os.chdir(lowlevel)

    for a in range(10):
        test = lowlevel+str(a)+'.test'
        shutil.copyfile('/tmp/files/1kb',test)

    for toplevel in tld:
        os.mkdir(toplevel)
        os.chdir(toplevel)

        for b in range(10):
            shutil.copyfile('/tmp/files/1kb',
                            toplevel+str(b)+'.test')
        os.chdir('..')
    os.chdir('..')

elapsed_clock = time.clock() - clock_start
elapsed_time = time.time() - time_start
avg_time = 1430/elapsed_time

os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
f.write("time/clock/operationspersec\n\n")
f.write("%f\n" % elapsed_time)
f.write("%f\n" % elapsed_clock)
f.write("%f\n" % avg_time)
f.close()
os.rename(stats_out+".tmp", stats_out)
```

Além dos *scripts* para a medição dos tempos, a ferramenta htop foi instalada com o intuito de medir o uso de CPU, devido a sua facilidade de instalação e clareza na exibição dos resultados.

3.3 Configuração do ambiente computacional

Os sistemas de arquivos distribuídos avaliados no presente trabalho foram instalados e configurados nas instalações do laboratório do Grupo de Sistemas Paralelos e Distribuídos (GSPD) do Departamento de Ciências de Computação e Estatística (DCCE) do Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto (IBILCE).

Para tal, foi utilizado o *cluster Beowulf*, que é composto por 10 máquinas com processadores *Intel® Pentium® Dual CPU E2160 @ 1.80 GHz*, 2GB de memória *RAM*, disco rígido de 40GB 7200RPM, com sistema operacional *Linux 32 bits*, distribuição *Debian 5.0*, *kernel* versão *2.6.26-2-686*, interligados pela rede *ethernet* num *switch 3Com Office Connect Dual Speed 16 Plus* de 16 portas *10/100Base-TX*.

A máquina principal do *cluster* recebe o nome de *front-end*, com endereço IP da rede local 192.168.0.1, enquanto os demais nós recebem os nomes *node1*, *node2*, *node3*, *node4*, *node5*, *node6*, *node7*, *node8* e *node9*, e seus respectivos endereços IP são 192.168.0.2, 192.168.0.3, até o IP 192.168.0.10 da máquina *node9*.

Além das máquinas do *cluster Beowulf*, foram utilizadas três máquinas de uso comum do laboratório, que foram conectadas junto à rede do *cluster* durante os testes, para serem utilizadas como clientes dos sistemas de arquivos distribuídos avaliados, e ainda para a instalação de máquinas virtuais.

Esses computadores são compostos por processadores *Intel® Core™2 Quad CPU Q8200 @ 2.33 GHz*, 4GB de memória *RAM*, disco rígido de 320GB 7200RPM, com sistema operacional *Linux 64 bits*, distribuição *Debian 6.0*, *kernel* versão *2.6.32-5-amd64*. Os nomes das máquinas são *camus*, *milo* e *shaka*, e seus respectivos IPs na rede local do cluster são: 192.168.0.16, 192.168.0.15 e 192.168.0.13.

3.3.1 Instalação dos servidores

O servidor do NFS foi instalado na máquina *front-end*. Os diretórios */shared/nfs/dir1/*, */shared/nfs/dir2/* e */shared/nfs/dir3/* foram compartilhados com permissões de acesso de leitura/escrita para os usuários da rede 192.168.0.0/24.

Para o Tahoe, o *introducer* foi configurado na máquina *front-end*, instalado na pasta */shared/tahoe/introducer/*. Os *storage nodes* foram instalados em todas as 10 máquinas do cluster, no diretório /*shared/tahoe/storage/* local de cada máquina.

Ainda, para fins de medição do uso de CPU pelos processos servidores, a ferramenta htop foi instalada na máquina *front-end*.

3.3.2 Instalação dos clientes

Para a instalação dos clientes dos sistemas de arquivos distribuídos avaliados foram utilizadas as máquinas *camus*, *milo* e *shaka*. Para o NFS, foi configurada a montagem dos diretórios compartilhados */shared/nfs/dir1/*, */shared/nfs/dir2/* e */shared/nfs/dir3/* nas máquinas *shaka*, *milo* e *camus*, respectivamente.

No Tahoe, as requisições de acesso foram feitas pelas máquinas *camus*, *milo* e *shaka*, diretamente aos servidores de armazenamento. A cada acesso, um dentre todos os servidores é escolhido de modo aleatório.

Da mesma forma que no servidor, a ferramenta htop foi instalada na máquina *shaka* para fins de medição do uso de CPU pelos processos clientes, uma vez que os clientes tendem a ter o mesmo comportamento por se tratarem de máquinas com as mesmas configurações.

3.3.3 Instalação em ambiente de máquinas virtuais

Para a instalação do Hadoop, foi utilizado um ambiente de máquinas virtuais. Foi feito o *download* da imagem utilizada nas máquinas virtuais através da página do *framework* Cloudera [Cloudera 2011]. Esta imagem contém sistema operacional *Linux 64 bits*, distribuição CentOS 5.6 Final, *kernel* versão *2.6.18-238.19.1.el5*, com o Hadoop instalado juntamente com alguns outros serviços do *framework* citado.

Ao todo, foram utilizadas quatro máquinas virtuais com 1GB de memória RAM, instaladas nas máquinas *shaka*, *milo* e *camus*. Na máquina *camus* foram montadas duas máquinas virtuais: uma para servir de *NameNode* e a outra como *DataNode*. Na máquina *milo* foi instalada outra máquina como *DataNode*, e na máquina *shaka* a última máquina, funcionando como cliente, totalizando dois *DataNodes*, um *NameNode* e um cliente no sistema completo.

Ainda, na máquina cliente foi instalado o suporte ao sistema de arquivos do usuário (*Filesystem Userspace* – FUSE) do Hadoop. Com isso, o sistema de arquivos

distribuídos pode ser montado em um diretório no sistema de arquivos local para a aplicação do cenário de avaliação.

Para obter um parâmetro viável de comparação com o HDFS e o sistema de arquivos distribuídos em desenvolvimento, foi feita a instalação do NFS em duas das máquinas virtuais. A máquina virtual da *camus* foi utilizada como servidor, enquanto a *shaka* foi utilizada como cliente. O diretório */shared/nfs/* foi compartilhado no servidor e montado no mesmo local na máquina cliente.

Em ambos os casos, foi utilizado o cenário de um cliente sem concorrência de processamento, e executados os testes de operações, leitura e escrita. A medição do consumo de uso de CPU não foi realizada, pois este é um componente virtualizado sobre os componentes físicos da máquina hospedeira, e não fornece uma medida precisa para tal métrica. Além disso, as máquinas virtuais estão executando concorrentemente com os processos da máquina hospedeira.

Com isso, o cenário de avaliação foi completamente definido. A Figura 3.1 ilustra as máquinas do *cluster Beowulf* com as instalações e configurações de cada uma, enquanto a Figura 3.2 ilustra as máquinas *camus*, *milo* e *shaka* utilizadas como clientes NFS, Tahoe e a montagem do ambiente das máquinas virtuais.

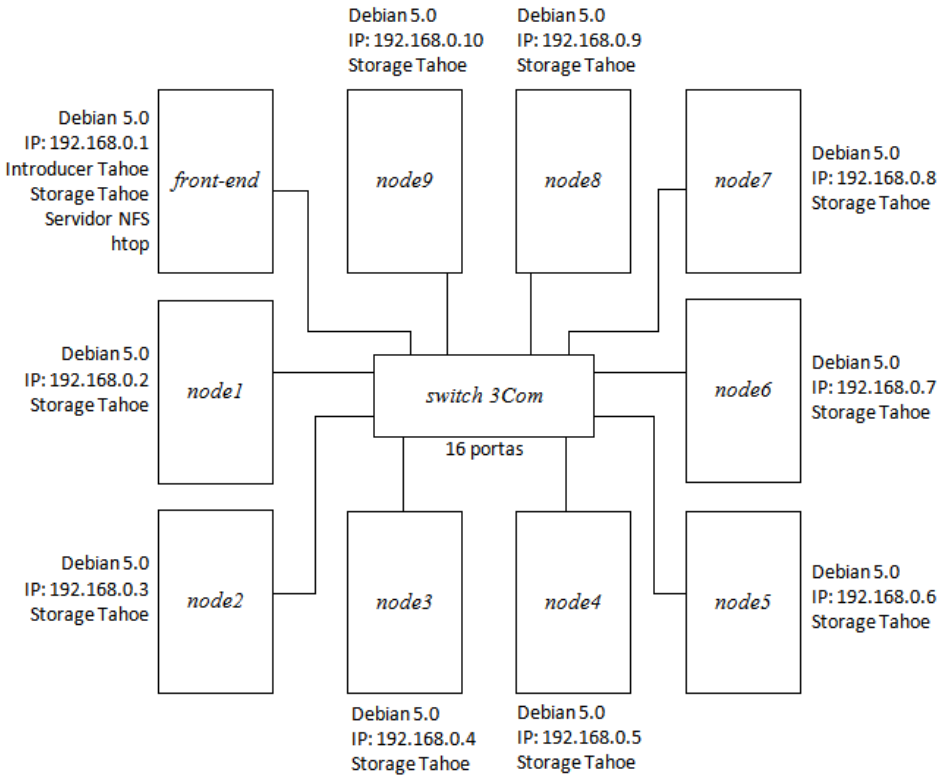

Figura 3.1: Instalações no *cluster Beowulf*

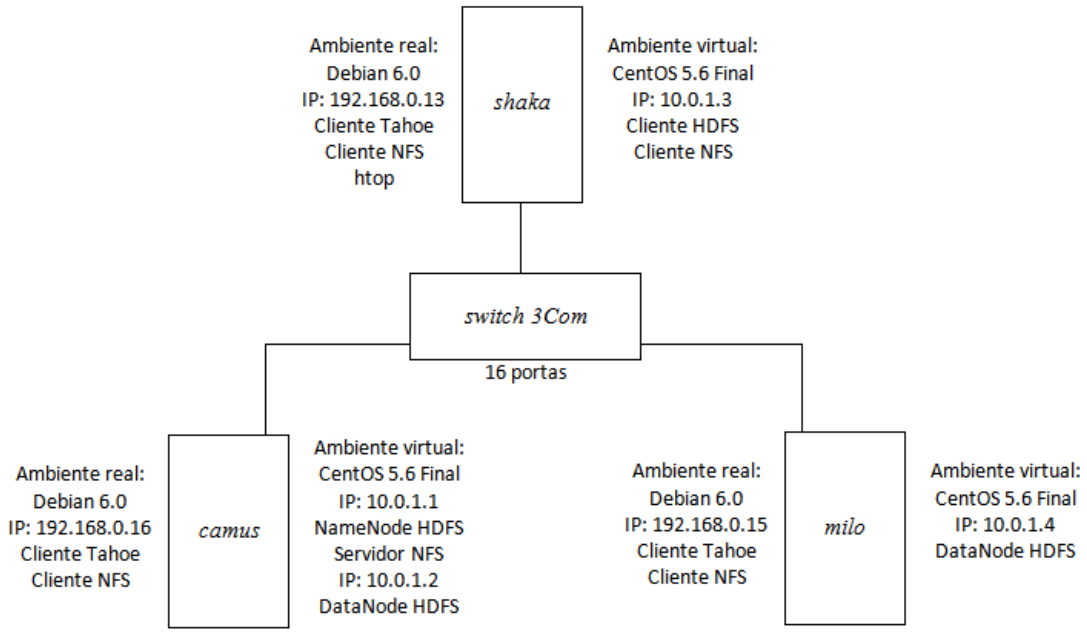

Figura 3.2: Instalações de ambiente real/virtual em *shaka, camus* e *milo*

Todas as configurações de *hardware* e *software* foram devidamente avaliadas quanto ao correto funcionamento, efetuando as operações de entrada e saída e acessos concorrentes.

Capítulo 4 – Resultados obtidos

Como explicado no capítulo anterior, a aplicação dos cenários de avaliação está dividida em dois ambientes: real e virtual. Dessa forma, os resultados serão exibidos e analisados de forma separada.

Os cenários de avaliação contemplados foram executados cinco vezes para cada tipo de situação, e a apresentação dos resultados será feita tomando como base a média de todas as iterações para cada caso.

No final deste capítulo, será feita uma ilustração dos resultados de maneira comparativa entre os sistemas de arquivos distribuídos. As tabelas com os resultados individuais completos de cada iteração para os diferentes cenários podem ser encontradas no Apêndice B.

4.1 Avaliação em ambiente real

Em ambiente real, foram instalados com sucesso o NFS e o Tahoe. Foram aplicados sobre eles o cenário completo de testes ilustrado no Capítulo 3, contemplando todos os casos de situações de concorrência, tanto de acesso como de processamento.

Para os cenários de leitura/escrita e operações, em ambientes sem concorrência de processamento e acesso, foi feita a medição da taxa de uso de CPU.

Para ilustrar os resultados, foram utilizados diversos gráficos. Em cada gráfico, foram utilizados alguns rótulos identificados pelas iniciais das situações de

concorrência de processamento, ou seja, sem concorrência (SC), concorrência no servidor (CS), concorrência no cliente (CC) e concorrência em ambos (CA).

4.1.1 NFS

As operações de acesso no NFS são feitas através de chamadas de procedimento remoto, onde o cliente faz a requisição para o servidor e este atende com a operação de E/S (Entrada/Saída) requisitada.

A leitura de arquivos no NFS foi efetuada desmontando e remontando o diretório compartilhado a cada iteração, a fim de evitar efeitos de *cache* nos resultados. Os resultados obtidos para situações sem concorrência de processamento podem ser observados na Figura 4.1.

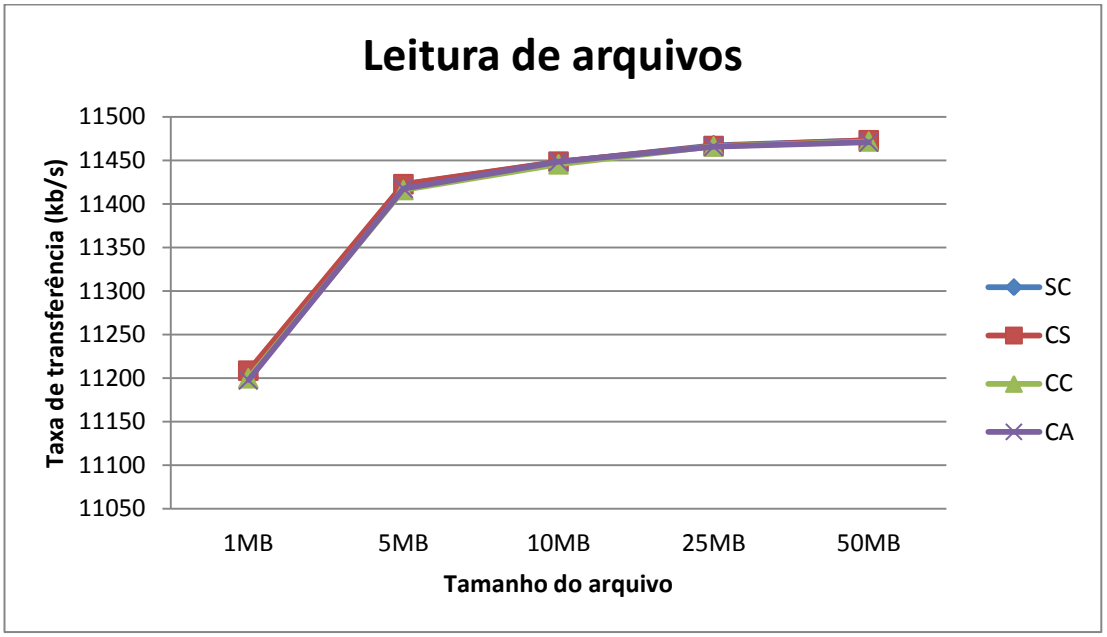

Figura 4.1: Leitura de arquivos sem concorrência de acesso NFS

O desempenho do sistema quando não havia concorrência de acesso manteve um padrão em todas as situações de concorrência de processamento, atingindo um ponto de saturação no limite da banda da rede. Entretanto, em situação de concorrência de acesso de três clientes simultâneos, passou a existir uma disputa de recursos da rede pelos usuários. Isso fez a taxa de transferência alcançar seu ponto de saturação variando em torno de 1/3 da taxa máxima, conforme verificado no gráfico da Figura 4.2.

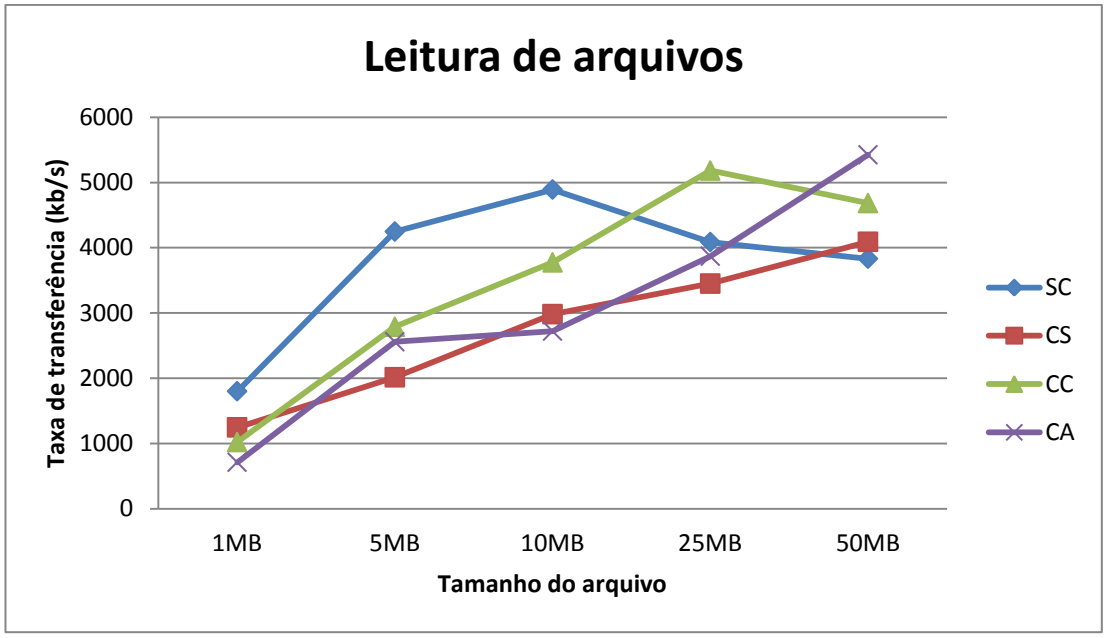

Figura 4.2: Leitura de arquivos com acesso concorrente NFS

No caso da escrita de arquivos, não foi necessária a desmontagem e remontagem dos diretórios compartilhados, uma vez que os arquivos eram apagados antes de serem reescritos a cada iteração. A Figura 4.3 ilustra os resultados obtidos para situações sem concorrência de acesso, enquanto a Figura 4.4 ilustra para as situações de acesso concorrente.

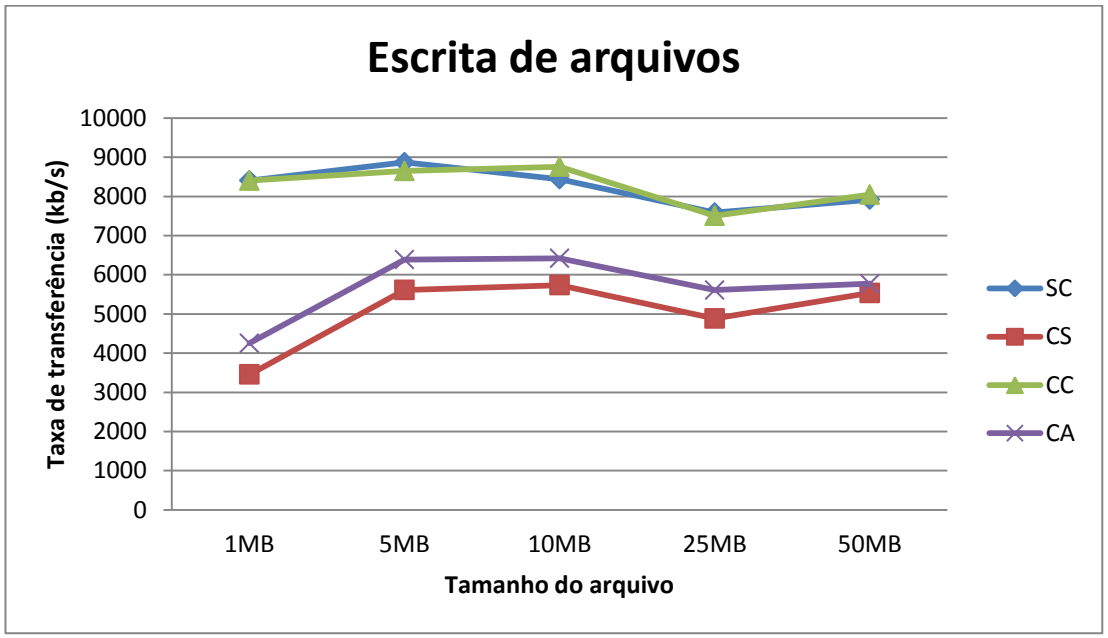

Figura 4.3: Escrita de arquivos sem concorrência de acesso NFS

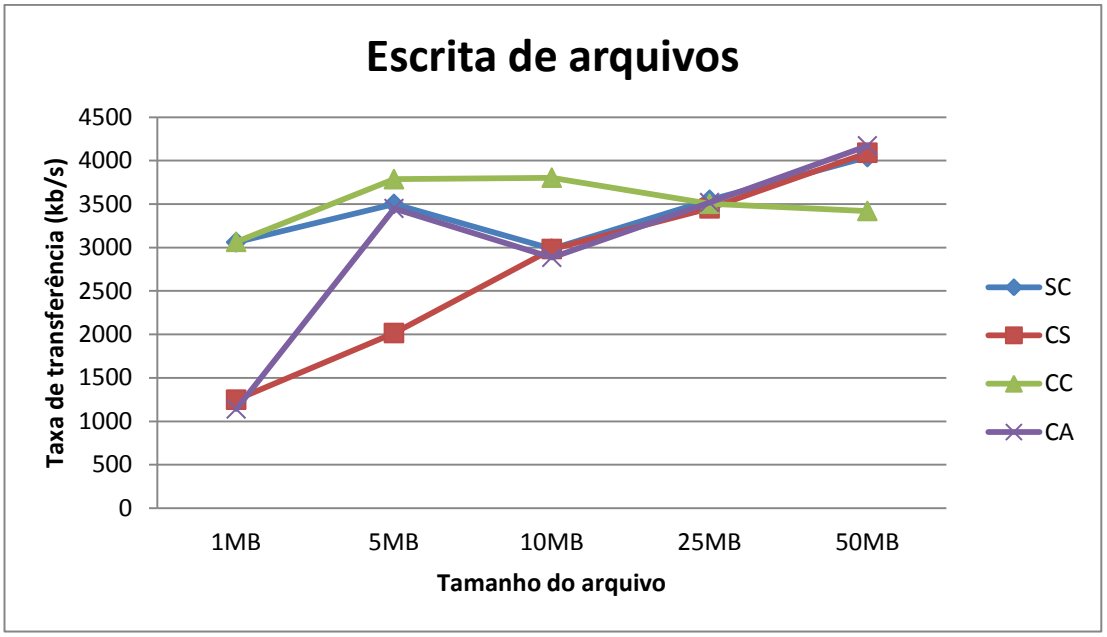

Figura 4.4: Escrita de arquivos com acesso concorrente NFS

Em relação à concorrência de processamento, quando essa existiu apenas no cliente, o desempenho da escrita de arquivos se portou de forma similar ao caso sem concorrência. Porém, quando essa concorrência aconteceu no servidor, o desempenho diminuiu de maneira considerável, visto que havia concorrência de E/S na escrita do arquivo em disco com o processo que gerava a concorrência de processamento.

Introduzindo a concorrência de acesso, o desempenho do sistema também foi depreciado. Todas as escritas tenderam a manter um padrão de escrita da mesma forma que se manteve um padrão de leitura com acesso concorrente, em torno de 1/3 do máximo da banda de rede.

No cenário de operações, o NFS totalizou 1.430 operações a cada iteração, que correspondem a 1.100 criações de arquivos de 1KB, 110 criações de diretórios e ainda 220 mudanças de diretório (*chdir*), suportadas no NFS. Os resultados de tempo real gasto para a execução de todo o processo estão ilustrados no gráfico da Figura 4.5.

A queda de desempenho do NFS pode ser observada em todos os casos de concorrência de acesso, e principalmente nas situações onde existe concorrência de processamento no servidor. Essa queda é explicada pelas trocas de contexto existentes no escalonamento dos processos, além de gerar concorrência de E/S no disco nas situações de concorrência de processamento.

Em situações onde não há nenhum tipo de concorrência no servidor, o tempo de resposta foi menor, pois não havia concorrência de E/S no disco do servidor.

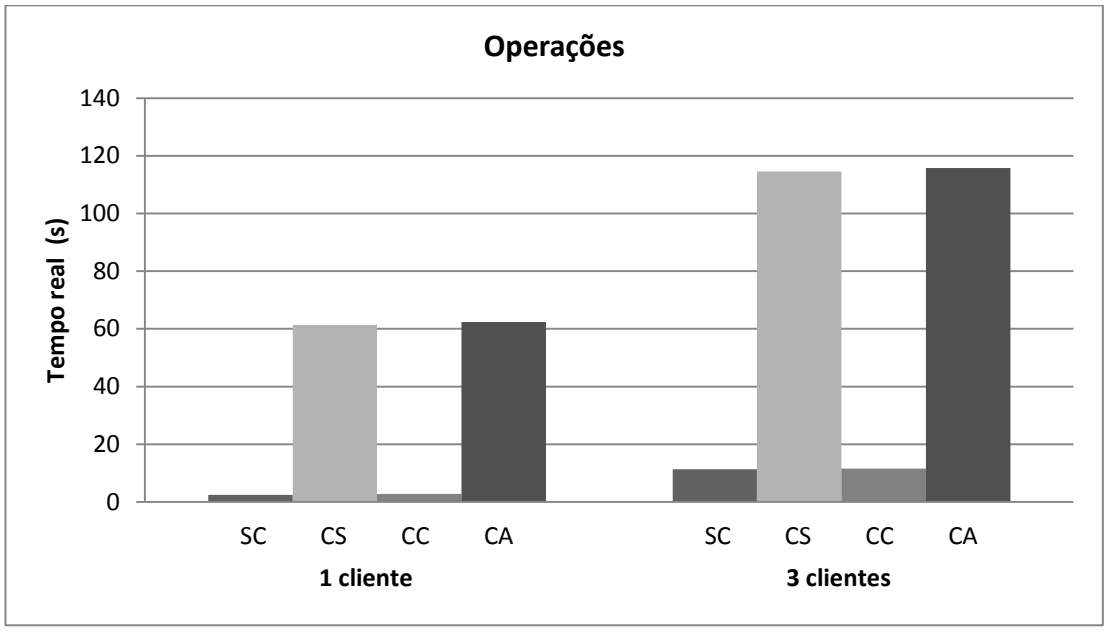

Figura 4.5: Tempo real gasto para realização de operações NFS

O tempo real decorrido variou de 2,43 segundos em média no caso sem concorrência para uma média de quase 2 minutos na situação de concorrência completa. Entretanto, o tempo de ocupação da CPU durante o processo de operações esteve na média de 0,13 segundos, exceto no caso de concorrência de processamento em ambos os lados que aumentou para 0,23 segundos.

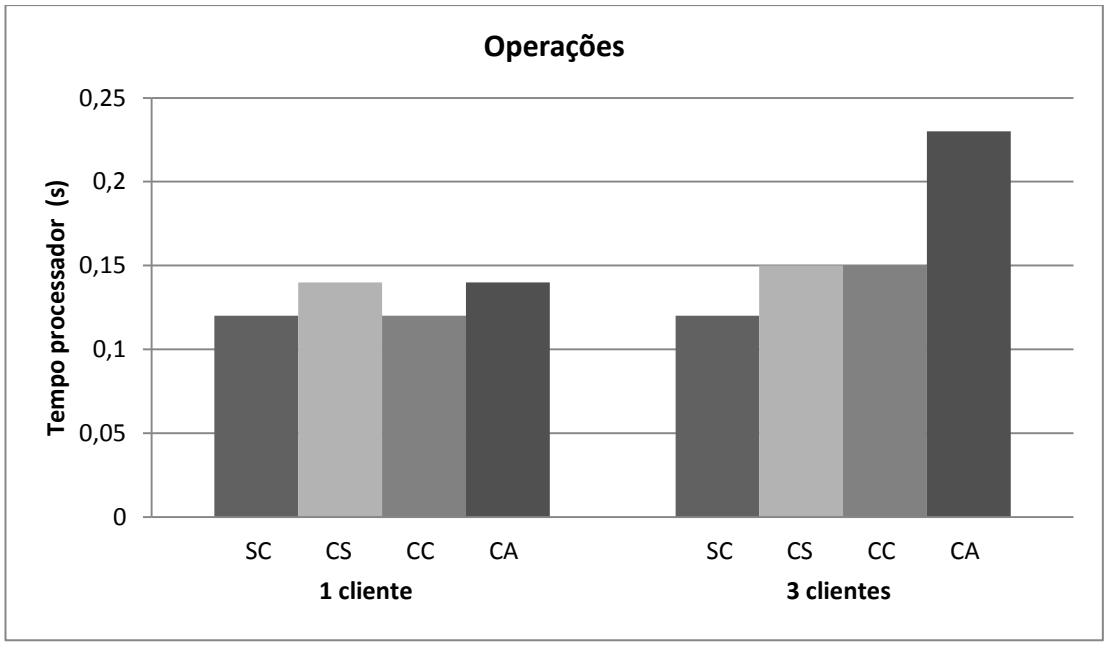

Figura 4.6: Tempo de processador gasto para realização de operações NFS

Tudo se reflete no número de operações realizadas por segundo, que é inversamente proporcional ao tempo real decorrido. O gráfico da Figura 4.7 exibe os resultados em relação a essa métrica.

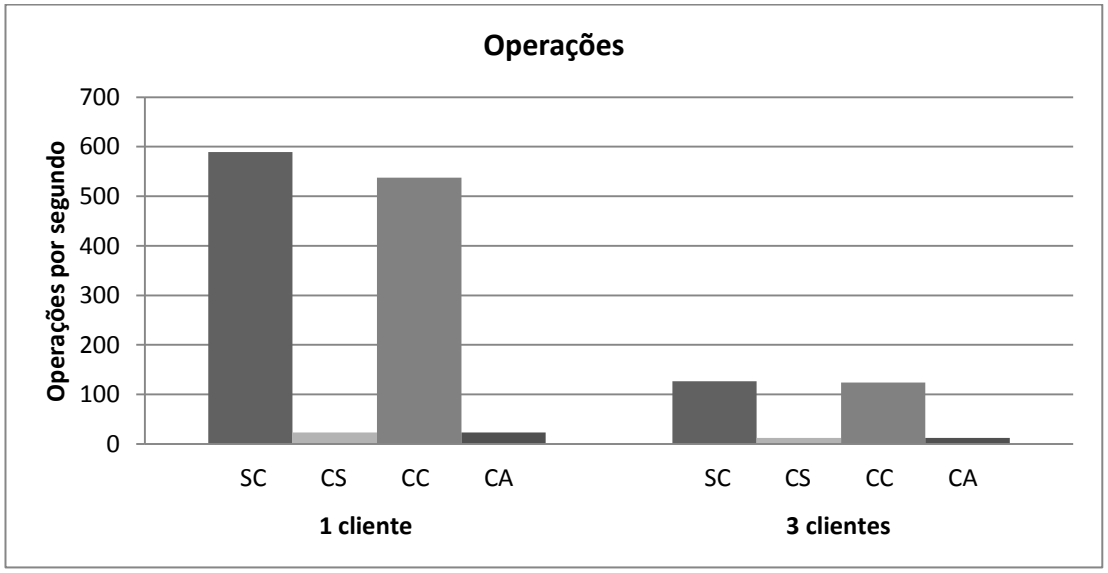

Figura 4.7: Operações realizadas por segundo NFS

Os resultados obtidos na medição feita do consumo da porcentagem de uso de CPU são exibidos em dois gráficos que ilustram o consumo obtido no servidor e no cliente, para situações de acesso isolado ou concorrente. A Figura 4.8 e Figura 4.9 ilustram respectivamente o consumo máximo e médio do uso de CPU.

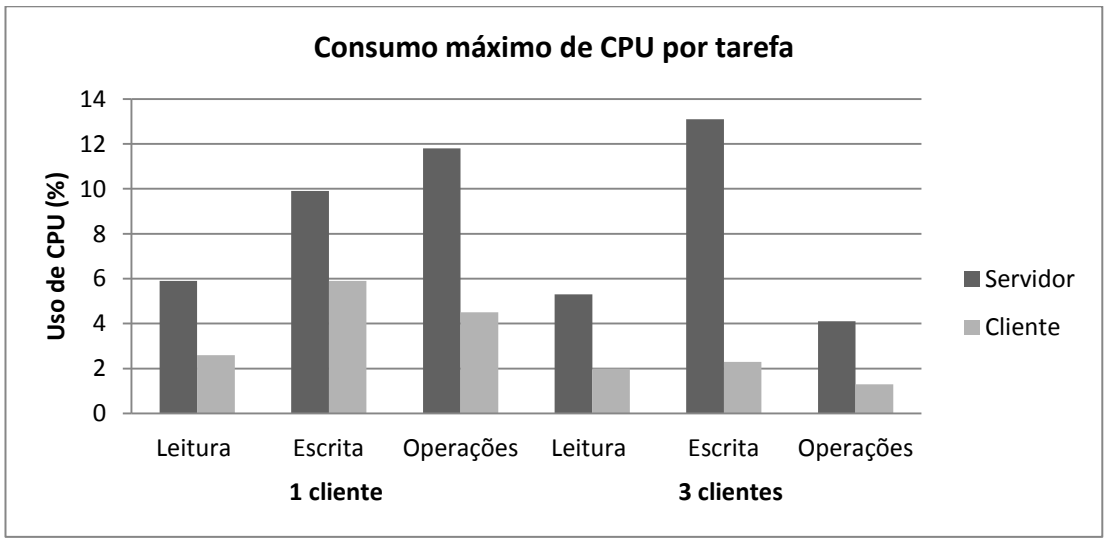

Figura 4.8: Consumo máximo de CPU por tarefa NFS

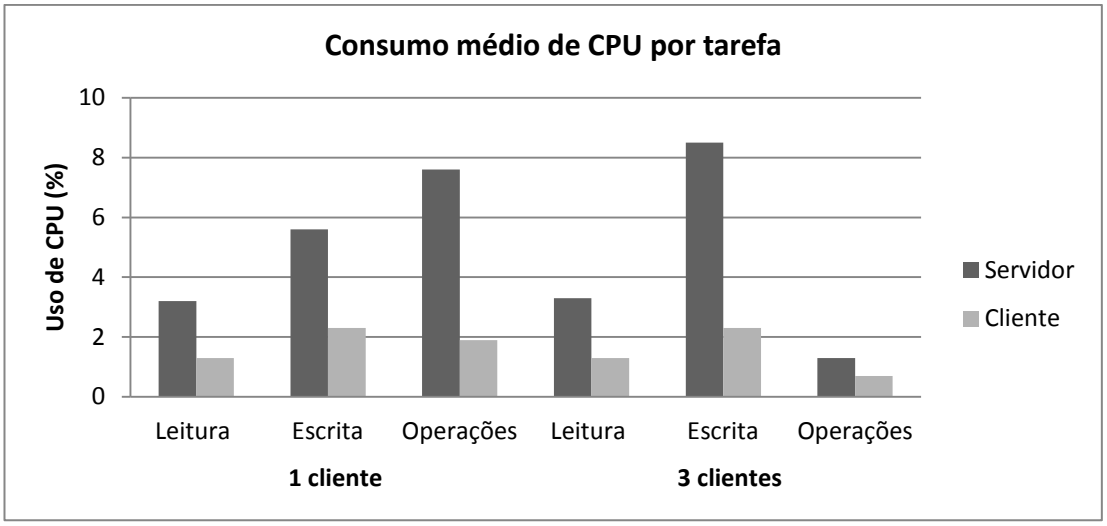

Figura 4.9: Consumo médio de CPU por tarefa NFS

No servidor, o consumo de CPU nos acessos de leitura teve média semelhante. Para a escrita, em situação concorrente houve um leve aumento no consumo, enquanto para as operações houve redução. No cliente, o consumo de CPU no geral foi pequeno, alcançando seus maiores índices na escrita de arquivos, de forma similar nas duas situações mensuradas.

4.1.2 Tahoe-LAFS

No Tahoe, os acessos ao sistema foram feitos por requisições HTTP diretamente a um servidor de armazenamento. As URLs (*Uniform Resource Locator*) eram formadas a partir do *dircap* do diretório utilizado, com um endereço de servidor de armazenamento escolhido de maneira aleatória, e ainda o caminho completo do arquivo na representação de diretórios do Tahoe.

Para efetuar a leitura, o cliente envia a URL do arquivo a um servidor, para este efetuar a tarefa. O servidor então busca no *introducer* informações de quais outros servidores utilizar para recuperar as três porções necessárias e remontar o arquivo original, e então enviar o arquivo ao cliente.

Os resultados da leitura de arquivos em situação de ausência de concorrência de acesso podem ser verificados na Figura 4.10, enquanto a Figura 4.11 ilustra os resultados para situação de acesso concorrente.

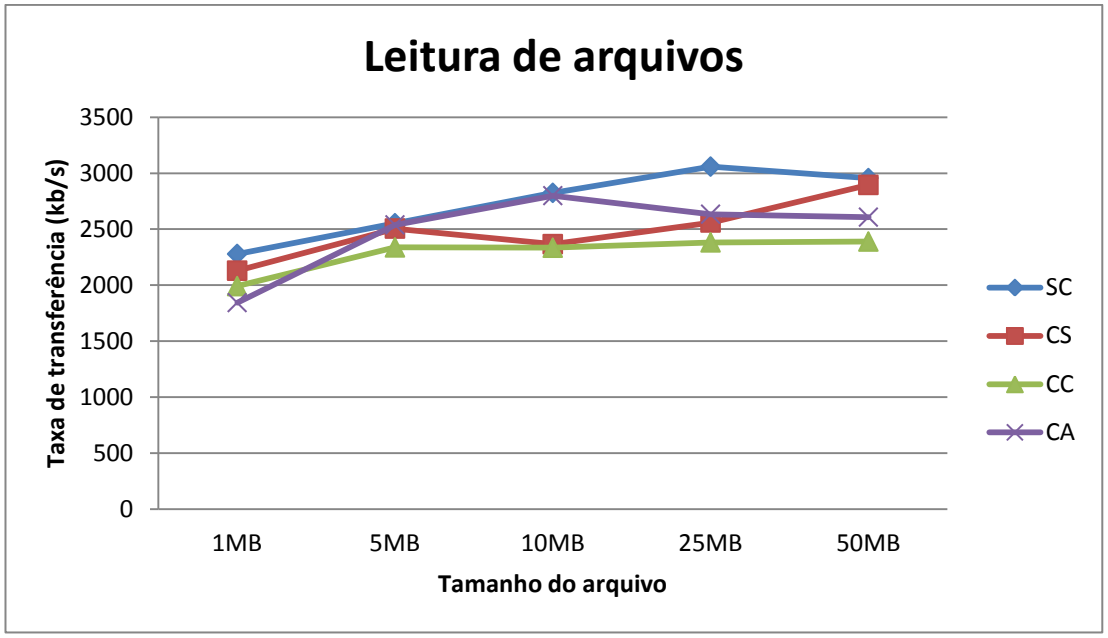

Figura 4.10: Leitura de arquivos sem concorrência de acesso Tahoe

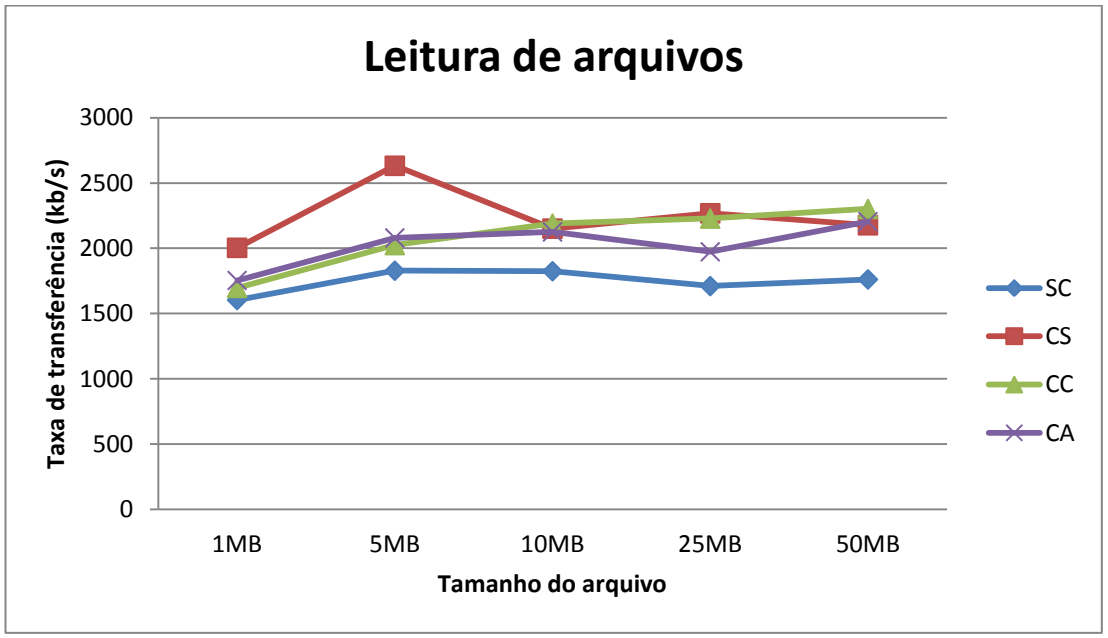

Figura 4.11: Leitura de arquivos com acesso concorrente Tahoe

Os casos de concorrência de acesso onde as taxas de transferência foram menores podem ser explicados pela escolha aleatória dos servidores, pois um mesmo servidor pode ter sido utilizado ao mesmo tempo por diferentes clientes para efetuar a requisição de transferência dos arquivos.

Para efetuar as operações de escrita, o cliente envia ao servidor a URL do arquivo que será criado, estabelece a conexão e envia os dados para a sua efetivação no sistema. No servidor, foram executados os procedimentos de criptografia e

codificação. Feito isso, foram geradas as 10 porções do arquivo que foram distribuídas aos demais nós de armazenamento.

Os resultados para escrita de arquivos podem ser verificados na Figura 4.12, no caso sem concorrência de acesso, e Figura 4.13, para situações onde há acesso concorrente.

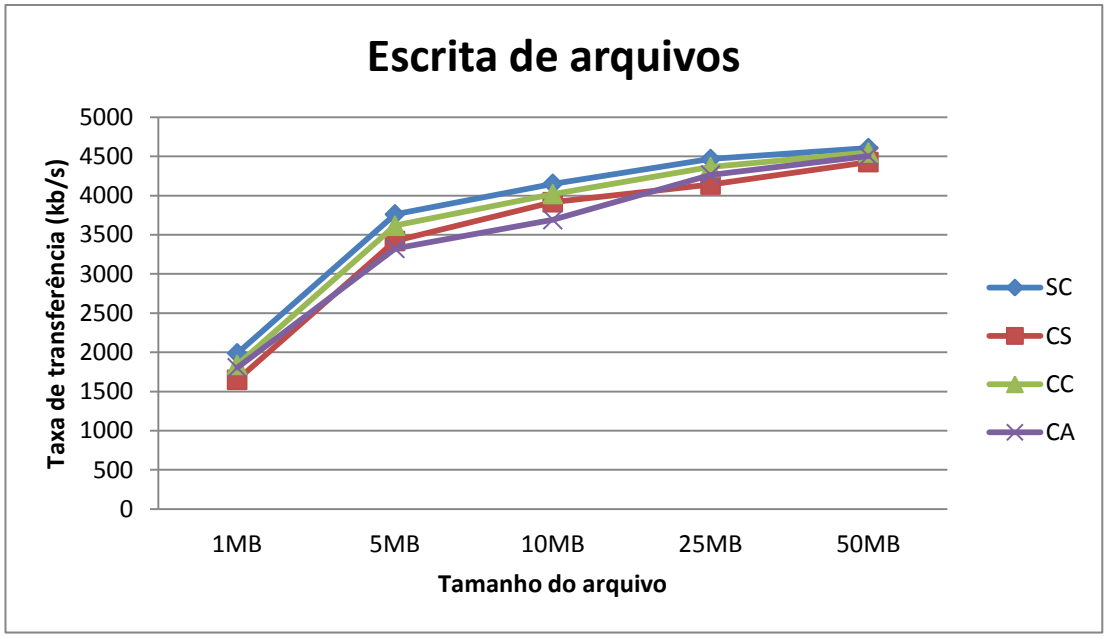

Figura 4.12: Escrita de arquivos sem concorrência de acesso Tahoe

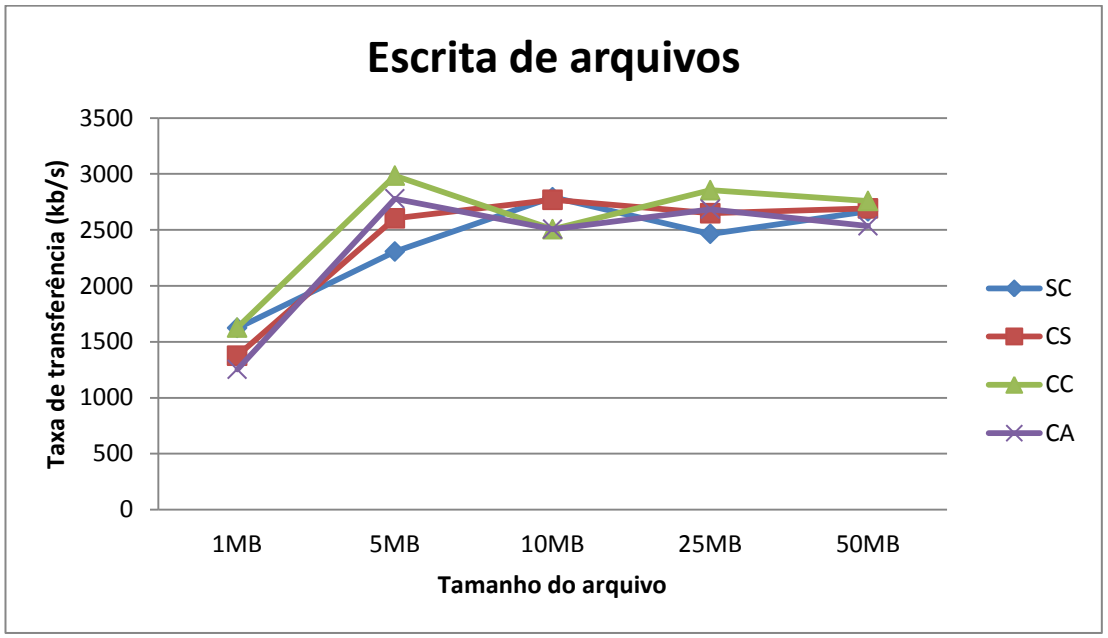

Figura 4.13: Escrita de arquivos com acesso concorrente Tahoe

O desempenho para arquivos pequenos foi menor, pois o esquema de criptografia e codificação utilizado no Tahoe consome um tempo inicial para o

processamento dessas operações. Com o aumento do tamanho do arquivo, o sistema atinge um ponto de saturação em torno de 4.500kb/s.

Nas situações de concorrência de acesso, houve uma variação maior na média dos resultados obtidos pela maior concorrência de recurso de rede gerado. A situação dos arquivos pequenos se manteve, tendo um desempenho inferior, e a saturação da taxa de transferência ocorreu para valores entre 2.500kb/s e 3.000kb/s, visto que as operações acontecendo ao mesmo tempo geravam um volume maior de dados, congestionando a rede.

Na avaliação de operações realizadas, foram consideradas 1100 criações de arquivos e 110 criações de diretórios, totalizando 1210 operações. Não foram computadas operações de mudança de diretórios, uma vez que estas não se fazem presente no Tahoe. Os resultados de tempo real gasto podem ser verificados na Figura 4.14, e o número de operações realizadas por segundo é exibido na Figura 4.15.

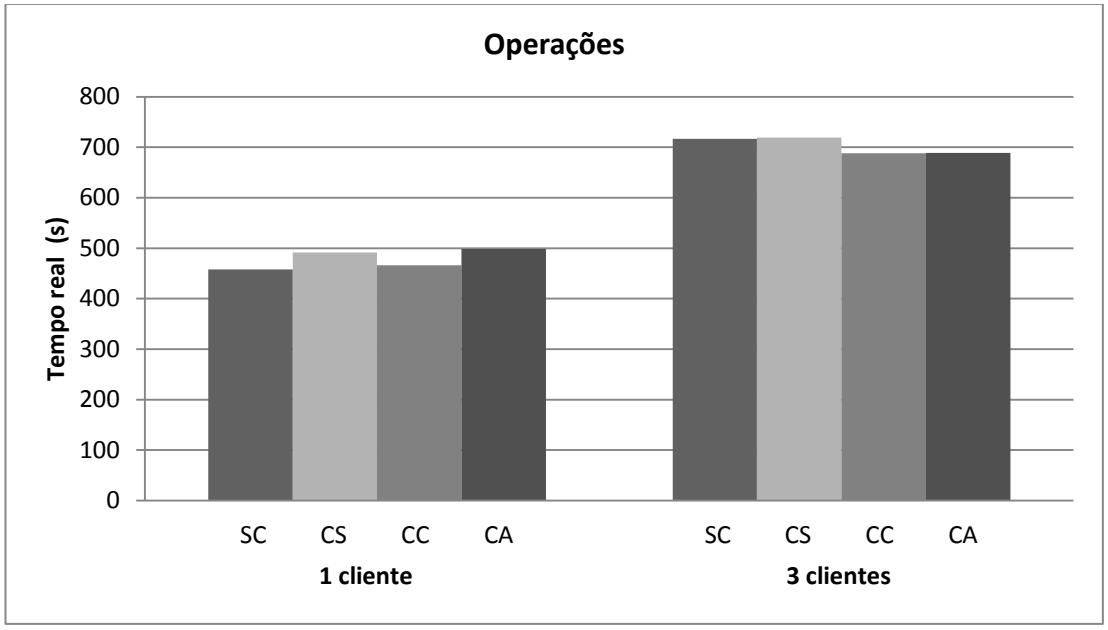

Figura 4.14: Tempo real gasto para realização de operações Tahoe

Nas situações sem concorrência de acesso, em média levou 8 minutos para a execução das operações. Em situações de acesso concorrente, esse tempo médio subiu para 11 minutos. As situações geram uma taxa média de duas operações por segundo. Isso se explica pelo esquema de criptografia e codificação adotado pelo Tahoe no instante de criação dos arquivos e diretórios, além do fato que as operações são realizadas por requisições HTTP.

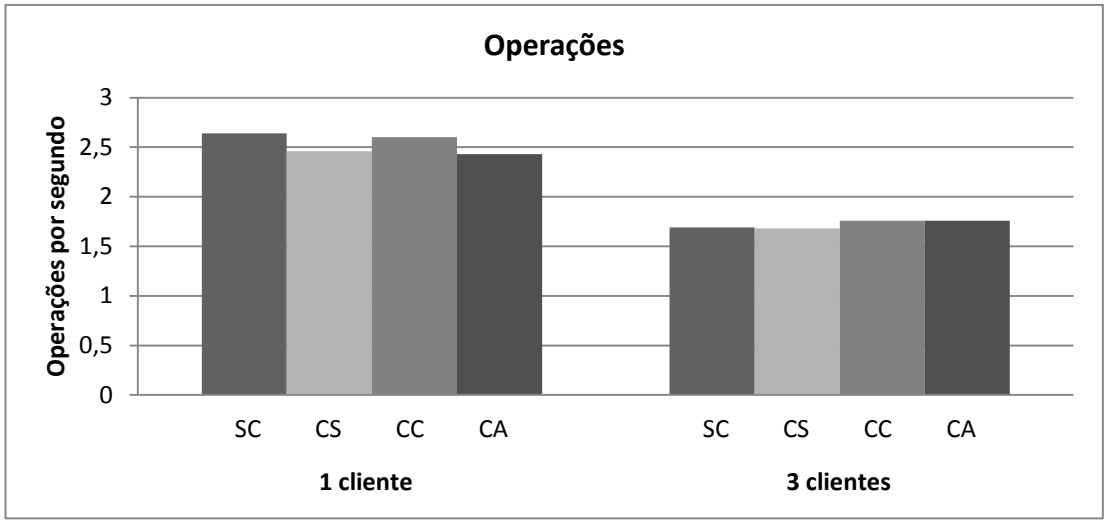

Figura 4.15: Operações realizadas por segundo Tahoe

O consumo médio de processamento ficou entre 0,65 e 0,75 segundos. Os resultados médios de cada situação para o tempo de uso da CPU podem ser vistos na Figura 4.16.

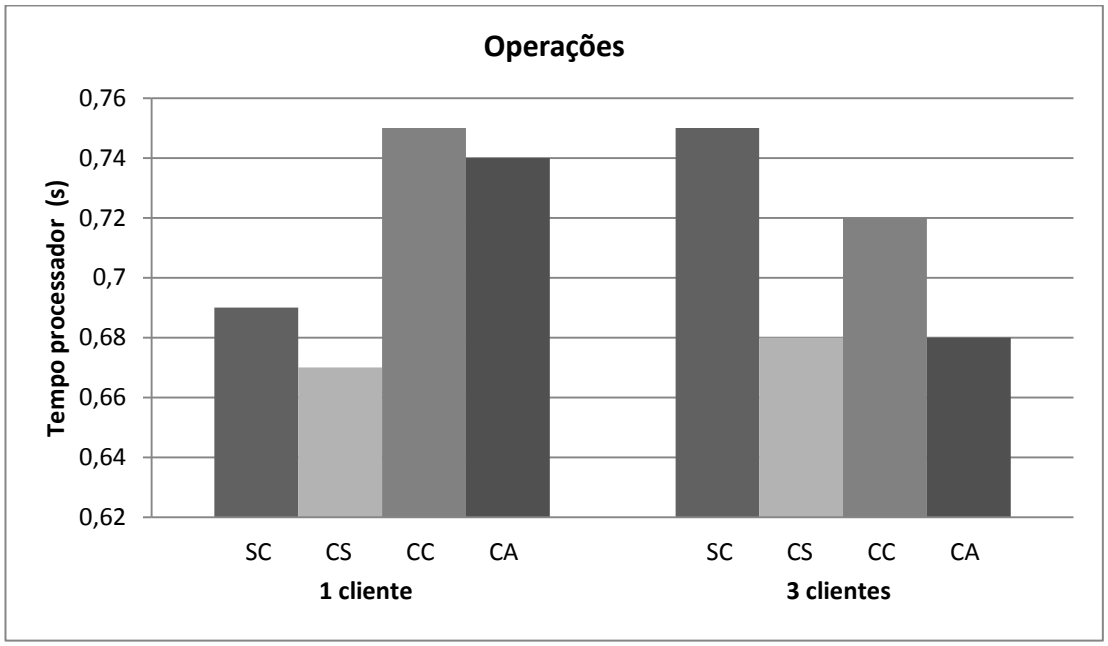

Figura 4.16: Tempo de processador gasto para realização de operações Tahoe

Da mesma maneira adotada para a exibição no NFS, a Figura 4.17 e Figura 4.18 exibem respectivamente o consumo máximo e médio do uso de CPU para o Tahoe.

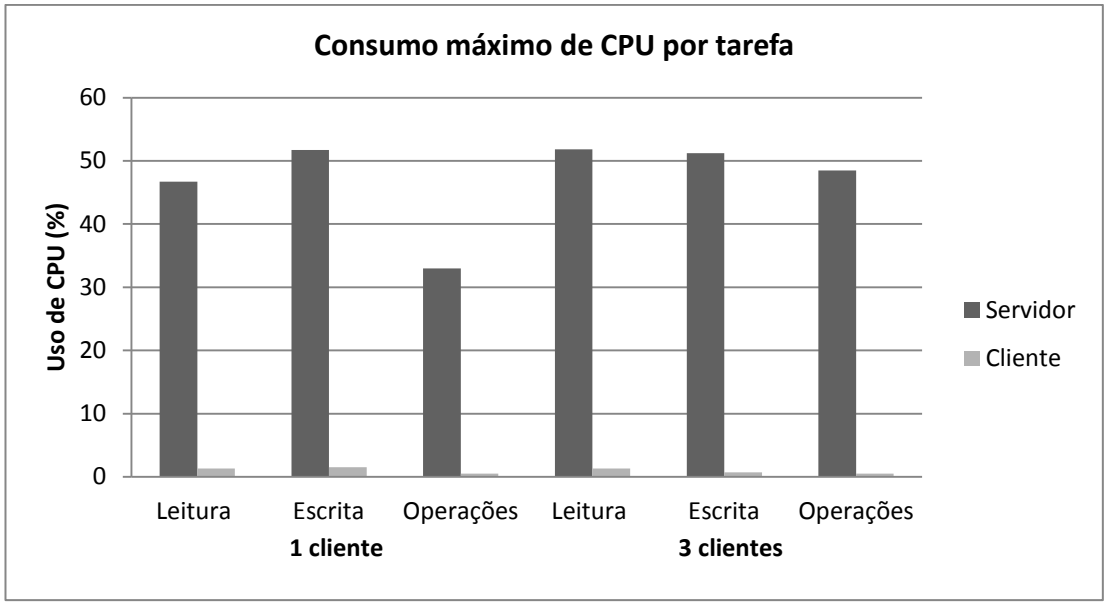

Figura 4.17: Consumo máximo de CPU por tarefa Tahoe

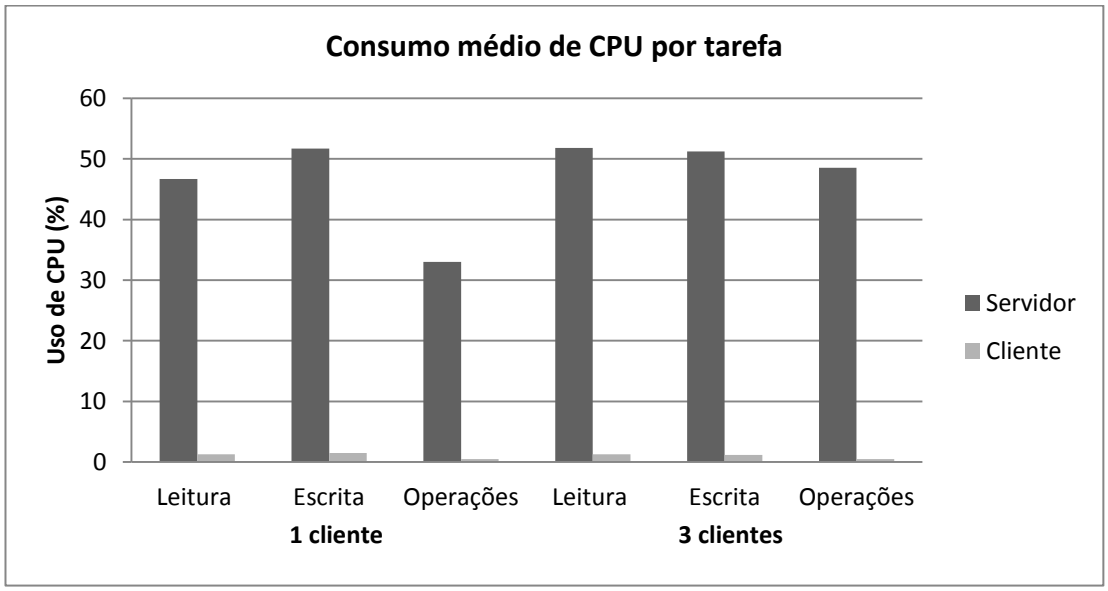

Figura 4.18: Consumo médio de CPU por tarefa Tahoe

Observa-se que no cliente, o consumo de recursos é muito baixo em todas as situações, enquanto o servidor realiza altas taxas de processamento durante todo o processo de leitura/escrita. Isso é explicado pela abordagem utilizada para as requisições de E/S, que são realizadas por meio de requisições HTTP diretamente para os servidores de armazenamento. Dessa forma, etapas de criptografia e codificação ficam por conta do servidor, além do envio das porções de arquivos para os demais servidores.

4.2 Avaliação em ambiente virtual

No ambiente virtual, foram instalados o HDFS, o sistema de arquivos em desenvolvimento e ainda o NFS. Este último foi instalado para se obter um parâmetro viável de comparação.

O cenário de testes utilizado aqui não tratou casos de concorrência de acesso e programação e nem foi feita a medição da taxa de consumo de memória RAM e CPU, por se tratar de um ambiente de máquinas virtuais, e devem ser analisados como parte de trabalhos futuros com a implantação dos sistemas em ambiente real.

4.2.1 NFS

A avaliação de desempenho do NFS em ambiente virtual foi executada a fim de se obter um parâmetro de comparação para o HDFS e o sistema de arquivos distribuídos em desenvolvimento. Os resultados para a leitura e escrita de arquivos são exibidos na Figura 4.19.

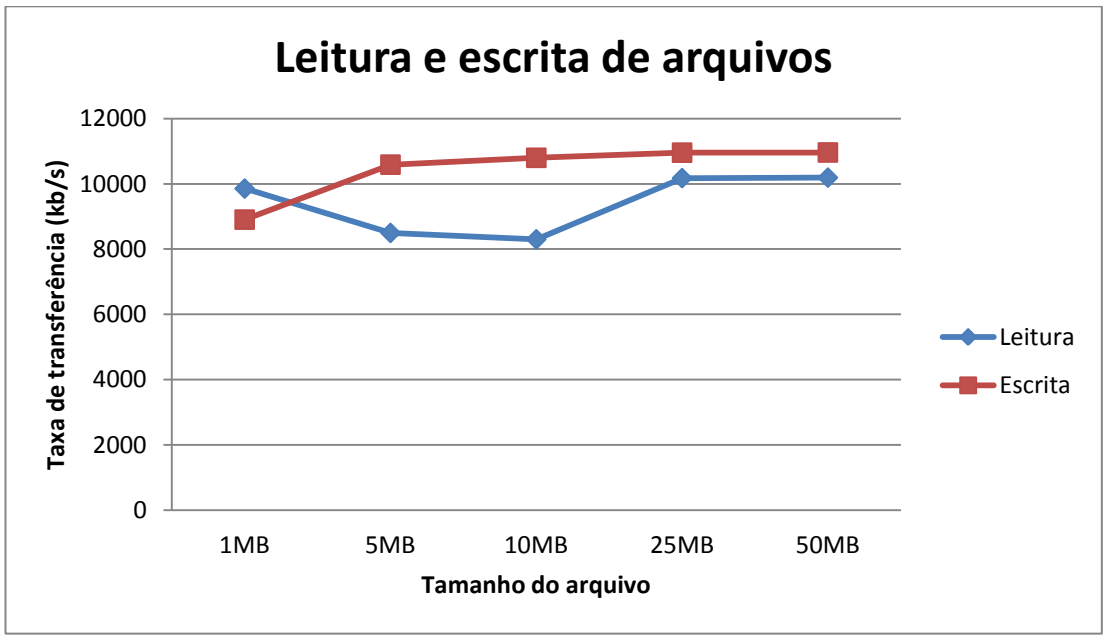

Figura 4.19: Leitura e escrita de arquivos sem concorrência de acesso NFS (virtual)

Em ambiente virtual, o NFS teve uma pequena oscilação no seu desempenho de leitura, tendo uma variação entre 8.300kb/s e 10.200kb/s. No ambiente real, esses resultados estavam todos na casa dos 11.000kb/s. O desempenho na escrita de arquivos seguiu caminho inverso, tendo aumentado a taxa de 8.000kb/s para resultados em torno de 9.000kb/s a 11.000kb/s.

Na etapa de operações realizadas, em média gastou-se 46 segundos para executar as 1430 operações, que fornece a taxa de praticamente 72 operações por segundo. Em relação ao ambiente real, o desempenho foi bastante inferior.

4.2.2 HDFS

O HDFS foi utilizado em ambiente virtual, visto que sua instalação em ambiente real foi mal-sucedida. Vários problemas com as dependências necessárias foram encontrados e não puderam ser contornados. Como alternativa já mencionada no Capitulo 3, uma imagem com a instalação do *framework* Cloudera, que contém o HDFS, foi utilizada.

Com o suporte ao FUSE instalado para o cliente HDFS, foi possível executar os mesmos *scripts* de avaliação utilizados no NFS, alterando apenas os diretórios de escrita onde o sistema foi montado. Os resultados para leitura e escrita do HDFS são mostrados na Figura 4.20.

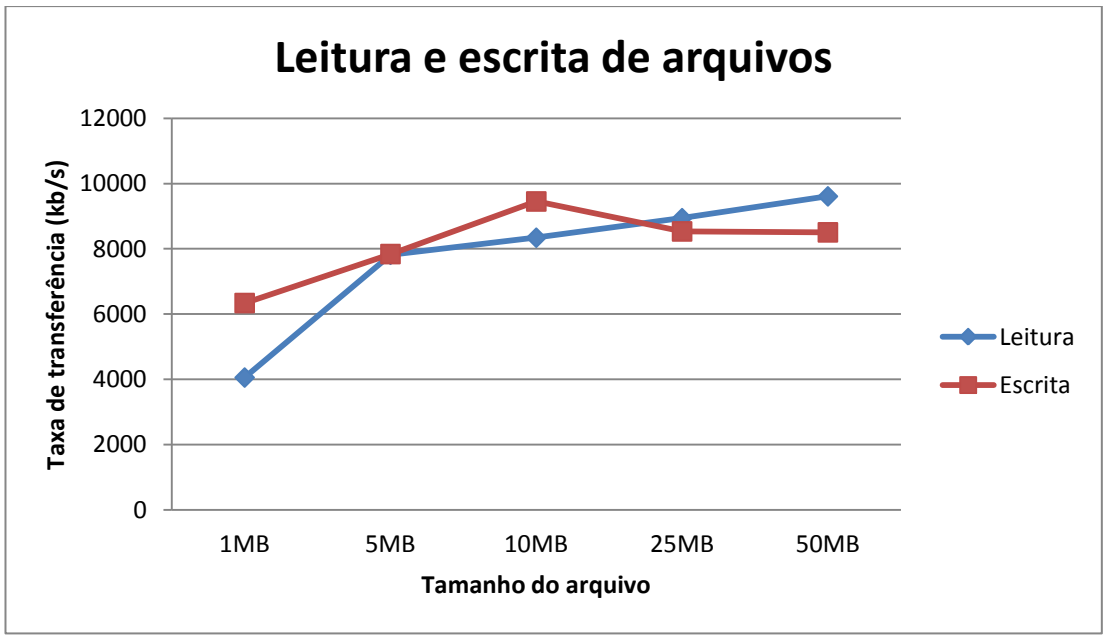

Figura 4.20: Leitura e escrita de arquivos sem concorrência de acesso HDFS (virtual)

Seu desempenho foi menor com arquivos de tamanho inferior a 10MB, e alcançou maiores taxas de transferências para os demais arquivos. Em alguns casos, houve uma grande variação nessa taxa, como por exemplo na leitura de arquivos de 5MB, onde oscilou entre 3.750kb/s e 9.250kb/s, e na escrita de arquivos de 1MB, onde oscilou entre 1.600kb/s e 9.000kb/s.

Em relação à etapa de operações, apesar do suporte ao FUSE permitir operações de mudança de diretórios (*chdir*), elas não foram utilizadas, visto que não são nativas do HDFS. Portanto, para executar as 1210 operações, em média o sistema levou 82 segundos, fornecendo uma taxa de aproximadamente 33 operações por segundo.

Entretanto, houve uma grande variação nos resultados. No melhor caso obteve-se uma taxa de 59,4 operações por segundo, enquanto no pior caso essa taxa foi de 5,79 operações por segundo.

4.2.3 Sistema de arquivos distribuídos em desenvolvimento

Os cenários de leitura/escrita foram aplicados utilizando-se aleatoriamente três servidores de dados num total de 10 para envio de arquivos. Assim como no Tahoe, uma estratégia de criptografia e codificação é utilizada no sistema de arquivos distribuídos em desenvolvimento. Os resultados obtidos podem ser verificados na Figura 4.21.

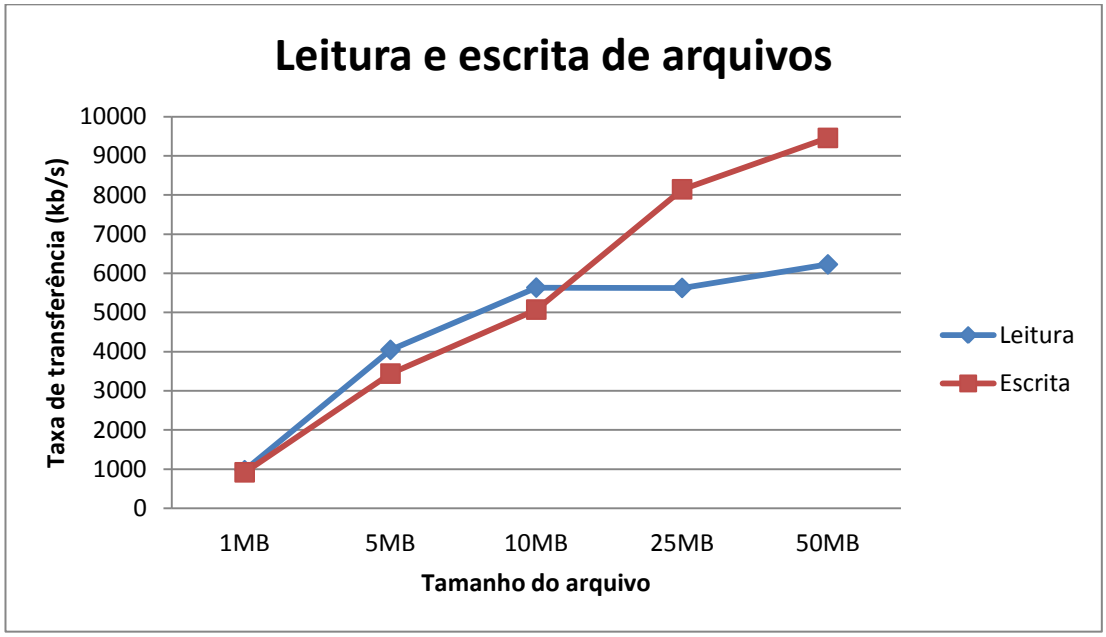

Figura 4.21: Leitura e escrita de arquivos sem concorrência de acesso do sistema de arquivos em desenvolvimento (virtual)

O desempenho para arquivos pequenos foi menor, visto que da mesma forma como acontece no Tahoe, uma estratégia de criptografia e codificação é utilizada. O desempenho do sistema aumenta conforme o volume de dados segue o mesmo caminho.

Não foi possível a aplicação da avaliação de operações no sistema de arquivos em desenvolvimento pois a organização da hierarquia de diretórios ainda não foi implementada, uma vez que este é um sistema de arquivos distribuídos em fase de desenvolvimento.

4.3 Análises comparativas

4.3.1 Ambiente real

Comparando os resultados obtidos entre o NFS e o Tahoe, verificados em ambiente real, é notável a grande diferença existente entre eles. Isso pode ser explicado por diversos fatores.

O NFS possui arquitetura de modelo cliente-servidor, e suas requisições são feitas por chamadas de procedimento remoto do cliente ao servidor. O Tahoe possui uma arquitetura híbrida, com suas requisições realizadas via HTTP aos servidores dados, o que aumenta o tempo de resposta a cada requisição.

Na Figura 4.22, o gráfico ilustra a diferença no desempenho de leitura de arquivos, enquanto a Figura 4.23 exibe essa diferença em relação à escrita. Ambos os casos tratam de acessos verificados sem situações de concorrência.

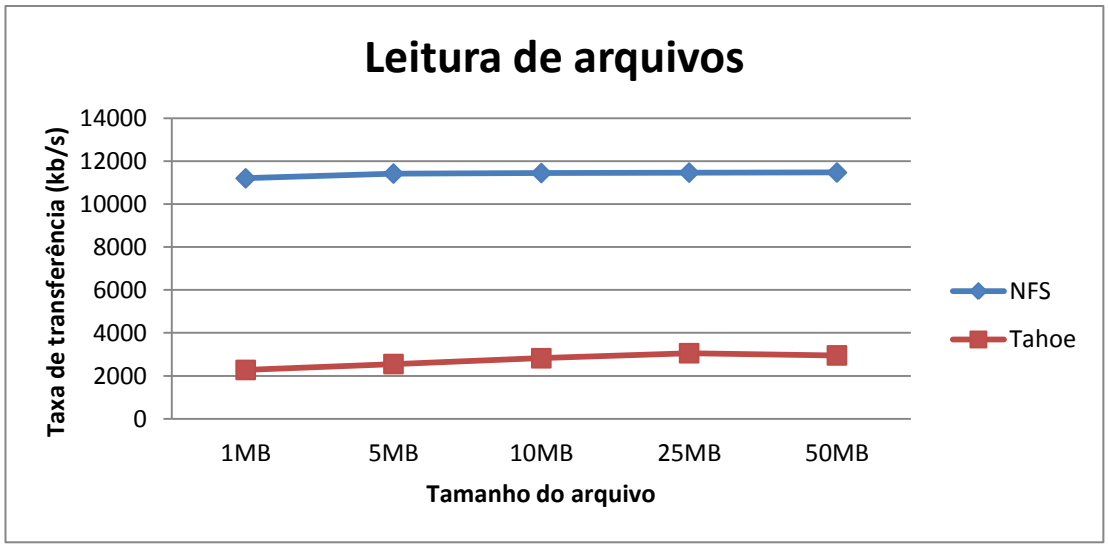

Figura 4.22: Leitura de arquivo NFS e Tahoe, sem concorrência

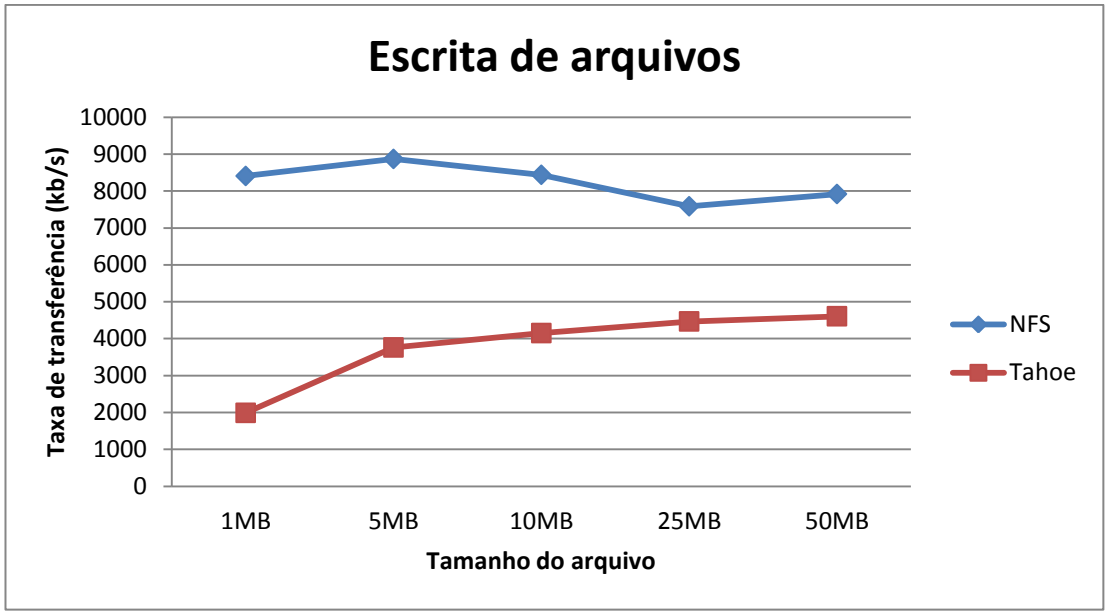

Figura 4.23: Escrita de arquivo NFS e Tahoe, sem concorrência

Entretanto, quando cenários de concorrência são inseridos, a oscilação dos resultados se dá de maneira distinta. No NFS, a oscilação foi maior, visto que todas as requisições e transferências são feitas em um único servidor de dados, caracterizando um gargalo. Para o Tahoe, as requisições são feitas para servidores distintos, distribuindo a carga de transferência pelos nós de armazenamento.

A Figura 4.24 ilustra a discrepância existente para a leitura concorrente de arquivos, enquanto a Figura 4.25 exibe para os acessos de escrita.

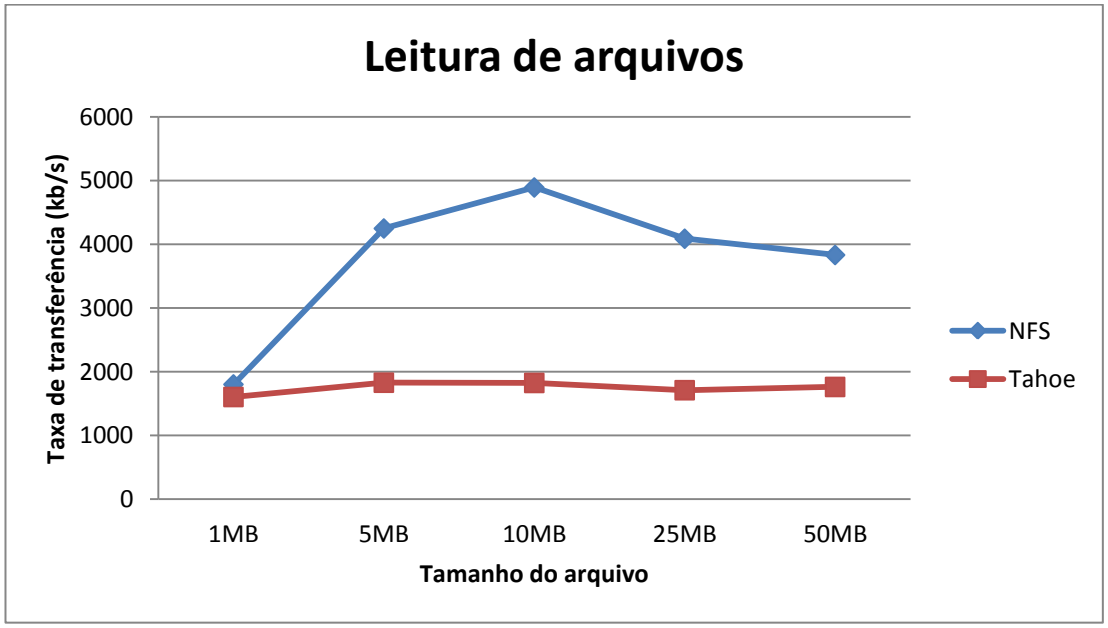

Figura 4.24: Leitura de arquivo NFS e Tahoe para três clientes simultâneos

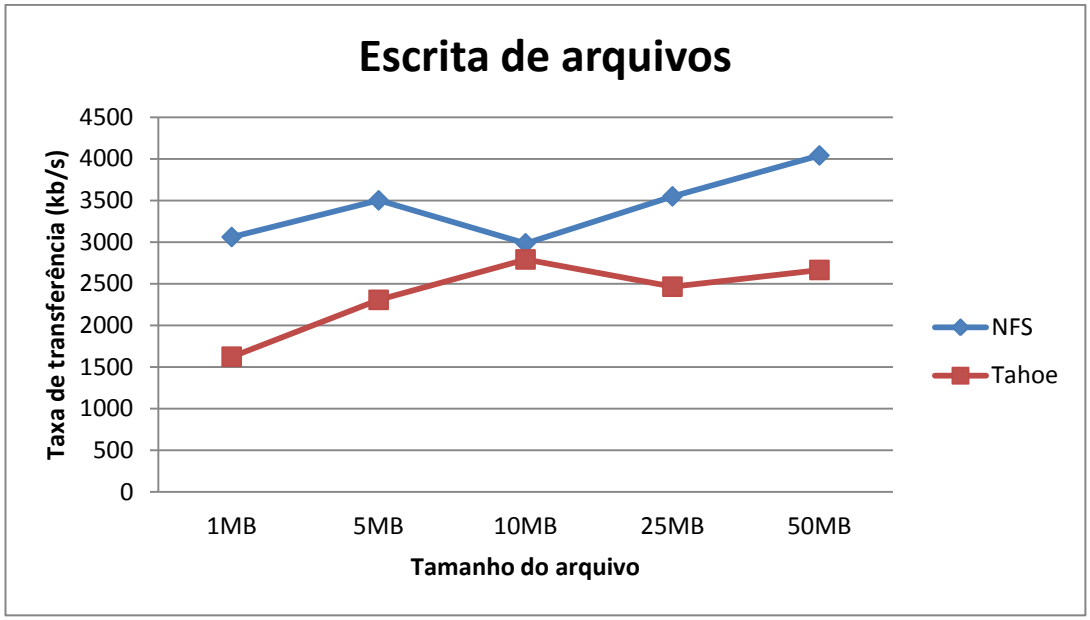

Figura 4.25: Escrita de arquivo NFS e Tahoe para três clientes simultâneos

Em relação ao número de operações realizadas por segundo, a diferença é ainda maior. Para acessos não concorrentes, em média o NFS obteve uma taxa de 589 operações por segundo, enquanto o Tahoe realizava 2,7 operações por segundo. Para a situação de acesso concorrente, o rendimento do NFS diminuiu para uma média de 126 operações por segundo, enquanto o Tahoe se manteve em torno de 1,7 operações realizadas por segundo.

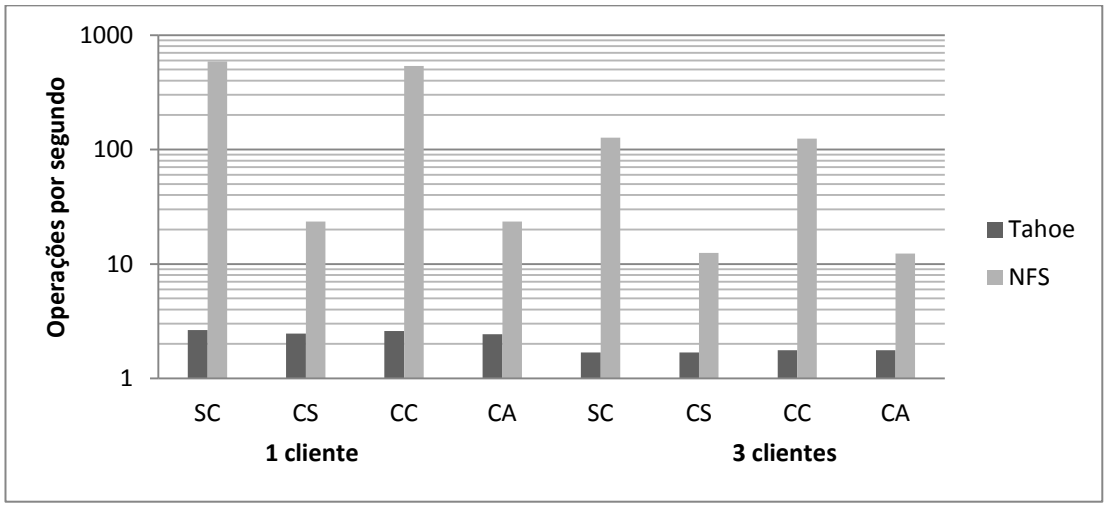

Figura 4.26: Operações realizadas por segundo NFS e Tahoe

Em situações de concorrência de processamento existentes no servidor, o NFS produz resultados muito inferiores em relação aos obtidos em situações sem concorrência. No Tahoe, este cenário não sofre grandes variações. Essa diferença

pode ser verificada na Figura 4.26. O eixo da métrica de operações realizadas por segundo está em escala logarítmica base 10 para uma melhor visualização.

Quanto ao consumo de recursos, o cliente NFS teve uma utilização ligeiramente maior do processador em relação ao Tahoe. Porém, quando visto no lado servidor, esses resultados se invertem e com uma maior discrepância. Isso se explica, pois o tratamento dado aos arquivos pelo Tahoe, como a criptografia e divisão dos arquivos em porções de dados, requer um maior gasto computacional que pela abordagem de acesso utilizada é atribuído ao servidor. No NFS, os arquivos são gravados diretamente no disco do servidor, recuperados e transferidos pela rede através de uma única conexão, sem nenhum tipo de tratamento especial.

4.3.2 Ambiente virtual

Os sistemas de arquivos distribuídos avaliados em ambiente virtual foram o NFS, HDFS e o sistema de arquivos distribuídos em desenvolvimento (SAD). Os resultados obtidos estão ilustrados através de gráficos, para leitura na Figura 4.27, e para a escrita na Figura 4.28.

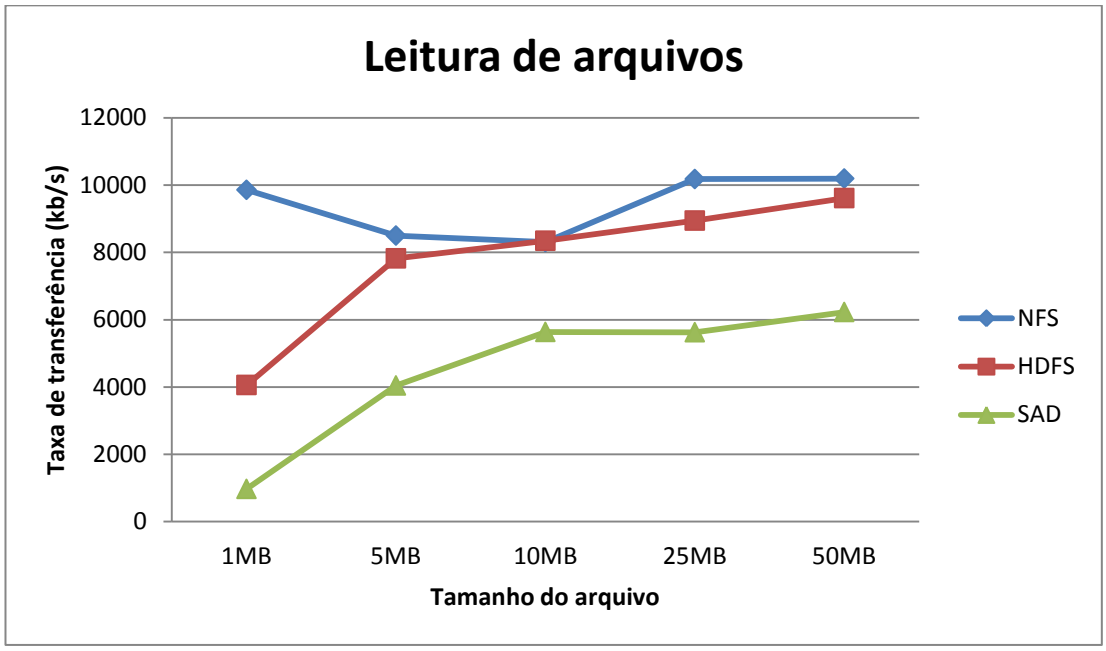

Figura 4.27: Leitura de arquivo NFS, HDFS e sistema de arquivos distribuídos em desenvolvimento

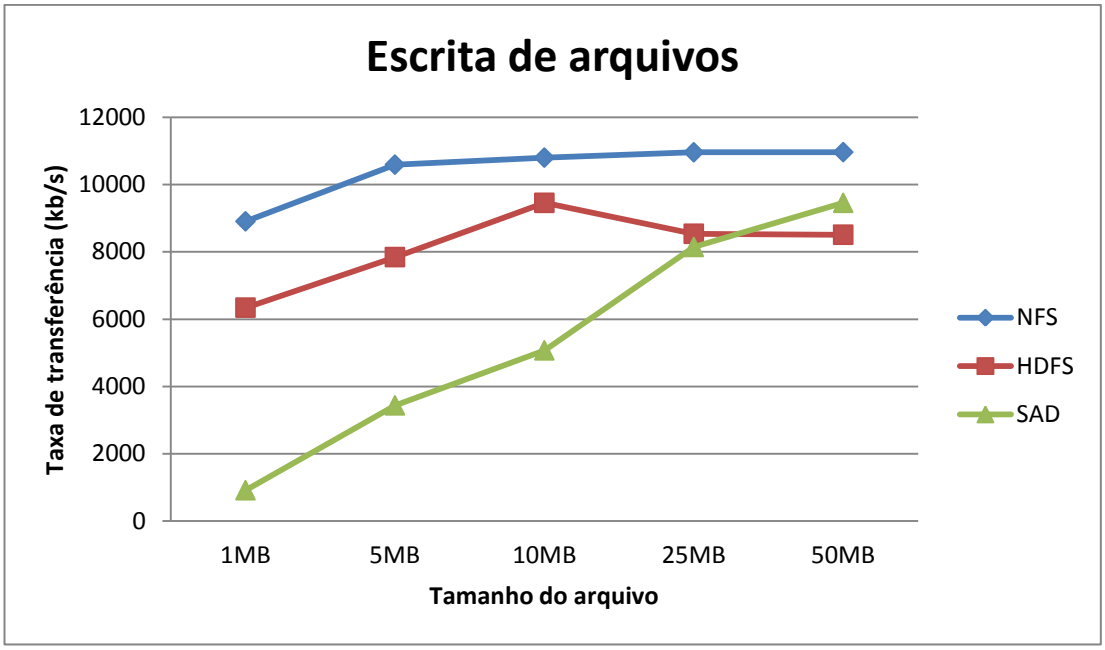

Figura 4.28: Escrita de arquivo NFS, HDFS e sistema de arquivos distribuídos em desenvolvimento

A instalação do HDFS se resumiu em 2 *DataNodes*, que são os servidores de dados, 1 *NameNode*, que é o servidor de informações do conteúdo dos *DataNodes* e um cliente, utilizado para acessar o sistema. Visto isso, o sistema se portou como um sistema cliente servidor, apenas alterando de acordo com os critérios do *NameNode* em qual servidor de dados armazenaria as informações. Dessa forma, em média seu desempenho foi parecido ao NFS, com uma perda maior em desempenho nos arquivos de 1MB.

Em ambos os casos, o desempenho do sistema de arquivos em desenvolvimento foi crescendo e ganhando uma maior estabilidade conforme aumentava o volume de dados. Em média, obteve taxa de transferência semelhante ao HDFS para arquivos de 25MB, e superior ao mesmo para arquivos de 50MB.

Na avaliação de operações, o NFS obteve uma taxa de 72 operações por segundo contra 33 do HDFS. Isso é explicado pela forma como os arquivos são armazenados em cada um desses sistemas de arquivos distribuídos. No NFS, eles são armazenados diretamente no disco, enquanto no HDFS são armazenados blocos de dados dos arquivos, e informações desses blocos são mantidas no *NameServer*.

Capítulo 5 – Conclusões

No presente trabalho foi interessante verificar o comportamento prático dos sistemas de arquivos distribuídos avaliados, trabalhando com as instalações e configurações de cada um.

O NFS obteve desempenho superior em ambos os cenários devido à sua simplicidade e facilidade de uso. Porém, o modelo cliente-servidor utilizado pode se tornar um gargalo com perda significativa de desempenho no sistema quando este aumentar significativamente o número de acessos concorrentes ao sistema, o que pôde ser verificado na prática.

No Tahoe, pode se destacar algumas características, como as etapas de criptografia e codificação aplicadas para o armazenamento dos dados. Elas são adotadas para garantir a integridade e confidencialidade dos arquivos. Além disso, a codificação gera uma distribuição de porções dos arquivos ao longo dos servidores de armazenamento para garantir uma maior disponibilidade dos mesmos. Entretanto, tais etapas exigem maior quantidade de processamento.

A codificação utilizada no Tahoe aumenta o volume de dados e o tempo gasto na transferência dos arquivos para o sistema de arquivos distribuído, pois este processo divide o arquivo em dez porções de mesmo tamanho de forma que quaisquer três utilizadas remontam o arquivo original. Dessa forma, o volume de dados cresce num fator de 3,3 vezes do tamanho original.

O sistema de arquivos distribuídos em desenvolvimento foi incluído no projeto e utilizado em ambiente virtual, pois ele está ainda em fase de desenvolvimento. Desse modo, não foi possível sua instalação no ambiente

distribuído do *cluster Beowulf*. Seu protótipo ainda está com funcionalidades limitadas, não fornecendo, por exemplo, suporte à estrutura de diretórios.

Mesmo não sendo instalado em ambiente real, e utilizado uma configuração limitada em ambiente virtual, obteve-se uma idéia do funcionamento do HDFS. As limitações de instalação em ambiente virtual se deram pelo fato dos processadores das máquinas do *cluster* não fornecerem suporte à virtualização de 64 *bits*. Com isso, a instalação foi possível apenas nas maquias máquinas que forneciam tal suporte, as máquinas *camus*, *milo* e *shaka*.

Sugere-se a instalação bem-sucedida do HDFS no ambiente distribuído do *cluster*, com maior quantidade de servidores de armazenamento e mais clientes realizando acessos concorrentes ao sistema, para verificar como esse se comportaria nesse tipo de situação. Dessa forma, poderia ser aplicado o cenário completo de avaliação que foi definido neste trabalho.

A requisição de transferência de arquivos no Tahoe é bastante flexível, podendo utilizar diferentes abordagens para tal. No presente trabalho foi utilizada a abordagem de requisições HTTP realizadas diretamente no servidor, transferindo para ele as etapas de criptografia, codificação e envio das porções de arquivos aos demais servidores. Fica a sugestão para trabalhos futuros da utilização de diferentes abordagens de requisições de arquivos, através da interface Web ou linha de comando. Dessa forma, a carga de processamento da criptografia, codificação e transferências de porções de arquivos serão avaliadas com comportamentos distintos.

Para evitar dificuldades de implantação dos sistemas reais para a avaliação de desempenho, uma sugestão para trabalhos futuros é utilizar técnicas de modelagem para efetuar tal tarefa. Dessa forma, problemas com instalação, configuração e até mesmo equipamentos necessários podem ser contornados de maneira alternativa.

Este trabalho contribui com a definição dos cenários de avaliação utilizados neste trabalho, para o método de aferição. Além disso, verificou-se que a modelagem não seria viável no presente projeto, por questões de tempo e complexidade do trabalho.

Essas informações podem ser utilizadas na proposta de um novo *benchmark*, ou ainda na proposta de um modelo que possa ser resolvido por simulação.

REFERÊNCIAS BIBLIOGRÁFICAS

[Barbosa Jr. et al. 2007] BARBOSA JR, A. A.; GREVE, F.; BARRETO, L. P. 2007. **Avaliação de Sistemas de Arquivos Distribuídos num Ambiente de Pequena Escala.** In *XXVII Congresso da SBC, IV Workshop de Sistemas Operacionais, Rio de Janeiro – RJ. Anais,* p. 852-862.

[Boito 2009] BOITO, F. Z. 2009. **Estratégias para avaliação do desempenho do sistema de arquivos Lustre.** Trabalho de Conclusão de Curso, Instituto de Informática da Universidade Federal do Rio Grande do Sul.

[Cloudera 2011] Hadoop | Hadoop Download | Cloudera Hadoop | Cloudera, http://www.cloudera.com/. Acessado em: nov. 2011

[Costa 2008] COSTA, F. 2008. ***Ambiente de rede monitorado com Nagios e Cacti.*** Editora Ciência Moderna.

[Coulouris et al. 2005] COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. (2005). ***Distributed systems: concepts and design.*** Addison-Wesley.

[Eisler 2000] EISLER, M. **RFC 2847: LIPKEY – A Low infrasctructure Public Key Mechanism Using SPKM.** Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. jun. 2000. Disponível em: <http://www.ietf.org/rfc/rfc2847>. Acessado em: out. 2011.

[Eisler et al. 1997] EISLER, M. *et al*. **RFC 2203: RPCSEC_GSS Protocol Specification.** Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. sep. 1997. Disponível em: <http://www.ietf.org/rfc/rfc2203>. Acessado em: out. 2011.

[Fernandes 2011] FERNANDES, S. E. N.; and LOBATO, R. S. 2011. **Sistema de Arquivos Distribuído no Espaço do Usuário**. In *Escola Regional de Alto Desempenho de São Paulo. ERAD-SP*.

[Fujimura et al. 2008] FUJIMURA, A.; OH, S.; and Gerla, M. 2008. **Network coding vs. erasure coding: Reliable multicast in ad hoc networks**. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1-7.

[Ghemwat 2003] Ghemawat, S.; Gobioff, H.; Leung, S. **The Google File System** In *ACM Symposium on Operating Systems Principles, 19, 2003. Lake George NY*. P1-11.

[Hadoop 2011] Apache™ Hadoop™. http://hadoop.apache.org/. Acessado em: nov. 2011.

[htop 2011] htop - an interactive process viewer for Linux, http://htop.sourceforge.net/. Acessado em: nov. 2011.

[Huang and Grimshaw 2010] Huang, H. H. and Grimshaw, A. S. 2011. **Design, implementation and evaluation of a virtual storage system**. *Concurr. Comput. : Pract. Exper.*, 23:311–331.

[IOzone 2011] IOzone Filesystem Benchmark, http://iozone.org/. Acessado em: nov. 2011

[Kassick 2008] KASSICK, R. V.; BOITO, F. Z.; NAVAUX, P. O. A. 2008. **Testing the performance of parallel file systems**. In: *VI Workshop de Processamento Paralelo e Distribuído*, 2008, *Porto Alegre - RS. Anais.*

[Kohl 1993] KOHL, J.; NEUMAN, C. **RFC 1510: The Kerberos Network Authentication Service (V5).** Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. set. 1993. Disponível em: <http://www.ietf.org/rfc/rfc1510>. Acessado em: out. 2011.

[Kon 1994] KON, F. **Sistemas de arquivos distribuídos.** *Master's thesis*, Instituto de Matemática e Estatística da Universidade de São Paulo, 1994.

[Kshemkalyani and Singhal 2008] ***Distributed Computing: principles, algorithms and systems***. Cambridge University Press.

[Lamport 1978] Lamport, L. 1978. **Time, Clocks, and the ordering of events in a distributed system.** *Commun., ACM*, 21:558-565.

[Machado 2008] MACHADO, D.J. 2008. **algSim – Linguagem algorítmica para simulação de redes de filas.** Trabalho de Conclusão de Curso, Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, São José do Rio Preto.

[Plank et al. 2009] PLANK, J. S.; LUO, J.; SCHUMAN, C. D.; XU, L.; and WILCOX-O'HEARN, Z. 2009. **A performance evaluation and examination of open-source erasure coding libraries for storage**. In *Proceedings of the 7th conference on File and storage technologies. USENIX Association*.

[Python 2011] Python Programming Language – Official Website, http://python.org/

[Shepler et al. 2000] SHEPLER, S. et al. **RFC 3010: NFS version 4 Protocol.** Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. dec. 2000. Disponível em: <http://www.ietf.org/rfc/rfc3010 >. Acessado em: out. 2011.

[Shepler et al. 2003] SHEPLER, S. et al. **RFC 3530: Network File System (NFS) version 4 Protocol.** Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. apr. 2003. Disponível em: <http://www.ietf.org/rfc/rfc3530>. Acessado em: out. 2011.

[Shvachko et al. 2010] SHVACHKO, K.; KUANG, H.; RADIA, S.; and CHANSLER, R. 2010. **The hadoop distributed file system**. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*.

[Tahoe 2011] Tahoe-lafs. http://tahoe-lafs.org/. Acessado em: nov. 2011

[Tanenbaum and Steen 2007] TANENBAUM, A. S.; STEEN, M. V. (2007). ***Distributed systems: principles and paradigms***. Pearson Prentice Hall.

[Vieira 2010] VIEIRA, M. F. 2010. **Análise de desempenho entre os sistemas de arquivo HDFS e LustreFS**, Universidade Luterana do Brasil, Relatório Técnico.

[Wilcox-O'Hearn 2008] WILCOX-O'HEARN, Z.; WARNER, B. 2008. **Tahoe – The Least-Authority Filesystem**. In *Proceedings of the 4th ACM international workshop on Storage security and survivability, StorageSS '08*, pages 21-26. ACM.

APÊNDICE A - Códigos-fonte dos *scripts* de avaliação

Neste apêndice serão exibidos todos os códigos-fonte utilizados para efetuar a avaliação de desempenho dos sistemas de arquivos distribuídos no presente trabalho.

Leitura NFS

```
#>script para avaliacao de leitura
#>diretorio utilizado: /shared/nfs/
#>arquivos e diretorios necessarios:
#/shared/nfs/{1024kb,5120kb,10240kb,25600kb,51200kb} >
arquivos a serem lidos
#/tmp/saidas/ > diretorio para a escrita dos resultados
#>executar como ./python read_nfs.py <arq_saida>

import os, sys, time
stats_out = sys.argv[1]
readfiles = [1024,5120,10240,25600,51200]
os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
os.chdir('/shared/nfs/')

for size in readfiles:
    filename = str(size)+'kb'
    clock_start = time.clock()
    time_start = time.time()
    file = open(filename, 'r')
    file.read()
    elapsed_clock = time.clock() - clock_start
    elapsed_time = time.time() - time_start
    avg_time = size/elapsed_time
    f.write("filesize: %d\n" % size)
```

```
    f.write("time elapsed: %f\n" % elapsed_time)
    f.write("clock elapsed: %f\n" % elapsed_clock)
    f.write("transfer rate: %f\n\n" % avg_time)

f.write("time in seconds, clock in seconds, transfer rate
in kbps\n")
f.close()
os.chdir('/tmp/saidas/')
os.rename(stats_out+".tmp", stats_out)
```

Escrita NFS

```
#>script para avaliacao de escrita
#>diretorio utilizado: /shared/nfs/
#>arquivos e diretorios necessarios:
#/tmp/files/{1024kb,5120kb,10240kb,25600kb,51200kb} >
arquivos a serem escritos
#/tmp/saidas/ > diretorio para a escrita dos resultados
#>executar como ./python write_nfs.py <arq_saida>

import os, sys, shutil, time
stats_out = sys.argv[1]
writefiles = [1024,5120,10240,25600,51200]
os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
os.chdir('/shared/nfs/')

for size in writefiles:
    filename = '/tmp/files/'+str(size)+'kb'
    dest = str(size)+'kb'
    start_clock, start_time = time.clock(), time.time()
    shutil.copyfile(filename,dest)
    t_up_clock = time.clock() - start_clock
    t_up_time = time.time() - start_time
    kbps = float(size/t_up_time)
    f.write("filesize: %d kb\n" % size )
    f.write("time elapsed: %f\n" % t_up_time)
    f.write("clock elapsed: %f\n" % t_up_clock)
    f.write("transfer rate: %f\n\n" % kbps)

f.write("time in seconds, clock in seconds, transfer
rate in kbps\n")
f.close()
os.chdir('/tmp/saidas/')
os.rename(stats_out+".tmp", stats_out)
```

Operações NFS

```
#>script para avaliacao de operacoes por segundo NFS
#>diretorio utilizada: /shared/nfs/
```

```
#>arquivos e diretorios necessarios:
#/tmp/files/1kb> arquivo de 1kb
#/tmp/saidas/> diretorio para a escrita dos resultados
#>executar como ./$python operations_nfs.py <arq_saida>

import os, sys, shutil, time
stats_out = sys.argv[1]
lld = ['dirA','dirB','dirC','dirD','dirE',
       'dirF','dirG','dirH','dirI','dirJ']
tld = ['dir1','dir2','dir3','dir4','dir5',
       'dir6','dir7','dir8','dir9','dir0']
os.chdir('/shared/nfs/')
clock_start = time.clock()
time_start = time.time()
for lowlevel in lld:
    os.mkdir(lowlevel)
    os.chdir(lowlevel)

    for a in range(10):
        test = lowlevel+str(a)+'.test'
        shutil.copyfile('/tmp/files/1kb',test)

    for toplevel in tld:
        os.mkdir(toplevel)
        os.chdir(toplevel)

        for b in range(10):
            shutil.copyfile('/tmp/files/1kb',
                            toplevel+str(b)+'.test')
        os.chdir('..')
    os.chdir('..')

elapsed_clock = time.clock() - clock_start
elapsed_time = time.time() - time_start
avg_time = 1430/elapsed_time

os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
f.write("time/clock/operationspersec\n\n")
f.write("%f\n" % elapsed_time)
f.write("%f\n" % elapsed_clock)
f.write("%f\n" % avg_time)
f.close()
os.rename(stats_out+".tmp", stats_out)
```

Leitura Tahoe

```
#>script para avaliacao de leitura Tahoe
#>diretorio utilizado: passado no arquivo ./root.cap
#>arquivos e diretorios necessarios:
```

```
#./root.cap>arquivo com o dircap do diretorio de escrita
#./server-URLs>arquivo com IP:porta dos storage nodes
#>executar como ./python read_tahoe.py <arq_saida>
import os, sys, httplib, urllib, random, time, urlparse
import json as simplejson

stats_out = sys.argv[1]
server_urls = []
for url in open("server-URLs", "r").readlines():
    url = url.strip()
    if url:
        server_urls.append(url)
root = open("root.cap", "r").read().strip()
filenames = [1024,5120,10240,25600,51200]

def read_and_discard(nodeurl, root, pathname):
    if nodeurl[-1] != "/":
        nodeurl += "/"
    url = nodeurl + "uri/%s/" % urllib.quote(root)
    if pathname:
        url += urllib.quote(pathname)
    f = urllib.urlopen(url)
    while True:
        data = f.read(4096)
        if not data:
            break
f = open(stats_out+".tmp", "w")
for size in filenames:
    pathname = str(size)+'kb'
    server = random.choice(server_urls)
    print "reading", pathname
    start_clock, start_time = time.clock(), time.time()
    read_and_discard(server, root, pathname)
    t_down_clock = time.clock() - start_clock
    t_down_time = time.time() - start_time
    kbps = float(size/t_down_time)
    f.write("file: %s\n" % pathname )
    f.write("time elapsed: %f\n" % t_down_time)
    f.write("clock elapsed: %f\n" % t_down_clock)
    f.write("transfer rate: %f\n\n" % kbps)
f.write("time & clock in seconds, transfer rate in
kbps\n")
f.close()
os.rename(stats_out+".tmp", stats_out)
```

Escrita Tahoe

```
#>script para avaliacao de escrita
#>diretorio utilizado: passado no arquivo ./root.cap
#>arquivos e diretorios necessarios:
```

```
#./root.cap > arquivo com o capability do diretorio de
escrita
#./server-URLs > arquivo com end. dos storage nodes
#>executar como ./python write_tahoe.py <arq_saida>

import os, sys, httplib
import json as simplejson
import urllib, random, time, urlparse

stats_out = sys.argv[1]
server_urls = []
for url in open("server-URLs", "r").readlines():
    url = url.strip()
    if url:
        server_urls.append(url)
root = open("root.cap", "r").read().strip()
filesizes = [1048576,5242880,10485760,26214400,52428800]
#filesizes: vetor de tamanhos de arquivos
(1mb,5mb,10mb,25mb,50mb), em bytes

def parse_url(url, defaultPort=None):
    url = url.strip()
    parsed = urlparse.urlparse(url)
    scheme = parsed[0]
    path = urlparse.urlunparse(('','')+parsed[2:])
    if defaultPort is None:
        if scheme == 'https':
            defaultPort = 443
        else:
            defaultPort = 80
    host, port = parsed[1], defaultPort
    if ':' in host:
        host, port = host.split(':')
        port = int(port)
    if path == "":
        path = "/"
    return scheme, host, port, path

def generate_and_put(nodeurl, root, remote_filename,
size):
    url = nodeurl + "uri/%s/" % urllib.quote(root)
    url += urllib.quote(remote_filename)
    scheme, host, port, path = parse_url(url)
    if scheme == "http":
        c = httplib.HTTPConnection(host, port)
    elif scheme == "https":
        c = httplib.HTTPSConnection(host, port)
    else:
        raise ValueError("unknown scheme '%s', need http
or https" % scheme)
```

```
    c.putrequest("PUT", path)
    c.putheader("Hostname", host)
    c.putheader("User-Agent", "tahoe-check-load")
    c.putheader("Connection", "close")
    c.putheader("Content-Length", "%d" % size)
    c.endheaders()
    while size:
        chunksize = min(size, 4096)
        size -= chunksize
        c.send("\x00" * chunksize)
    return c.getresponse()
current_writedir = "/"
f = open(stats_out+".tmp", "w")

for size in filesizes:
    pathname = str(size/1024)+"kb"
    print "  writing", pathname
    server = random.choice(server_urls)
    start_clock, start_time = time.clock(), time.time()
    generate_and_put(server, root, pathname, size)
    t_up_clock = time.clock() - start_clock
    t_up_time = time.time() - start_time
    size = size/1024
    kbps = float(size/t_up_time)
    f.write("filesize: %d kb\n" % size )
    f.write("time elapsed: %f\n" % t_up_time)
    f.write("clock elapsed: %f\n" % t_up_clock)
    f.write("transfer rate: %f\n\n" % kbps)
f.write("time in seconds, clock in seconds, transfer
rate in kbps\n")
f.close()
os.rename(stats_out+".tmp", stats_out)
```

Operações Tahoe

```
#>script para avaliacao de operacoes por segundo
#>diretorio utilizado: passado no arquivo ./root.cap
#>arquivos e diretorios necessarios:
#./root.cap > arquivo com o capability do diretorio de
escrita
#./server-URLs > arquivo com end. dos storage nodes
#>executar como ./python operations_tahoe.py <arq_saida>

import os, sys, httplib
import json as simplejson
import urllib, random, time, urlparse
stats_out = sys.argv[1]
server_urls = []
for url in open("server-URLs", "r").readlines():
    url = url.strip()
```

```
    if url:
        server_urls.append(url)
root = open("root.cap", "r").read().strip()

lld = ['dirA','dirB','dirC','dirD','dirE',
       'dirF','dirG','dirH','dirI','dirJ']
tld = ['/dir1','/dir2','/dir3','/dir4','/dir5',
       '/dir6','/dir7','/dir8','/dir9','/dir0']

def parse_url(url, defaultPort=None):
    url = url.strip()
    parsed = urlparse.urlparse(url)
    scheme = parsed[0]
    path = urlparse.urlunparse(('','')+parsed[2:])
    if defaultPort is None:
        if scheme == 'https':
            defaultPort = 443
        else:
            defaultPort = 80
    host, port = parsed[1], defaultPort
    if ':' in host:
        host, port = host.split(':')
        port = int(port)
    if path == "":
        path = "/"
    return scheme, host, port, path

def generate_and_put(nodeurl, root, remote_filename):
    if nodeurl[-1] != "/":
        nodeurl += "/"
    url = nodeurl + "uri/%s/" % urllib.quote(root)
    url += urllib.quote(remote_filename)
    scheme, host, port, path = parse_url(url)
    if scheme == "http":
        c = httplib.HTTPConnection(host, port)
    elif scheme == "https":
        c = httplib.HTTPSConnection(host, port)
    else:
        raise ValueError("unknown scheme '%s', need http
or https" % scheme)
    c.putrequest("PUT", path)
    c.putheader("Hostname", host)
    c.putheader("User-Agent", "tahoe-check-load")
    c.putheader("Connection", "close")
    c.putheader("Content-Length", "%d" % 1024)
    c.endheaders()
    c.send("\x00" * 1024)
    return c.getresponse()

clock_start = time.clock()
```

```
time_start = time.time()

for lowlevel in lld:

    for a in range(10):
        server = random.choice(server_urls)
        current_writedir = lowlevel+"/"+str(a)
        generate_and_put(server, root, current_writedir)

    for toplevel in tld:

        for b in range(10):
            server = random.choice(server_urls)
            current_writedir = lowlevel + toplevel + "/"
                                + str(b)
            generate_and_put(server, root,
                             current_writedir)

elapsed_clock = time.clock() - clock_start
elapsed_time = time.time() - time_start

avg_time = 1210/elapsed_time

f = open(stats_out+".tmp", "w")
f.write("time/clock/operationspersec\n\n")
f.write("%f\n" % elapsed_time)
f.write("%f\n" % elapsed_clock)
f.write("%f\n" % avg_time)
f.close()
os.rename(stats_out+".tmp", stats_out)
```

Leitura HDFS

```
#>script para avaliacao de leitura
#>diretorio utilizado: /shared/hadoop/test/
#>arquivos e diretorios necessarios:
#/shared/hadoop/test/{1024kb,5120kb,10240kb,25600kb,51200
kb} > arquivos a serem lidos
#/tmp/saidas/ > diretorio para a escrita dos resultados
#>executar como ./python read_nfs.py <arq_saida>

import os, sys, time
stats_out = sys.argv[1]
readfiles = [1024,5120,10240,25600,51200]
os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
os.chdir('/shared/hadoop/test/')

for size in readfiles:
    filename = str(size)+'kb'
```

```
    clock_start, time_start = time.clock(), time.time()
    file = open(filename, 'r')
    file.read()
    elapsed_clock = time.clock() - clock_start
    elapsed_time = time.time() - time_start
    avg_time = size/elapsed_time
    f.write(" size %d\n" % size)
    f.write(" time %f\n" % elapsed_time)
    f.write("clock %f\n" % elapsed_clock)
    f.write("trate %f\n\n" % avg_time)

f.write("time in seconds, clock in seconds, transfer rate
in kbps\n")
f.close()
os.chdir('/tmp/saidas/')
os.rename(stats_out+".tmp", stats_out)
```

Escrita HDFS

```
#>script para avaliacao de escrita
#>diretorio utilizado: /shared/hadoop/test/
#>arquivos e diretorios necessarios:
#/shared/hadoop/test/{1024kb,5120kb,10240kb,25600kb,51200
kb} > arquivos a serem escritos
#/tmp/saidas/ > diretorio para a escrita dos resultados
#>executar como ./python write_nfs.py <arq_saida>

import os, sys, shutil, time
stats_out = sys.argv[1]
writefiles = [1024,5120,10240,25600,51200]
os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
os.chdir('/shared/hadoop/test/')

for size in writefiles:
    filename = '/tmp/files/'+str(size)+'kb'
    dest = str(size)+'kb'
    start_clock, start_time = time.clock(), time.time()
    shutil.copyfile(filename,dest)
    t_up_clock = time.clock() - start_clock
    t_up_time = time.time() - start_time
    kbps = float(size/t_up_time)
    f.write(" size %d\n" % size )
    f.write(" time %f\n" % t_up_time)
    f.write("clock %f\n" % t_up_clock)
    f.write("trate %f\n\n" % kbps)

f.write("time in seconds, clock in seconds, transfer rate
in kbps\n")
f.close()
```

```
os.chdir('/tmp/saidas/')
os.rename(stats_out+".tmp", stats_out)
```

Operações HDFS

```
#>script para avaliacao de operacoes por segundo
#>diretorio de escrita: /shared/hadoop/test/
#>arquivos e diretorios necessarios:
#/tmp/files/1kb > arquivo de 1kb
#/tmp/saidas/ > diretorio de resultados
#>executar como ./python operations_nfs.py <arq_saida>

import os, sys, shutil, time
stats_out = sys.argv[1]
lld = ['dirA','dirB','dirC','dirD','dirE',
       'dirF','dirG','dirH','dirI','dirJ']
tld = ['/dir1','/dir2','/dir3','/dir4','/dir5',
       '/dir6','/dir7','/dir8','/dir9','/dir0']

os.chdir('/shared/hadoop/test/')
clock_start = time.clock()
time_start = time.time()

for lowlevel in lld:
    os.mkdir(lowlevel)

    for a in range(10):
        shutil.copyfile('/tmp/files/1kb',lowlevel
                        +'/'+lowlevel+str(a)+'.test')

    for toplevel in tld:
        os.mkdir(lowlevel+toplevel)

        for b in range(10):
            shutil.copyfile('/tmp/files/1kb',lowlevel
                            +toplevel+'/'+toplevel
                            +str(b)+'.test')

elapsed_clock = time.clock() - clock_start
elapsed_time = time.time() - time_start

avg_time = 1210/elapsed_time

os.chdir('/tmp/saidas/')
f = open(stats_out+".tmp", "w")
f.write(" time %f\n" % elapsed_time)
f.write("clock %f\n" % elapsed_clock)
f.write(" op/s %f\n" % avg_time)
f.close()
os.rename(stats_out+".tmp", stats_out)
```

APÊNDICE B - Tabelas de resultados

Neste apêndice serão exibidos as tabelas resumidas dos resultados, e também todas as tabelas com os resultados completos para cada iteração de cada um dos cenários de avaliação aplicados neste trabalho.

Tabela 1: Leitura de arquivos NFS (taxa de transferência, em KB/s)

Concorrência		Tamanho do Arquivo				
Acesso	Proc.	1MB	5MB	10MB	25MB	50MB
1 Cliente	SC	11201,37	11422,43	11448,08	11467,21	11472,97
	CS	11208,40	11422,54	11448,36	11466,33	11472,94
	CC	11199,76	11415,89	11445,34	11465,94	11471,64
	CA	11197,62	11417,60	11448,45	11465,76	11470,68
3 Clientes	SC	1801,62	4248,01	4891,50	4087,30	3832,21
	CS	1248,07	2015,29	2982,15	3450,51	4090,64
	CC	1023,22	2788,82	3775,32	5182,07	4684,41
	CA	713,05	2558,79	2721,91	3869,47	5423,99

Tabela 2: Escrita de arquivos NFS (taxa de transferência, em KB/s)

Concorrência		Tamanho do Arquivo				
Acesso	Proc.	1MB	5MB	10MB	25MB	50MB
1 Cliente	SC	8406,64	8868,50	8438,92	7585,64	7918,23
	CS	3458,30	5612,53	5736,25	4884,29	5534,49
	CC	8403,04	8649,15	8758,16	7506,32	8046,17
	CA	4253,58	6387,20	6421,04	5608,64	5770,39
3 Clientes	SC	3060,86	3501,38	2985,73	3548,95	4040,33
	CS	1248,07	2015,29	2982,15	3450,51	4090,64
	CC	3065,20	3786,45	3803,78	3503,24	3420,44
	CA	1140,70	3449,61	2885,78	3517,89	4171,54

Tabela 3: Operações NFS (tempo em segundos)

Concorrência		Métrica		
Acesso	Proc.	Tempo real	Tempo proc.	Op/s
1 Cliente	SC	2,43	0,12	589,37
	CS	61,42	0,14	23,46
	CC	2,80	0,12	537,29
	CA	62,41	0,14	23,46
3 Clientes	SC	11,30	0,12	126,65
	CS	114,49	0,15	12,51
	CC	11,56	0,15	124,33
	CA	115,78	0,23	12,36

Tabela 4: Consumo de memória RAM (em MB) e CPU (%) para um cliente NFS

		Leitura			Escrita			Operações		
Máquina	Recurso	Inicio	Máx.	Média	Inicio	Máx.	Média	Inicio	Máx.	Média
front-end	RAM	338	343	342	336	339	337	313	313	313
	CPU	0,7	5,9	3,2	0,7	9,9	5,6	0,7	11,8	7,6
shaka	RAM	474	523	498	474	480	480	411	420	415
	CPU	0,3	2,6	1,3	0,3	5,9	2,3	0,3	4,5	1,9

Tabela 5: Consumo de memória RAM (em MB) e CPU (%) para três clientes NFS

		Leitura			Escrita			Operações		
Máquina	Recurso	Inicio	Máx.	Média	Inicio	Máx.	Média	Inicio	Máx.	Média
front-end	RAM	341	358	352	340	341	340	272	274	273
	CPU	0,7	5,3	3,3	0,7	13,1	8,5	0,7	4,1	1,3
shaka	RAM	479	532	500	479	486	485	352	385	356
	CPU	0,3	2,0	1,3	0,3	5,5	2,3	0,3	1,3	0,7

Tabela 6: Leitura de arquivos Tahoe (taxa de transferência, em KB/s)

Concorrência		Tamanho do Arquivo				
Acesso	Proc.	1MB	5MB	10MB	25MB	50MB
1 Cliente	SC	2278,27	2552,76	2824,17	3058,57	2955,71
	CS	2129,09	2505,30	2365,64	2559,59	2895,10
	CC	1991,29	2337,03	2334,31	2381,97	2389,59
	CA	1843,55	2538,51	2799,04	2633,72	2607,36
3 Clientes	SC	1604,27	1828,65	1824,78	1710,78	1761,99
	CS	2004,16	2632,60	2151,13	2268,64	2176,96
	CC	1697,42	2025,16	2189,48	2228,23	2301,73
	CA	1754,34	2080,03	2125,64	1973,81	2204,22

Tabela 7: Escrita de arquivos Tahoe (taxa de transferência, em KB/s)

Concorrência		Tamanho do Arquivo				
Acesso	Proc.	1MB	5MB	10MB	25MB	50MB
1 Cliente	SC	1988,53	3760,63	4150,79	4466,74	4606,54
	CS	1649,76	3420,48	3915,81	4141,00	4427,48
	CC	1840,24	3616,13	4020,32	4364,81	4550,33
	CA	1804,61	3324,43	3691,86	4265,08	4506,33
3 Clientes	SC	1623,63	2306,19	2791,52	2464,46	2665,34
	CS	1374,95	2602,63	2768,30	2648,34	2691,60
	CC	1626,85	2985,06	2506,13	2854,49	2758,62
	CA	1253,58	2778,84	2507,00	2684,80	2535,87

Tabela 8: Operações Tahoe (tempo em segundos)

Concorrência		Métrica		
Acesso	Proc.	Tempo real	Tempo proc.	Op/s
1 Cliente	SC	457,69	0,69	2,64
	CS	491,86	0,67	2,46
	CC	466,04	0,75	2,60
	CA	498,29	0,74	2,43
3 Clientes	SC	716,67	0,75	1,69
	CS	718,93	0,68	1,68
	CC	688,42	0,72	1,76
	CA	688,53	0,68	1,76

Tabela 9: Consumo de memória RAM (em MB) e CPU (%) para um cliente Tahoe

		Leitura			Escrita			Operações		
Máquina	Recurso	Inicio	Máx.	Média	Inicio	Máx.	Média	Inicio	Máx.	Média
front-end	RAM	348	350	350	346	349	348	343	345	344
	CPU	0,7	46,7	43,3	0,7	51,7	50,0	0,7	33,0	15,7
shaka	RAM	429	433	433	429	433	433	429	433	433
	CPU	0,3	1,3	1,3	1,3	1,5	0,9	0,3	0,5	0,3

Tabela 10: Consumo de memória RAM (em MB) e CPU (%) para três clientes Tahoe

		Leitura			Escrita			Operações		
Máquina	Recurso	Inicio	Máx.	Média	Inicio	Máx.	Média	Inicio	Máx.	Média
front-end	RAM	331	334	334	329	332	331	327	327	327
	CPU	0,7	51,8	50,0	0,7	51,2	50,0	0,7	48,5	25,0
shaka	RAM	431	435	435	432	436	436	431	435	435
	CPU	0,3	1,3	0,7	0,3	1,2	0,7	0,3	0,5	0,5

Tabela 11: Leitura e escrita de arquivos NFS (taxa de transferência, em KB/s)

	Tamanho do Arquivo				
Tipo de Acesso	1MB	5MB	10MB	25MB	50MB
Leitura	9855,13	8496,94	8301,95	10176,31	10191,26
Escrita	8904,66	10588,83	10798,50	10955,93	10961,03

Tabela 12: Leitura e escrita de arquivos HDFS (taxa de transferência, em KB/s)

	Tamanho do Arquivo				
Tipo de Acesso	1MB	5MB	10MB	25MB	50MB
Leitura	4058,43	7820,02	8343,49	8942,60	9610,22
Escrita	6341,31	7842,45	9453,95	8535,93	8507,22

Tabela 13: Leitura e escrita de arquivos SilasFS (taxa de transferência, em KB/s)

	Tamanho do Arquivo				
Tipo de Acesso	1MB	5MB	10MB	25MB	50MB
Leitura	971,27	4043,94	5634,23	5625,46	6226,50
Escrita	917,44	3438,23	5072,41	8143,91	9455,10

Operações NFS

Tabela 14: Operações para um cliente, sem concorrência NFS

Iteração	1	2	3	4	5	Média
Time (s)	2,51	2,37	2,30	2,51	2,45	2,43
Clock (s)	0,13	0,12	0,11	0,11	0,12	0,12
Op/s	570,48	603,52	620,65	569,15	583,05	589,37

Tabela 15: Operações para um cliente, concorrência no cliente NFS

Iteração	1	2	3	4	5	Média
Time (s)	2,23	2,29	2,28	3,40	3,82	2,80
Clock (s)	0,13	0,12	0,12	0,12	0,13	0,12
Op/s	640,00	624,48	627,16	420,05	374,74	537,29

Tabela 16: Operações para um cliente, concorrência no servidor NFS

Iteração	1	2	3	4	5	Média
Time (s)	66,41	57,27	57,69	56,33	69,41	61,42
Clock (s)	0,14	0,13	0,14	0,14	0,14	0,14
Op/s	21,53	24,97	24,79	25,39	20,60	23,46

Tabela 17: Operações para um cliente, concorrência em ambos NFS

Iteração	1	2	3	4	5	Média
Time (s)	58,28	68,54	65,77	66,40	53,06	62,41
Clock (s)	0,18	0,21	0,20	0,19	0,19	0,19
Op/s	24,54	20,86	21,74	21,54	26,95	23,13

Tabela 18: Operações para três clientes, sem concorrência NFS

Iteração	1	2	3	4	5	Média
Time (s)	11,20	11,20	11,27	10,86	11,99	11,30
Clock (s)	0,12	0,12	0,12	0,13	0,12	0,12
Op/s	127,72	127,68	126,94	131,72	119,22	126,65

Tabela 19: Operações para três clientes, concorrência no cliente NFS

Iteração	1	2	3	4	5	Média
Time (s)	10,56	12,70	10,89	12,33	11,32	11,56
Clock (s)	0,14	0,14	0,16	0,15	0,16	0,15
Op/s	135,43	112,64	131,31	115,97	126,33	124,33

Tabela 20: Operações para três clientes, concorrência no servidor NFS

Iteração	1	2	3	4	5	Média
Time (s)	111,74	111,30	118,62	110,76	120,01	114,49
Clock (s)	0,15	0,15	0,16	0,14	0,16	0,15
Op/s	12,80	12,85	12,06	12,91	11,92	12,51

Tabela 21: Operações para três clientes, concorrência em ambos NFS

Iteração	1	2	3	4	5	Média
Time (s)	119,68	114,26	115,15	111,16	118,65	115,78
Clock (s)	0,23	0,23	0,23	0,22	0,22	0,23
Op/s	11,95	12,52	12,42	12,86	12,05	12,36

Operações Tahoe

Tabela 22: Operações para um cliente, sem concorrência Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	453,42	451,69	456,80	463,00	463,54	457,69
Clock (s)	0,69	0,69	0,69	0,68	0,68	0,69
Op/s	2,67	2,68	2,65	2,61	2,61	2,64

Tabela 23: Operações para um cliente, concorrência no cliente Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	470,92	465,52	465,89	461,26	466,61	466,04
Clock (s)	0,75	0,75	0,74	0,75	0,74	0,75
Op/s	2,57	2,60	2,60	2,62	2,59	2,60

Tabela 24: Operações para um cliente, concorrência no servidor Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	485,08	476,74	485,24	522,37	489,88	491,86
Clock (s)	0,67	0,66	0,68	0,67	0,67	0,67
Op/s	2,49	2,54	2,49	2,32	2,47	2,46

Tabela 25: Operações para um cliente, concorrência em ambos Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	492,14	480,89	487,73	519,98	510,70	498,29
Clock (s)	0,75	0,74	0,74	0,74	0,73	0,74
Op/s	2,46	2,52	2,48	2,33	2,37	2,43

Tabela 26: Operações para três clientes, sem concorrência Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	698,15	693,26	689,48	681,73	680,02	688,53
Clock (s)	0,68	0,68	0,67	0,69	0,69	0,68
Op/s	1,73	1,75	1,75	1,77	1,78	1,76

Tabela 27: Operações para três clientes, concorrência no cliente Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	692,27	688,04	688,60	689,51	683,67	688,42
Clock (s)	0,75	0,69	0,72	0,76	0,69	0,72
Op/s	1,75	1,76	1,76	1,75	1,77	1,76

Tabela 28: Operações para três clientes, concorrência no servidor Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	718,44	702,41	713,39	723,41	736,99	718,93
Clock (s)	0,68	0,69	0,67	0,68	0,67	0,68
Op/s	1,68	1,72	1,70	1,67	1,64	1,68

Tabela 29: Operações para três clientes, concorrência em ambos Tahoe

Iteração	1	2	3	4	5	Média
Time (s)	706,65	731,44	711,81	717,77	715,67	716,67
Clock (s)	0,75	0,75	0,74	0,75	0,75	0,75
Op/s	1,71	1,65	1,70	1,69	1,69	1,69

Leitura NFS completo

Tabela 30: Avaliação de leitura 1 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,01	0,03	0,05
Taxa de transferência (KB/s)	11206,32	11421,28	11449,52	11467,07	11472,98

Tabela 31: Avaliação de leitura 2 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,00	0,02	0,02	0,06
Taxa de transferência (KB/s)	11183,44	11420,92	11447,63	11467,01	11472,88

Tabela 32: Avaliação de leitura 3 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,02	0,02	0,06
Taxa de transferência (KB/s)	11207,05	11423,29	11448,68	11467,15	11472,61

Tabela 33: Avaliação de leitura 4 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,00	0,02	0,02	0,06
Taxa de transferência (KB/s)	11204,94	11422,96	11447,66	11467,82	11473,19

Tabela 34: Avaliação de leitura 5 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,00	0,04	0,06
Taxa de transferência (KB/s)	11205,12	11423,70	11446,89	11467,02	11473,20

Tabela 35: Avaliação de leitura 1 NFS - 1 cliente, concorrência no servidor

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,01	0,03	0,05
Taxa de transferência (KB/s)	11206,32	11421,28	11449,52	11467,07	11472,98

Tabela 31: Avaliação de leitura 2 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,00	0,02	0,02	0,06
Taxa de transferência (KB/s)	11183,44	11420,92	11447,63	11467,01	11472,88

Tabela 32: Avaliação de leitura 3 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,02	0,02	0,06
Taxa de transferência (KB/s)	11207,05	11423,29	11448,68	11467,15	11472,61

Tabela 33: Avaliação de leitura 4 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,00	0,02	0,02	0,06
Taxa de transferência (KB/s)	11204,94	11422,96	11447,66	11467,82	11473,19

Tabela 34: Avaliação de leitura 5 NFS - 1 cliente, sem concorrência

Tamanho do arquivo	1MB	5MB	10MB	25MB	50MB
Tempo real (s)	0,09	0,45	0,89	2,23	4,46
Tempo processamento (s)	0,00	0,01	0,00	0,04	0,06
Taxa de transferência (KB/s)	11205,12	11423,70	11446,89	11467,02	11473,20