Sistemas de Arquivos de Uso Especial

Eider Oliveira - eider@ime.usp.br
Orientador: Prof. Arnaldo Mandel - am@ime.usp.br

20/06/2005

Resumo

Estudo de sistemas de arquivos de uso especial. Mecanismos de projeto, implementação para sistemas de arquivos, em particular sistemas de grande escala, com uso restrito a aplicações específicas.

1 Introdução

Muitos estudos sobre sistemas de arquivos vêm sendo conduzidos bem como diversos projetos experimentais podem ser encontrados facilmente. Embora o conceito básico da distribuição de um sistema de arquivos seja comum a todos, notavelmente há uma diferenciação quanto ao tipo de aplicação que obtém de um determinado sistema de arquivos sua melhor performance.

Neste estudo, estamos interessados em comparar os sistemas de arquivos distribuídos existentes sob o ponto de vista de sua aplicação. Para melhor compreendê-los, é necessário realçarmos as características chave de sua implementação, além da motivação básica do projeto.

Os seguintes sistemas de Arquivos foram estudados

- Write Anywhere File Layout File System (Dispositivo dedicado NFS)
- HTTPFS
- Google File System
- Andrew File System
- CODA
- Intermezzo
- Lustre
- Global File System (GFS)

Em [8] é descrita a dificuldade de se escrever um texto abordando sistemas de arquivos em português. A falta de um jargão em português que traduza os termos comumente usados em inglês ainda é notória. Neste texto encontram-se neologismos que embora atentem contra a língua, são de largo uso tanto no meio acadêmico como no meio empresarial.

2 Write Anywhere File Layout File System

O WAFL [5] foi desenhado para executar em um servidor NFS dedicado, não em computadores de uso geral. Este tipo de equipamento possui

características de hardware especiais que possibilitam ao WAFL suas características especiais.

2.1 Metainformações

As metainformações do sistema de arquivos são um arquivo no WAFL. São três os possíveis tipos:

Inode Table contém os inodes do sistema de arquivos

Block Map identifica os blocos livres

Inode Map que identifica os inodes livres

Ao manter as metainformações em arquivos, torna-se possível que estas informações estejam em qualquer lugar no disco. Esta é a origem do nome WAFL. O WAFL é otimizado para escritas em RAID, onde as escritas são feitas no mesmo *stripe* o máximo possível, reduzindo a penalidade existente em RAID quando da atualização de um único bloco do sistema de arquivos.

2.2 *Snapshots*

Snapshot é uma cópia somente de leitura do sistema de arquivos. Esta é a característica principal do WAFL. O WAFL cria e apaga *snapshots* periodicamente e mantém até 255 *snapshots* disponíveis, o que possibilita um acesso simples e eficiente a versões anteriores de arquivos.

Snapshots usam a técnica de *copy-on-write* para evitar duplicar blocos de dados que são os mesmos da versão atual do sistema de arquivos. Somente quanto blocos do sistema de arquivo são alterados ou modificados é que um *snapshot* começa a efetivamente ocupar espaço no disco. Os usuários podem acessar um *snapshot* via NFS e obter uma versão anterior de um arquivo recém alterado ou recuperar um arquivo apagado erroneamente.

O WAFL pode ser melhor compreendido como uma árvore de blocos, cuja a raiz é o *root inode*. Para criar um novo *snapshot* o WAFL apenas duplica o *root inode*. Este novo *inode* torna-se a raiz do *snapshot* da mesma forma que o *root inode* é a raiz do sistema de arquivos corrente. No momento da criação de um novo *snapshot*, ele corresponde exatamente ao mesmo sistema de arquivos corrente. À medida que alterações são feitas no sistema de arquivos, os blocos alterados são copiados (*copy on write*) e o novo bloco é inserido na árvore corrente em lugar do bloco original. O bloco original continua sendo parte da árvore que representa o *snapshot*. Ao duplicar apenas o *root inode* um *snapshot* é criado de forma bastante rápida de eficiente, com um consumo mínimo de espaço.

2.3 Consistência do Sistema de Arquivos

O WAFL evita a necessidade de verificação de consistência após um reinício forçado criando um *snapshot* especial chamado *checkpoint* (pontos de controle) em intervalos regulares. Após um reinício forçado, o sistema simplesmente usa a raiz do último *checkpoint* como *root inode*. Isto evita que muito tempo seja necessário para recuperar um sistema de arquivo muito grande.

WAFL utiliza memória não volátil para armazenar as ocorrências de escrita entre os pontos de controle. Após um reinício forçado, o WAFL

verifica as operações que estão listadas nesta memória e as executa novamente, de forma que o sistema de arquivos volte ao estado em que se encontrada no momento da interrupção forçada.

3 Google File System

Este sistema de arquivos, conforme descrito em [4], é um sistema de arquivos altamente distribuído e redundante, voltado para aplicações que demandam dados de forma intensa. Seu desenho leva em consideração as características das aplicações no Google, bem como o ambiente tecnológico utilizado na empresa.

Entre os seus pressupostos estão:

- O sistema de arquivo será composto por centenas ou milhares de máquinas de baixo custo e baixa qualidade que sempre falham. O sistema precisa ser capaz de monitorar-se, tolerar e se recuperar de falhas de software ou hardware.
- Serão poucos os arquivos armazenados neste sistema, mas estes arquivos são imensos, considerando os padrões tradicionais. São esperados alguns milhões de arquivos com tamanho superior a 100MB cada, sendo que arquivos com vários gigabytes são comuns.
- Em sua maioria, as alterações em arquivos se dão pelo acréscimo de novos dados, não pela alteração ou remoção do conteúdo já existente
- O projeto da aplicação é feito em conjunto com o projeto do sistema de arquivos, simplificando o desenho de ambos.
- A demanda de leitura é basicamente uma leitura seqüencial de grande quantidade de dados. Desta forma, a latência não é importante, mas a capacidade de banda é crucial.
- O sistema possui semântica especial e eficiente para que múltiplos clientes possam anexar dados aos arquivos simultaneamente.

3.1 Interface

O sistema possui uma interface similar aos sistemas de arquivos convencionais, mas esta interface não atende ao padrão POSIX. Os arquivos são organizados em diretórios estes se organizam em árvores.

O sistema de arquivo possui os comandos

snapshot para criar cópias eficientes de arquivos o ou diretórios

anexar para que múltiplos clientes possam de forma atômica anexar dados ao arquivo concorrentemente

3.2 Arquitetura

Um *cluster* GFS consiste de um único *master*, diversos *chunkservers* e é acessado por diversos clientes. Cada um destes é uma máquina rodando Linux e um processo em modo de usuário. É possível executar tanto um *chunkserver* e um cliente na mesma máquina desde que os recursos sejam suficientes e que a pouca confiabilidade do sistema seja aceitável.

Os arquivos são divididos em seções de tamanho fixo chamados *chunks*. Cada *chunk* possui um identificador numérico de 64 bits que é definido pelo *master* no momento da criação do *chunk*.

O *Master* é responsável por toda metainformação do sistema, tais como as definições de controle de acesso, mapeamento de arquivos para lista de *chunks* e localização dos *chunks*. Além disto controla o acesso aos *chunks*, recuperação de *chunks* órfãos e migração de *chunks* entre *chunkservers*.

Além disto, o *Master* periodicamente consulta os *chunkservers* obtendo seu estado. Nem o *master*, os *chunkservers* ou os clientes mantém *cache* dos arquivos. Manter um único *Master* simplifica a arquitetura e permite ao *Master* decidir sobre a alocação dos *chunks*, bem como sua replicação usando informação do sistema como um todo. Contudo, o papel do *Master* é minimizado nas operações de leitura e escrita para evitar que ele se torne um gargalo no sistema.

Para efetuar a leitura de um arquivo, o cliente consulta o *Master* para obter a localização dos *chunks* que compõe o arquivo e de posse desta informação ele consulta diretamente os *chunkservers* sem a interferência ou conhecimento do *Master*.

3.3 Replicação

Cada *chunk* está presente em pelo menos 3 *chunkservers*, mas caso seja necessário o fator de replicação pode ser maior para arquivos muito acessados. O *Master* é responsável por recriar cópias dos *chunks* que forem perdidos em caso de perda de um *chunkserver*.

4 Andrew File System

Um sistema usando AFS [6] é organizado em células, que são compostas de máquinas clientes e servidores. As máquinas servidores executam diversos programas servidores, onde cada um deles cuida de um serviço diferente, como serviço de cache, serviço de arquivos, segurança, localização de arquivos, dentre outros. Uma célula é um agrupamento de administração independente. Um administrador de uma célula não necessita conhecer ou compartilhar da configuração de outras células.

Embora as células sejam independentes, a coleção de arquivos que uma célula deseja exportar para outras células se torna parte de um espaço de nomes global. Desta forma, todos os arquivos exportados são visíveis a todos os clientes, se estes clientes possuírem as credenciais necessárias. Os arquivos são agrupados em **volumes**, que são a unidade de compartilhamento do AFS.

4.1 Volumes

Volume é um agregado conceitual para um conjunto de arquivos relacionados, que permanecem em um mesmo servidor. Embora possam variar em tamanho, em geral são menores que uma partição. Seu tamanho reduzido facilita sua movimentação entre partições e até entre servidores. Para aumentar a eficiência do sistema, é possível movimentar os volumes entre os servidores de forma a manter a carga balanceada entre eles.

Cada volume corresponde logicamente a um diretório na árvore de arquivos. Assim pode-se manter um volume para cada diretório raiz de um usuário por exemplo. A correspondência entre diretório e volumes também torna o acesso transparente possível, uma vez que para encontrar um volume é necessário apenas saber o nome do diretório acima na árvore de diretórios. Os volumes mais comumente usados podem ser replicados

entre servidores, mantendo o serviço disponível mesmo com a queda de um servidor. Estas cópias, também chamadas clones, são apenas para leitura.

4.2 Performance e Cache

Nas máquinas clientes são mantidos caches dos arquivos em utilização, visando melhora na performance do sistema. Recursos de rede são economizados quando um cliente obtém um arquivo diretamente do seu cache, sem necessidade de obtê-lo remotamente. O sistema mantém um mecanismo de *callback* para garantir que os caches dos clientes estejam atualizados em caso de alterações feitas por outros clientes.

4.3 Segurança

Um mecanismo de autenticação mútua garante aos servidores que eles apenas disponibilizam os arquivos aos clientes autorizados, e garante aos clientes que ele estão obtendo os arquivos também dos servidores corretos. O sistema também mantém lista de controle de acessos mantidas pelos usuários que permitem uma configuração mais fina e precisa do controle de acesso.

4.4 Transparência de Localização

Os arquivos disponíveis num sistema AFS fazem parte de um espaço de nomes único e global. É completamente transparente aos clientes a localização física de um arquivo. Os clientes acessam os arquivos como se eles estivessem efetivamente em seu computador local. Além disto, o espaço de nomes é uniforme entre os diversos clientes.

5 CODA

CODA [2] tem sua origem no AFS, com destaque para as seguintes características:

- Operação desconectada, possibilitando computação móvel
- Disponível em uma licença bastante liberal
- Alta performance obtida por um cache persistente nos clientes
- Replicação de servidores
- Modelo de segurança para autenticação, criptografia e controle de acesso
- Operação continuada em caso de falhas temporárias na rede
- Boa escalabilidade
- Semântica bem definida de compartilhamento, mesmo em caso de falha na rede

5.1 Operação desconectada

Um problema encontrado no final da década de 80, quando o AFS já era executado com mais de mil clientes é a forma como o AFS lida com falha de rede entre servidores e clientes [7]. O esforço empregado no CODA para lidar com este problema se mostrou conveniente com o advento de

clientes móveis. Quando um usuário atualiza um arquivo, esta alteração precisa ser propagada aos servidores que armazenam este arquivo. Caso o cliente esteja conectado ao servidor, esta atualização é feita de forma síncrona, isto é, a alteração é propagada ao servidor no momento de sua gravação no cliente.

Caso haja algum problema nesta comunicação, ao invés de reportar erro ao usuário, é gravado localmente um registro de uma alteração pendente, e assim que a comunicação com os servidores for restabelecida, estas alterações são propagadas automaticamente aos servidores.

Há dois conceitos muito importantes relacionados à operação desconectada. O primeiro é a estoque de arquivos. Como o sistema não consegue resolver uma ausência de cache com o sistema desconectado, ele procura manter seu cache atualizado antecipadamente. Estes arquivos são descobertos pelo acompanhamento do uso dos arquivos pelo usuário. Na prática, muitos arquivos de sistema e aplicativos são armazenados nos computadores clientes, como pacotes gráficos e de escritório.

A outra questão é a atualização de um mesmo arquivo por dois clientes desconectados. Isto é chamado conflito. Em alguns casos, os conflitos podem ser resolvidos automaticamente, como dois usuários inserindo eventos em horários distintos num calendário. Na maioria dos casos, no entanto, é necessária a intervenção manual para resolução dos conflitos.

5.2 Volumes, Servidores e Replicação

A maioria dos sistemas de arquivos remotos permitem o compartilhamento de uma árvore. Para estes sistemas não é prática a montagem de um novo volume na árvore já importada. A organização de arquivos nos servidores não é como nos sistemas de arquivos tradicionais. Partições nos servidores CODA podem ser disponibilizadas pelos Servidores de Arquivos. Estas partições contém arquivos que são agrupados em volumes. Cada volume contém uma estrutura de diretórios como um sistema de arquivos tradicional, ou seja, um diretório raiz que contém uma árvore de arquivos e sub-diretórios. Um volume é uma unidade de dados que pode ser naturalmente vista como uma unidade do ponto de vista administrativo. Um volume possui um nome e um identificador, e pode ser montado em qualquer posição abaixo de /coda. Por exemplo, o volume eider pode ser montado em /coda/usr/eider. Coda não permite que os pontos de montagem sejam arquivos já existentes, mas vai criar os diretórios como parte do processo de montagem. Coda identifica um arquivo através de uma tripla de 32 bits: VolumeId, VNodeId, e um *Uniquefier*. Esta tripla é única em um cluster de servidores Coda.

Coda possui replicação de servidores em modo de leitura e escrita. Uma gravação em um arquivo replicado é realizada em todos os servidores do grupo. A vantagem clara é a alta disponibilidade dos dados, mesmo em caso de falha nos servidores.

6 Intermezzo

Intermezzo é um projeto derivado do CODA [3], com foco em alta disponibilidade. Conceitos como replicação de servidores são mais desenvolvidos e sofisticados neste sistema de arquivos. O operação desconectada de Coda utiliza um mecanismo de *callback* para manter o cache utilizado no cliente atualizado. Esta técnica implica que os servidores necessitam manter

Figura 1: Operações Intermezzo no Cliente

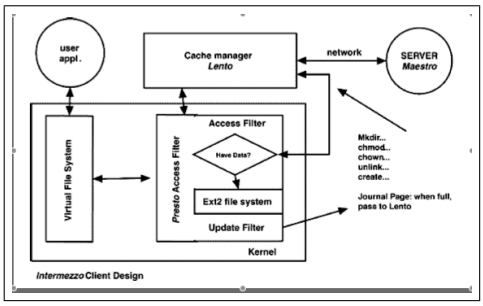
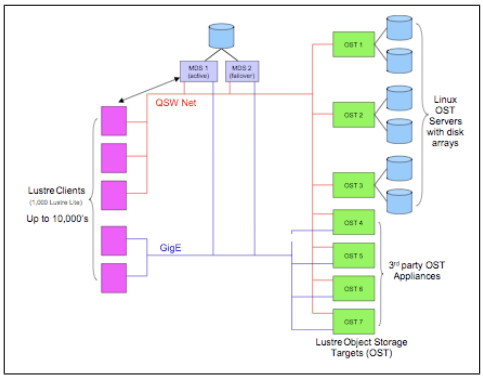


Figura 2: Arquitetura do lustre



uma lista de clientes e quais arquivos estão em uso por eles, embora isto somente seja necessário para os arquivos abertos em modo de escrita.

Intermezzo implementa uma diferente estratégia de cache cliente [1], que possibilita uma performance superior a Coda nas operações de escrita e consulta ao cache. No Intermezzo, o Cache Manager (responsável pelo cache no cliente em Coda) é dividido em dois componentes, *lento* e um módulo de filesystem no kernel chamado *presto*. *Lento* mantém as metainformações que não podem ser armazenadas em um filesystem local, como *Version Stamps*, VolumeId e lista de controle de acessos. *Presto* filtra as operações de leitura e escrita no kernel de forma a obter melhor performance nas atualizações remotas. A figura 1 detalha como uma operação é realizada no Intermezzo.

7 Lustre

A proposta do Lustre é de ser um filesystem distribuído que suporte milhares de clientes simultâneos, endereçando um sistema de arquivo da ordem de petabytes [10]. A figura 2 contém um esquema da arquitetura do Lustre. Os servidores em um cluster Lustre são divididos em 3 grupos:

Logical Object Volume (LOV) Driver Que executa no cliente

Meta Data Servers (MDS) Mantém as metainformações além de ser

responsável pela alocação e distribuição dos arquivos.

Object Storage Targets (OST) Mantém os arquivos em disco local

7.1 Meta Data Servers

Estes servidores mantém todos os dados críticos do sistema de arquivos, como a árvore de diretórios que compõe o espaço de nomes, controla a criação, distribuição, alocação e disponibilidade dos OST e mantém o registro das alterações no sistema de arquivos. Estas informações são replicadas em um MDS que atua como *failover*. Além disto, o MDS opera de forma transacional o que implica na atomicidade, durabilidade, unicidade e consistência das operações nele realizadas.

Também é papel do MDS comunicar-se periodicamente com os OST para identificar possíveis falhas nestes equipamentos de forma a redirecionar novas alocações apenas para servidores que estiverem operacionais.

7.2 Object Storage Targets

O OST é responsável por todas as operações de leitura e escrita nos arquivos. Enquanto o MDS cuida da localização e das metainformações, o OST é quem realiza as operações de dados e quem mantém fisicamente os arquivos. Os sistemas de armazenagem chamados *Object-Based Disks (OBD)* são representantes dos discos, e a interface com o OST se dá através de *device drivers*. Esta separação permite uma maior flexibilidade e um incremento na performance do sistema é possível pela mudança dos sistemas de disco por sistemas mais novos e eficientes.

Novos OST podem ser adicionados ao sistema, promovendo crescimento vegetativo do sistema de arquivos sem interrupção do sistema, e também novos OBD podem ser adicionados a um OST em funcionamento.

7.3 Logical Object Volume

LOV é o nome dado ao código cliente do Lustre, que permite ao usuário enxergar um cluster Lustre como um sistema de arquivos montado localmente. Ele é responsável por identificar qual MDS está ativo, através de uma consulta a um servidor LDAP, enviar as requisições de controle ao MDS e obtendo do MDS uma referência para os OST que contém as informações desejadas ou que será o destinatário dos dados a serem gravados, promover e manter a comunicação diretamente com os OST envolvidos. A figura 3 mostra um esquema detalhado do LOV.

7.4 Independência de Rede

O Lustre utiliza um mecanismo de abstração de redes, chamado NAL (*Network Abstraction Layer*), que lhe permite funcionar em uma variedade de configurações. Atualmente, existem implementações para TCP e Quadrics. Estão ainda em desenvolvimento o suporte a Mirynet, Fibre Channel, Stargen e InfiniBand. Além disto, um cluster Lustre pode operar sob redes heterogêneas.

8 Global File System

Originalmente desenvolvido para ambientes IRIX da Silicon Graphics, o GFS, agora sob propriedade da Red Hat Inc, está sendo portado para

Figura 3: Logical Object Volume

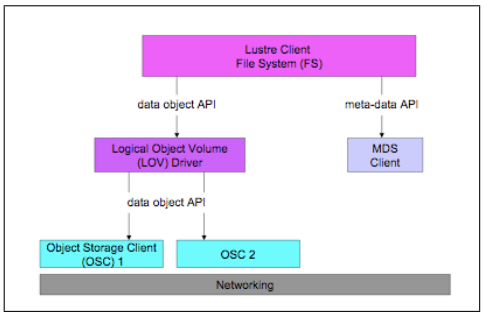
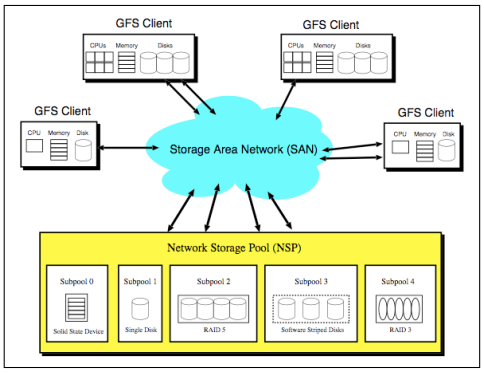


Figura 4: Arquitetura do GFS



o Linux. Inicialmente o projeto foi apresentado na Quinta Conferência de Armazenamento em Massa da NASA [11] Diferentemente de outros sistemas de arquivos, o GFS não é baseado em nenhum outro sistema de arquivos já existente, sendo desenvolvido independente de outros projetos.

O GFS consegue garantir alterações concorrentes em seus arquivos através de mecanismos de bloqueio. Um cliente necessita obter permissão exclusiva para escrita antes de efetuá-la, e enquanto este cliente não liberar sua permissão, outros clientes ficam bloqueados, em estado de espera. Exceto para manter as garantias de consistência, os clientes não dependem de outros clientes.

A figura 4 descreve a arquitetura do GFS. O foco do GFS é de promover a melhor performance possível para um número pequeno de clientes, divergindo da proposta de outros sistemas de arquivos como o Lustre. As aplicações alvo do GFS são aplicações que tem grande demanda por vasto espaço de armazenamento e de grande largura de banda, tais como multimídia, computação científica e visualização. Esta grande capacidade de armazenamento tem impacto direto nas estruturas do sistema de arquivos e nos mecanismos de cache. O GFS também não oferece mecanismos de segurança além dos oferecidos pelo Unix.

8.1 Network Storage Pools

Um NSP *(Network Storage Pool)* é um conjunto de dispositivos compartilhados fisicamente. *Subpools* são partições de um NSP de acordo com as características dos dispositivos. Estas características podem ser tanto baixa latência quanto grande largura de banda. Uma implementação pode tirar proveito das diferenças entre *subpools* alocando objetos de acesso freqüente, como diretórios em dispositivos de baixa latência. Embora os arquivos possam ser particionados e cada parte armazenada em um servidor diferente, isto é desejável em poucas situações. Um esquema desta distribuição pode ser visto na figura 5.

Se os acessos estiverem uniformemente distribuídos entre os dispositivos, o paralelismo resultante implica em melhor performance geral. Isto pode ser obtido através da migração de arquivos entre os dispositivos.

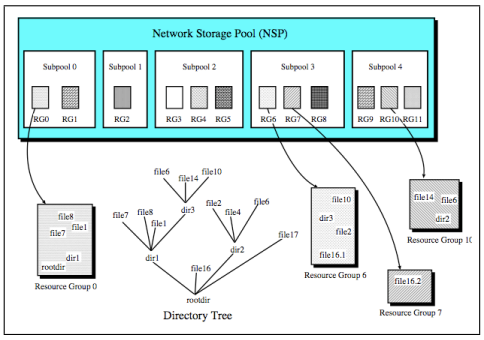
8.2 Hierarquia de Memória

A hierarquia de memória do GFS é inclui a memória das máquinas clientes, os caches dos dispositivos de armazenamento, e a mídia de armazenamento propriamente dita. Embora esta hierarquia se assemelhe a um sistema de arquivos local, manter a consistência desta hierarquia no GFS é mais complexo. Mecanismos de exclusão mútua são utilizados para garantir esta consistência. O tempo de acesso a cada nível desta hierarquia difere em uma ordem de magnitude. Sistemas de arquivos locais subutilizam o cache dos dispositivos de armazenamento. Em geral, eles se limitam à utilização da memória da máquina para este fim. O cache dos dispositivos tem um papel mais importante no GFS, atuando como um cache intermediário do qual se beneficiam todos os clientes do cluster. Tipicamente os dispositivos implementam uma estratégia LRU *(Least Recently Used)* associada a leitura adiantada dos dados. O GFS aprimora este mecanismo sugerindo que dados devem ser mantidos no cache.

8.3 Vantagens Arquiteturais

- O desenho simétrico com suporte a múltiplos clientes permite ao cluster operar como se fosse um computador com SMP (*Multi Symmetric Processors*). Os recursos do cliente como CPU e memória não são compartilhados entre clientes
- GFS tira proveito das arquiteturas modernas de NAS ao remover os servidores do caminho de dados.
- Por ser projetado desde o início como um sistema de arquivos distribuídos, ao invés de ser uma modificação de um sistema de arquivos local, o GFS pode contar com recursos de cache mais elaborados, serialização de metainformações e um desenho que permitem uma escalabilidade maior.
- Grupos de dispositivos de armazenamento são combinados logicamente em *pools* compartilhados de armazenamento. Uma implementação GFS pode mapear os dados do sistema de arquivos para vários grupos de acordo com as características destes dispositivos que compõe o grupo, conforme demonstrado na figura 5. Redundância de hardware e software garantem a disponibilidade dos dados.
- GFS garante consistência total nos dados. Requisições de leitura retornam o dado mais recentemente escrito, mesmo para arquivos compartilhados entre diversos clientes.

Figura 5: Mapeamento de Arquivos em um NSP



9 Classificação dos Sistemas de Arquivos

Uma nova classificação para os sistemas de arquivos é apresentada em [9] baseando-se nas aplicações que se utilizam dos sistemas de arquivos:

Application Transparent Adaptation Identifica sistemas de arquivos em que as aplicações que o utilizam não tem conhecimento do sistema de arquivos que estão utilizando, não podendo explorar características pertinentes a este sistema de arquivo.

Application Aware Adaptation A aplicação utiliza recursos específicos do sistema de arquivos utilizados, e em geral, seu desenho contempla o desenho do sistema de arquivos. Um exemplo é o *Google File System* em que o desenho do sistema de arquivos não pode ser desassociado do desenho da aplicação.

Referências

[1] P. J. Braam, M. Callahan, and P. Schwan.
The intermezzo file system.
Technical report, 1999.
http://www.inter-mezzo.org/docs/perlintermezzo.pdf.

[2] Peter J. Braam.
The coda distributed file system.
School of Computer Science, Carnegie Mellon University, http://www.coda.cs.cmu.edu/ljpaper/lj.html.

[3] Peter J. Braam and Philip A. Nelson.
Removing bottlenecks in distributed filesystems: Coda intermezzo as examples, 1999.
citeseer.ist.psu.edu/235665.html.

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.
The google file system.
SOSP, 2003.
http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf.

[5] Dave Hit, James Lau, and Michael Malcolm.
File system design for an nfs file server appliance.
Network Appliance Inc, 1995.
http://www.netapp.com/tech_library/3002.html.

[6] IBM.
The open afs.
Technical report, IBM Corporation, 2000.
http://www.openafs.org/doc/index.htm.

[7] James Jay Kistler.
Disconnected operation in a distributed file system, volume 1002.
Springer-Verlag Inc., New York, NY, USA, 1995.
citeseer.ist.psu.edu/kistler93disconnected.html.

[8] Fabio Kon.
Sistemas de arquivos distribuídos.
Master's thesis, IME USP, 1994.
http://choices.cs.uiuc.edu/~f-kon/thesis/kon-master.ps.gz.

[9] Mahadev Satyanarayanan.
Mobile information access.
In *IEEE Personal Communications*, volume February 1996, pages 26–33, 1996.
http://www-2.cs.cmu.edu/afs/cs/project/coda-www/ResearchWebPages/docdir/ieeepcs95.pdf.

[10] P. Schwan.
Lustre: Building a file system for 1000-node clusters.
Technical report, Cluster File Systems, Inc, 2003.
http://citeseer.ist.psu.edu/schwan03lustre.html.

[11] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe.
The Global File System.
In *Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems*, pages 319–342, College Park, MD, 1996. IEEE Computer Society Press.
citeseer.ist.psu.edu/soltis96global.html.