UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Survey sobre Sistemas de Arquivos

Miguel Leonardo C. Cartagena

Florianópolis – SC

2008

Lista de Tabelas

Lista de Figuras

Sumário

1 Introdução

Sistemas de arquivo são parte integrande da grande maioria dos sistemas operacionais modernos. Seu estudo tem por objetivo obter maior conhecimento das soluções já empregadas e consagradas, para que estas possam ser aplicadas à dados científicos. É importante ressaltar que, mesmo com a diferença na natureza dos dados, conceitos de manipulação e organização de arquivos podem ser sempre absorvidos e utilizados de maneira que aproxime o usuário/aplicação aos sitemas de arquivos convencionais.

O trabalho aborda sistemas de arquivos especialmente projetos para tratar dados científicos, isto é, grandes volumes de dados que uma vez escritos raramente são alterados, no entanto, são consultados com frequência. Por se tratar de dados com características específicas, seu armazenamento também requer cuidados diferenciados, especialmente quanto à indexação e estruturação de arquivos. Tais aspectos são apresentados no presente trabalho, bem como uma comparação com os sistemas de arquivos de propósito geral.

O capítulo seguinte do trabalho apresenta os sistemas de arquivos tradicionais, bem como suas características e aplicações. Posteriormente, no capítulo 3, são discutidas soluções em formatos de arquivos científicos, bem como suas particularidades. Finalmente o capítulo 4 traz uma conclusão acerca do tema abordado.

2 *Sistemas de Arquivos*

Sistemas de arquivos surgiram para solucionar três principais problemas: pouco espaço em memória para armazenamento de informações; impossibilidade de persistir informações, já que os dados armazenados na memória eram perdidos ao final do processo; e falta de acesso múltiplo, isto é, mais de um processo acessandos informações ao mesmo tempo. A partir dos problemas acima, é possível listar três requisitos essenciais para armazenamento de informações (TANENBAUM, 2003):

- Possibilidade de armazenamento de grandes quantidades de informação;
- Informações devem persistir mesmo após o encerramento de um processo;
- Vários processos devem ser capazes de acessar informações concorrentemente.

Frente as estas necessidades e requisitos, surgiu o conceito de arquivo, uma unidade de informação que pode ser persistida em discos e outros meios, independente do ciclo de vida de um processo. Além disso, processos podem manipulá-los através de leitura e escrita. É tarefa do sistemas operacional a organização dos arquivos dentro de um disco, devendo criar mecanismos para nomeação, acesso e estruturação. Esta parte de um sistema operacional é chamada de sistema de arquivos. A seguir será mostrado como arquivos e diretórios trabalham, posteriormente as estruturas de organização de arquivos em disco serão abordadas.

2.1 Arquivos

Um arquivo, de modo geral, possui diversas funcionalidades importantes onde é possível destacar sua nomeação e forma de acesso. Tais características são diferenciais para o usuário do

sistema operacional, já que grande parte das operações envolvem os dois atributos citados. A nomeação de arquivos é uma das funções mais básicas de um sistema de arquivos, uma vez que é através do nome do arquivo que este será chamado e identificados pelos processos. Diferentes sistemas operacionais utilizam mecanimos distintos de nomeação de arquivos, dentre os quais pode-se destacar o limite de caracteres dos nomes de arquivos e a diferenciação entre letras maiúsculas e minuscalas. Muitos sistemas também adotam extenções de arquivos, um pequeno conjunto de caracteres (em geral três, após um ponto que separa o nome propriamente dito) que indica alguma informação sobre os arquivos. Tais extenções podem ser utilizadas por alguns sistemas operacionais para definir os tipos de arquivos, isto é, um arquivo de texto sempre terá a extenção *.txt*. Além do nome, arquivos possuem outros atributos que o caracterizam. Estes atributos, em geral, dizem respeito a sua forma e data de criação e/ou modificação. Outros atributos, as *flags*, controlam alguma característica específica. A Tabela 2.1 traz os alguns dos possíveis atributos de um arquivo (TANENBAUM, 2003).

Atributo	**Significado**
Proteção	Quem pode acessar o arquivo e como.
Criador	ID do usuário que criou o arquivo.
Flag do Sistema	0 para arquivos normais; 1 para arquivos do sistema
Flag ASCII/Binário	0 para ASCII; 1 para binário.
Flag de acesso aleatório	0 se apenas acesso sequêncial; 1 para acesso aleatório
Tamanho do registro	Número de bytes em um registro
Posição da chave	Deslocamento da chave dentro de cada registro
Tamanho da chave	Número de bytes no campo-chave
Momento da criação	Data e horário em que o arquivo foi criado
Tamanho atual	Número de bytes do arquivo
Tamanho máximo	Número de bytes que o arquivo pode vir a ter

Tabela 2.1: *Alguns atributos de arquivos.*

Um aspecto importante dos arquivos é sua estruturação interna. Sistemas operacionais costumam tratar de forma diferente a estrutura dos arquivos, onde o tipo mais comum é a sequência de bytes. Tanto sistemas UNIX quanto Windows utilizam esta técnica, que permite maior flexibilidade aos programadores já que as especificações de leitura e escrita serão feitas pelo responsável

pelo arquivos. Outra abordagem utiliza blocos de registro de tamanho fixo, assim arquivos são visto como sequências de registros. Métodos como este favorecem a segmentação interna já que arquivos nem sempre possuem um tamanho divisível pelo tamanho do registro. Na terceira técnica para estruturação interna de arquivos, este são vistos como árvores de registros, nem sempre de tamanho fixo, que possuem uma chave. Assim como outras estruturas de dados clássicas, os registro são ordenados por sua chave e o acesso é feito através da busca por tal chave (TANENBAUM, 2003)(SILBERSCHATZ, 1997).

Como citado anteriormente, arquivos possuem tipos que os distinguem em funcionalidade e, principalmete, em estrutura. Sistemas como o UNIX e Windows apresentam dois tipos de arquivos, regulares e diretórios, onde os arquivos regulares são os arquivos de usuário e os diretório são parte da estruturação dos arquivos. Arquivos especiais, como o de caracteres, são utilizados por alguns sistemas para representar dispositívos de E/S. Os tipos mais comuns de arquivos, os regulares, são em geral arquivos ASCII ou binários. Arquivos ASCII são sequências de caracteres, delimitados por caracteres especiais de quebra de linha, que podem ser enviados à dispositivos de E/S, como impressoras. Já arquivos binários são específicos de cada sistema operacional, sendo que estes guardam informações de programas que executam sob o sistema operacional.

Como destacado no início da seção, o acesso aos arquivos é uma importante ferramenta do sistemas operacionais. Historicamente arquivos eram acessados de forma sequêncial, ou seja, os processos liam byte a byte o arquivo, não sendo possível a leitura apenas da parte de interesse. Posteriormente, com a utilização de discos magnéticos para o armazenamento de arquivos foi possível implementar acesso randômico aos arquivos. Para algumas aplicações, este tipo de acesso é fundamental, a fim de evitar que milhares de bytes sejam lidos sem necessidade.

Haja vista que arquivos foram projetados para persistir informações de processos, estes devem possuir um conjunto de operações capazes de manipular arquivos. Estas operações, oferecidas pelo sistema operacional, são em geral bastante parecidas, não tendo grandes variações de sistema para sistema. A Tabela 2.2 sumariza as principais operações realizadas com arquivos (TANENBAUM, 2003)(SILBERSCHATZ, 1997).

Operação	Descrição
create	O arquivo é criado sem dados. Há alocação de espaço para o arquivo.
delete	Apaga o arquivo quando este não é mais necessário.
open	Busca o arquivo e o coloca na memória para que possa ser acessado.
close	Após a utilização do arquivo, este deve ser retirado da memória.
read	São lidos dados do arquivo, de forma sequêncial ou aleatória.
write	Escreve dados no arquivo na posição atual do ponteiro.
append	Tipo especial de *write*, onde dados são escrito no final do arquivo.
seek	Reposiciona o ponteiro do arquivo para um local específico.
rename	Permite que o arquivo senha renomeado.

Tabela 2.2: *Principais operações sobre arquivos.*

2.2 Diretórios

Diretórios são ferramentas que sistemas operacionais utilizam para organização e controle de arquivos, sendo possível que alguns sistemas tratem diretórios como arquivos especiais. Diretórios podem ainda ser organizados em vários níveis, como será visto a seguir. Para tornar o diretório funcional, é interessante que algumas operações possam ser realizadas sobre o mesmo. Abaixo estão listadas as operações disponíveis para diretórios em sistemas UNIX (TANENBAUM, 2003).

- ***Create.*** Cria um diretório vazio, exceto pelo ponto e pontoponto.
- ***Delete.*** Apaga um diretório (vazio) que não é mais utilizado.
- ***Opendir.*** Abre o diretório para operações dentro do mesmo.
- ***Closedir.*** Após ser aberto e utilizado, o diretório deve ser fechado.
- ***Readdir.*** Retorna a próxima entrada em um diretório aberto.
- ***Rename.*** Troca o nome de um diretório, assim como funciona para arquivos.
- ***Link.*** Cria uma ligação de um arquivo para outro diretório.

- ***Unlink*.** Remove a ligação de um arquivo com determinado diretório.

A estruturação de diretório aparece como um aspecto bastante importante de um sistema de arquivos, autores como (TANENBAUM, 2003) dividem sistemas de diretórios em três classificações: os de nível único; de dois níveis; e os hierárquicos. Já outros autores, como (SILBERSCHATZ, 1997), apresentam classificações mais aprofundadas, considerando pelo menos cinco diferentes sistemas de diretórios. A fim de apresentar o tema com maior riqueza, será apresentada a classficação adotado por (SILBERSCHATZ, 1997).

- **Nível Único** - São os sistemas mais simples de diretórios, onde todos os arquivos ficam em um único diretório, comumente chamado de diretório-raiz. Foram adotados nos primeiros computadores pessoais, já que, em geral, havia apenas um usuário. Possui diversas limitações, como a impossibilidade de existir dois arquivos com nomes iguais (dois arquivos *inbox* de usuários diferentes, por exemplo). Além disso, não favorece a organização dos arquivos se estes forem em grande número.

- **Dois Níveis** - Visando solucionar os problemas do sistema de diretórios de nível único, a solução foi criar um diretório para cada usuário. Desta forma, dois níveis de diretórios estão presentes, o diretório-raiz e os diretórios de usuário, acabando com a impossibilidade de arquivos com nomes iguais, pertencentes a usuários diferentes. Apesar de resolver o problema de nomes duplicados, este sistema de diretório não favorece a cooperação de dois usuário que desejam compartilhar um arquivo. Além disso, arquivos do sistema devem ser tratados de forma especial. Por exemplo, para não haver replicação de um arquivo binário do sistema no diretório de cada usuário, um usuário especial deve ser criado e todos os arquivos do sistema devem ser colocados lá. Para utilização destes arquivos, quando é invocado pelo seu nome, o sistema operacional deve buscar primeiro no diretório correntes (do usuário atual) e, em caso de falta, o diretório especial deverá ser consultado.

- **Estrutura em Árvore** - Os sistemas de diretórios em árvore são os mais utilizados por sistemas operacionais. Como o próprio nome diz, a estruturação dos diretórios se dá em árvore, onde a raiz é o diretório inicial, ou diretório-raiz, e os demais diretórios são ramos que partem da raiz. Seguinte a lógica, arquivos são as folhas, ou o ponto final da árvore. Usuários possuem diretórios correntes (geralmente seus diretórios de usuário ou *home*), de

onde seus arquivos são chamado. Caso algum arquivo fora do diretório corrente precisar ser acessado, o caminho completo do arquivo deve ser especificidade ou deve-se mudar o diretório corrente para o diretório onde o arquivo está presente. Fica claro que neste sistema de diretórios a especificação do caminho de arquivos é fundamental, de forma que há dois meios para determinar caminhos: *absoluto*, onde todo caminho, desde o diretório-raiz, é descrito; e o *relativo* que define o caminho a partir do diretório corrente. Esta abordagem possibilita ainda maior flexibilidade de organização, já que usuário podem criar e definir seus próprios diretórios a fim de sistematizar seus arquivos. Um aspecto importante que surgem com este esquema de diretórios é a política de remoção. A maioria dos sistemas operacionais só permite a remoção de diretórios vazios, forçando o usuário a remover primeiro arquivos e subdiretórios para depois remover o diretório. Outros sistemas, como o UNIX, permite que através da passagem de alguns parâmetros diretório não vazios sejam apagados. Isto pode pode ser mais conveniente para o usuário, porém apresenta grande risco de remoção acidental de diretórios essenciais do sistema operacional.

- **Estrutura de Grafo Acíclico** - É a evolução natural da estrutura em árvore como solução para compartilhamento de arquivos. Supondo que dois usuários queiram compartilhar um arquivo no qual estejam trabalhando juntos, é natural que as modificações feitos por um sejam visíveis ao outro. Desta forma, sistemas de diretórios em estrutura de grafo acíclico permitem que um mesmo arquivo ou subdiretório "apareça"em dois ou mais diretórios, sem que seja necessário copiá-lo. A figura 2.1 mostra um exemplo de diretório seguindo o esquema apresentado. Existem muitas maneiras de implementar o compartilhamento de arquivos e subdiretórios, sendo a mais comum delas a adotada por sistemas UNIX: a criação de *links*. Nesta solução um ponteiro para o arquivos ou diretório original é criado, e partir deste é possível ter acesso ao destino através da verificação de seu caminho. *Links* podem ser um problema quando o arquivo original é apagado e perde-se sua referência. Sistemas operacionais tratam este caso de duas formas: esperando até que o *link* seja acessado, para então acusar uma evocação ilegal de arquivo; ou aculumando um contador de referências ao arquivo e só apagando-o quando o contador for igual a zero. Uma alternativa aos *links* é a duplicação de arquivos ou diretórios. Neste caso, o sistema deve manter cópias fiéis em diversos diretórios, o que pode despender blocos de disco e processamento extra.

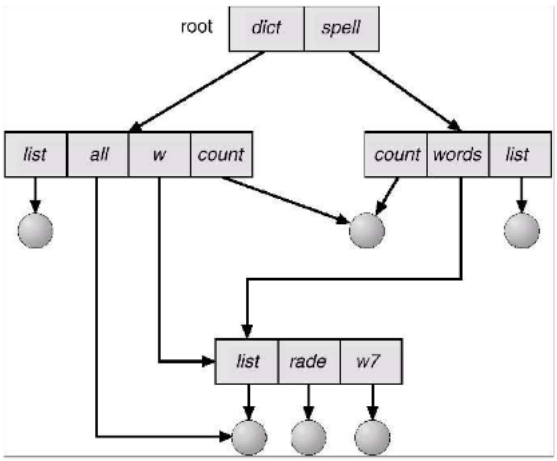

Figura 2.1: *Exemplo de diretórios com estrutura de grafo acíclico.*

2.3 Implementação de Sistemas de Arquivo

Nesta seção será abordado temas como indexação de arquivos, gerência de espaço livre, entre outros. Grande parte dos assuntos têm fundamental importância para as discuções apresentadas no capítulo seguinte.

Uma vez que sistemas de arquivos são armazenados em discos, é necessário haver alguma forma de organização interna. Esta tarefa está a cargo das partições, divisões lógicas de disco que podem conter algum sistema de arquivo. A tabela de partições de um disco é armazenada em setor especial chamado MBR (*master boot record*), que também é responsável indicar qual partição deve ser iniciada durante o book. Visando descrever o conteúdo das partições, estas as seguintes contém informações (TANENBAUM, 2003):

- **Bloco de boot.** Responsável por carregar o sistema operacional contído na partição.
- **Superbloco.** Contém informações sobre o sistema de arquivos (número mágico).
- **Gerenciamento de espaço livre.** Informações sobre os blocos livres da partição.
- ***I-nodes*.** Informações sobre os arquivos da partição.
- **Diretório-raiz.** Contém o topo da árvore do sistema de arquivos.
- **Arquivos e diretórios.** Conteúdo de dados propriamente dito.

2.3.1 Métodos de Alocação

Arquivos podem ser alocados de diversas maneiras dentro de um disco rígido, dependendo de suas capacidades e exigências de performance. Alguns deles serão apresentados abaixo.

- **Alocação contígua** - É o esquema mais simples de alocação, onde os arquivos são continuamente em blocos de tamanho fixo. Considerando um arquivo de 100 KB e blocos de 5 KB, o armazenamento ocupará 20 blocos consecutivos do disco. Esta abordagem possui vantagens de performance e simplicidade, já que é possível ler um arquivo através de um simples posicionamento de disco e guardar informações de um arquivo com apenas endereços dos blocos de início e final. Estas vantagens, porém, têm seu preço. Uma grande desvantagem é o gerenciamento de espaço livre em disco, já que após sucessivas alocações (apenas incluindo blocos no final) e remoções de arquivos o disco estará fragmentado, havendo possibilidade de não haver espaço contínuo para receber um arquivo. Este problema pode ser contornado através da realocação de todos os blocos do disco, no entando, seria muito custoso. Sistemas desta natureza são geralmente utilizados em mídias onde não há alocação e remoção de arquivos, como o CD-ROM.

- **Alocação por lista encadeada** - Nesta abordagem, cada bloco possui um ponteiro nos primeiros bytes para outro bloco, sendo possível alocar um arquivo de forma não contínua. Embora não haja fragmentação, já que todos os blocos podem ser utilizados, a leitura aleatória de um arquivo se torna bastante custosa. Caso se queira ler o enésimo bloco de um arquivo, será necessário consultar todos os blocos anteriores, ou seja, *n - 1* leituras de blocos.

- **Alocação por lista encadeada utilizando uma tabela na memória** - Utilizando o mesmo conceito da abordagem anterior, está técnica se diferencia por guardar toda a tabela de blocos na memória. Esta tabela recebe o nome de FAT (*file allocation table*). O ganho de desempenho é fruto da consulta à memória ao invés do disco. Além disso, cada bloco de disco pode ser utilizado inteiramente para alocação de dados, uma vez que não são necessários mais ponteiros em disco. A grande desvantagem deste método é a utilização de memória, já que a tabela deve estar inteiramente na memória todo o tempo.

- ***I-nodes*** - São estruturas de tamanho fixo que possuem informações dos blocos de um

arquivo. Cada *i-node* é um vetor de endereços de blocos, onde o enésimo membro do vetor possui o endereço do enésimo bloco do arquivo. Haja vista que *i-nodes* possuem tamanho fixo, é necessário encadea-los quando o arquivo não couber em apenas um. A técnica é bastante semelhante a alocação por lista encadeana, porém são encadeados *i-nodes* e não blocos de arquivos. Diferente da abordagem anterior, não é preciso manter um tabela correspondente a todo o sistema de arquivos, apenas os *i-nodes* de arquivos abertos devem estar na memória. Isto pode fazer com que o sistema operacional limite o número máximo de arquivos abertos simultaneamente, para que não haja esgotamento de memória.

2.3.2 Implementação de Diretórios

Diretório são importantes, pois podem conter informações sobre os arquivos, como proprietário, data de criação e principalmente um ponteiro para o bloco do arquivo (este ponteiro pode variar de acordo com a alocação adotada pelo sistema de arquivos). Sua principal função é mapear o nome ASCII de um arquivo em uma informação que possa localizá-lo. Sistemas operacionais mais antigos limitavam em poucos caracteres o tamanho dos nomes de diretórios, o que era mais conveniente, pois a alocação deste nomes na memória é simples. No entando, sistemas modernos permitem que diretórios e arquivos tenham nomes de até 255 caracteres, o que raramente é atingido. Assim, alocar espaços de tamanho fixo para entradas de diretórios e arquivos se torna dispendioso e causa desperdício. Uma alternativa é fazer com que cada entrada de diretório contenha um cabeçalho de tamanho fixo (composto do tamanho do diretório, proprietário, data da criação, etc.) e uma parte de tamanho variável para alocar o nome do diretório. Neste esquema, além da fragmentação interna, já que os bytes reservados para os nomes de diretórios são múltiplos de quatro, pode ocorrer o mesmo problema da alocação contígua de arquivo quando não há mais espaço para alocação de arquivos de determinado tamanho. Isto é resolvido separando a área de cabelhaço e de nomes, de modo que os cabelhos ficam todos aclomerados e possuam um ponteiro para os nomes de diretórios, que ficam em uma área temporária de memória (*heap*). Esta técnica não causa fragmentação interna e diretórios podem ser retirados sem que haja danos a alocação. A figura 2.2 exemplifica as duas abordagens discutidas anteriormente.

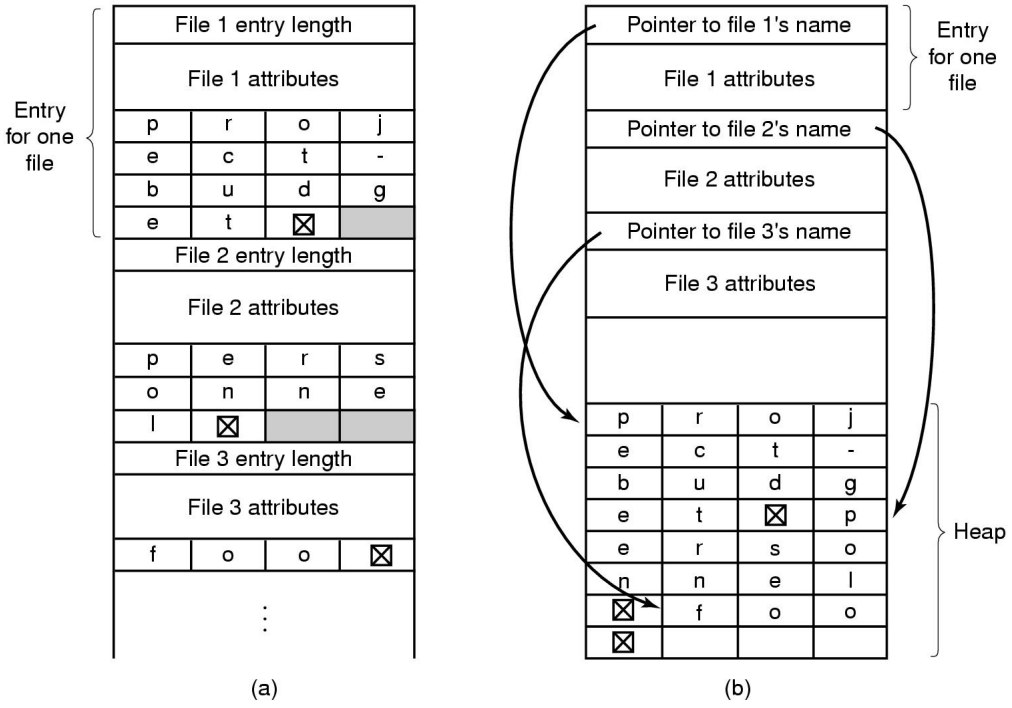

Figura 2.2: *(a) Mostra do primeiro esquema discutido; (b) Nomes da heap e seus ponteiros.*

2.3.3 Gerenciamento de Espaço em Disco

No projeto de um sistema de arquivo, um aspecto bastante importante é o tamanho dos blocos de disco. Blocos muito pequenos podem acarretar em muitas encadeações para persistir um arquivo, o que tornará sua leitura e escrita posterior bastante custosa. Por outro lado, blocos muito grandes causam demasiada fragmentação interna, já que arquivos podem ocupar apenas um pequena fração do bloco, inutilizando todo o restante. Para sistemas de propósito comum, é interessante que sejam feitos estudos para mensurar o tamanho médio de arquivos dos usuários, para melhor adaptar o tamanho do bloco às necessidades do usuário. Sistemas como o UNIX e o Windows utilizam tamanhos de blocos que variam de 1 KB à 32 KB.

O monitoramento de blocos livre depende diretamente do tamanho de bloco escolhido. Dependendo da técnica utilizada, guardar informações sobre blocos livre pode ser bastante dipendioso. Existem duas técnicas bastante utilizadas para gerenciar blocos livres: armazenar blocos livre em uma lista encadeada ou utilizar um mapa de bits. O mapa de bits é uma abordagem mais simples, onde para cada bloco há um entrada de um bit no mapa, ou seja, um disco com n blocos terá um mapa de n bits. Neste mapa, naturalmente, apenas dois valores são aceitos, 0 para blo-

cos livres e 1 para blocos ocupados. As grandes vantagens desta técnica é a baixa utilização de blocos para guardar o mapa de bits, que pode residir na memória; e sua simples implementação.

Já a lista encadeada de blocos livres segue uma abordagem semelhantes à lista encadeada de arquivos. Aqui, porém, cada bloco encadeado pode conter informações de *n* blocos livres, dependendo do tamanho do endereço de cada bloco. A lista, no entanto, ocupa maior espaço que o mapa de bits, além de poder acarretar em muitas operações de E/S quando um bloco é removido ou adicionado na lista. Visando minimizar as operações de E/S, o sistema pode manter na memória sempre lista meio cheias, para que um alocação de arquivo não cause transbordamento, forçando o sistema a escrever a lista em disco e ler outra lista com blocos livres.

3 *Formatos de Arquivos para Dados Científicos*

Experimentos científicos, como simulações físicas tridimensionais, geram grande massas de dados que podem chegar a ordem de terabytes (SHASHARINA et al., 2007). Deste modo, persistir e gerenciar tais dados não é uma tarefa trivial, bem como exige meios de armazenamento especialistas. Neste capítulo serão apresentados e analisados alguns sistemas de arquivos voltados para armazenamento de dados científicos ou de larga escala.

3.1 netCDF - network Common Data Form

O netCDF é uma iniciativa da Unidata, uma comunidade de pesquisadores de mais de 120 universidades e organizações (UNIDATA, 2008) e consiste em uma modelo para dados científicos, que também envolve bibliotecas para acesso aos dados e um format de dado binário. O modelo de dados do netCDF é baseado em dimensões, variáveis e atributos, ou seja, dados pode ser representados utilizando estas entidades de modo individual ou combinado. Cada um deles possui um name e um número de ID relacionado, para que seja possível realizar buscar dentro de arquivos netCDF. As dimensões podem representar alguma grandeza física do mundo real, como tempo ou temperatura; já os atributos guardam características sobre os dados armazenados, isto é, metadados, geralmente trazem informações de alguma variável; finalmente as variáveis são utilizadas para armazenar os dados concretamente. A figura 3.1 ilustra as possíveis relações em atributos, variáveis e dimensão.

Arquivos netCDF podem ainda conter grupos, noção semelhante a de diretórios apresentada anteriormente. São organizados de modo hierárquico e utilizados para esquematização de ar-

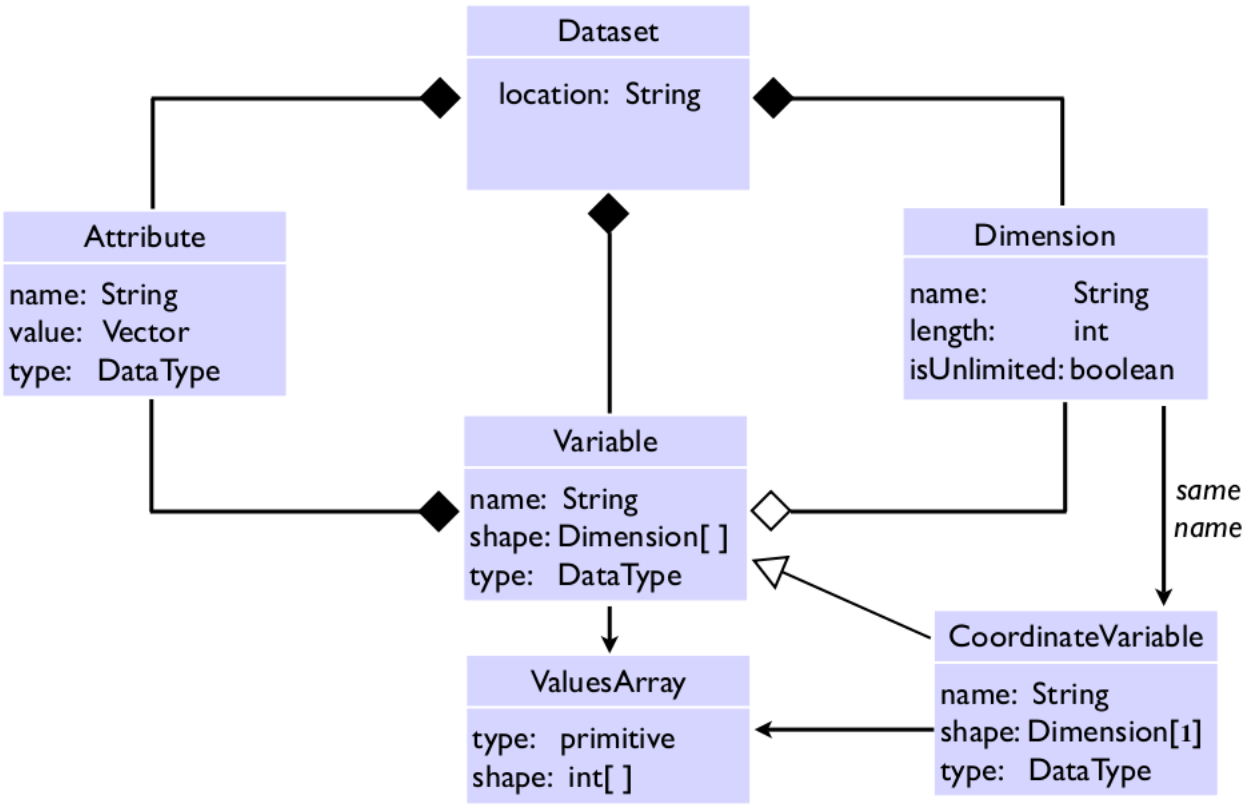

Figura 3.1: *Relações internas de um arquivos netCDF.*

quivos netCDF. Assim como o arquivo em si, podem conter atributos, dimensões e variáveis, bem como outros grupos. Abaixo um exemplo de arquivo netCDF onde variáveis de tempo, localização e condição atmosférica são criadas.

O netCDF ainda possui interfaces para diversas linguagens de programação, como C++, C, Java, Python, Ruby, entre outros. As interfaces permitem arquivos sejam manipulados através de operações semelhantes as apresentadas para sistemas de arquivos tradicionais, acrescentando métodos para acesso de alto nível no caso de algumas linguagens. Por fim, o formato dos arquivos netCDF foi projetado para isolar usuários, aplicações e dados das arquiteturas de máquinas, isto é, tornar o formato portável. Assim, foi criado um formato auto-descritivo, portável, de acesso direto e extensível (UNIDATA, 2008). O formato é composto de um cabelhaço que contém informações de dimensão, atributos e variáveis, seguido dos valores de variáveis de tamanho fixo e posteriormente variáveis de tamanho não fixo.

```
dimensions:
  station = 1000; time = UNLIMITED;
variables:
  double time(time);
    time:units = ‘‘hours since 2004-01-01’’;
  double lat(stationID); lat:units = ‘‘degrees_N’’;
  double lon(stationID); lon:units = ‘‘degrees_E’’;
  int elev(stationID);    elev:units = ‘‘meters’’;
  float temperature(time, stationID);
    temperature:units = ’’celsius’’;
  float pressure(time, stationID);
     pressure:units = ‘‘millibars’’;
```

Código 3.1.1: *Exemplo de um arquivo netCDF.*

O netCDF possui algumas limitações em relação a outras técnicas adotadas para armazenamento de dados científicos. Primeiramente, por não se tratar de uma base de dados, não possui indexação por chaves, o que pode ser prejudicial ao desempenho em consultas. Além disso, não permite escritas paralelas, que podem acarretar em inconsistência de dados. Outros dois problemas podem ser apontados: o tratamento de *strings*, que são vistas apenas como um vetor de caracteres; e a ausência de estruturas aninhadas, como árvores ou estruturas recursivas (UNIDATA, 2008).

3.2 HDF5 - Hierarchical Data Format

O HDF5 é a quinta versão de outro formato de dados científicos bastante utilizado no meio acadêmico (SHASHARINA et al., 2007), baseado em estruturação hierárquica de dados. Fruto da pesquisa do departamento de *Supercomputing* da Universidade de Illinois nos Estados Unidos, representa hoje não apenas um formato de dados, mas um conjunto de APIs e aplicações para manipulação de arquivos no formato HDF. Seu modelo de dados é semelhantes ao modelo do netCDF, porém com propriedades distintas (MACEDO, 2008).

O modelo de dados do HDF5 é divido em outros submodelos, como: modelo abstrado de dados, que define como são dados são representados conceitualmente; modelo de programação que manipula computacionalmente objetos do modelo abstrato de dados; e o modelo de armazenamento, o qual trata do armazenamento físico dos arquivos HDF5. A figura 3.2 mostra as

relações entre os modelos citados.

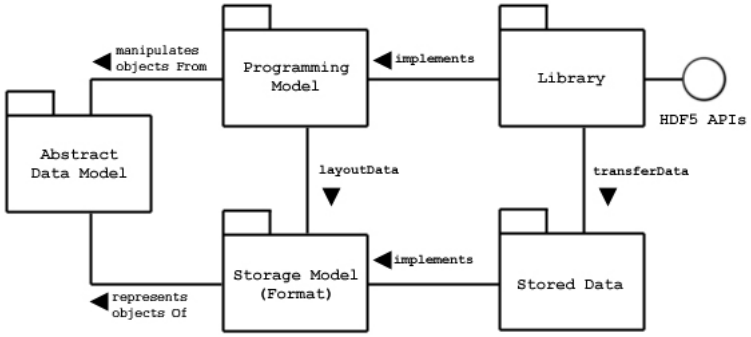

Figura 3.2: *Relacionamento entre os modelos do HDF5.*

Para definir e descrever dados no HDF5 os seguintes conceitos são utilizados:

- **Arquivo** - Conceitualmente é um contêiner que pode conter grupos e conjuntos de dados, além de outros objetos definidos posteriormente. Todo arquivo deve conter pelo pelo um grupo raiz, onde todos os objetos devem pertencer ou ser descendentes deste grupo. Além disso, cada objeto dentro de um arquivo deve ter um nome único, para que possa ser identificado e acessado. Bem como sistemas UNIX, o HDF permite que arquivos sejam montados dentro de outros arquivos (como subgrupos), evitando assim replicação desnecessária de dados.

- ***Group*** - A exemplo do netCDF, grupos em arquivos HDF equivalem a diretórios de sistemas UNIX. A única diferença, no entando, é a obrigatoriedade de pelo menos um grupo (o grupo-raiz) em arquivos HDF.

- ***Dataset*** - É um *array* multidimensional de elememtos de dados, o qual possui descrições de metadados e outroa atributos.

- ***Dataspace*** - Descreve um *array* multidimensional através de espeficicação de seus elementos (tamanho atual e tamanho máximo). Na para seleção de subconjuntos de dados, pode ser utilizado para descrever *hyperslab*.

- ***Datatype*** - É a descrição de cada dado de um elemento, por exemplo, a descrição de cada

valor de um *array*, se é um inteiro ou um caracter. Além disso, descreve também como este tipo de dado deve ser armazenado.

- ***Attribute*** - Assim como no netCDF, atributos descrevem o dado representado. No HDF, porém, cada grupo, *datatype* ou *dataset* pode possuir, ou não, atributos. Todos os atritutos são nomeados e possuem valor.

- ***Property List*** - Coleção de parâmetros que controlam opções da biblioteca, assim como os atritutos descrevem os dados. Cada propriedade tem um nome, um *datatype* e um valor. A lista de propriedades é anexada ao objeto e pode ser usado por qualquer elemento. Em geral, este recurso é utilizado para passar parâmetros para programas externos, como um *driver* VLF.

Por fim, o modelo de formato de arquivo do HDF define como objetos serão mapeados para espaços de endereços lineares, assumindo que que o espaço de endereço é um *array* contínuo de *bytes*. Sua organização se dá em três partes, o Nível 0, 1 e 2. O Nível 0 descreve o cabeçalho do arquivo, com informações sobre sua versão, assinatura, parâmetros de descrição do arquivo e ponteiros para os outros níveis do arquivos. Modelo semelhante foi apresentado quando diretórios foram discutidos. O Nível 1, por sua vez, define como serão as estruturas internas do arquivo, tal como árvores, *heaps* ou grupos. Por fim, o Nível 2 trata da organização do dados e objetos, de forma semelhante ao Nível 1, porém focando nas estruturas para dados propriamente ditos (GROUP, 2008). A figura 3.3 sintetiza a organização de um arquivo HDF5.

1. **Level 0**: File Signature and Super Block
2. **Level 1**: File Infrastructure
    a. **Level 1A**: B-link Trees and B-tree nodes.
    b. **Level 1B**: Group
    c. **Level 1C**: Group Entry
    d. **Level 1D**: Local Heaps
    e. **Level 1E**: Global Heap
    f. **Level 1F**: Free-space index
3. **Level 2**: Data Object
    a. **Level 2A**: Data Object Headers
    b. **Level 2B**: Shared Data Object Headers
    c. **Level 2C**: Data Object Data Storage

Figura 3.3: *Estruturação interna de um arquivo HDF5.*

4 Conclusão

Este trabalho apresentou conceito de sistemas de arquivos e suas principais implementações. Foram apresentados conceitos básicos de arquivos e implementação de sistemas de aquivos, onde foi possível estudar e notar as principais diferenças das soluções expostas. Fica claro que não há uma solução ótima genérica, que abrage todos os casos. Um vez que sistemas operacionais estão presentes sem grande partes dos aparelhos eletrônicos, escolher ou implementar um sistema de arquivos deve ser um tarefa cuidadosa e criteriosa, para que a melhor solução possa ver escolhida de acordo com as necessidades.

O trabalho também concentrou seu foco em formatos de arquivos para dados científicos, que vem crescendo muito. Por se tratarem de dados especiais, com características próprias, é desejável que haja alguma otimização quando a persistência destes dados. As soluções apresentadas são amplamente utilizados no meio acadêmico e provem bastante flexibilidade na representação de dados das mais diversas naturezas. No entando, é importante concentrar ações de pesquisa também em métodos para recuperação e consultas destes dados. Um vez que, em geral, são grandes massas de dados, a pesquisa por algum parâmetro espefícido não é algo trivial. Por fim, espera-se que o trabalho tenha atingido seu objetivo no sentido de esclarer dúvidas e abrir caminhos para pesquisas mais profundas.

Referências Bibliográficas

GOSINK, L. et al. Hdf5-fastquery: Accelerating complex queries on hdf datasets using fast bitmap indices. In: *Scientific and Statistical Database Management, 2006. 18th International Conference on*. [s.n.], 2006. p. 149–158. Disponível em: <http://dx.doi.org/10.1109/SSDBM.2006.27>.

GROUP, T. H. *HDF Group - HDF5*. 2008. Acesso em: Agosto de 2008. Disponível em: <http://hdf.ncsa.uiuc.edu/HDF5/>.

LI, J. et al. Parallel netcdf: A high-performance scientific i/o interface. In: *Supercomputing, 2003 ACM/IEEE Conference*. [s.n.], 2003. p. 39. Disponível em: <http://dx.doi.org/10.1109/SC.2003.10053>.

MACEDO, D. D. J. de. *Um Estudo de Estratégias de Sistemas Distribuídos Aplicadas a Sistemas de Telemedicina*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2008.

SHASHARINA, S. G. et al. Distributed technologies for remote access of hdf data. In: *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Washington, DC, USA: IEEE Computer Society, 2007. p. 255–260. Disponível em: <http://portal.acm.org/citation.cfm?id=1339263.1339599>.

SILBERSCHATZ, A. *Operation System Concepts*. 5. ed. New York: John Willey and Sons, 1997.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 2. ed. São Paulo: Prentice Hall, 2003.

UNIDATA. *NetCDF (network Common Data Form)*. 2008. Acesso em: Agosto de 2008. Disponível em: <http://www.unidata.ucar.edu/software/netcdf/>.