Confiabilidade e Desempenho 1

Marcelo Johann

Lembrando: gerenciamento de espaço livre

- 2 problemas foram vistos:
  - Escolha do tamanho de bloco adequado
    - Se for muito pequeno, se gasta muito em seek/latência
    - Se for muito grande, se chega a fragmentação.
    - Mais um meio-termo a achar!
  - Mecanismo de gerenciamento dos blocos livres
    - Lista encadeada vs. Bitmap.

30 segundos de filosofia: informação x processamento

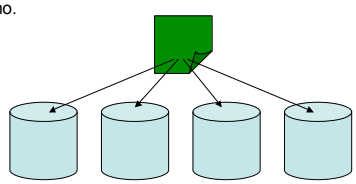
- Informática = **processamento** de **dados**.
- **Processamento** é cada vez mais rápido e barato
  - Ciclo de relógio menor, integração maior, paralelismo, nanotecnologias, lei de Moore, etc...
- Dados = **informação** é cada vez mais preciosa:
  - Sucesso da rede (Web, redes P2P...)
  - Banco de dados / data mining
  - Bibliotecas digitais (Google)
- Perder um computador pode ser chato, mas nem tão problemático.
- Perder um disco pode ser extremamente prejudicial!

Confiabilidade & Desempenho

- O disco deve armazenar dados de forma consistente e duradoura
  - Confiabilidade é uma característica fundamental do sistema de arquivos.
    - O HW é falível!
  - Pode ser auxiliada pelo HW, e/ou aumentada pelo SW
    - Discos RAID
    - Diagnóstico/manutenção/conserto de problemas pelo Sis. Op.
- O acesso ao disco é naturalmente demorado
  - Desempenho deve ser garantido.
  - Emprego de cache de HW e de técnicas de SW (vide tabelas Hash, escalonamento de acessos...)

Discos RAID

- Redundant Array of Independent Disks
  - Combinação de vários discos físicos para prover redundância
  - O usuário só enxerga um disco único!
  - Visa o aumento do desempenho e da confiabilidade
    - Paraleliza os acessos e garante *backups*.
  - Existem diversos níveis de discos RAID conforme o grau de paralelismo.



Níveis de RAID

- **RAID-0**: os arquivos estão recortados em *strips*, e os mesmos estão espalhados entre os discos.

  - **RAID-1**: espelhamento
    - Há cópia física de todo um disco em (pelo menos) um outro.
  - **RAID-2-3-4**: stripping + paridade armazenada separadamente.
    - 2: paridade / bit ; 3: paridade Byte ; 4: paridade / bloco.

RAID (fim)

- Existem outras combinações...
  - Espelhamento + stripping
  - Distribuição da informação de paridade...
- Cada vez que há redundância, se perde espaço "útil"
  - Mas se ganha em segurança e em tempo de leitura!
- Usar discos RAIDs necessita de um sistema de arquivo especial em nível do Sis. Op.
- Discos RAID podem também ser simulados em software...
  - Diminui o tempo de E/S!

Backups

- Estratégico!
  - Um dos problemas é: onde fazer o backup...
- Backup integral: todo um sistema de arquivos é copiado.
- Backup incremental:
  - Parte-se de uma versão inicial do sistema de arquivos
  - só as diferenças (atualizações) são armazenadas a partir da última versão.
  - No Linux: existe o comando 'rsync'
- Política de backup:
  - 1 backup integral cada semana;
  - Backups incrementais todos os dias.

O problema dos blocos defeituosos

Existem duas grandes fontes de problemas para um sistema de arquivo:

- De repente, um bloco (setor) do disco **estraga**
  - Problema material
  - Perda dos dados no bloco, mais problema potencial ao acessar o bloco.
- **Perda da coerência** entre as estruturas de dados
  - Queda de luz, "CTRL-ALT-DEL" violento...
  - Parte dos blocos copiados na RAM ou em Cache não esteve atualizada no disco.
  - Perda de dados, ou pior, de acesso a parte do sistema de arquivos.

Blocos defeituosos

- Os discos incluem pelo menos um setor extra por trilha, que servem para compensar um defeito em um (vários) setor(es) da trilha.
  - O próprio controlador do disco pode remapear os setores se for preciso.
- O sistema de arquivos mantém uma **lista dos blocos defeituosos**.
  - Inicializada desde a formatação do sistema de arquivos!
  - Evita trancar o sistema de arquivos, mas não prevê a perda dos dados no(s) bloco(s) defeituosos.

Consistência do sistema de arquivos

- Problema crucial quando blocos "perdidos" contêm inodes, listas (de blocos defeituosos por exemplo), diretórios...
- Uso de programas de sistema especiais que verificam a consistência de um sistema de arquivos
  - fsck no linux/Unix.
- Podem ser executados ao boot
  - Útil após um crash.
  - Demorado, pois varre todo o sistema de arquivos!

Para piorar...

- O que acontece com escritas?
- Considere a criação de um novo arquivo em um diretório:
  - /home/johann/arquivo.txt
  - Deve-se, potencialmente, escrever dados:
    - No inode do diretório /home/johann (para alterar os números de links, por exemplo);
    - Em um bloco de dados do mesmo inode (para inserir uma entrada)
    - No inode do novo arquivo 'arquivo.txt' (nome do arquivo...)
    - Em pelo menos um bloco apontado pelo mesmo.
  - Têm pelo menos 4 blocos envolvidos.
- Se houver um *crash* enquanto isso, há possibilidade séria de alguma inconsistência.

Fsck: consistência de blocos

Constrói duas tabelas:

- Cada uma contém um contador por bloco, inicializado com zero.
  - 1a tabela: quantas vezes um bloco está referenciado por um arquivo
  - 2a tabela: quantas vezes um bloco está na lista de blocos disponíveis.
- Lê todos os inodes e varre recursivamente todos os blocos usados
  - Atualiza a 1a lista
- Lê a lista de blocos disponíveis
  - Atualiza a 2a lista

Resultados e medidas a tomar

- **1o caso**: cada bloco aparece UMA vez só em apenas uma tabela.
  - Tudo está bem.
- **2o caso**: um bloco não aparece em nenhuma tabela (***missing block***)
  - Espaço perdido...
  - Solução simples: o bloco entra na tabela de blocos livres.
  - (obs: perder-se-á o conteúdo do bloco!)
- **3o caso**: duplicação de um bloco na lista de blocos disponíveis
  - Limpa a lista.
- **4o caso**: duplicação de um bloco na lista de blocos em uso.
  - Recupera um bloco livre, copia o duplicado nele, e usa o novo bloco em um dos dois arquivos.

fsck: consistência de diretórios

- Verifica a consistência dos diretórios
  - Operação semântica.
- A partir da raiz, varre a árvore de diretórios
  - Cria uma lista: para cada inode/arquivo, contabiliza o número de diretórios que o contêm.
  - Após isso, compara com o contador contido no inode (número de links)
    - Ambos devem ser iguais!
- Caso haja mais links do que caminhos:
  - O arquivo não será deletado em devido tempo.
  - Solução: seta o número de links igual ao número de caminhos.
- Caso haja menos links do que caminhos:
  - Muito grave! Vai ser (ou já foi) perdido dados...
  - Solução: seta o número de links igual ao número de caminhos.

Desempenho e Cache de disco

- Tempo de acesso ao disco >> tempo de acesso à RAM.
- Deve-se usar mecanismos de Cache para agilizar o acesso.
  - “Cache” vem do francês “Cacher” = “esconder”. :o)
- Cache = coleção de blocos, logicamente pertencendo ao disco, mas temporariamente na RAM.
- Enquanto só têm leituras, não tem dificuldade:
  - A 1a leitura traz o bloco para a Cache, as leituras sucessivas lêem õ bloco na Cache.

Políticas de atualização de Cache

Quando a cache está cheia...

- É preciso descartar um bloco que está na Cache;
  - Vide políticas LRU/FIFO,2a chance da paginação...
  - Os acessos são mais raros e mais bem ordenados do que no caso da memória.
  - Antes disso, se foi alterado, é preciso copiá-lo no disco.
    - Vide bit de sujeira das páginas!
- Problema com LRU:
  - a política é deixar dados “envelhecer” na Cache.
  - Quanto mais tempo passa antes de ser gravado no disco, maior o risco de crash/inconsistência!

Resumo – sistemas de arquivos

Setores, blocos e clusters, dados e meta-dados...

- Em nível de Hardware, existem **setores**
  - Trilhas, cilindros...
- Em nível do FS, os setores são agrupados em **blocos** (ou **clusters**)
  - Problema do tamanho do bloco/cluster
- Blocos são usados pelo FS para:
  - Armazenar dados "brutos" (**blocos de dados**)
  - Armazenar informação de gerenciamento dos blocos de dados (**meta-dados**)
- Meta-dados e dados formam o **Sistema de Arquivos**, enxergado pelo usuário como:
  - Diretórios, arquivos, informações de acontabilidade...

O que está nos meta-dados?

- Todo o necessário à administração dos dados:
  - Endereços de blocos (de dados ou de outros meta-dados, por exemplo de diretório)
  - Quando foi acessado um arquivo/diretório
  - O dono do arquivo/diretório
  - O tamanho (número de blocos usados)
  - Direitos de acesso
  - Blocos livres
    - Bitmap
    - lista

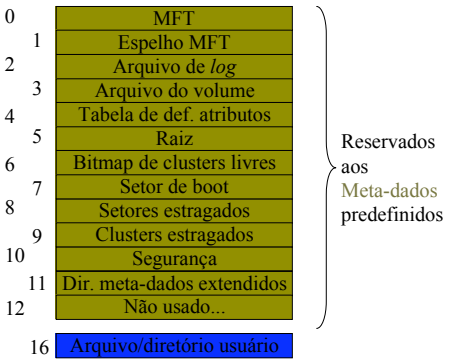
Windows 2000+ --- NTFS

- FS do Windows desde 2000 (sucessor da FAT)
- Agrega os setores em clusters
  - 512 Bytes (partições de 512 MBytes)
  - Até 4 KB (partições de mais de 2 GB)
- Define volumes (partições);
- Incluí links, compressão, jornalização...
  - Atomicidade (transações) das operações críticas
    - Criação de arquivos/diretórios, aumento de tamanho, remoção...
  - Redundância de arquivos críticos
    - Caso estrague um setor.
  - garante os meta-dados, não os dados.

Implementação da NTFS: MFT

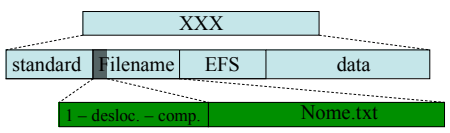
- Descritores de arquivos/diretórios são índices de entradas na ***Master File Table*** (MFT)
- A MFT é uma tabela de entradas (***records***), cada uma de tamanho fixo 1 KB.
  - Obs: se um cluster é identificado por um endereço de 4B, cada entrada contém no máximo 256 endereços de clusters.
    - 256 x 1K = 256 KB.
- Há um *record* por arquivo/diretório no volume.
  - Não estoura, graças ao mecanismo de jornalização!
- A cada *record* é associado um arquivo extra, que contém os meta-dados.
  - O resto dos arquivos do volume contém os dados dos usuários.

Os *records* da MFT



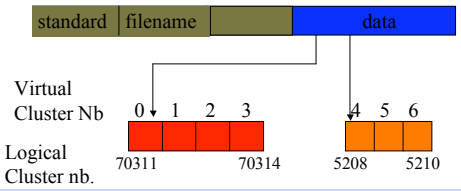
Conteúdo de um *record*

- Para o NTFS, um arquivo é um conjunto de pares (atributo, valor)
  - Dados constituem apenas um atributo (unnamed data attribute)
- Exemplos de atributos:
  - Standard information: Read-only, data de criação, número de links...
  - Filename: Nome,
  - Data: dados no caso de arquivos ou endereço de arquivos na MFT no caso de diretórios
  - EFS: criptografia, etc...
- Atributos podem ser residentes ou não.



Atributos não residentes

- Quando um atributo se torna muito grande (> 1 KB), ele se torna não-residente.
  - O record armazena apenas ponteiros sobre um espaço no disco **fora** da MFT (*run*);
  - Pode haver mais de um *run* por atributo.
    - Usado quando há muitos arquivos num diretório, quando um arquivo é muito grande, ...
    - Guarda uma tabela de mapeamento VCN/LCN
  - É um mecanismo de lista!



NTFS e abertura de um arquivo

- A partir da raiz, se percorre o caminho até o arquivo
  - Ex. \D:\Windows\My Documents\aula.ppt
  - Cada record associado a um diretório será lido:
    - Deve ser carregado do disco para a memória
      - Na verdade, já há cópia na RAM.
    - Verifica-se os direitos de acesso
      - Na verdade, já há cópia na RAM.
    - Encontra-se o diretório seguinte no caminho
      - Na verdade, já há cópia na RAM.
  - Determina-se os clusters onde se encontram os dados

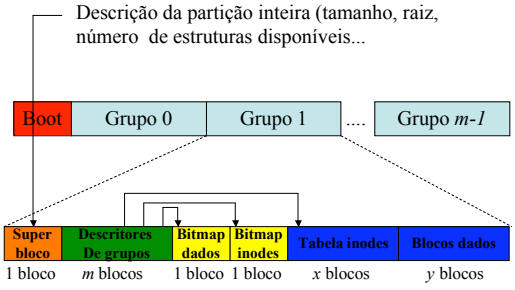
Linux – ext3fs

- Diretórios e arquivos são implementados através de inodes:
  - Inode = alguns campos de meta-dados + ponteiros para blocos de dados.
  - A tabela de inodes contém todos os inodes existentes no FS. Inodes são referenciados na tabela.
- Não há Master File Table (tabela de entradas):
  - Os blocos são organizados em **grupos**
  - Cada grupo é descrito por um **super-bloco**
  - O superbloco aponta para os grupos que contém apontadores para suas estruturas internas
    - Estrutura de lista.

Descrição dos grupos no ext3fs

- Super-bloco:
  - Contém meta-dados relevantes a toda a partição (duplicado em cada grupo)
    - Número mágico da partição,
    - número de *mount*s,
    - tamanho do bloco,
    - tamanho do grupo (*m*),
    - ponteiro para o 1o inode do sistema de arquivos (/),
    - Número de inodes, de inodes livres, de blocos livres...
- Descritor do grupo:
  - 1 entrada por grupo
  - Cada entrada fornece:
    - O endereço do bloco onde está o bitmap dos blocos livres;
    - O endereço do bloco onde está o 1o inode na tabela.
    - O endereço do bloco do bitmap dos inodes

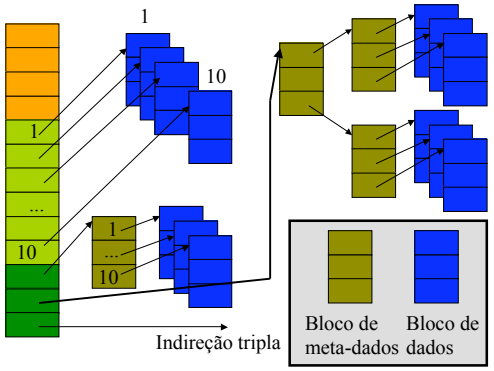
Representação dos grupos



Conteúdo do inode

- Meta-dados:
  - Tipo de arquivo (4 bits), direitos de acesso
  - Número de links
  - Dono (UID+GID)
  - Tamanho em bytes
  - Hora de último acesso
  - ...
- 10 ponteiros para blocos de dados
- 1 ponteiro para blocos que vão conter ponteiros sobre blocos de dados
  - Indireção dupla
- 1 ponteiro de indireção tríplice

Linux – inode



Ext3fs e abertura de um arquivo

- A partir da raiz, se percorre o caminho até o arquivo
  - Ex. /home/nicolas/aula.pdf
  - Cada inode associado a um diretório/arquivo será lido:
    - Carrega-se do disco para a memória
      - Na verdade, já há cópia na RAM.
    - Verifica-se os direitos de acesso
    - Encontra-se o bloco que contém os dados
  - Determina-se os blocos onde se encontram os dados

Próxima aula…

- **Confiabilidade e desempenho 2**
- **Sistemas de Arquivos Jornalizados**