Confiabilidade e Desempenho 2 Jornalização

Marcelo Johann

Lembrando: 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...)

Qual é o problema?

- CPUs estão se tornando mais rápidas;
- RAMs estão se tornando (bem) maiores;
- Conseqüência 1: pode-se usar caches de disco na RAMs cada vez maiores.
- Conseqüência 2: acessos em leitura nos discos se torna uma coisa menos freqüente.
  - "Read-ahead"
  - As escritas ainda devem ser feitas para garantir a coerência.

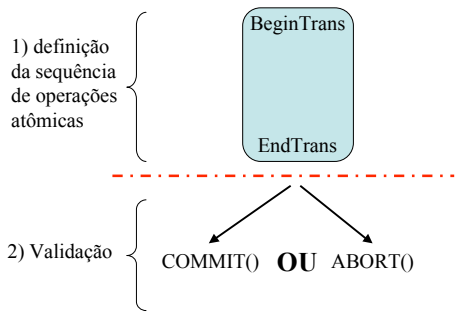
Para piorar...

- O que há com escritas?
- Considere a criação de um novo arquivo em um diretório:
  - /home/nicolas/arquivo.txt
  - Deve-se, potencialmente, escrever dados:
    - No inode do diretório /home/nicolas (para alterar os número 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.

Escrita seqüencial e atômica

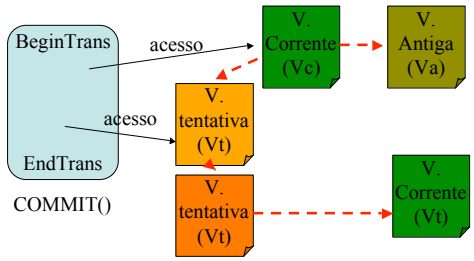
- Deve-se usar um mecanismo de **transação**:
  - Conceito de banco de dados.
  - "um negócio é fechado somente após assinatura"
  - Verifica 4 propriedades:
    - Atomicidade (não pode ser interrompido)
    - Consistência (coerência interna)
    - Isolamento (independência da multiprogramação)
    - Durabilidade (uma vez concluída, fica permanente)
- Efetuadas em 2 fases: commit/validação.

As duas fases da transação



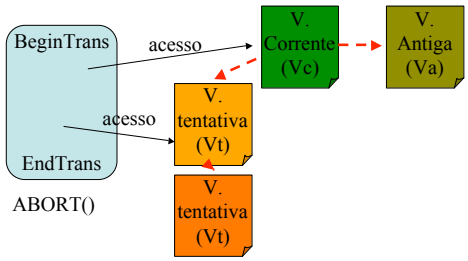
Commit e versionamento

- Até o commit ter sido efetuado, é preciso manter o sistema anterior do FS
  - Em caso de crash, será recuperada a versão anterior (*roll-back*)



Commit e versionamento

- Até o commit ter sido efetuado, é preciso manter o sistema anterior do FS
  - Em caso de crash, será recuperada a versão anterior (*roll-back*)



Princípio da jornalização

- Idéia simples:
  - Virtualizar o disco em um grande arquivo de *log*, onde se pode apenas acrescentar dados no fim.
  - Escrita **seqüencial** do arquivo de *log* no disco, por grandes pedaços (*chunks*)
    - Aumenta a vazão de escrita!
  - Quando há um *crash*, basta examinar o fim do arquivo de *log* para verificar as incoerências.
- Primeiro projeto a usar isso: Log-structured File System (Berkeley, 1991).

Elementos a considerar (1)

- **O que entra** no *log*?
  - Tudo...
    - Perfeito, mas caro!
  - Ou apenas os meta-dados.
    - Econômico e garante a coerência, mas o usuário pode perder (blocos de) dados.
- **Operações *vs.* Valores**
  - Pode-se logar as ações do usuário, ou o resultado dessas ações.
  - O interesse depende do tamanho da informação processada.
- **Trocar todo o FS** ou apenas melhorá-lo?
  - Re-implementar tudo (vide NTFS vs. FAT) implica em abrir mão de algumas estruturas... Perde-se compatibilidade!
  - Pode-se apenas acrescentar um FS com funcionalidades de log.

Elementos a considerar (2)

- **Redo *vs.* Undo**
  - Redo-only log: armazena apenas os valores que foram alterados;
    - Simplifica o trabalho
    - Ncessita garantia sobre a ordem de armazenamento dos valores.
  - Undo-redo log: armazena antigos e novos valores dos dados.
    - Logs maiores, mas mais poderoso.
- **Coletor de lixo**
  - O log será de tamanho finito...
  - Vai ser preciso re-aproveitar lacunas nele!
- **Escrita dos *chunks***
  - Quando, com qual freqüência,...
  - Qual será o tamanho do *chunk*?
- **Como recuperar os dados** do log?

Estudo de caso: Unix BSD 4.4

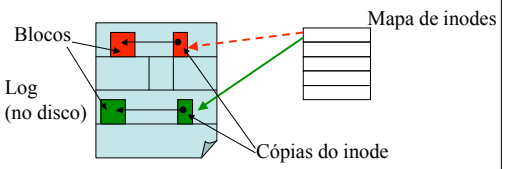
- O sistema de arquivos é chamado BSD-LFS.
- Todo o disco é usado como um grande arquivo de log:
  - Cada escrita é feita no fim do log;
  - O log é divido em **segmentos**, encadeados numa lista.
    - Pode haver seek entre dois segmentos.
  - Um processo 'cleaner' recupera espaço e gerencia um acesso circular aos segmentos do log.
- Sistema de log acrescentado à estrutura clássica de FS:
  - Inode, diretórios...
  - Mas cada inode pode ser colocado num lugar aleatório do disco, conforme for o segmento!
  - Deve-se rastrear os inodes dentro dos segmentos: usa-se um mapa de inodes.

O segmento no BSD-LFS

- Um segmento tem tamanho típico ½ MBytes.
  - Olhem o tamanho!!! É grande, pois visa o uso de toda a vazão de escrita!
- É grande demais para poder garantir a atomicidade da escrita
  - Risco de crash entre duas escritas;
  - A Cache pode encher;
  - Pode-ser exigida a sincronização (sync) antes de ter completado um segmento.
- Define-se um **segmento parcial**, para qual se garante a escrita atômica:
  - Constituído de dados + um header:
    - Checksums, endereço no disco de cada inode no segmento parcial, dos blocos de dados...
- Mantém-se também uma tabela de uso dos segmentos.

Escrita do Log

- Os blocos usados estão agregados nos segmentos.
- Eles estão ordenados pelo número de bloco, no segmento, para agilizar a gravação no disco.
  - Importante: essa ordenação dinâmica implica em **atualizar os endereços** dos blocos acessados **nos inodes**!
  - Cada vez que se altera o endereço de um bloco, a versão antiga do bloco é descartada
    - Re-aproveitamento possível pelo cleaner.
- A escrita limpa o segmento da Cache



Funcionamento

- A tabela de uso de segmentos e o mapa de inodes são copiados mais frequentemente para o disco.
  - Eles são cruciais!
- **Para recuperar** os dados:
  - É preciso localizar os inodes no disco
  - Usa-se o mapa de inodes para obter o endereço físico!
- Usa-se uma cache grande para evitar de sempre procurar no disco os inodes.
- O processo **cleaner**: coleta de lixo!
  - Procura-se segmentos com lacunas;
    - A tabela de uso dos segmentos é usada!
  - Copia-se o que ainda está no segmento num outro segmento no fim do log;
  - Recupera-se assim um segmento todo.

Quando há crash...

- Recupera-se a versão mais recente da tabela de uso de segmentos e o mapa de inodes no disco.
- Efetua-se o *replay* do que ficou gravado no log desde então.
  - Usa-se informações de data para se ter certeza que as informações estão válidas!
- Os *checksums* são usados para garantir a correção dos segmentos parciais recuperados.
- Não há como recuperar o que estava em setores defeituosos!

Limitações do BSD-LFS

- Pode haver vários meta-dados a respeito de um arquivo/diretório; todos devem constar no mesmo segmento parcial, se não pode-se chegar a inconsitências.
- Alocação de espaço livro no disco
  - Deve ser feita desde a escrita no log!
- É preciso de muita memória RAM.
- Sobrecusto de gerenciamento:
  - O desempenho aumenta nitidamente, devido ao bom uso da vazão, quando se medem FS com muitos metadados
    - Muitos diretórios, muito arquivos pequenos.
  - Quando se têm poucos arquivos, muito grandes, aumento menor.

Outros sistemas de arquivos jornalizados

- Para o Unix/Linux:
  - XFS (SGI/Linux)
  - JFS (IBM-AIX/Linux)
  - Reiser-FS (Linux)
  - Ext3fs (Linux)
- Microsoft: NTFS

ext3fs - Linux

- Evolução recente do ext2fs
  - Ficam 100% compatíveis
    - Pode-se passar transparentemente de um para o outro.
      - mke2fs / tune2fs
    - O ext3fs cria um arquivo .jornal na raiz do FS
  - Conforme escolha do usuário, pode ser configurado para jornalizar:
    - Os metadados só;
    - Ou os metadados e os dados
      - Atrasa o processo.
  - Os metadados incluem os inodes, os superblocos, descritores de grupos e os bitmaps.
- Jornalização feita em 2 *commits*:
  - Copia dos metadados alterados para o log
    - Se há falha antes, as alterações são perdidas
  - Escrita do log no disco
    - Se há falha antes, recupera os metadados do log.

NTFS

- Chkdsk é o equivalente do fsck do Unix
- Usa transações para garantir a coerência
  - Garante a jornalização dos meta-dados só.
- Usa jornalização
  - Arquivo de log armazenado dentro da MFT
  - O log é circular (= o caso do FS de Berkeley!)
- A manipulação do arquivo de log é feito pelo Log Fileystem Service
  - = daemon
  - Efetua os trabalhos de escrita, leitura e "cleaner" do BSD-LFS.
- Possibilita o Undo (roll-back)/Redo
- Exemplos de transações logadas:
  - Criação, remoção, extensão, alteração das informações sobre o arquivo,...

Resumindo...

- Todos os FS recentes provêm jornalização
  - NTFS, ext3fs...
- Mecanismo baseado na transação
  - Commit
  - Possibilidade de roll-back.
- O jornal armazena as alterações feitas no FS
  - Meta-dados, em geral.
- E necessário gravá-lo no disco regularmente para poder re-usá-lo quando necessário.
- Implica em um sobrecusto de gerenciamento, limitado pelo fato de o mesmo ser feito na RAM.

Próxima aula…

Sistemas de Arquivos Distribuídos

Dúvidas