

Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/S Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

*Sistemas de Ficheiros:
Recuperação de falhas: journal*

Alguns cenários... (relembrando)

- **Cenário 1:** Todos ☺ os processos a ler ficheiros
 - Aqui, assume-se que *nenhum processo está a escrever em ficheiros.*
- **Cenário 2:** Alguns processos a escrever em ficheiros
 - 2a) Os processos escrevem, mas a dimensão dos ficheiros não se altera.
 - 2b) A dimensão de alguns ficheiros cresce.
- **Cenário 3:** Um cenário real!
 - Há ficheiros a serem lidos e outros lidos/escritos; há-os a serem criados e apagados...

Anomalias e file system checkers

- Que se pode esperar num SF, como resultado das anomalias, nos diferentes cenários?
 - Inconsistências no superbloco
 - “magic number” indica um FS desconhecido ou incorrecto
 - Dimensões inconsistentes para as estruturas de metadados
 - Inconsistências entre os bitmaps e os blocos ou inodes
 - Blocos de dados marcados livres que os inodes dizem estar ocupados ou vice-versa; inodes marcados livres com conteúdo correcto ou ocupados com conteúdo incorrecto
 - Inconsistências entre as directorias e os inodes
- Solução usando file system checkers
 - Muito lenta em volumes com centenas de milhar de ficheiros

Solução #2: journaling

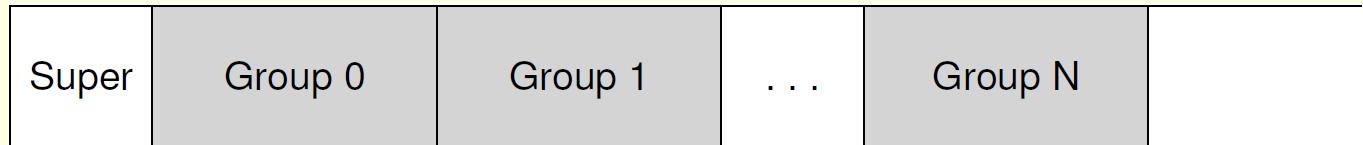
□ *Journaling*

- Parte de uma técnica usada nas BDs e que nesse contexto é designada write-ahead (ou redo) logging.
- Consiste em escrever, numa localização precisa do file system (por vezes num volume ou num grupo de blocos dedicado a esse efeito), informação sobre as alterações que se vão efectuar antes de se fazer essas alterações nos estruturas do SF (bitmaps, inodes, etc.)
- Se houver um CFE (crash ou falha de energia) que deixe estruturas inconsistentes, faz-se o replay do log para as corrigir.

Journaling no ext3 (1)

□ Linux ext3

- Pode, de uma forma simples, ser visto quase como um SF ext2 com acrescento de logging.
- Em vez de

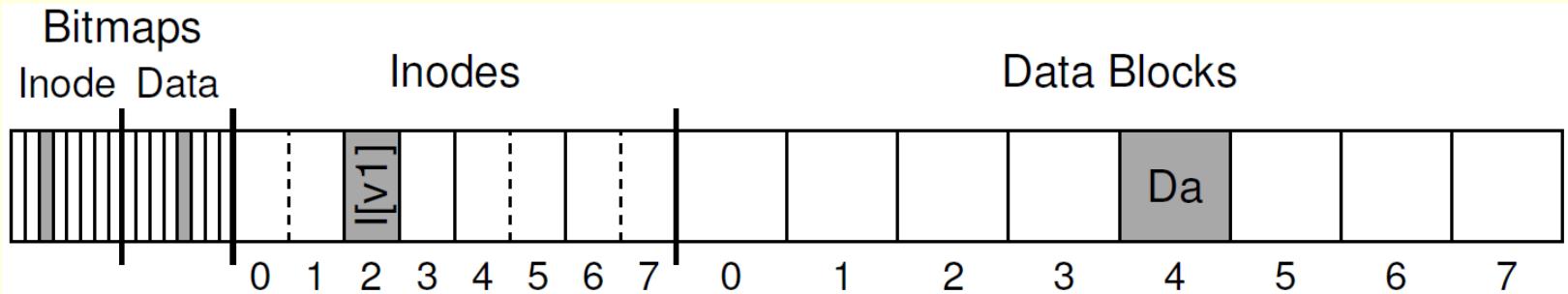


- Tem-se uma zona de **blocos** (não é um SF!) reservados para o journal

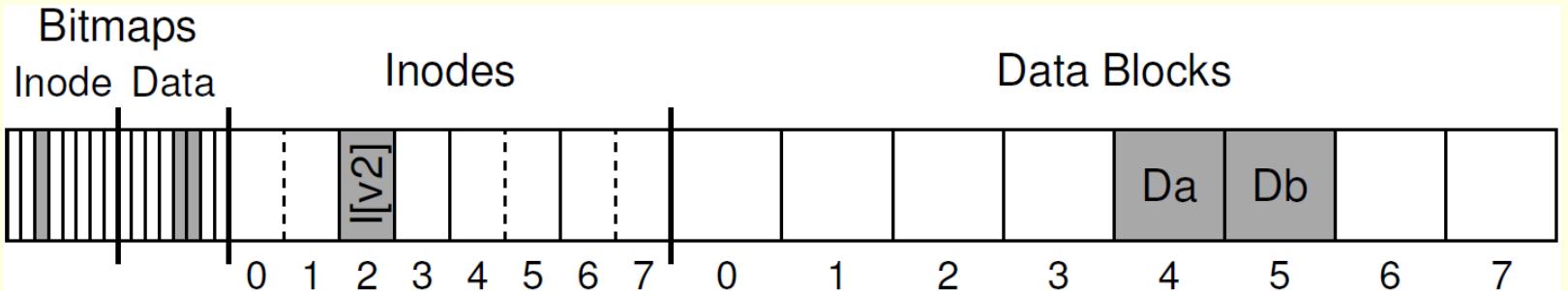


ext3: Cenário 2b (1)

- Consideremos o estado *initial* do SF:

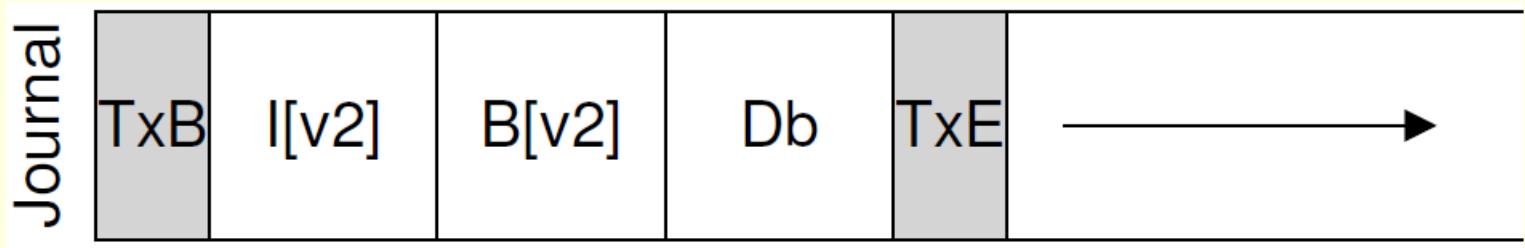


O ficheiro ocupa o inode 2 e gasta um data block. Vamos fazer write(s) que obrigarão a alocar mais um data block, o que daria



ext3: Cenário 2b (2)

- *Contudo, antes de fazer as alterações,*
 - *Escreveríamos no journal o seguinte*



- *Em que*
 - *TxB e TxE são marcas (estruturas de dados que marcam) de início e fim de transacção*
 - *I[v2], B[v2] e Db são, respectivamente, as imagens alteradas do inode, bitmap de blocos e bloco de dados. Esta informação é escrita imediatamente no disco (write síncrono) antes de fazer as alterações “in-place”*

ext3: Cenário 2b (3)

- **Nota intercalar:** o que é uma transacção?
 - É uma acção, composta por várias sub-acções, que conduz um sistema de um estado consistente a um outro, também consistente
 - A transacção é atómica, isto é, ou as sub-acções são todas executadas com sucesso, ou então nenhuma é executada.
- Depois da transacção estar escrita no disco,
 - Cada uma das sub-acções – actualizar o inode, actualizar o bitmap de dados, escrever o novo bloco – pode permanecer em cache algum tempo, na certeza de que no caso de um CFE se pode recuperar a informação a partir do log, mas acaba por ser efectuada (nessa altura diz-se que se fez o **checkpoint**)

ext3: Cenário 2b (4)

□ Consequências de um CFE

- A preocupação é agora com o *log*: e se este estava a ser escrito na altura do CFE?. Podemos considerar 2 situações:
 1. O *log* está *incompleto* – por exemplo, falta o *TxE*.
 2. O *log* está aparentemente *completo*, mas há informação em falta.

□ Soluções

1. Log incompleto: fácil

os dados *in-place* (isto é, nos sítios certos: a alteração ao *inode* escrita na tabela de *inodes*, a alteração ao *bitmap* escrita no *bitmap*, etc.) só são escritos após escrita correcta do *log*; assim, se esta falhou, os dados *in-place* nem sequer começaram a ser escritos...

2. Log aparentemente completo: a ver a seguir...

Semântica das operações em disco (1)

□ Cenário

- O SF pede num único `write()` a escrita de um buffer que contém um conjunto de “estruturas de dados”, E_1, E_2, \dots, E_n , cuja dimensão requer vários blocos logicamente contíguos, $B_1, \dots, B_k, \dots, B_j, \dots, B_m$.

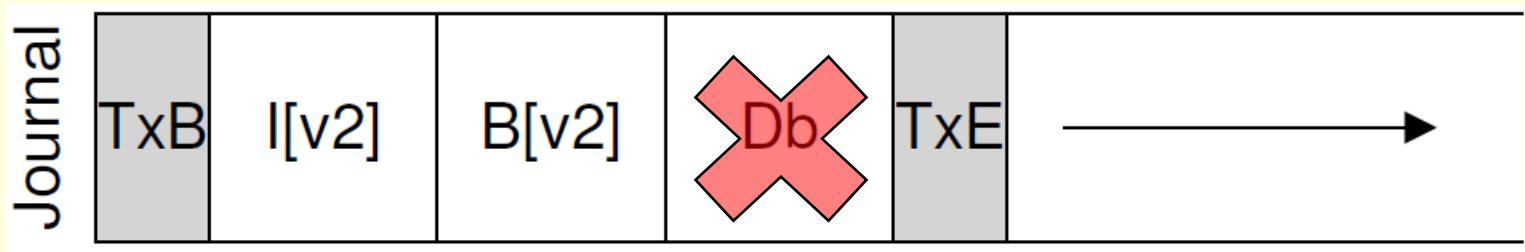
□ Questão

- É possível garantir que os blocos são efectivamente escritos pela ordem dos seus índices, i.e., que o bloco B_k , não é escrito após B_j ?
- E a resposta é **não**, não é possível garantir! De facto, o controlador do disco pode reordenar os pedidos...

ext3: Cenário 2b (5)

□ Assim pode acontecer que, na escrita do journal,

- O bloco *TxE* seja escrito antes do *Db*, e o *CFE* aconteça imediatamente a seguir, deixando “lixo” no lugar onde seria escrito o bloco *Db*



□ Solução

- Dois writes síncronos sucessivos, sendo o primeiro para todos os itens **excepto** o *TxE*, e o segundo **apenas** para o *TxE*, que deverá ocupar um bloco inteiro para que o controlador garanta a escrita atómica e sem o tentar recombinar com mais dados...

Optimizações (1)

- *Um journal pode diminuir o desempenho...*
 - A realização de múltiplas escritas síncronas degrada o desempenho e por isso podem fazer-se optimizações que se destinam a recuperar parte dessa perda de desempenho
- *Soluções*
 - Não fazer logs dos dados, apenas dos metadados
 - e.g., no exemplo que apresentamos, não guardar o Db, apenas inode e bitmap. É um sacrifício que passa a pesar mais sobre o utilizador (que tem de ter backups) mas continua a garantir a consistência do SF.
 - Nota: **os dados de uma directória são metadados!**

(continua)

Optimizações (2)

□ Soluções

(continuação)

- *Combinar múltiplos updates das mesmas estruturas do SF numa única transacção maior*
 - e.g., numa situação em que há uma sequencia de operações **numa mesma directória** envolvendo criação, remoção e alteração de nome de ficheiros, em vez de criar uma transacção por cada operação, aglutinar várias operações **numa única transacção** – em caso de CFE, ou todas as operações são committed, i.e., têm sucesso, ou nenhuma “entra” no SF.

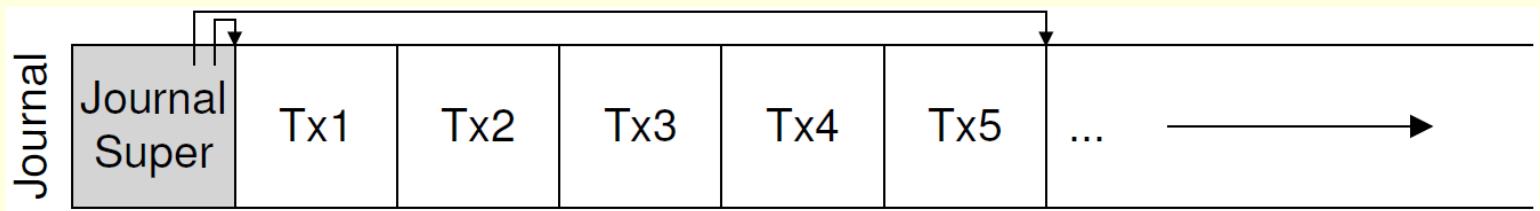
Que espaço para journal? (1)

- *Um journal não pode “crescer eternamente”...*
 - *Como o espaço em disco é finito e, por razões de desempenho, o espaço para logs é reservado em avanço e não é elástico,*
 - *... e um grande log significa que o seu replay vai demorar bastante,*
 - *É necessário manter o espaço usado no log sob controle*
- *Solução*
 - *Usar um log circular, no qual as transacções committed (i.e., os dados in-place já estão em disco) são descartadas e o seu espaço libertado no log.*

Que espaço para journal? (2)

(continuação)

- Usar um log circular, no qual as transacções *committed* (i.e., os dados *in-place* já estão em disco) são descartadas e o seu espaço libertado no log
 - Uma técnica é ter um “journal superblock” que mantém informação sobre o espaço livre e as transacções ainda activas (*uncommitted*), possivelmente usando uma lista ligada.



Algoritmo final

- Caso de *journaling* de dados e metadados
 1. *Escrita no journal*: escrever sincronamente numa única operação o TxB e os dados e metadados.
 2. *Commit do journal*: escrever sincronamente numa única operação o TxE
 3. *Checkpoint*: escrever os dados e metadados nos seus locais (*in-place*) do SF – o que pode acontecer mais tarde... mas só aí é que o *checkpoint* está completo.
 4. Marcar no journal a transacção como terminada e libertar o espaço por esta ocupado

Timeline do algoritmo final

□ Caso de *journaling* de dados e metadados

