IMPLEMENTAÇÃO DE UM SERVIDOR DE ARQUIVOS COM TRANSAÇÕES ATOMICAS

MARIA BEATRIZ FELGAR DE TOLEDO

IMPLEMENTAÇÃO DE UM SERVIDOR DE ARQUIVOS

COM TRANSAÇÕES ATOMICAS

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Maria Beatriz Felgar de Toledo e aprovada pela comissão julgadora.

Campinas, 3 de Abril de 1986.

Prof. Dr. Célio C. Guimarães
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

março/1986

SUMÁRIO

O servidor de arquivos implementado como trabalho de tese suporta transações atômicas sobre vários arquivos, garantindo a atomicidade da transação em face de quedas do processador ou acessos concorrentes ao mesmo arquivo. A interface oferecida para os clientes é do tipo "servidor universal" e permite acesso randômico a páginas de um arquivo.

ÍNDICE

INTRODUÇÃO

O servidor de arquivos implementado como trabalho de tese suporta transações atômicas sobre vários arquivos, garantindo a atomicidade da transação, mesmo em face de quedas do processador ou acessos concorrentes ao mesmo arquivo. A interface oferecida para os clientes permite acesso randômico a páginas de um arquivo, criação e remoção de arquivos, sendo semelhante à interface do servidor universal. Para controle de concorrência, o servidor usa o método de trancas ("locks") do tipo um escritor ou vários leitores.

O servidor de arquivos foi escrito em Turbo-Pascal, para o microcomputador I-7000, utilizando, como dispositivo de armazenamento, uma unidade de disco Winchester.

O capítulo 1 é dedicado ao conceito de servidor de arquivos e servidor universal e à apresentação das interfaces que podem ser oferecidas aos clientes por um servidor de arquivos.

Transações, operações elementares de uma transação e o conceito de atomicidade são descritos no capítulo 2. O capítulo 2 também apresenta uma revisão das noções básicas de controle de concorrência e os métodos que podem ser usados na sua implementação - trancas ("locks") e "timestamps".

O capítulo 3 é dedicado à revisão de algumas técnicas usadas para recuperação de falhas.

As características principais - interface com clientes, estruturas de dados, mecanismos usados para garantir atomicidade- dos servidores que serviram de modelo para este trabalho, (Cambridge, Paxton e XEROX), estão no capítulo 4 e as caracterís- ticas do servidor implementado estão no capítulo 5.

No capítulo 6 são apresentados os testes, e extensões ao sistema.

1. CARACTERÍSTICAS GERAIS DOS SERVIDORES DE ARQUIVOS

1.1. O que é um servidor de arquivos:

Um servidor é um sistema dedicado ao controle de um ou mais recursos compartilhados por uma rede de computadores.

Um servidor de arquivos tem por objetivo armazenar, de forma confiável, grandes quantidades de dados para os outros computadores da rede, chamados clientes. Além de facilitar o compartilhamento de dados, reduz o custo de hardware, já que permite que dispositivos de memória secundária caros sejam compartilhados por processadores relativamente baratos [13].

1.2. Interface oferecida pelo servidor de arquivos:

Os serviços fornecidos pelo servidor de arquivos para seus clientes podem ser invocados através das várias operações que constituem a interface entre o servidor e o cliente [2].

Essa interface pode ser de alto nível, oferecendo para os clientes um sistema de arquivos completo. Neste caso, os objetos das operações são arquivos identificados por nomes alfanuméricos. O servidor teria implementado controle de acesso, estruturas hierárquicas de diretório, trancas para acessos simultâneos a arquivos, isto é, todas as funções características de um sistema de arquivos.

Como interface de mais baixo nível, o servidor aparece para um cliente como um simples gerenciador de espaço em disco com

operações sobre blocos de disco, identificados por seu endereço. As operações fornecidas pelo servidor seriam:

. alocar/desalocar espaço em disco de tamanho N a partir do endereço X

. apresentar o conteúdo do endereço X até o endereço Y

. escrever a partir do endereço X, N unidades de armazenamento

Enquanto no primeiro caso, a interface apresenta a vantagem do pouco software a implementar em cada máquina cliente, no segundo, torna possível a cada cliente implementar com o máximo de flexibilidade, a aplicação mais adequada para o seu caso, por exemplo, um sistema de arquivos, banco de dados ou memória virtual. Existe entre as duas interfaces extremas, uma variedade de opções que podem ser tomadas, levando-se em conta os serviços requeridos em cada aplicação.

1.3 Servidor universal:

Um servidor universal oferece uma interface entre os dois extremos vistos acima, de modo a suportar aplicações diferentes de vários clientes. É mais sofisticado que um gerenciador de espaço em disco, mas flexível o bastante para que cada cliente tenha organização e proteção de dados adequadas para o seu caso.

O servidor universal é capaz de suportar acesso a nível de arquivo, página ou bytes. O modo de acesso é expresso através do par (deslocamento, comprimento). O deslocamento é a distância em

bytes ou palavras do início do arquivo e o comprimento é o número de bytes ou palavras a serem acessadas a partir do deslocamento.

O servidor implementado e descrito no capítulo 5 oferece, como um servidor universal, essa interface intermediária a seus clientes, mas suporta acesso a nível de páginas de arquivos somente.

O servidor universal de Birrell e Needham mantém objetos que podem ser "segmentos" ou "índices" [2]. Um objeto é composto por uma sequência de unidades elementares de informação, tipicamente, bytes ou palavras.

Um segmento pode ser manipulado pelo cliente, enquanto um índice somente pelo servidor. Um índice contém um conjunto de nomes internos de objetos. Fica a cargo de um cliente (por exemplo, um sistema de arquivos) assegurar, através de pedidos ao servidor, que os nomes internos dos objetos relevantes à aplicação sejam armazenados em um índice.

As operações básicas oferecidas pelo servidor são:

. criar um objeto e preservar seu nome interno em um índice

. preservar o nome interno de um objeto em um índice

. remover uma entrada de um índice

. ler n unidades de um objeto a partir da unidade X

. escrever n unidades em um objeto a partir da unidade X

Com essas facilidades, um sistema de arquivos pode ser implementado: um diretório é construído com um índice e um segmento

cujo nome interno é preservado neste índice, em uma posição fixa. O segmento é usado para armazenar os nomes alfanuméricos das entradas do diretório, deslocamentos do nome interno no índice, controle de acesso, data e hora da criação, etc. Birrell e Needham exemplificam como o servidor universal pode ser usado para implementar um sistema de arquivos de dois níveis, o que é descrito a seguir.

Considera-se a existência de um índice mestre onde o nome interno do índice inicial do cliente deve ser preservado.

No índice inicial, a primeira entrada vai ser o nome interno para um objeto do tipo segmento (diretório mestre) e as entradas seguintes, nomes internos para objetos do tipo índice (diretórios do usuário). O sistema de arquivos manterá no diretório mestre nomes alfanuméricos e controle de acesso para os diretórios de usuário.

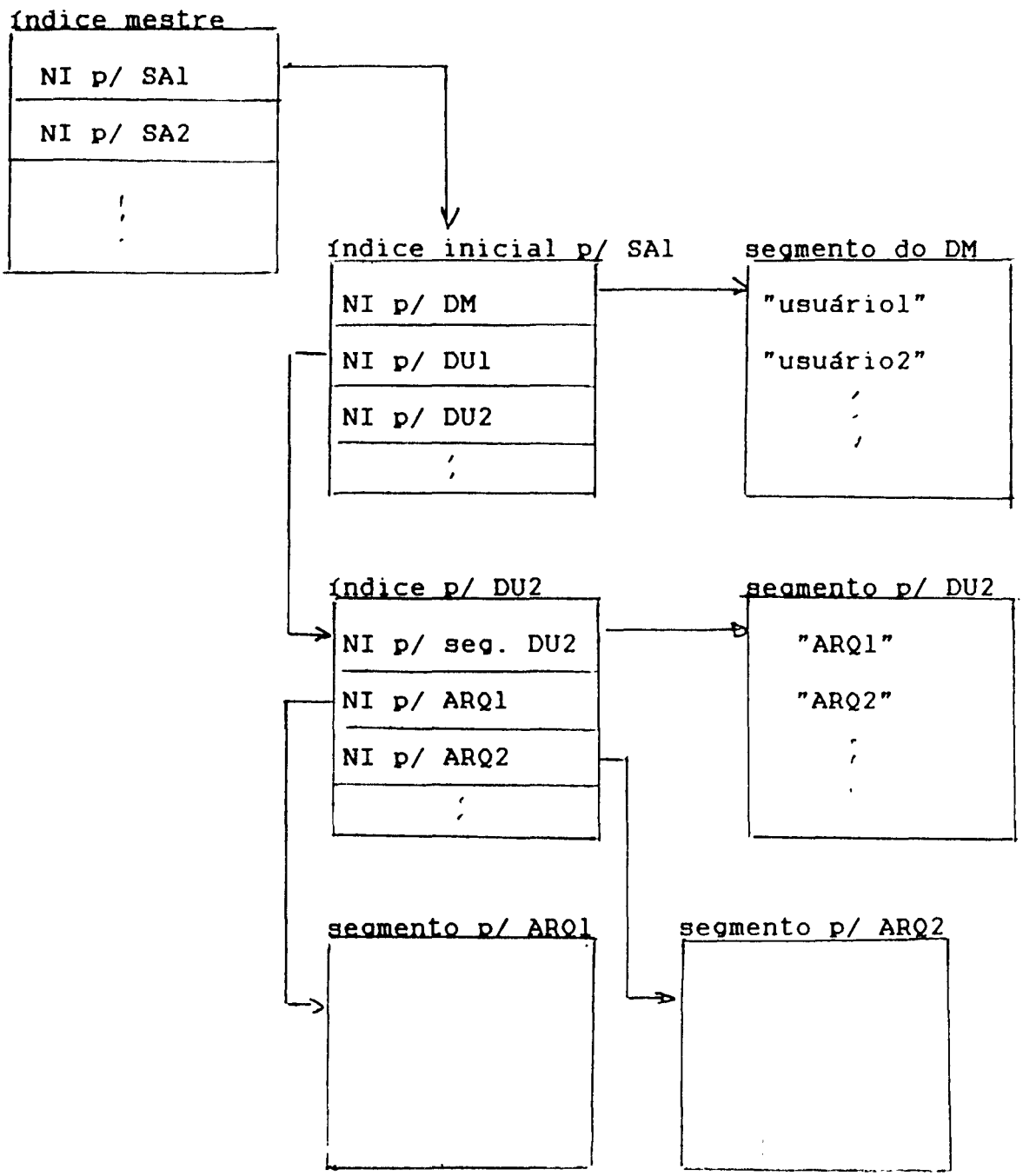
Em cada diretório de usuário, o sistema de arquivos manterá um segmento com nomes alfanuméricos e controle de acesso para os arquivos do usuário.

NI = nome interno

SA = sistema de arquivos

DM = diretório mestre

DUi = diretório do usuário i


índice mestre
NI p/ SA1
NI p/ SA2
índice inicial p/ SA1
NI p/ DM
NI p/ DU1
NI p/ DU2
segmento do DM
"usuário1"
"usuário2"
índice p/ DU2
NI p/ seg. DU2
NI p/ ARQ1
NI p/ ARQ2
segmento p/ DU2
"ARQ1"
"ARQ2"
segmento p/ ARQ1
segmento p/ ARQ2

2.PROCESSAMENTO DE TRANSAÇÕES

Uma transação, no contexto desse trabalho, é uma sequência de operações elementares sobre objetos armazenados pelo servidor. As operações elementares que podem ser invocadas por um cliente são para criar, remover, recuperar ou modificar um objeto, além das operações para indicar início e fim de transação.

Num sistema, os objetos estão relacionados de diversas maneiras. O sistema está em estado consistente, se estas relações são satisfeitas. Uma transação atômica ou termina com sucesso, deixando o sistema em um novo estado consistente, ou não modifica o estado do sistema. Concorrência e falhas requerem tratamento especial para não decompor uma transação atômica. Duas transações que compartilhem dados e executem concorrentemente podem interagir de modo a apresentar dados em estado intermediário e inconsistente que pode ser observado por outra transação. Além disso, uma transação que é interrompida devido a uma falha, pode deixar dados em estado intermediário. Para que a consistência seja mantida, atomicidade deve ser garantida mesmo quando transações sejam executadas concorrentemente e em presença de falhas. No primeiro caso, para que várias transações executando concorrentemente tenham visão consistente dos dados, elas devem parecer executar uma depois da outra, isto é, o efeito da execução concorrente é equivalente ao de alguma execução estritamente serial das transações [4]. Este é o chamado "princípio de serialização" que é a

base dos algoritmos de controle de concorrência a serem discutidos adiante. No segundo caso, todas as operações elementares devem ser completadas ou nenhuma é realizada para que a transação seja atômica com relação a falhas [9].

A seguir são descritas as operações usuais oferecidas por um servidor de arquivos.

2.1. Operações de controle da transação:

As operações de controle são as seguintes:

. inicia transação

retorna - identificador único para transação ou código de falha

Ao enviar esse pedido, o cliente inicia a transação e recebe o identificador da transação que será usado em todas as operações seguintes.

. termina transação

parâmetro - identificador de transação

retorna - confirmação ou código de falha

O pedido para que a transação termine torna as atualizações permanentes.

. aborta transação

parâmetro - identificador de transação

retorna - confirmação ou código de falha

Um pedido para abortar termina a transação e as atualizações realizadas durante a transação não terão efeito.

2.2 Operações elementares sobre objetos do tipo arquivo:

As operações sobre arquivos são as seguintes:

. cria arquivo

parâmetro - identificador da transação

retorna - identificador do arquivo ou código de falha

. lê

parâmetro - identificador da transação, identificador do arquivo, seletor de dado

retorna - dado ou código de falha

. escreve

parâmetro - identificador da transação, identificador do arquivo, seletor de dado, dado

retorna - confirmação ou código de falha

. remove arquivo

parâmetro - identificador da transação, identificador do arquivo

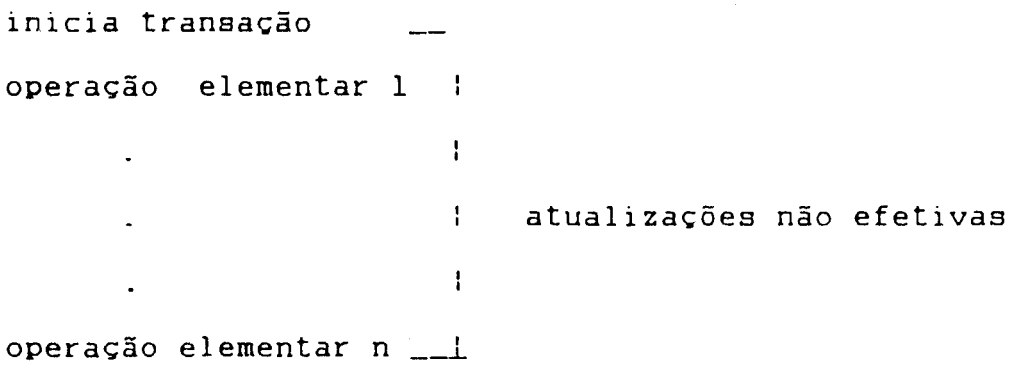
retorna - confirmação ou código de falha

Antes que uma transação termine, ela pode ser abortada

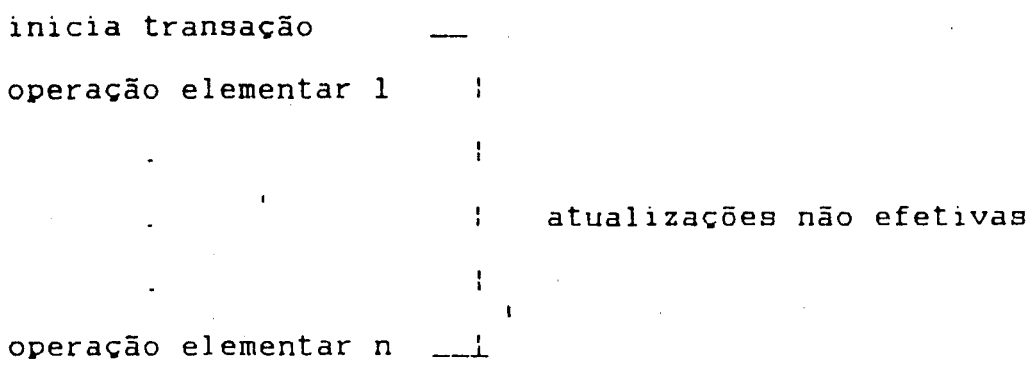
tanto pelo cliente como pelo próprio servidor (devido à limitação de recursos, "deadlock", etc). Se a transação é abortada, as atualizações realizadas pela transação até esse ponto não têm efeito e os dados permanecem inalterados (caso 2 e 3, adiante).

Uma transação que termine com sucesso terá o estado alterado pelas atualizações realizadas durante a transação (caso 1, adiante).

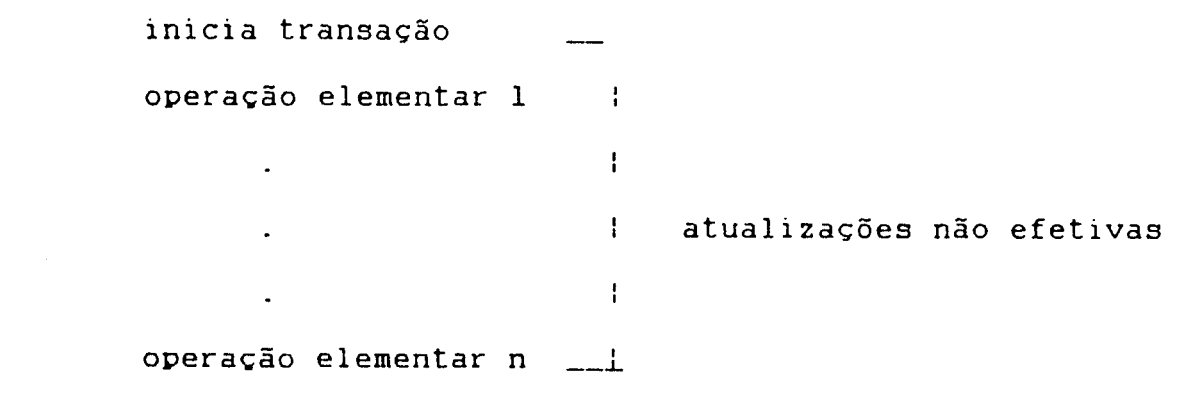
1. transação que termina com sucesso



2. transação abortada pelo cliente



3. transação abortada pelo servidor



Enquanto a transação está em progresso, as atualizações são não efetivas. O servidor deve manter dois estados dos dados para cada transação em progresso:

. o estado em que o sistema se encontrava quando a transação iniciou e que deve ser restaurado, em caso de queda do sistema ou a transação ser abortada.

. o novo estado do sistema com as atualizações requisitadas e que se torna efetivo se a transação terminar com sucesso.

O pedido para terminar uma transação envolve os passo seguintes:

. armazenar, em disco, toda informação necessária para que a transação termine, mesmo se ocorrer queda do servidor

. ponto de decisão ("commit point"): mudar o estado da transação para decidida, isto é, a transação deve ser terminada e o estado do sistema deve ser alterado e permanecer consistente, mesmo se ocorrer queda do servidor

. tornar as atualizações realizadas pela transação permanentes

. mudar o estado da transação para terminada

Antes do ponto de decisão, o estado do sistema é reversível. Após o ponto de decisão, as mudanças se tornam efetivas, mesmo que ocorra queda do servidor. Os procedimentos para este fim serão descritos em 4.1.3., para a extensão de Paxton; 4.2.5., para o servidor de Cambridge; 4.3.6, para o servidor da XEROX e 5.7, para o servidor implementado.

2.3. Transações atômicas em sistemas distribuídos

Se os dados atualizados por uma transação estiverem armazenados em mais de um servidor, um protocolo especial de decisão deve ser usado para garantir atomicidade [6]. Uma transação só pode decidir se todos os servidores envolvidos na transação puderem completar atualizações locais.

O protocolo para decidir transações em dados distribuídos é comumente chamado de "protocolo de decisão em duas fases". A transação passa a ter um nó coordenador que mantém um registro do estado da transação e se comunica com todos os demais servidores participantes.

Na primeira fase, o coordenador envia mensagem para que todos os participantes se preparem para decisão. Os participantes que puderem terminar a transação, armazenam, em disco, a informação necessária para completar as atualizações e enviam uma mensa-

gem para o coordenardor indicando que estão prontos para decidir a transação. Um outro servidor que não puder terminar envia mensagem indicando que abortou a transação.

Na segunda fase, o coordenador deve armazenar atomicamente, em disco, o estado final da transação ao receber resposta de todos os participantes. O estado da transação será decidido ou abortado dependendo das respostas. Se um dos participantes tiver abortado a transação ou não tiver enviado resposta dentro de um período de tempo determinado, o coordenador enviará uma mensagem a cada participante para que todos abortem. No caso em que todos os servidores puderem decidir, o coordenador envia uma mensagem a cada participante para que termine a transação e quando receber confirmação dos participantes, apaga o registro da transação.

Se ocorrer queda do coordenador, ao reiniciar a operação, este verifica o estado da transação e envia mensagem para os outros servidores conforme esse estado. Se um participante cair na fase 1, a transação será abortada, já que o coordenador supõe queda do participante que não responder dentro de um certo prazo. Se um participante cair na fase 2, ao reiniciar sua operação, ele consulta o coordenador para saber se aborta ou termina a transação.

Este é o método usado para decisão de transações no sistema da XEROX, descrito em 4.3.

2.4. Controle de concorrência

Transações têm a propriedade de serem consistentes - elas encontram dados num estado consistente e preservam consistência depois de sua execução. Se várias transações são executadas uma depois da outra, cada transação verá um estado consistente deixado pela antecessora. Mas, se várias transações são executadas concorrentemente, os dados manipulados por uma transação podem estar inconsistentes quando são compartilhados por outra transação e acessados em estado intermediário [4].

O controle de concorrência deve assegurar que as operações de um conjunto de transações executadas de forma entrelaçada produzam o mesmo resultado que seria obtido se as transações fossem executadas serialmente, isto é, as entradas e saídas para cada transação de um conjunto de transações executando concorrentemente devem ser as mesmas que cada transação teria se o conjunto fosse executado serialmente. Cada transação veria objetos acessados em estado consistente e o estado final resultante da execução concorrente do conjunto de transações seria também consistente. Dessa forma, a execução concorrente de várias transações seria equivalente à execução serial dessas transações.

2.4.1. Serialização de transações:

Histórico ("schedule") para um conjunto de transações: o histórico indica a ordem em que as operações dessas transações

foram executadas.

O histórico para um conjunto de transações T1,T2,...,Tn é a sequência

$$H = \left(\bigcup_{i=1}^{m} (Ti, Opi, Oi) \right)$$

onde,

Ti representa quaisquer das transações T1,T2,...,Tn

Opi representa operação no passo i sobre o objeto Oi. As operações consideradas são leitura/escrita.

m é a soma do número de passos de todas as transações.

Histórico serial: é o histórico mais simples que corresponde à execução de todas as operações de uma transação seguidas das operações de outra e assim por diante.

Entradas e saídas de uma transação: com respeito às entradas e saídas de cada transação, o número de históricos permitidos diminui para que consistência seja mantida. Na verdade, só interessam os históricos em que cada transação tem as mesmas entradas e as mesmas saídas que teria se o conjunto de transações fosse executado serialmente. Examinando-se um histórico, podem-se determinar as entradas e saídas de cada transação.

Relação de dependência de um histórico: para cada histórico-

co H, existe uma relação de dependência DEP(H) sobre transações X objetos X transações, de modo que

$$(T,O,T') \in DEP(H)$$

se

$$H=(...(T,escrita,O),...,(T',escrita,O)...)$$

ou

$$H=(...(T,escrita,O),...,(T',leitura,O)...)$$

ou

$$H=(...(T,leitura,O),...,(T',escrita,O)...)$$

e não existe outras operações de escrita sobre o objeto O entre as operações especificadas.

O objeto O é uma saída de T e entrada para T'.

Históricos equivalentes: se dois históricos têm o mesmo conjunto de dependência, eles oferecem para cada transação as mesmas entradas e saídas, são ditos, então, equivalentes. Se um histórico é equivalente a um histórico serial, cada transação verá estado consistente e o estado final será também consistente. Por outro lado, se não for equivalente a algum histórico serial, é possível que ocorra inconsistência. Para que a consistência seja mantida, só os históricos equivalentes ao serial interessam.

Operações em conflito: duas operações de acesso de diferentes transações sobre o mesmo objeto estão em conflito se uma

delas é uma operação de escrita e nenhuma das duas transações foi terminada. Um conflito é causado por um pedido de leitura de um objeto, quando o pedido anterior foi de escrita ou por um pedido de escrita, quando o pedido anterior foi de escrita ou leitura. Na ausência de controle de concorrência, as transações em conflito poderiam interferir uma com a outra.

Quando um pedido de uma transação A está em conflito com outro pedido anterior de outra transação sobre o mesmo objeto, o controle de concorrência pode:

- fazer a transação A esperar até que a outra transação (ou as outras) com a qual está em conflito termine
- abortar a transação A ou a transação (ou as transações) com a qual A está em conflito e possivelmente reiniciar a transação abortada.

O controle de concorrência seleciona uma resposta para as operações de escrita e leitura, conforme estejam em conflito ou não com a operação anterior sobre o mesmo objeto.

Para regular o acesso a recursos compartilhados, o controle de concorrência pode se utilizar de duas técnicas - trancas ("locks") [4], [6] ou timestamps [1].

2.4.2. Método de trancas:

Uma tranca sobre um objeto indica que ele está sendo usado

por uma transação, evitando que outras transações vejam o objeto em estado intermediário inconsistente. Antes de realizar qualquer operação sobre um objeto, uma transação deve adquirir uma tranca sobre ele, tornando-o inacessível a outras transações. A tranca só será liberada quando a transação terminar de usar o objeto.

Uma tranca sobre um objeto pode ser de dois tipos:

. tranca exclusiva: somente uma transação pode acessar o objeto. Enquanto a transação tiver a tranca, pode realizar operações de leitura ou escrita sobre esse objeto.

. tranca compartilhada: várias transações podem acessar o objeto. Essas transações só podem realizar operações de leitura sobre o objeto.

O controle de concorrência exige operações para obtenção e liberação de tranca dos objetos acessados durante a transação. A responsabilidade para controlar trancas pode ser relegada ao cliente ou servidor. Se for da responsabilidade do cliente, a programação se torna menos confiável. Quando o sistema controla as trancas, elas passam a ser obtidas implicitamente nas operações de acesso ao objeto e liberadas no final da transação. O servidor implementado utiliza o método de trancas para controle de concorrência, sendo que a operação de tranca está implícita na operação de abertura do arquivo e a operação de liberação de tranca ocorre no término da transação (seção 5.4.).

A um histórico descrevendo a execução de um conjunto de transações acrescentam-se as operações para tranca e liberação de

tranca. Com respeito à tranca, um histórico será legal quando não há uma operação de tranca exclusiva sobre um objeto já trancado (tranca compartilhada ou exclusiva). Operações de tranca compartilhada sobre um objeto já trancado em modo compartilhado são permitidas.

Transações bem formadas: uma transação é bem formada, se tranca um objeto antes de acessá-lo.

Transações "duas fases": uma transação é dita "duas fases", se não mais realiza operações de tranca depois que já liberou alguma tranca sobre qualquer objeto. A primeira fase é a fase de crescimento da transação, em que trancas são adquiridas. A segunda fase é a de encolhimento, a partir da primeira operação em que uma tranca é liberada.

Dado um conjunto de transações bem formadas e duas fases, então qualquer histórico legal é equivalente ao serial [4].

Transações recuperáveis: para que a consistência seja mantida, basta que uma transação seja bem formada e duas fases. Mas, para fazer transações separadamente recuperáveis, isto é, para que uma transação possa ser desfeita sem que outras transações que acessaram objeto decidido tenham também que ser desfeitas, a maneira mais simples, é só liberar trancas no final da transação.

Neste caso, a segunda fase (encolhimento) da transação ficaria reduzida a uma única operação, quando todas as atualizações são decididas juntas e todas as trancas liberadas [6].

Tratamento de "deadlock":

"Deadlock" pode ocorrer quando duas ou mais transações se bloqueiam mutuamente, esperando tranca sobre objetos. Para resolver o problema de "deadlock", o controle de concorrência pode optar por duas técnicas - prevenção de "deadlock" ou deteção de "deadlock" [6], [1].

I. Prevenção de "deadlock"

. nenhuma transação fica esperando para obter tranca: quando uma tranca sobre um objeto é pedida mas está em conflito com uma tranca já existente, o sistema escolhe uma das transações para ser abortada. A escolha pode ser baseada em prioridades entre as transações.

. ordenação de recursos: as trancas são declaradas no início da transação. Os objetos são numerados e as trancas sobre eles são pedidas em ordem numéricas. A prioridade de uma transação é determinada pelo maior número entre todos os objetos acessados, dessa maneira, uma transação só espera por outra transação com prioridade maior.

. temporização: à cada transação está associado um período

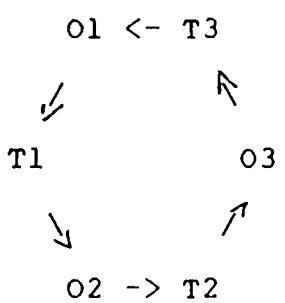
de tempo que ao se esgotar faz com que a tranca seja liberada e a transação que possuia a tranca abortada, se outra transação quiser acessar o mesmo objeto.

II. Deteção de "deadlock"

Transações podem ficar esperando uma por outra e serão abortadas somente se deadlock ocorrer.

"Deadlocks" podem ser detetados construindo-se um grafo de espera para o conjunto de transações em execução e procurando-se ciclos nesse grafo.

Os vértices do grafo são transações e trancas. Uma aresta de um vértice V1 que é uma tranca para um vértice V2 que é uma transação indica que a tranca V1 foi concedida à transação V2. Uma aresta de um vértice V1 que é uma transação até um vértice V2 que é uma tranca indica que a transação V1 está esperando pela tranca V2.



Ciclos indicam a presença de "deadlock". Quando um ciclo

no grafo é encontrado, uma ou mais transações no ciclo devem ser abortadas para a tranca ser concedida a uma outra transação também no ciclo e que está esperando. A escolha da transação a ser abortada é geralmente baseada na quantidade de recursos usados pelas transações no ciclo.

2.4.3. Método de "timestamps":

"Timestamp" é um número único que está associado a uma transação ou objeto, escolhido de uma sequência crescente de números que pode ser função da hora do dia. Em sistemas distribuídos, cada nó da rede tem um identificador único e o "timestamp" é gerado concatenando-se o tempo local (mais significativo) com o identificador do nó (menos significativo). Isto garante que o "timestamp" seja unívoco num sistema distribuído.

O controle de concorrência fornece para cada transação que é iniciada um "timestamp". A cada operação de leitura ou escrita de uma transação está associado o "timestamp" da transação. a serialização das transações executando concorrentemente é obtida utilizando-se o "timestamp" para ordenar a execução das operações em conflito.

Para cada objeto X armazenado, os "timestamps" da última operação de leitura, R_ts(x), e da última de escrita, W_ts(x), são também armazenados.

Para resolver o conflito leitura-escrita, quando:

I- operação de leitura com "timestamp" TS é solicitada,

se TS < W_ts(x)

então o controle de concorrência rejeita a operação e aborta a transação que a solicitou

senão a operação é executada e R_ts(x):=max(R_ts(x),TS)

II- operação de escrita com "timestamp" TS é solicitada,

se TS < R_ts(x)

então o controle de concorrência rejeita a operação e aborta a transação que a solicitou

senão a operação é executada e W_ts(x):=max(W_ts(x),TS)

Para resolver o conflito escrita-escrita, quando operação de escrita com "timestamp" TS é solicitada,

se TS < W_ts(x)

então o controle de concorrência rejeita a operação e aborta a transação que a solicitou

senão a operação é executada e W_ts(x):=TS

Uma prova de que o controle de concorrência baseado em "timestamps" garante serialização pode ser encontrada em [1].

3. RECUPERAÇÃO DE FALHAS

Um sistema se torna robusto com relação a falhas através do uso de redundância. Na recuperação de uma falha, a informação redundante é usada para reconstruir o estado do sistema.

Conforme a falha, a informação destruída poderá ser:

1. o conteúdo da memória volátil do processador (depois de uma queda do processador): o estado do sistema é reconstruído das informações em disco. Denominamos este tipo de falha de "queda do sistema" ou "queda do processador".
2. um bloco de disco que foi corrompido durante um acesso
3. toda a superfície de um disco, devido a um colapso do funcionamento da cabeça ("head crash") ou outra falha mecânica do disco: o estado do sistema é reconstruído de uma versão do sistema arquivada externamente mais um "log" das atividades realizadas desde que a versão arquivada foi armazenada.

Nenhum dos servidores estudados têm o grau de redundância necessária para recuperação de uma falha do tipo 3. Nesses servidores qualquer falha diferente de queda de processador ou um bloco de disco corrompido será catastrófica. O servidor implementado se recupera de falhas do tipo "queda de processador".

Técnicas usadas para recuperação do estado do sistema depois de uma falha dos tipos 1 e 2:

1. memória estável ("stable storage") [9]

As operações sobre memória estável são atômicas. Cada página em memória estável é representada por duas páginas normais de disco.

Escrever uma página estável exige que as duas páginas que a representam sejam escritas, a segunda depois que a escrita da primeira terminou com sucesso. O término com sucesso de uma escrita é verificado lendo-se a página imediatamente após a escrita e comparando-se o valor lido com o originalmente escrito. As duas páginas são alocadas distantes uma da outra, preferivelmente em dispositivos distintos, para evitar falhas que possam destruir ambas as páginas.

Se uma falha ocorrer durante a escrita de uma página normal de disco, a página pode ficar em três estados:

. inalterada, quando a falha ocorrer antes da escrita

. atualizada com sucesso, quando a falha ocorrer depois da escrita

. inconsistente (a inconsistência pode ser detetada através do CRC), quando a falha ocorrer durante a escrita da página

Se uma falha ocorrer durante a escrita de uma página estável, as duas páginas que a representam podem ficar:

. iguais e consistentes

. diferentes e consistentes

. diferentes e uma página inconsistente

Na recuperação de escrita, as duas páginas são lidas e

comparadas; se diferentes, a primeira página (ou a página consistente) é copiada sobre a outra.

Ler uma página estável implica na leitura de uma das páginas normais. Se a página estiver em estado inconsistente, a outra é lida.

2. páginas "shadow" (para arquivos construídos como árvores de blocos de tamanho fixo) [7]

Páginas modificadas não são escritas sobre páginas armazenadas anteriormente, mas escritas em um bloco de disco livre. Enquanto uma transação está executando, existe duas versões para cada arquivo atualizado: a versão antiga e a versão "shadow". Depois de uma queda do sistema, o arquivo é restaurado para a versão antiga, se a transação interrompida não foi decidida, ou colocado no novo estado (versão "shadow"), se a transação foi decidida.

3. listas de intenções [7]

Uma lista de intenções armazena as mudanças de um arquivo ou conjunto de arquivos atualizados durante uma transação e que devem se tornar efetivas, se a transação decidir. O ponto de decisão da transação consiste no armazenamento seguro com relação a falhas, da lista de intenções associada à transação. Depois do ponto de decisão, o servidor passa a realizar as intenções, tornando as mudanças permanentes. Já que a execução das intenções

pode ser interrompida por uma nova queda do sistema, elas tem que ser implementadas de maneira que sua reexecução produza o mesmo efeito no sistema.

I)

4.SERVIDORES DE ARQUIVOS REPRESENTATIVOS

Neste capítulo serão descritos alguns servidores de arquivos representativos e que são relevantes seja pela relação com este trabalho, seja por sua importância na área. Será dada ênfase aos mecanismos de finalização e de recuperação de transações interrompidas por quedas do sistema.

4.1. Extensão do servidor de arquivos Woodstock (Paxton)

A extensão do servidor Woodstock [11], [13], suporta transações atômicas em vários arquivos e vários servidores. O cliente é o responsável pelo gerenciamento da transação e pela manutenção da integridade dos dados. O servidor Woodstock, desenvolvido em XEROX Palo Alto Research Center, [13], [14], fornece acesso aleatório a setores de um arquivo, permite acesso único de um cliente a um arquivo, usando o método de trancas ("locks") e tem a capacidade de liberar tranca sobre um arquivo, se ele não for acessado por um certo período de tempo.

4.1.1. Tipos de arquivos envolvidos em uma transação

. arquivo de dados:

Contém informação relevante ao cliente e um setor de cabeçalho. O cabeçalho contém o mapeamento de setor lógico, usado pa-

ra endereçar dados, para setor real, como entendido pelo servidor. O cabeçalho está localizado no primeiro setor real do arquivo. Além do mapa de setores, o cabeçalho contém a lista de setores reais livres e um registro indicando a transação em que o arquivo estiver envolvido, se houver.

O mapeamento, descrito no cabeçalho, se torna necessário para que as escritas, realizadas por uma transação, não sejam feitas sobre as páginas armazenadas anteriormente. As páginas atualizadas permanecem "shadows", até que a transação seja decidida. Embora ineficiente, o duplo mapeamento para determinar o endereço físico de uma página do arquivo é necessário para manter a integridade dos dados, em presença de quedas do processador.

Arquivos de dados são convertidos em uma única escrita (setor do cabeçalho) para a nova versão. Para arquivos grandes, o cabeçalho pode ser estendido para mais de um setor. O primeiro setor real contém a parte inicial do mapa, a lista de setores livres e o apontador para uma árvore de setores que são extensões ao mapa.

. arquivo de intenções:

Contém registros sobre arquivos de dados modificados pela transação. É usado para a recuperação dos arquivos de dados depois de uma queda do servidor.

Cada cliente tem um arquivo de intenções alocado permanentemente com informações relativas à transação em progresso num

dado instante. Contém o estado da transação (TI- iniciada, TD- decidida, TC- completada), número da transação, lista de mudanças e sequência de cabeçalhos dos arquivos de dados atualizados. A lista de mudanças consiste dos identificadores dos arquivos de dados e dos apontadores para os cabeçalhos que se encontram mais adiante no arquivo de intenções.

Enquanto uma transação estiver no estado TI, ela pode ser abortada e os arquivos de dados, acessados pela transação, não serão afetados. Mas, uma vez que o estado passa para TD, todas as atualizações que ocorreram durante a transação serão feitas e a transação será terminada eventualmente, mesmo em caso de falha.

4.1.2. Operações fornecidas pelo sistema

O sistema de gerenciamento de transações implementado por Paxton fornece as seguintes operações:

. begin transaction

retorna - identificador da transação

Muda estado da transação para estado TI, na memória local.

. open

parâmetro - identificador do arquivo (que indica também em qual servidor se encontra o arquivo)

retorna - chave para acessar arquivo

A operação open tranca o arquivo e lê seu cabeçalho, armazenando o cabeçalho em memória local. Se o cabeçalho indicar que o arquivo está envolvido em uma transação, deve ter ocorrido uma queda que impediu o término de uma transação. Neste caso, a transação interrompida deve ser terminada e o cabeçalho relido em seguida.

. read

parâmetro - chave para arquivo, número do setor lógico, buffer para receber dados

retorna - dado ou código de erro, indicando que transação foi abortada, se a tranca do arquivo tiver sido quebrada

A operação read converte o setor lógico em setor real, usando uma cópia local do cabeçalho, e o dado é transferido do servidor para o buffer.

. write

parâmetro - chave para arquivo, número do setor lógico, buffer para conter dados

retorna - código de erro, se a tranca do arquivo foi quebrada

A operação write aloca um setor real da lista de setores livres do cabeçalho - nenhuma escrita é feita sobre dados já existentes. Verifica se o setor lógico que está sendo escrito já existe no mapa de setores para que, em caso afirmativo, o setor

real correspondente seja incluído na lista de setores livres quando a transação for completada. Atualiza a cópia local do cabeçalho e transfere o buffer para o servidor.

. abort transaction

parâmetro - identificador da transação

As trancas de todos os arquivos abertos na transação são liberadas e o estado da transação muda para TC.

. end transaction

parâmetro - identificador da transação

Se nenhum arquivo foi modificado, a operação end libera as trancas de todos os arquivos abertos.

Se apenas um arquivo foi modificado, os arquivos que foram lidos somente têm as trancas liberadas, o cabeçalho do arquivo modificado é escrito e sua tranca liberada.

Se vários arquivos foram modificados:

1. libera tranca dos outros arquivos que foram lidos somente
2. obtém tranca do arquivo de intenções
3. marca cada arquivo de dados que foi modificado como envolvido na transação, colocando no cabeçalho o número da transação e o identificador do arquivo de intenções. Esta operação é feita em disco.
4. escreve os cabeçalhos da memória para o arquivo de intenções, depois de atualizar as listas de setores livres de cada arquivo,

inserindo, na lista, os setores reais que foram liberados pela transação

5. escreve lista de mudanças (identificadores de arquivos de dados e apontadores para cabeçalho no arquivo de intenções), número da transação e estado (TD) no arquivo de intenções. Este é o ponto de decisão da transação.

6. escreve cabeçalho dos arquivos, indicando que não estão envolvidos em transação e libera tranca do arquivo de dados, depois que seu cabeçalho foi escrito

7. atualiza arquivo de intenções com estado TC e libera tranca sobre o arquivo

4.1.3. Recuperação de uma transação interrompida

A recuperação é feita sob demanda quando um arquivo envolvido numa transação é acessado pela primeira vez depois de uma queda do cliente. A recuperação pode ser inclusive feita por um cliente que não invocou a transação interrompida. Uma transação interrompida é indicada pelo cabeçalho do arquivo de dados e/ou pelo estado da transação igual a decidido no arquivo de intenções. Em qualquer dos dois casos, todos os arquivos de dados atualizados pela transação são testados e colocados em estado apropriado.

Quando uma operação open encontra um arquivo de dados envolvido em uma transação, verifica se o arquivo de intenções

apontado pelo cabeçalho do arquivo de dados, em questão, pertence ao cliente que invocou a transação atual.

Se pertence ao mesmo cliente, a queda deve ter ocorrido depois que o arquivo de dados foi marcado como envolvido na transação e antes que a transação tivesse sido decidida. Pois, se o estado fosse decidido, a transação teria sido completada, quando o cliente iniciou a nova transação e readquiriu o arquivo de intenções.

Se pertence a um outro cliente, a tranca sobre o arquivo de intenções deve ser adquirida. Se o arquivo de intenções já estiver trancado, o sistema libera a tranca sobre o arquivo de dados, pois o cliente que tem a tranca sobre o arquivo de intenções pode estar tentando obter tranca sobre o arquivo de dados para terminar a transação. Ao conseguir tranca sobre o arquivo de intenções, o sistema relê o cabeçalho do arquivo de dados para verificar se continua indicando envolvido em transação. Se estiver, verifica o estado (deve ser decidido), se o arquivo é referenciado na lista de mudanças e se o número da transação é o mesmo no arquivo de intenções e no arquivo de dados. Caso todas as condições sejam satisfeitas, a transação deve ser completada.

Para cada arquivo da lista de mudanças, obtém tranca, esperando se necessário; lê o cabeçalho, verificando se envolvido em uma transação, o identificador do arquivo de intenções e o número da transação; se a informação estiver consistente, atualiza o cabeçalho do arquivo de dados com o novo cabeçalho no arquivo

de intenções e libera a tranca do arquivo de dados.

4.2. Servidor de arquivos Cambridge

O servidor Cambridge [3], [10], [13] foi desenvolvido na Universidade de Cambridge para fornecer a seus clientes que são sistemas operacionais a interface do servidor universal. É usado por dois sistemas operacionais diferentes que implementam seu próprio sistema de arquivos. Também suporta memória virtual de um dos sistemas operacionais. Fornece transação atômica sobre um único objeto.

4.2.1. Tipos de objetos armazenados pelo servidor

Os objetos são de dois tipos: arquivos e índices, identificados por um número de 64 bits (32 bits aleatórios e 32 bits para o endereço do objeto em disco). Um índice é uma lista de identificadores únicos de objetos. Um índice pode conter qualquer identificador, incluindo o seu próprio.

Operações de controle sobre uma transação:

. open

parâmetros - identificador do objeto

A operação open inicia uma transação sobre um objeto e es-

tabelece tranca do tipo vários leitores ou um escritor. Retorna um identificador único para a transação que deve ser usado como parâmetro nas operações seguintes sobre o objeto.

. close

parâmetros - identificador da transação, variável booleana que indica se a transação deve terminar normalmente ou ser abortada

A operação close termina a transação, tornando as atualizações realizadas durante a transação permanentes ou restaurando o objeto para o estado em que se encontrava quando a transação iniciou, conforme o valor da variável booleana.

Operações sobre arquivos:

. read

parâmetros - identificador da transação, número de palavras a serem lidas, deslocamento no arquivo

Transmite para o cliente as palavras especificadas do arquivo.

. write

parâmetros - identificador da transação, número de palavras a serem escritas, deslocamento no arquivo

Atualiza o arquivo a partir da palavra especificada.

. create-file

parâmetros - identificador do índice, número da entrada no índice

Cria um objeto do tipo arquivo, preserva seu identificador na entrada do índice especificado e retorna identificador do arquivo criado

Operações sobre índices:

. preserve

parâmetros - identificador da transação, entrada no índice, identificador do objeto

Armazena identificador do objeto, se válido, na entrada do índice especificado.

. retrieve

parâmetros - identificador da transação, identificador do índice, entrada no índice

Retorna identificador armazenado na entrada do índice especificado.

. delete

parâmetros - identificador da transação, entrada no índice

Remove identificador armazenado na entrada do índice es-

pecificado.

. **create-index**

parâmetros - identificador do índice, entrada no índice

Cria objeto do tipo índice, preserva seu identificador na entrada do índice especificado e retorna identificador do índice criado

4.2.2. Sistema de armazenamento

O sistema de armazenamento aparece como um grafo orientado para os clientes com arquivos como nós terminais e índices contendo as ligações para outros nós. A raiz é um índice cujo identificador único não é distribuído para nenhum cliente.

Quando um novo cliente desejar armazenar informação, um índice é criado, seu identificador único é colocado na raiz e enviado para o cliente.

Um cliente que desejar compartilhar um objeto, deve reter, em um índice do grafo, o identificador único do objeto e remover essa entrada quando terminar de usar o objeto.

Não há operações explícitas para remover um objeto. Toda vez que o identificador do objeto é preservado em um índice, um contador associado ao objeto é incrementado e quando seu identificador é removido de um índice, o contador é decrementado. Se o contador atinge zero, o objeto fica inacessível pois não há cami-

nho no grafo da raiz até o objeto e o espaço ocupado por ele pode ficar disponível. Devido à possibilidade de existência de ciclos no grafo, um coletor de espaço deve correr periodicamente para determinar que objetos estão inacessíveis.

4.2.3. Estruturas de dados usadas pelo servidor

. objetos:

Objetos são representados por árvores. As folhas da árvore são blocos de dados e os nós internos são blocos de mapa, contendo um conjunto de endereços de disco para outros nós. Inicialmente, um objeto é representado por um único bloco de dado apontado pelo próprio identificador do objeto e a medida que escritas são feitas, a altura da árvore aumenta.

. mapa de cilindro:

Existe um mapa de cilindro para cada cilindro de cada disco. Um mapa de cilindro é representado por uma tabela indexada por número de setor. Cada entrada dessa tabela contém o estado de alocação do setor (ocupado, livre, com intenção de alocar, com intenção de desalocar), o identificador do objeto ao qual pertence o setor (se não estiver livre) e a posição na árvore que representa o objeto. Para a raiz do objeto, a entrada contém também o bit de decisão.

Os mapas de cilindro e os mapas de objeto são redundan-

tes, o que permite a reconstrução da primeira estrutura a partir da segunda, ou vice-versa, conforme explicação em 4.2.5..

4.2.4. Mecanismos internos e decisão de uma transação

Os arquivos dos clientes podem ser de dois tipos: normais ou especiais. Um arquivo especial pode ser revertido para o estado anterior ao início da transação, se a transação sobre ele for abortada. As atualizações sobre um arquivo normal são irreversíveis porque feitas sobre os nós já existentes da árvore.

As atualizações sobre arquivos especiais têm que ser feitas de maneira reversível, por isso exigem mecanismo especial para garantir a propriedade atômica da transação.

Qualquer nó (bloco de dados) da árvore que deva ser modificado requer que um novo bloco seja selecionado, onde os dados atualizados serão escritos. Esse bloco é marcado no mapa de cilindro "com intenção de alocar" e o bloco (mais antigo) correspondente ao nó da árvore é marcado "com intenção de desalocar". Blocos de dados já podem ser escritos em disco, enquanto blocos de mapa são atualizados somente na memória.

Durante a transação, as atualizações dos mapas de cilindro são feitas na memória. No final da transação, os mapas de cilindro modificados são escritos em disco.

O bit de decisão na entrada correspondente à raiz é atualizado e o mapa de cilindro é escrito em disco.

Em seguida, os blocos correspondentes aos nós internos da árvore que foram modificados são escritos em disco.

Então, as entradas no estado "com intenção de alocar" são modificadas para "ocupado" e "com intenção de desalocar" para "livre" e os mapas de cilindro são escritos em disco. Com ambas as estruturas (mapas de cilindro e árvore) refletindo o novo estado, o bit de decisão pode ser desligado e o mapa de cilindro que contém o bit de decisão pode ser escrito em disco.

4.2.5. Recuperação de uma queda do servidor

A recuperação deve garantir que os mapas de cilindro e os mapas de objeto reflitam o mesmo estado para todos os blocos. Para isso, ao reiniciar operação, o servidor deve percorrer os mapas de cilindro e verificar se todas as intenções foram realizadas. Em caso negativo, uma transação não foi completada e o arquivo envolvido nesta transação deve ser colocado no estado consistente com o mapa de cilindro e as intenções devem ser realizadas ou não conforme o bit de decisão. A maneira de atualizar um mapa de objeto é a mesma usada no caso de um mapa de objeto corrompido tiver que ser reconstituído.

Se uma transação foi interrompida durante uma escrita, um bloco do mapa de objeto ou mapa de cilindro pode ter ficado corrompido.

Se um mapa de objeto estiver corrompido, os mapas de ci-

lindro são percorridos para achar todos os blocos pertencentes ao objeto. A posição na árvore indica quais os setores filhos do nó corrompido e seus endereços podem ser usados para reconstituir o nó.

Se um mapa de cilindro estiver corrompido, os mapas de objeto podem ser usados para reconstruí-lo, percorrendo-se o grafo desde o índice raiz e examinando o contéudo de cada índice para achar todos os mapas de objeto. Os mapas de objeto também devem ser examinados para que se determine quais blocos pertencem a esse cilindro.

4.2.6. Protocolo de comunicação

O protocolo de comunicação foi projetado tendo em vista a confiabilidade e o mecanismo de controle de fluxo do anel de Cambridge.

Dados solicitados pelas operações de escrita ou leitura são enviados através de sequências de mini-pacotes do anel. No final da sequência, o receptor verifica se houve erro ou não testando checksum. Em caso de erro, a transferência é reiniciada desde o primeiro mini-pacote.

Na ausência de resposta do servidor a um pedido, o cliente pode enviar o pedido novamente, pois as operações são repetíveis.

O servidor controla fluxo, usando bits do mini-pacote. O transmissor ao detetar pacote rejeitado, reinicia a transmissão

depois de um intervalo de tempo.

4.3. Sistema de arquivos distribuído da XEROX

O sistema de arquivos distribuído da XEROX [10], [12], [13], desenvolvido em XEROX Palo Alto Research Center foi projetado para servir de suporte a aplicações de banco de dados. Fornece transações atômicas envolvendo vários arquivos em vários servidores, permitindo que mais de um cliente compartilhe uma transação.

4.3.1. Operações oferecidas aos clientes

As operações oferecidas aos clientes são:

. open transaction:

Estabelece conecção de um cliente ao sistema. Além de fazer autenticação do cliente, inicia uma transação. Se o acesso do cliente ao sistema for permitido, o servidor retorna o identificador único da transação que deve ser incluído em todas as outras mensagens da transação. O servidor que receber essa mensagem será o coordenador da transação.

. read

. write

. create file

. destroy file

. close transaction:

Torna as atualizações realizadas durante a transação efetivas, disconecta do cliente.

. add server:

Quando um cliente desejar acessar arquivos em mais de um servidor, deve enviar a mensagem de add server para o novo servidor.

O servidor estabelece, então, conecção com o cliente e se comunica com o coordenador (o identificador da transação contém identificação do coordenador).

A mesma mensagem é usada para incluir um novo cliente na transação.

4.3.2. Controle de concorrência

O controle de concorrência se utiliza de trancas para serializar transações executando concorrentemente.

As trancas são estabelecidas inplicitamente pelas operações de leitura ou escrita de dados, a nível de bytes, permitindo acesso de vários leitores e um escritor ao mesmo tempo. Existe três tipos de trancas: de leitura, de intenção de escrita, de decisão de escrita. Zero ou mais transações podem ler um byte de um arquivo juntamente com uma transação com intenção de escrita sobre o mesmo byte. Quando a transação com tranca de intenção de escrita decidir, as trancas de leitura são quebradas. Para evitar

que uma transação seja abortada, se tiver alguma tranca de leitura quebrada, o cliente pode escolher "limpar" a tranca quebrada.

Para prevenção de "deadlock", o sistema da XEROX associa uma temporização às trancas concedidas. Quando o período de tempo tiver esgotado, a tranca fica vulnerável e pode ser quebrada, se outra transação quiser acessar o mesmo dado e as trancas estiverem em conflito.

Esse mecanismo com três tipos de trancas foi elaborado para permitir que clientes mantenham "caches" locais (em memória ou disco), ao invés de enviar pedidos cada vez que quiserem acessar dados. Inicialmente, o cliente que deseja manter um "cache" local deve ler a informação necessária e periodicamente testar se alguma tranca foi quebrada,relendo os dados que tiverem a tranca quebrada. Se tranca fosse de escrita ou leitura, poderia ocorrer "deadlock", quando duas transações compartilhassem o mesmo dado:

1. a transação A tem tranca de leitura e mantém "cache" local
2. a transação B solicita tranca de escrita para atualização do dado e quebra tranca de A
3. A percebe que a tranca foi quebrada e tenta ler novo valor do dado
4. Se B não tiver terminado, pode ter a tranca quebrada e o dado retorna ao estado original. B pode iniciar outra vez e a mesma sequência de eventos poderia se repetir.

Também para permitir "caches" de dados, o servidor deve avisar quando alguma tranca de leitura foi quebrada. Um cliente

para decidir uma transação deve "limpar" essas trancas, indicando então que as escritas que ocorreram durante a transação não dependem do dado cuja tranca foi quebrada.

4.3.3. Representação de arquivos

Em cada unidade de disco, existe uma árvore B para armazenar as estruturas de todos os arquivos dessa unidade. A árvore B faz o mapeamento do par (identificador do arquivo, número da página) para endereço físico no disco.

4.3.4. Mecanismos usados para garantir atomicidade

1. memória estável ("stable storage")

Somente as informações essenciais como listas de intenções, páginas de mapas da árvore-B, mapa de alocação de disco, são armazenadas em memória estável.

2. páginas "shadow":

O servidor aloca páginas "shadow" para as páginas de dados atualizadas. O novo mapeamento é registrado, numa lista de intenções, para atualização posterior das páginas de mapa.

Se uma falha ocorrer durante a transação, as páginas "shadow" são liberadas e nada precisa ser feito para restaurar a árvore-B já que nenhuma escrita foi feita sobre informações arma-

zenadas anteriormente à transação interrompida.

3. listas de intenções:

Como uma página da árvore-B pode conter entradas de vários arquivos, ela pode ser compartilhada por diversas transações concorrentes. As novas entradas da página são mantidas na lista de intenções da transação. As páginas de dados são atualizadas através de páginas "shadow" somente e por isso, uma única transação pode ter uma tranca de escrita sobre dados de uma página.

Uma transação só está decidida, depois que a lista de intenções com informações sobre todas as escritas da transação for gravada no "log" de intenções (em memória estável).

4. protocolo de decisão duas fases:

fase 1: o coordenador da transação envia mensagem para todos os servidores envolvidos na transação para que armazenem as listas de intenções. Cada lista de intenções em um servidor participante deve conter a identificação do coordenador e o estado da transação, na fase 1, o estado da transação será "tentando decidir". O coordenador espera respostas dos participantes, incluindo identificação de cada servidor que envia resposta na sua lista de intenções. Se algum participante não concordar em terminar a transação ou não responder dentro de um certo período de tempo, a transação será abortada; caso contrário, será decidida.

fase 2: Conforme as respostas dos participantes, o coordenador

envia mensagem para que todos decidam ou todos abortem. No primeiro caso, cada participante muda o estado da transação de "tentando decidir" para "decidido", realiza as intenções, envia confirmação para o coordenador e apaga sua lista de intenções. Quando todos os participantes tiverem respondido, o coordenador marca suas intenções como realizadas.

4.3.5. Decisão de uma transação

Uma transação só é considerada decidida quando sua lista de intenções é armazenada no "log" de intenções. Então, o cliente pode ser informado que a transação terminou com sucesso e as intenções podem ser realizadas.

A árvore B é atualizada de baixo para cima com páginas "shadow" e memória estável até que a página de nível mais alto da árvore B (não necessariamente a raiz) seja atualizada.

O mapa de bits que indica o estado (livre, alocada) de cada página do disco é atualizado - as páginas da versão antiga são liberadas. Uma transação abortada torna as páginas "shadow" alocadas, livres novamente.

4.3.6. Recuperação de uma transação interrompida

Ao reiniciar operação, o servidor percorre o "log" de intenções e realiza as intenções das transações decididas. A árvore

B é atualizada da forma descrita anteriormente e no caso de as intenções terem sido parcialmente realizadas antes da queda do servidor, as páginas "shadow" que foram escritas são esquecidas e escritas novamente. Quando as intenções forem totalmente realizadas, o "log" de intenções é atualizado atomicamente para refletir o novo estado da transação.

As páginas "shadow" de transações não decididas que foram interrompidas são marcadas livres no mapa de bits porque não ocorrem no "log" de intenções e nem na árvore B - ambos percorridos como parte da recuperação do sistema.

4.3.7. Transações com múltiplos clientes

A mesma mensagem add server, para adicionar um servidor a uma transação, é usada para conectar um novo cliente à transação. A identificação da transação, incluída na mensagem, deve ser obtida externamente ao sistema. Quando um servidor recebe um pedido de add server, verifica se já tem um cliente conectado envolvido na mesma transação, estabelece conecção com o cliente e se for o primeiro cliente dessa transação, comunica-se com o coordenador.

Uma tranca é associada com o cliente que a solicitou, para que o cliente possa ser notificado se tiver alguma tranca quebrada. Diferentes clientes têm privilégios de acesso diferentes.

Exemplo de uma transação com mais de um cliente: um cliente que cria um arquivo e o servidor de diretório que mapeia

nomes alfanuméricos em identificadores únicos. Ambos os clientes devem compartilhar a transação.

5. SERVIDOR DE ARQUIVOS IMPLEMENTADO

O servidor, desenvolvido nesta tese, suporta transações atômicas sobre vários arquivos. A interface oferecida aos clientes é semelhante a do servidor universal, podendo ser usada por sistemas de arquivos diferentes, embora não tenha sido implementado o tipo de objeto índice que facilitaria a construção de diretórios. É um sistema robusto a falhas do tipo "queda do servidor". Fornece acesso randômico a nível de páginas de arquivo.

5.1. Interface oferecida aos clientes

Operações de controle da transação:

. inicia transação

retorna - identificador da transação

. termina transação

parâmetros - identificador da transação

Termina a transação, fechando todos os arquivos e tornando todas as atualizações feitas pela transação efetivas.

. aborta transação

parâmetros - identificador da transação

Termina a transação, fechando todos os arquivos, mas ignorando qualquer atualização feita pela transação (todos os arqui-

vos permanecem no estado em que se encontravam antes de a transação iniciar).

Operações sobre arquivos:

. cria arquivo

parâmetros - identificador da transação

retorna - identificador do arquivo

. abre arquivo

parâmetros - identificador da transação, identificador do arquivo, tipo da tranca (escrita ou leitura)

Abre o arquivo, se a tranca solicitada for compatível com a tranca já existente sobre o arquivo ou se não existir tranca sobre o mesmo. Se a tranca for de leitura, várias transações podem acessar o arquivo, mas se a tranca for de escrita apenas uma transação acessa o arquivo.

. lê uma página do arquivo

parâmetros - identificador da transação, identificador do arquivo, número da página

retorna - página do arquivo , se existir

. escreve uma página do arquivo

parâmetros - identificador da transação, identificador do arquivo, número da página, dado

. remove arquivo

parâmetros - identificador da transação, identificador do arquivo

Remove o arquivo, se a transação terminar com sucesso.

Além dessas operações, um cliente pode invocar a operação que retorna o número de setores disponíveis em disco.

5.2. Identificação de um arquivo

Cada arquivo possui um identificador único que é enviado ao cliente no momento em que cria o arquivo. O servidor não tem conceito de usuário e basta a apresentação do identificador único do arquivo para que se tenha acesso a ele.

O identificador único é um inteiro sem sinal de 15 bits criado randomicamente. Não houve preocupação com proteção de arquivos, o que poderia ser obtido escolhendo-se o identificador a partir de um espaço de nomes maior.

5.3. Mecanismos internos da transação

Uma transação é um conjunto de operações elementares de escrita ou leitura sobre vários arquivos, além de criação ou remoção de arquivos.

A operação de início de transação retorna um identificador para a transação que deve ser usado em todas as operações seguintes da transação. A transação pode terminar com sucesso, neste caso, todas as atualizações realizadas pela transação se tornam efetivas ou então ser abortada (pelo cliente ou pelo próprio sistema), caso em que todas as atualizações são ignoradas.

Antes de realizar operações de acesso sobre um arquivo, o cliente deve invocar a operação para abrir arquivo, especificando o identificador do arquivo. O arquivo pode ser aberto para leitura somente (pode ser compartilhado por várias transações) ou para escrita (envolvido somente em uma transação). Também podem ser invocadas operações de remoção de arquivo, desde que não tenha sido aberto anteriormente.

É a operação de término de transação que decide a transação e libera todos os arquivos envolvidos, quando a transação for completada. Depois que a transação for decidida, todos os arquivos envolvidos serão atualizados para o novo estado.

A transação é executada com segurança suficiente para garantir atomicidade, mesmo em caso de falha, isto é, ou todas as operações elementares são realizadas (a transação tem efeito) ou nenhuma delas ocorre (a transação não tem efeito). Para isso, o servidor cria uma estrutura auxiliar - páginas "shadow", onde as atualizações são feitas (nenhuma alteração é feita sobre as páginas já existentes). Em caso de falha, o servidor garante que cada arquivo envolvido em uma transação interrompida seja recuperado

para o estado consistente anterior ao início da transação (se uma falha ocorrer antes que a transação seja decidida ou se a transação for abortada) ou que o arquivo seja modificado para novo estado contendo todas as atualizações feitas pela transação (se falha ocorrer depois que a transação tiver decidido, mas não completa).

Se os dados de um arquivo não forem críticos e o cliente desejar otimizar o acesso ao arquivo, pode realizar operações de escrita diretamente sobre as páginas já existentes do arquivo, dispensando a estrutura auxiliar de páginas "shadow". O servidor não garante consistência dos dados, caso ocorra falha durante uma operação de escrita no arquivo. Ao invés de determinar o tipo da atualização (atômica ou não) sobre o arquivo no momento da criação - como foi feito no Servidor Cambridge, que trata os arquivos como normais ou especiais - a escolha é feita no momento em que o arquivo é aberto. Durante a execução de uma transação, antes de acessar o arquivo, o cliente deve invocar operação para abrir arquivo, especificando escrita não atômica como parâmetro e o servidor em resposta, retorna um número de transação que deverá ser especificado nas operações de acesso sobre o arquivo. A operação para término de transação só tem a função de liberar o arquivo, já que o mecanismo de decisão não tem sentido, neste caso.

5.4. Controle de concorrência

O controle de concorrência regula o acesso a arquivos através de trancas.

Quando o cliente abre um arquivo, uma tranca é estabelecida sobre ele e mantida até o final da transação. A tranca indica a outras transações que o arquivo está sendo acessado.

A tranca é do tipo vários leitores/um escritor, isto é, vários clientes podem acessar um arquivo, se a tranca for de leitura, mas somente um cliente tem acesso sobre um arquivo, se a tranca for de escrita.

A operação para abrir arquivo verifica se já existe tranca sobre o arquivo. Se existir e a tranca pedida for incompatível com a já existente, a transação que pede a tranca tem que esperar a liberação do arquivo e tentar a operação novamente para poder acessá-lo. Isso pode provocar "deadlock", se duas ou mais transações ficarem esperando uma por outra para obter tranca. Temporização pode ser usada para evitar que transações fiquem esperando indefinidamente (ver extensões).

tranca existente	tranca pedida	
nenhuma	leitura/escrita	concedida
leitura	leitura	concedida
leitura	escrita	negada
escrita	leitura/escrita	negada

5.5. Estruturas de dados utilizadas pelo servidor

5.5.1. Estruturas armazenadas em disco

5.5.1.1. Representação de arquivos:

Informação num arquivo é dividida em páginas de tamanho fixo (512 bytes) mapeadas em setores não contíguos (não necessariamente) de disco.

A estrutura usada para representar arquivos é a de árvore-B [8], como no sistema distribuído da XEROX, com a diferença de que cada arquivo corresponde a uma árvore-B, enquanto que o sistema da XEROX utiliza somente uma árvore-B para todos os arquivos de um disco. Isso torna a atualização de um nó da árvore mais complexa, pois esse nó pode ser compartilhado por mais de uma transação: além de páginas shadow, o sistema da XEROX necessita de listas de intenções para atualizar a árvore.

Os nós da árvore são de dois tipos:

. páginas de dado

. páginas de índice

Um nó do tipo página de índice contém o mapeamento página-setor:

n	npag1	end1	...	npagn	endn

n= número de entradas na páginas (maximo-127)

npag i=número da página

end i = apontador para a página npag i do arquivo ou página de índice para as páginas do arquivo menores ou iguais a npag i e maiores que npag i-1. A altura da árvore é usada para determinar se o apontador é para uma página de índice ou para uma página de dado.

A raiz do arquivo contém na última entrada:

npag 127 = altura da árvore

end 127 = apontador para versão atual do arquivo (durante atualização atômica do arquivo)

As folhas da árvore representam as páginas de informação do arquivo.

Algoritmo para manipulação da árvore-B: na seção 5.8.1.

5.5.1.2. Tabela de identificadores de arquivos:

Os apontadores para as raízes das árvores, representando cada arquivo, se encontram numa tabela de identificadores nos primeiros setores do disco. Na implementação atual, a tabela ocupa quatro setores, parâmetro que pode ser ajustado para que o número de arquivos no sistema seja maior.

Um setor da tabela de identificadores contém as informações:

n	identl	endl	ntransl	...	identn	endn	ntransn

n= número de entradas (máximo - 85)

ident i = identificador único do arquivo

bit 15 - indica se arquivo consistente (0) ou não (1), para o caso de escrita não atômica interrompida

end i = apontador para raiz do arquivo

ntrans i = bits 13 a 0 indicam:

identificador da transação em que o arquivo está envolvido

=0 se não participa de nenhuma transação, ou se transação não foi decidida

≠0 quando a transação foi decidida (até ser completada)

bits 15-14 indicam:

00 - arquivo aberto para escrita atômica (após a decisão)

01 - arquivo aberto para escrita não atômica (na abertura do arquivo)

10 - arquivo que foi criado pela transação (após a decisão)

11 - arquivo que foi removido pela transação (após a decisão)

Algoritmo para inserção na tabela de identificadores: os bits 7-6 do identificador único do arquivo são usados como índice da tabela de identificadors. Se o setor indexado não tiver mais entradas livres, o novo identificador é inserido no próximo setor que tiver uma entrada livre.

Algoritmo para retirada da tabela de identificadores: o identificador do arquivo é procurado sequencialmente no setor indexado pelos bits 7-6 do identificador único. Se o identificador a ser retirado da tabela não for encontrado nesse setor, é procurado nos setores seguintes.

5.5.1.3. Mapa de disco:

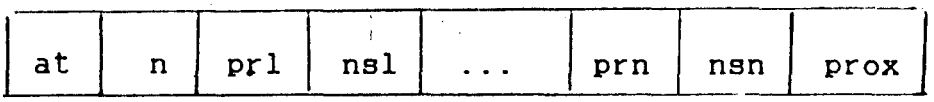
O mapa de disco contém a lista de blocos livres no disco. Um bloco é uma sequência de setores consecutivos.

O mapa de disco é gravado duplamente no disco. A redundância é necessária para preservar a consistência, em caso de queda do servidor durante a escrita do mapa (se o mapa tiver sido estendido para mais de um setor). O mapa ocupa inicialmente um setor (o segundo setor do disco) e se for necessário, pode ser estendido.

Nos outros servidores, a estrutura usada é um mapa de bits (sistema da XEROX) ou então uma tabela com o estado de cada bloco de disco (mapa de cilindro no Servidor Cambridge). Embora mais fácil de ser manipulado, esse tipo de tabela ocupa muito espaço para discos com grande capacidade de armazenamento. Numa tentativa de minimizar o espaço ocupado em disco pelo mapa, foi idealizada outra estrutura para o mapa de disco. Ela só se torna ineficiente à medida que os arquivos ficam fragmentados no disco e extensões ao mapa se tornam necessárias. Para resolver esse proble-

ma, um programa que compactasse os arquivos poderia correr periodicamente.

Um setor do mapa contém as informações:



at = indica qual dos dois mapas é o mais atual, pode valer 0,1,2

n = número de entradas no setor (máximo=126)

pr i = primeiro setor livre do bloco i

ns i = número de setores livres do bloco i

prox = apontador para extensão do mapa

A primeira entrada corresponde ao primeiro bloco livre do disco e assim por diante.

Algoritmo para atualização do mapa de disco: a atualização do mapa de disco é feita na memória - o mapa só será escrito em disco, quando a transação for decidida.

A escrita em disco é feita alternadamente sobre a cópia mais antiga. As extensões, se existirem, são escritas primeiro e por último o primeiro setor do mapa com o campo at atualizado (incrementado em módulo 3) para indicar que esta cópia é a mais atual. O mapa de disco que tiver o campo at com o maior valor será o mais atual. Assim, o mapa com at=1 é mais atual que o mapa com at=0, o mapa com at=2 é mais atual que o mapa com at=1 e o

mapa com at=0 é mais atual que o mapa com at=2.

5.5.2. Estruturas armazenadas na memória

5.5.2.1. Buffer de páginas: contém as páginas lidas do disco. Uma página de um arquivo permanece na memória, enquanto a transação que solicitou acesso a essa página estiver executando ou até que seja retirada para dar lugar a uma outra página.

Algoritmo para gerenciamento do buffer: cada página no buffer de páginas tem associado um contador na tabela de páginas (seção 5.5.2.3.) que indica se a página foi acessada recentemente. Além disso, existe um contador geral que é incrementado a cada acesso a uma página. O contador associado à página que está sendo acessada, recebe o valor do contador geral.

Se durante a execução de uma transação, uma determinada página deve ser lida do disco e guardada no buffer, caso não exista espaço disponível, uma página do buffer é escolhida para ser retirada. A página escolhida será a que tiver o contador com o menor valor, ou seja, a que foi acessada mais antigamente. E mais, se a página foi trazida para a memória durante uma operação de escrita (portanto deve ter sido atualizada), ela é escrita em disco antes de ser retirada.

As duas primeiras posições do buffer são reservadas para

o mapa de disco; na primeira posição, o mapa decidido, na segunda posição, o mapa atual. As extensões também são atualizadas no buffer, antes de serem escritas no disco.

5.5.2.2. Mapas de disco (no buffer de páginas):

O mapa decidido reflete a alocação dos setores de disco para as transações que foram decididas. É usado para a atualização do mapa (em disco), depois que uma transação decide. O mapa decidido é alterado somente pelas transações que decidem. Só é alterado depois do ponto de decisão.

Qualquer transação em execução retira do mapa atual os setores que vai utilizar. O mapa atual pode ser alterado a qualquer momento por uma transação.

Algoritmo para inicialização dos mapas de disco: quando o sistema é inicializado, após queda do servidor, as duas cópias do mapa de disco são lidas do disco. O campo at das duas cópias é comparado para decidir qual das cópias é a mais atual. Somente a mais atual permanece duplicada na memória, nas posições 1 e 2 do buffer de páginas. À medida que transações são executadas, os mapas de disco (no buffer de páginas) são usados diferentemente. O primeiro mapa na memória (mapa decidido) corresponde às atualizações feitas somente pelas transações que foram decididas. O segundo mapa na memória (mapa atual) reflete as modificações feitas por todas as transações em execução, decididas ou não.

Algoritmo para atualização dos mapas de disco: quando uma transação solicita um setor livre, o servidor retira o primeiro setor livre do mapa atual. Todos os setores solicitados pela transação para suas atualizações são inseridos na lista de setores ocupados (uma lista para cada transação, na tabela de transações, ver 5.5.2.4.). Os setores que correspondem às páginas da árvore que foram atualizadas e portanto, substituídas pelas páginas shadow, são inseridos na lista de setores livres (uma lista para cada transação, na tabela de transações, ver 5.5.2.4.).

Quando uma transação decide, todos os setores solicitados (na lista de setores ocupados) são retirados do mapa decidido. Os setores da lista de setores livres são inseridos no mapa atual e no mapa decidido. Então o mapa decidido pode ser escrito em disco.

Se uma transação é abortada, os setores solicitados (na lista de setores ocupados) são retornados ao mapa atual.

5.5.2.3. **Tabela de páginas:** contém o endereço em disco das páginas no buffer de páginas, a transação a que pertencem e indicador se foram recentemente acessadas. O campo identificador da transação é usado para que páginas associadas a uma transação sejam retiradas do buffer de páginas, quando a transação terminar.

5.5.2.4. **Tabela de transações:** contém informações sobre

todas as transações em execução.

Uma entrada na tabela de transações contém o identificador da transação, o apontador para a lista de setores ocupados pela transação, o apontador para a lista de setores livres da transação e o apontador para a lista de arquivos envolvidos na transação.

5.6. Atualização de arquivos

Um arquivo durante uma transação pode ser tratado como recuperável ou não recuperável. No primeiro caso, o arquivo é atualizado atomicamente, sendo possível qye seu estado reverta para um estado anterior consistente, se ocorrer uma falha durante a transação ou se a transação for abortada.

Se o arquivo armazenar dado não crítico, o cliente pode optar por atualização não atômica. Neste caso, se ocorrer uma falha, o arquivo pode ficar em estado inconsistente.

5.6.1. Arquivo recuperável - atualização atômica

As atualizações que ocorrem durante uma transação estão sujeitas a falhas, por isso devem ser armazenadas de maneira reversível. O servidor mantém para cada arquivo envolvido numa transação e aberto para escrita atômica, dois estados:
-o estado em que o arquivo se encontrava antes de a transação

iniciar (versão antiga) e que deve ser restaurado em caso de falha antes de a transação decidir ou término da transação pela operação de abortar

-o novo estado, com as atualizações que o cliente fez progressivamente (versão "shadow").

Somente no final da transação, quando ela é decidida, o novo estado está completo e o estado anterior pode ser descartado.

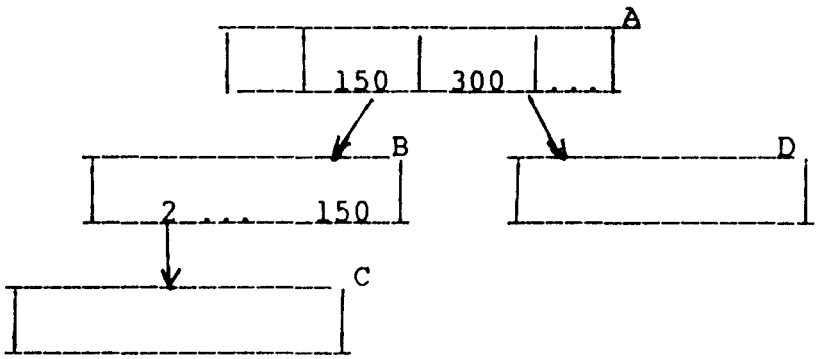
Quando uma página do arquivo é atualizada pela primeira vez durante uma transação, um setor de disco é alocado para ela e o novo conteúdo é escrito neste setor. Nenhuma nova informação é escrita na versão antiga - as atualizações afetam somente a versão "shadow".

Durante a transação, os setores de disco onde as páginas da versão "shadow" foram escritas são guardados na lista de setores ocupados da transação. Os setores de disco correspondentes às páginas da versão antiga que tiverem nova representação na versão "shadow" são guardados na lista de setores livres.

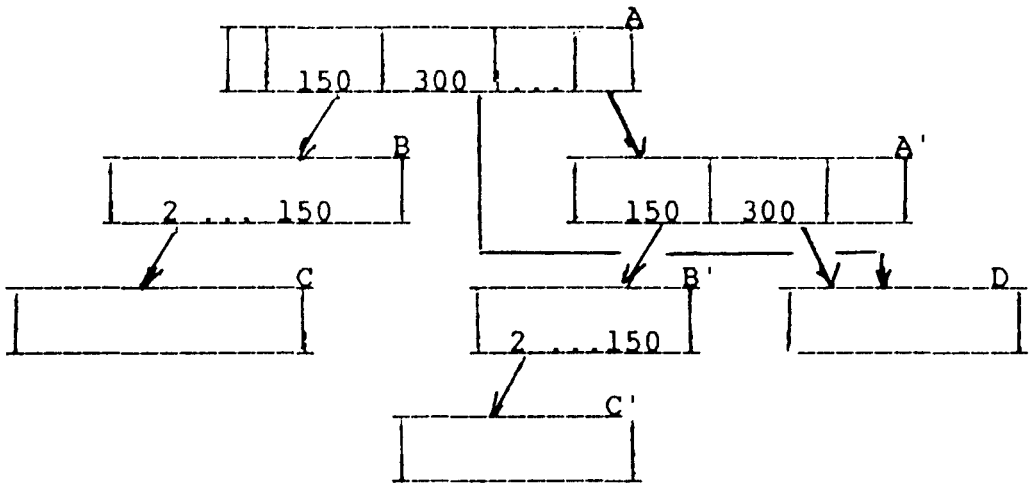
A operação de remoção, no momento em que é invocada, faz com que o arquivo seja percorrido e os setores de disco ocupados por ele inseridos na lista de setores livres da transação.

Esquema de uma atualização:

antes da atualização da página 2:



após a atualização da página 2:



O caminho da árvore até a página 2 é copiado em novos setores. A raiz A' da árvore atual é guardada na última posição da raiz da versão antiga. Os setores A,B,C são inseridos na lista de setores livres e os setores A',B',C' são inseridos na lista de setores ocupados para atualização do mapa do disco no final da

transação.

5.6.1.1. Transação termina com sucesso:

A operação para terminar a transação consiste em:

1. as páginas associadas à transação que ainda não foram gravadas e que foram atualizadas no buffer de páginas são escritas no disco.

2. todos os arquivos envolvidos na transação (com exceção dos arquivos abertos para leitura) são marcados, em disco, - o identificador da transação é guardado na entrada correspondente ao arquivo na tabela de identificadores.

O campo identificador da transação na tabela de identificadores indica, para cada arquivo, além da transação (bits 13 a 0) em que está envolvido, se a atualização feita sobre ele foi do tipo escrita atômica (bits 15-14 = 00), escrita não atômica (bits 15-14 = 01) ou remoção (bits 15-14 =11).

3. o identificador da transação é armazenado no primeiro setor de disco para indicar que a transação foi decidida. Este é o ponto de decisão. A partir desse momento, a transação será terminada mesmo em caso de queda do servidor durante a execução dos passos 4 a 8.

4. o mapa de disco é atualizado na memória - os setores de disco alocados pela transação (na lista de setores ocupados) são retirados do mapa (mapa decidido) e os setores liberados (na lista de setores livres) são inseridos no mapa (mapa atual e mapa decidi-

do).

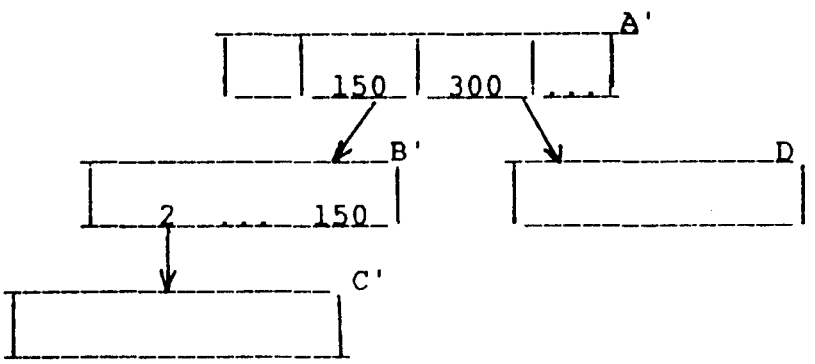
5. o mapa (decidido) é escrito em disco sobre o mapa de disco mais antigo. Se houver extensões, elas são gravadas primeiro e por último o primeiro setor do mapa com o campo at (que indica o mapa mais atual em disco) atualizado (algoritmo em 5.5.1.3.).

6. com o mapa refletindo o estado atual, a lista de arquivos envolvidos na transação é percorrida para que a tabela de identificadores seja atualizada - cada entrada correspondente a um arquivo (na lista de arquivos envolvidos na transação e aberto para escrita atômica) deve apontar para nova raiz (da versão atual) e o campo de identificador da transação deve valer zero (o arquivo não participa de nenhuma transação). Se o arquivo foi removido, sua entrada é retirada da tabela de identificadores.

7. a tabela de identificadores atualizada é escrita em disco.

8. o identificador da transação (no primeiro setor de disco) é atualizado para zero (transação está completa).

Se a transação que atualizou o arquivo mostrado anteriormente terminasse com sucesso, a nova representação do arquivo seria:



e o campo raiz do arquivo na tabela de identificadores que estava apontando para o setor A, depois de atualizado apontaria para o setor A'.

5.6.1.2. Transação é abortada:

Se uma transação é abortada, somente o mapa de disco na memória deve ser restaurado, pois nenhuma alteração foi feita em disco (com exceção, possivelmente, das páginas "shadow").

Os setores alocados durante transação estão presentes na lista de setores ocupados e são inseridos novamente no mapa atual.

5.6.2. Arquivo não recuperável:

Um arquivo aberto para atualização não atômica só tem uma versão e qualquer modificação sobre as páginas do arquivo é feita diretamente sobre a única versão do arquivo - as escritas são irreversíveis. Por isso, em caso de queda do servidor, o arquivo pode ficar em estado inconsistente.

O campo identificador da transação correspondente ao arquivo, na tabela de identificadores, indica que atualização sobre o arquivo é não atômica (bits 15-14 -01), desde o momento em que o arquivo é aberto para escrita não atômica (o tipo da atualização é um parâmetro da operação que abre arquivo). Depois que o mapa de disco é escrito em disco com as alterações necessárias, o

campo identificador de transação, na tabela de identificadores, é feito zero para indicar término de atualização do arquivo. Os passos 1, 4 a 7 (descritos na seção 5.6.1.1.) são executados.

5.7. Recuperação de uma transação em caso de falha

Falhas como queda do servidor não causam perda de consistência dos dados (se o arquivo tiver sido aberto para escrita atômica), pois o servidor garante a recuperação de cada arquivo envolvido na transação para o estado consistente igual ao estado em que se encontrava antes de a transação interrompida ter iniciado (se a transação não tiver sido decidida - número da transação = 0 no primeiro setor de disco) ou o estado que reflete as atualizações realizadas durante a transação (se a transação tiver sido decidida - o número da transação interrompida foi gravado no primeiro setor de disco). Qualquer outro tipo de falha será catastrófica.

A reconstrução do estado consistente é feita a partir de informações no disco e o conteúdo da memória volátil é considerado perdido. A rotina de recuperação pode ser reiniciada mais de uma vez, se ocorrer queda do servidor durante a sua execução, causando o mesmo efeito no estado do sistema.

Depois que o servidor volta de uma queda, a rotina de recuperação lê o primeiro setor do disco e verifica se uma transação foi decidida mas não completada. Se o identificador da

transação, gravado no primeiro setor, for diferente de zero, existe uma transação que precisa ser terminada e a rotina de recuperação percorre a tabela de identificadores verificando para cada arquivo , se está envolvido na transação interrompida.

5.7.1. Arquivo aberto para escrita não atômica:

Se um arquivo está marcado como envolvido em uma escrita não atômica (bit 15-14 do campo identificador da transação = 1) pode ter ficado em estado não consistente. A rotina de recuperação marca o arquivo como inconsistente (bit 15 do campo identificador do arquivo = 1) e o arquivo não deve mais ser acessado.

5.7.2. Transação interrompida antes da decisão:

A queda do servidor ocorreu antes da execução do passo 3 (descrito na seção 5.6.1.1). Uma transação pode ter sido interrompida:

1. antes de os arquivos envolvidos terem sido marcados (campo identificador de transação na tabela de identificadores)
2. depois que um arquivo foi marcado como envolvido em transação e antes de o número da transação ter sido escrito no primeiro setor de disco (número da transação igual a zero).

Em qualquer um dos dois casos acima, a transação foi interrompida antes de ser decidida e o estado dos arquivos deve permanecer inalterado. Como as atualizações que ocorrem durante a transação, só acontecem na memória, com excessão de alguma página

shadow escrita em disco, o estado de cada arquivo envolvido permanece consistente e igual ao que se encontrava antes de a transação iniciar.

O mecanismo de recuperação deve apenas alterar a entrada do arquivo na tabela de identificadores, se esta indicar que o arquivo está envolvido em uma transação (fazendo o campo identificador da transação igual a zero).

5.7.3. Transação interrompida depois da decisão:

A queda do servidor ocorreu depois da execução do passo 3 (descrito na seção 5.6.1.1).

Neste caso, a entrada na tabela de identificadores correspondente a um arquivo envolvido na transação tem o campo identificador de transação diferente de zero e igual ao identificador de transação gravado no primeiro setor do disco.

Todos os arquivos abertos para escrita têm duas versões:
- a versão antiga (igual ao estado do arquivo antes de a transação iniciar). O apontador para raiz do arquivo está na tabela de identificadores.
- a versão "shadow" (atualizações feitas pela transação). O apontador para a raiz da versão shadow está na última entrada da raiz da versão antiga. Se o arquivo tiver sido criado pela transação (bits 15-14 = 10), a raiz está na tabela de identificadores.

A rotina de recuperação deve reconstituir a transação interrompida e o mapa de disco. Os mapas de disco são lidos do dis-

co e somente o mais atual permanece na memória (o algoritmo para inicialização do mapa, foi descrito em 5.5.2.2.). A tabela de identificadores é percorrida e todos os arquivos marcados como envolvidos na transação são inseridos na lista de arquivos da transação.

Se um arquivo tiver sido aberto para escrita, a versão antiga é percorrida e os setores de disco correspondentes aos nós da árvore são inseridos na lista de setores livres da transação. A árvore que representa a versão "shadow" é percorrida e os setores de disco correspondentes aos nós são inseridos na lista de setores ocupados da transação.

Se um arquivo tiver sido marcado como removido, a árvore é percorrida e os setores ocupados pelo arquivo inseridos na lista de setores livres da transação.

Reconstituída a transação, o mapa de disco é atualizado, primeiramente na memória (mapa decidido), com a inserção dos setores livres da lista de setores livres e a remoção dos setores ocupados da lista de setores ocupados, nesta ordem. O mapa atual é atualizado da mesma maneira que o mapa decidido. Então, o mapa decidido é escrito em disco.

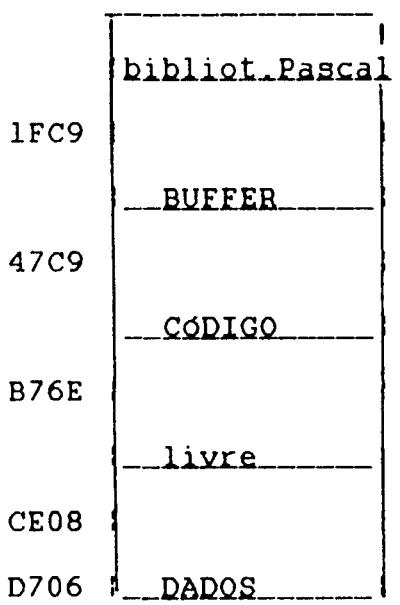
A tabela de identificadores é atualizada para que cada arquivo atualizado passe para a nova versão e seja marcado como não envolvido em transação.

O primeiro setor de disco é atualizado para indicar que todas as transações estão completas.

Este algoritmo tem a seguinte propriedade: caso o servidor caia durante a sua execução e o algoritmo seja reexecutado desde o início, quando o servidor voltar, o resultado final será o mesmo caso não houvesse queda do servidor.

5.8. Detalhes de implementação

Este servidor de arquivos ocupa 28k de código, 2k de dados, além de 10k para buffer de páginas. Roda no microcomputador I7000, utilizando uma unidade de disco Winchester para o armazenamento de arquivos. O mapa de memória do sistema é:



O servidor foi escrito em Turbo-Pascal, com exceção do código relacionado com a E/S em disco que foi inserido diretamente em linguagem de máquina.

Foram utilizadas as rotinas de EPROM da placa do contro-

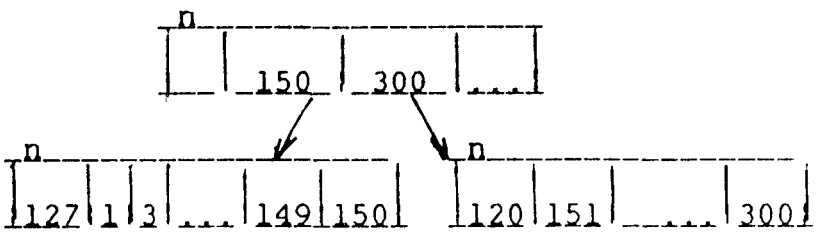
lador de disco. Nessa EPROM estão disponíveis as rotinas básicas de uso da placa, mas somente duas delas foram utilizadas - as rotinas de leitura e escrita de um setor (512 bytes) de disco. O código de preparação de parâmetros para essas rotinas - número de disco, número de trilha, número de setor, endereço para transferência, foi inserido diretamente através do comando inline.

5.8.1. Implementação de atualizações na árvore-B [8]:

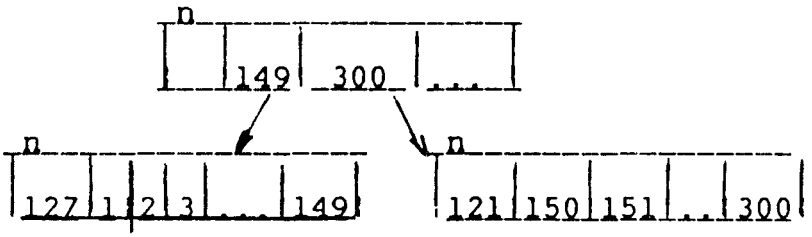
Uma página de índices contém no máximo 127 entradas (apontadores para outras páginas). Quando uma página está cheia e mais uma entrada tem que ser inserida, o servidor verifica os vizinhos. Se não houver espaço nos nós vizinhos então o nó e um de seus vizinhos é particionado em três.

a) overflow à direita: se houver espaço para mais uma entrada no nó vizinho à direita, uma entrada da página cheia é inserida no vizinho à direita, criando espaço para mais uma entrada na página que estava cheia.

Esquema para inserir página 2 do arquivo:

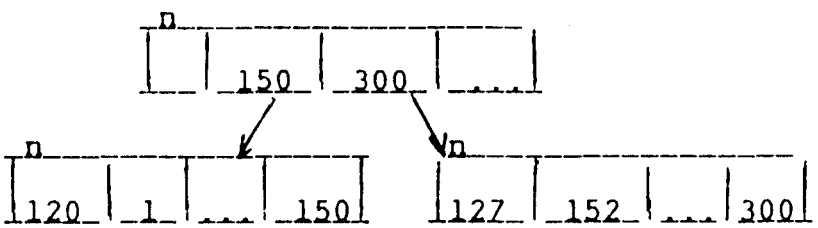


Depois da chamada da rotina que trata overflow à direita:

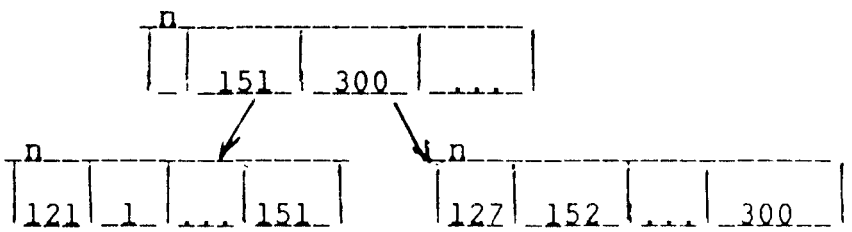


b) overflow à esquerda: Se houver espaço para mais uma entrada no vizinho à esquerda, a primeira entrada da página índice cheia ou apontador para página que está sendo acrescentada (o menor entre os dois) é colocado no vizinho à esquerda, criando espaço para mais uma entrada na página de índices que estava cheia.

Esquema para inserir página 151 do arquivo:

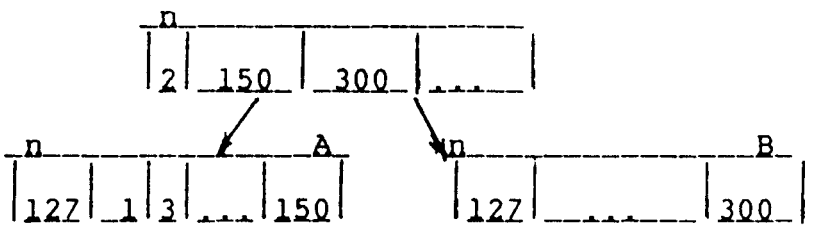


Depois da chamada da rotina que trata overflow à esquerda:

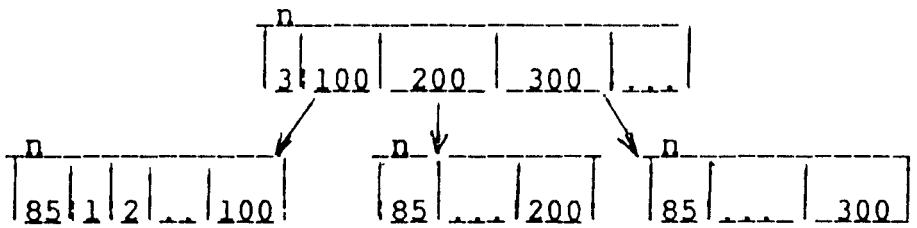


Se a página de índices onde mais um apontador tem que ser inserido estiver cheia e overflow não puder ser feito, o nó cheio e um de seus vizinhos é particionado em três nós.

Esquema para inserir página 2:



Depois da chamada da rotina que particiona nó, os nós A e B são particionados em três:



O mesmo esquema é aplicado para os níveis superiores da árvore, quando o apontador para o novo setor (alocado pela rotina que particiona nó) deve ser inserido.

5.9. Comparação entre o servidor implementado e os outros servidores

O servidor de arquivos implementado oferece uma interface flexível a seus clientes, de maneira que aplicações diferentes possam utilizar o servidor. Um cliente identifica um arquivo através de seu nome interno, sem ter que usar um sistema de arquivos para poder acessá-lo. Além disso, o servidor fornece atualizações atômicas sobre um conjunto de arquivos para manter a integridade dos dados. Todas as características mencionadas anteriormente são as mesmas do servidor universal. Difere na maneira como os dados são recuperados e atualizados pelos clientes - suporta acesso randômico a páginas de um arquivo, enquanto o servidor universal de Birrell e Needham permite desde o acesso a nível de palavras até o acesso a um arquivo inteiro.

A redundância necessária para assegurar a atomicidade de atualizações em um arquivo é obtida através de páginas "shadow", sendo similar à extensão de Paxton e ao sistema da XEROX e distante da técnica usada pelo servidor de Cambridge, que embora utilizando páginas "shadow", baseia a recuperação de falhas na redundância das estruturas de mapas de objeto e mapas de cilindro. O mecanismo implementado para a atualização atômica de vários arquivos (intenções na tabela de identificadores) é uma simplificação do arquivo de intenções de Paxton e da lista de intenções gravada em um "log" do sistema da XEROX.

Como o servidor de Cambridge, a unidade de tranca é um

arquivo inteiro. O tipo da tranca permite que vários leitores ou que somente um escritor acesse o arquivo.

O mapa de alocação proposto é uma lista ligada contendo os blocos livres do disco (com alguma semelhança com o sistema UNIX [15], que também usa lista ligada), na tentativa de minimizar o espaço ocupado pelo mapa de bits usado na maior parte dos servidores. Como o mapa é gravado duplamente em disco, fica mais simples implementar memória estável para representar o mapa.

6-TESTES E EXTENSÕES

Testes:

O servidor de arquivos utiliza um trecho do disco Winchester para simular um disco virtual, onde os arquivos são armazenados. As operações para o servidor foram invocadas pelo teclado e as mensagens de erro exibidas na tela.

Foram executadas transações que criavam, atualizavam e removiam arquivos. Transações concorrentes foram simuladas, intercalando-se as operações das transações.

Para testar a recuperação do sistema, transações que atualizavam e removiam arquivos foram executadas, mas não terminadas (algumas foram decididas, outras não chegaram a ser decididas). Na reinicialização do sistema, foi invocada a rotina de recuperação que tornou efetivas as atualizações (se a transação chegou a ser decidida) ou restaurou o estado do sistema, conforme se encontrava antes de a transação iniciar (se a transação não foi decidida).

Extensões:

Apresentaremos a seguir algumas extensões ao trabalho que tornariam o sistema mais robusto numa aplicação prática e que são relativamente simples de implementar:

. temporização: o microcomputador I7000 possui uma pasti-

lha 8253 que pode ser usada para implementar temporização. Para prevenção de "deadlock", pode-se associar uma temporização a cada transação em execução. Se durante um certo período de tempo, o servidor não receber nenhum pedido de uma transação, ela pode ser abortada, se uma segunda transação quiser acessar um arquivo aberto pela primeira transação. Dessa maneira, nenhuma transação ficará esperando indefinidamente uma outra terminar.

. serviço de diretório: um arquivo pode ser usado como diretório para armazenar os nomes e identificadores únicos dos arquivos criados. O cliente pode invocar, durante uma transação, uma operação que insere nome-identificador único no arquivo de diretório. O arquivo de diretório seria tratado como qualquer outro arquivo que participasse de uma transação e seu estado retornaria para estado anterior, se a transação abortasse. Outra operação disponível seria a tradução de nome para identificador único, tendo como parâmetro uma password. O mesmo serviço é oferecido pelo servidor Felix [5].

. verificação de erros de E/S em disco: as rotinas em EPROM para acesso a disco retornam um código de erro. Conforme o erro, poderia ser tentada uma recuperação (o que não foi feito nesta implementação), utilizando as outras rotinas disponíveis na placa ou tentando novo acesso.

. spooler: o servidor pode se encarregar de listar arquivos para os cliente.

. tabela de identificadores em memória estável: durante a

execução de uma transação, todas as informações relativas às atualizações dos arquivos são redundantes, com exceção da tabela de identificadores. As atualizações de um arquivo são feitas na versão "shadow" e a versão antiga permanece inalterada. Se ocorrer queda do servidor, durante a escrita de uma página do arquivo, nenhuma página da versão antiga ficará corrompida.

No momento da decisão, a tabela de identificadores deve ser atualizada e pode ficar corrompida, se houver queda do servidor durante a escrita. O problema poderia ser resolvido, se a tabela de identificadores fosse armazenada em memória estável: uma escrita de um setor da tabela que não chegou a ser completada, mas se tornou irrecuperável, poderia ser detetada através do CRC, gravado em cada setor do disco, e a outra cópia da tabela seria utilizada pelo mecanimsmo de recuperação. A raiz da versão "shadow" de um arquivo que nesta implementação é armazenada na raiz da versão antiga, passaria a ser armazenada na tabela de identificadores para evitar o mesmo tipo de problema.

BIBLIOGRAFIA

[1] Bernstein, P.A. e Goodman, N.

"Concurrency control in distributed database systems",

ACM Comput. Surv. 13, 2 (junho de 1981), 185-221

[2] Birrell, A.D. e Needham, R.M.

"A universal file server",

IEEE Trans. Software Eng. SE-6, 5

(setembro de 1980), 450-453

[3] Dion, J.

"The Cambridge file server",

ACM SIGOPS Operating Syst. Rev. 14, 4

(outubro de 1980), 26-35

[4] Eswaran, K.P., Gray J. N., Lorie R. A., Traiger I.L.

"Notions of consistency and predicate locks in a database system",

CACM 19, 11 (novembro 1976), 624-633

[5] Fridrich, M. e Older, W.

"The Felix file server"

Proc. 8th. ACM Symp. Operating Systems Principles,

ACM SIGOPS Operating Syst. Rev. 15, 5

(dezembro de 1981), 37-44

[6] Gray, J. N.

"Notes on database operating systems",

Lecture Notes in Computer Science 60, Springer-Verlag

(1978), 393-481

[7] Gray, J. N., McJones, P., Blasgen, M.W., Lorie, R.A., Price, T.G.,
Putzulu, G.F. e Traiger, I.L.
"The recovery manager of a data management system",
ACM Comput. Sur. 13, 2 (junho 1981), 223-242

[8] Knuth, D.E.
"The art of computer programming", vol. 3

[9] Lampson, B. W. e Sturgis, H.E.
"Crash recovery in a distributed data storage system",
Tech. Rep. XEROX Palo Alto Research Center, 1979

[10] Mitchell, J.G. e Dion, J.
"A comparison of two network-based file servers",
CACM 25, 4 (abril de 1982), 233-245

[11] Paxton, W.H.
"A client-based transaction system to maintain data integrity",
Proc. 7th. ACM Symp. Operating Systems Principles,
ACM SIGOPS Operating Syst. Rev. 13, 4
(dezembro de 1979), 18-23

[12] Sturgis, H.E., Mitchell, J.G., Israel, J.E.
"Issues in the design and use of a distributed file system",
ACM SIGOPS Operating Syst. Rev. 14, 3 (julho 1980), 55-69

[13] Svobodova, L.
"File Servers for network-based distributed systems"

ACM Comput.Surv., 16, 4 (dezembro 1984), 353-398

[14] Swinehart, D., McDaniel, G., Boggs, D.

"WFS: a simple shared file system for a distributed environment",

Proc. 7th. ACM Symp. Operating Systems Principles,

ACM SIGOPS Operating Syst. Rev. 13, 4 (dezembro 1979), 9-17

[15] Thompson, K.

"UNIX implementation"

The Bell System Technical Journal, 57, 6, parte 2

1978), 1931-1946