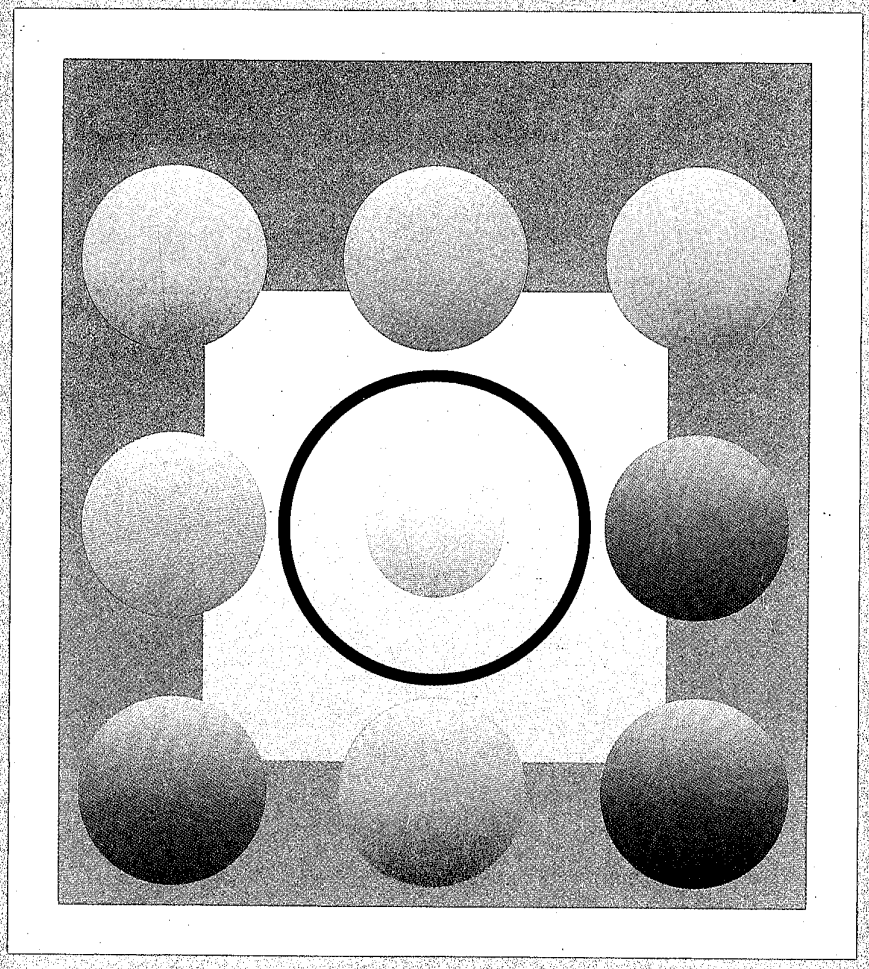
REVISTA
BRASILEIRA
DE COMPUTAÇÃO

RBC

ISSN 0101-0883

Uma publicação da SBC SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

1981, Vol. 1, Nº 2



NESTE NÚMERO

EDITORA CAMPUS

RBC, Rio de Janeiro, V. 1, nº 2, 1981, pp. 127 a 144

ESPECIFICAÇÃO DE UM COMPONENTE DE ARMAZENAMENTO CONFIÁVEL PARA UM SISTEMA DE ARQUIVOS DISTRIBUÍDO

OSCAR E. LANDES
MEDIDATA SISTEMAS
RUA DONA MARIANA, 166
CEP 22280 – RIO DE JANEIRO – RJ – BRASIL

DANIEL A. MENASCÉ
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO/DEPARTAMENTO DE INFORMÁTICA
RUA MARQUÊS DE SÃO VICENTE, 225
CEP 22453 – RIO DE JANEIRO – RJ – BRASIL

SUMÁRIO

Um sistema de arquivos distribuído deve ser tolerante a falhas e preservar a integridade dos dados. Técnicas existentes para recuperação de falhas em sistemas distribuídos são integradas no projeto de um componente de armazenamento confiável para um sistema de arquivos distribuído (SAD). O componente de armazenamento é a porção do SAD responsável pelo gerenciamento dos acessos aos arquivos e pela implementação de mecanismos de recuperação de falhas. A descrição apresentada aqui é informal, mas detalhada o suficiente para servir de base para uma implementação.

ABSTRACT

The property of a distributed file system to be able to recover from failures and maintain data integrity is extremely important. Existing crash recovery techniques in distributed systems are integrated into the design of a storage component for a stand-alone distributed file system. The storage component is the portion of the file system responsible for managing access to files and for implementing crash recovery mechanisms. The description given here is informal but detailed enough to lead easily to an implementation.

1. INTRODUÇÃO

A propriedade de um sistema distribuído de se recuperar de falhas e manter a integridade dos dados é extremamente importante. Nestes últimos anos, muitos estudos foram feitos na área de recuperação de falhas em sistemas de bancos de dados [1,2,3,4,5,9 e 11]. Landes em [6] apresenta uma especificação de um componente de armazenamento para sistemas de bancos de dados distribuídos.

O objetivo deste trabalho é integrar as idéias apresentadas por Israel et. al . em [10] e o componente de armazenamento especificado por Landes em [6 e 12], a fim de propor um componente de armazenamento confiável específico para sistemas de arquivos distribuídos independentes que não estão embutidos no sistema operacional. O componente de armazenamento é a porção do sistema de arquivos responsável pelo mapeamento dos arquivos em memória secundária e pela transferência dos mesmos entre a memória secundária e a memória principal.

O componente de armazenamento aqui proposto é robusto com relação a ocorrência de falhas de transação, de sistema e de memória secundária. Para tanto, o componente implementa mecanismos de recuperação. A idéia básica destes mecanismos é a noção de listas de intenções e memória com propriedade atômica forte. Estas e outras idéias consideradas importantes para o entendimento deste trabalho são apresentadas em [11].

Este artigo apresenta uma descrição informal, mas completa, do componente de armazenamento. Uma descrição formal de um componente semelhante a este pode ser encontrada em [6]. O componente de armazenamento aqui proposto trabalha a nível de transação.

A seção 2 do artigo apresenta algumas definições básicas e alguns conceitos preliminares sobre sistemas de arquivos distribuídos. É apresentada a definição de um sistema de arquivos distribuídos encontrada em [10]. A interface entre o componente e o restante do sistema é definida aqui. A seção 3 descreve a estrutura básica do componente de armazenamento, a estratégia de atualização utilizada e as funções que compõem sua interface com o resto do sistema de gerenciamento de arquivos (SGA). Finalmente, na última seção, são apresentadas as conclusões deste trabalho.

2. DEFINIÇÕES BÁSICAS E CONCEITOS PRELIMINARES

Segundo Israel et. al. [10], um sistema de arquivos distribuídos (SAD) é constituído por um conjunto de computadores de vários tipos conectados por uma rede de comunicação. Cada um destes computadores é chamado de servidor e armazena um conjunto de arquivos que podem ou não estar duplicados pela rede. Outros computadores atuam como clientes do sistema de arquivos distribuídos , para os quais o sistema provê facilidades para criar, deletar e acessar arquivos. A figura 1 mostra esta arquitetura.

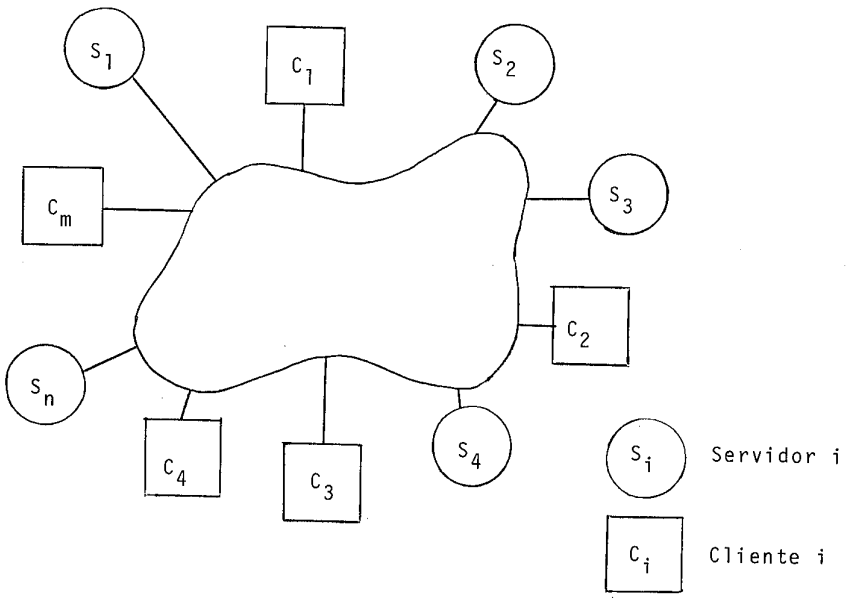


Figura 1 - Arquitetura de uma Rede de Computadores com um Sistema de Arquivos Distribuido

Cada servidor possui um sistema de gerenciamento de arquivos (SGA) que pode ser conceitualmente dividido em dois componentes:

a - Componente Lógico : responsável pela manutenção da visão do usuário, pelo mapeamento dos nomes lógicos dos arquivos fornecidos pelos usuários em identificações internas ao sistema e pelo mapeamento dos registros lógicos em registros físicos.

b - Componente de Armazenamento: responsável pelo mapeamento dos arquivos na memória auxiliar, pelo controle da transferência destes arquivos entre a memória secundária e a memória principal e pela implementação dos mecanismos de recuperação.

A interface entre o componente de armazenamento e o componente lógico consiste de um conjunto de funções ou operações implementadas pelo componente de armazenamento. Estas funções serão descritas na seção 3.

Cada arquivo é batizado com um identificador único (IDU). O mapeamento dos nomes dados aos arquivos pelos usuários para os IDUS é feito através de um diretório.

Um único serviço pedido ao SAD será geralmente providenciado por um conjun

to cooperante de computadores servidores. Os programas dos clientes e seus usuários podem tratar o SAD como um sistema lógico, e por isto serem independentes da distribuição dos arquivos pelas máquinas servidoras. A estruturação do SAD desta forma é motivada:

a) pela economia de dispositivos de armazenamento em disco. Usando-se computadores pequenos como servidores de arquivos permite-se um largo alcance de capacidade de armazenamento de arquivos porque é possível expandir o SAD desde um único microcomputador com um disco até vários servidores, cada um com um conjunto de discos.
b) pela necessidade de compartilhamento de dados comuns por pessoas situadas em localizações diferentes.
c) pela independência de funcionamento de cada servidor. Usando-se um conjunto de pequenos servidores a robustez do SAD é aumentada porque problemas de hardware e software em cada servidor não afetam os outros.

É interessante salientar neste ponto, que o componente de armazenamento aqui proposto, funciona perfeitamente quando as funções do cliente e do servidor são executadas em um único computador.

Os usuários interagem com o SGA através de transações. Uma transação é uma sequência de ações do tipo leia arquivo, atualize arquivo, crie arquivo e delete arquivo. Os usuários tem permissão de introduzir transações em qualquer um dos computadores clientes.

A interface entre os servidores e os clientes se dá através de um protocolo básico que garante a confiabilidade da transmissão de mensagens na rede.

As transações devem preservar a consistência do sistema de arquivos. Por causa da existência de múltiplas cópias dos arquivos, existem dois conceitos de consistência em SADs: consistência interna e consistência externa. A consistência interna de um arquivo é preservada se um determinado conjunto de assertivas ou restrições de integridade é sempre válido. No caso de um sistema de arquivos, as restrições de integridade não podem envolver mais do que um arquivo. A consistência externa ou mútua de um SAD é preservada se todas as cópias dos arquivos convergirem para o mesmo valor quando toda a atividade do sistema cessa.

As transações, por preservarem a consistência do SAD, são consideradas unidades de consistência. Cada transação isoladamente transforma um estado consistente em un novo estado consistente. Para tanto, o SGA distribuído deve garantir a propriedade atômica de uma transação: "ou todas ou nenhuma das modificações de uma transação devem ser executadas". Neste sentido, a transação também é considerada como uma unidade de recuperação.

Uma transação, ao ingressar no sistema, deve receber uma identificação única (geralmente um número). O mecanismo de geração destas identificações deve ser tal que não possibilite a geração de duas identificações iguais durante um certo período de tempo.

A sincronização de transações que atualizam os mesmos arquivos é um problema de grande importância. A metodologia de sincronização utilizada deve garantir que todos os arquivos sejam atualizados correta e consistentemente.

O método mais simples utilizado para sincronizar atualizações distribuídas em um sistema de arquivos convencional é o bloqueio dos arquivos que estão sendo atualizados por transações concorrentes. Quando um arquivo vai ser atualizado a transação adquire um bloqueio em modo exclusivo referente aquele arquivo e o mantém até que a transação seja completada. O bloqueio é um mecanismo de serialização que garante que somente uma transação acessa a entidade bloqueada a cada vez.

O problema central do funcionamento confiável de um sistema de arquivos distribuído é a manutenção da consistência dos arquivos na presença de falhas durante a execução de transações de atualização. Estas falhas ameaçam destruir a atomicidade destas transações, fazendo com que a atualização seja parcialmente executada no sistema. É importante portanto, que mecanismos apropriados sejam incorporados ao sistema a fim de permitir o bom funcionamento do mesmo na presença dos mais variados tipos de falhas. Estes mecanismos são chamados de mecanismos de recuperação.

Existem vários tipos de falhas, que serão classificadas em 3 categorias de acordo com o tipo da ação de recuperação que é necessária: falhas de transação, falhas de sistema e falhas de memória secundária. Uma falha de transação ocorre quando o término normal de uma transação não pode ser alcançado e a transação deve ser cancelada. Uma falha de sistema é caracterizada pela perda do conteúdo da memória principal. Como consequência, todas as transações que estavam sendo executadas na hora da ocorrência da falha são perdidas. Uma falha de memória secundária ocorre quando parte ou todo o conteúdo de um arquivo é perdido e necessita ser recuperado a partir de uma cópia previamente criada.

A próxima seção descreve o componente de armazenamento. Antes porém, é importante salientar, neste ponto, quais as responsabilidades atribuídas ao SGA (servidor) e ao cliente para o perfeito funcionamento do componente de armazenamento que será descrito a seguir (daqui para frente, sempre que mencionarmos SGA estaremos nos referindo ao SGA menos o componente de armazenamento).

Ao SGA são atribuídas as seguintes responsabilidades:

. mapear os nomes dados aos arquivos pelos usuários para IDUs através de um diretório.

. resolver pedidos de acesso concorrente.
. tratar impasses.
. decidir qual é o pedido que deve ser atendido.
. identificar as credenciais do cliente (controlar o acesso aos arquivos).
. criar uma identificação única para as transações.
. manter um *log* de transações para poder recuperar os arquivos caso ocorra alguma falha de memória secundária.
. traduzir os comandos dos usuários nas funções oferecidas pelo componente de armazenamento.
. implementar a política de *dump*.
. identificar os diferentes tipos de falhas e tomar as devidas providências.
. recuperar os arquivos através do *log* no caso da ocorrência de uma falha de memória secundária.
. monitorar a distribuição dos comandos entre os diversos servidores.
. mapear os dados dos arquivos em páginas lógicas e manter índices para acessar as páginas lógicas dos arquivos.
. implementar o algorítmo de Lampson e Sturgis para controlar a execução distribuída das listas de intenções.

Ao cliente são atribuídas as seguintes responsabilidades:

. ter conhecimento do domínio da aplicação.
. ter conhecimento do significado dos arquivos.
. ter conhecimento da estrutura lógica dos arquivos.
. manter a ilusão de um sistema de arquivos único.
. manter um *log* de transações para poder refazer uma transação cancelada.

3. O COMPONENTE DE ARMAZENAMENTO

Estrutura Básica. O sistema de arquivos é uma coleção de arquivos. Cada arquivo é um espaço de endereçamento linear de tamanho variável. Os arquivos são divididos em páginas lógicas de mesmo tamanho. As páginas lógicas são armazenadas em páginas físicas na memória secundária.

Podem existir até k arquivos em cada servidor. Uma página lógica é referenciada através de seu número sequencial dentro do arquivo Ak, digamos i, onde $1 \leq i \leq Mk$ e Mk é o número máximo de páginas que o arquivo Ak pode conter. Duas páginas lógicas consecutivas do mesmo arquivo podem estar mapeadas em duas páginas físicas não consecutivas na memória secundária.

Para cada arquivo, existe uma tabela de páginas, que é uma sequência de endereços de páginas físicas. O i-ésimo elemento desta tabela indica a página física que é usada para armazenar o conteúdo da página lógica i. Cada tabela de páginas é implementada em páginas físicas, chamadas de páginas de ponteiros.

A figura 2 ilustra a estrutura básica do sistema de arquivos.

A tabela de páginas pode ocupar mais do que uma página física. Por esta razão, um arquivo é composto de uma ou mais páginas de ponteiros e várias páginas de dados. As páginas de ponteiros de cada arquivo são numeradas sequencialmente a partir de 1.

A fim de acessar as páginas de ponteiros que formam um arquivo, o componente de armazenamento mantem um diretório. Existe uma entrada no diretório para cada arquivo existente no servidor local. Cada entrada deste diretório contem uma lista de todos os endereços das páginas físicas que armazenam as páginas de ponteiros que formam a tabela de páginas do arquivo em questão. Note que a i-ésima entrada do diretório corresponde ao arquivo i, e o j-ésimo endereço desta entrada corresponde a j-ésima página de ponteiros deste arquivo. O diretório também é implementado usando páginas físicas que, neste caso, são chamadas de páginas de diretório.

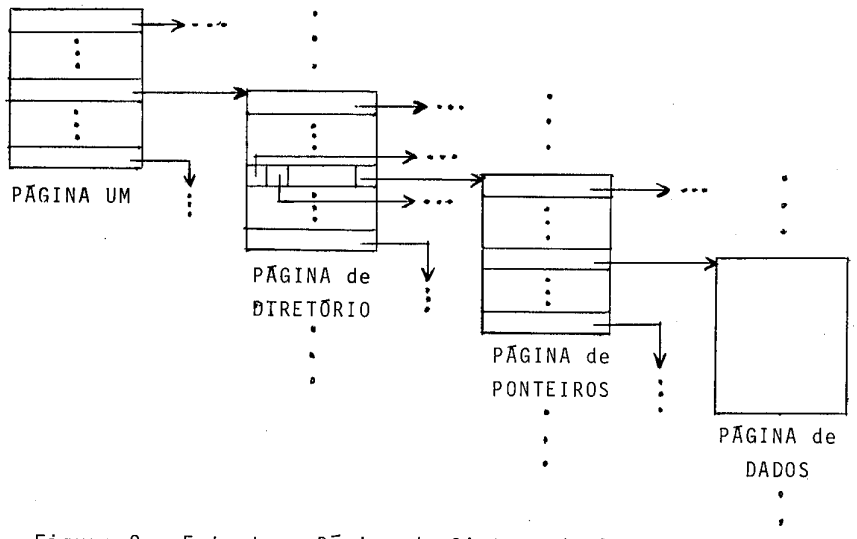


Figura 2 - Estrutura Básica do Sistema de Arquivos

Finalmente, para cada servidor, existe uma página no sistema que contem uma lista de todas as páginas de diretório, na ordem sequencial, existentes naquele servidor. Esta página é chamada página um, pois ela é alocada estáticamente na página física um. Todas as demais páginas no sistema são alocadas dinamicamente.

As páginas de diretório, páginas de ponteiros e a página um são chamadas de páginas de acesso. As páginas físicas que armazenam o conteúdo dos arquivos

são chamadas de páginas de dados.

A fim de acessar a página lógica i do arquivo k, o componente de armazenamento necessita executar os seguintes passos: encontrar a página de diretório que contém a entrada correspondente ao arquivo k; encontrar a página de ponteiros que contém o ponteiro para a página lógica i e finalmente, encontrar na página de ponteiros o endereço que referencia a página de dados desejada.

O componente de armazenamento proposto aqui, usa um mecanismo semelhante ao apresentado por Lorie em [3] com respeito a criação e modificação de páginas de dados. As novas versões das páginas de dados modificadas, bem como a primeira versão de uma nova página, são construídas em páginas físicas livres. A fim de encontrar estas páginas livres, o componente de armazenamento mantem, na memória secundária, um vetor de bits chamado MAP. O i-ésimo elemento deste vetor é um se a i-ésima página física do sistema está ocupada e zero caso contrário.

A seguir serão feitos alguns comentários sobre os *buffers* utilizados pelo componente. Todos os *buffers* são do mesmo tamanho. A cada *buffer* estão associadas as seguintes informações:

a. endereço na memória secundária da página física para a qual o conteúdo do *buffer* deve ser transferido.
b. identificação da transação que está usando o *buffer* (somente no caso de atualizações de páginas de dados).
c. bit de modificação que indica se o conteúdo do *buffer* deve ou não ser transferido para a memória secundária antes de sua reutilização.
d. contador de bloqueio: quando uma página física é transferida para um *buffer* a transação que originou esta transferência bloqueia o *buffer*. O *buffer* permanece bloqueado enquanto a transação estiver utilizando-o. Em outras palavras, o sistema de gerenciamento de *buffers* não pode liberar um *buffer* bloqueado. Note que várias transações podem estar usando o mesmo *buffer* concorrentemente(é o caso dos *buffers* que contem páginas de diretório). Por causa disso, o mecanismo de bloqueio foi implementado como se segue:

Para bloquear um *buffer*, a transação incrementa o contador de bloqueio associado ao *buffer*. A fim de liberar o *buffer*, a transação necessita decrementar uma unidade do contador. Dessa forma, o sistema de gerenciamento de *buffers* somente reusará aqueles *buffers* que não estão sendo utilizados por nenhuma transação, isto é, aqueles que possuem contador de bloqueio igual a zero. O contador de bloqueio é automaticamente incrementado pelo componente de armazenamento quando ele entrega um *buffer* para a transação. A transação necessita decrementar explicitamente o contador de bloqueio deste *buffer* através da função DECRBUFF oferecida pelo componente de armazenamento.

O componente de armazenamento mantem, para cada transação, uma lista de *buffers* que contêm páginas de dados. Quando uma transação é cancelada, os contadores de bloqueio de todos os *buffers* desta lista são decrementados de uma unidade pela função CANCELE_TRANSAÇÃO (veja seção 3).

Em resumo, o sistema de arquivos, em cada servidor, é composto da página um, das páginas de diretório, das páginas de ponteiros, das páginas de dados e das páginas que armazenam o vetor MAP. Todas estas páginas com exceção das páginas de dados são chamadas de páginas de acesso.

Estratégia de Atualização. Esta seção descreve a estratégia de atualização utilizada pelo componente de armazenamento para prover uma atualização confiável do sistema de arquivos. As páginas de acesso são sempre atualizadas por ações das listas de intenções. Relembre que existe uma lista de intenções associada a cada transação. As ações destas listas atualizam somente as páginas de acesso. As páginas de acesso refletem sempre um estado consistente dos arquivos do sistema. As páginas de arquivos são acessadas concorrentemente por várias transações. Portanto, a fim de evitar uma inconsistência, todas as atualizações feitas sobre estas páginas precisam ser vistas por todas as transações. Por exemplo, considere a seguinte situação: uma transação T1 cria a página de diretório que conterá as entradas dos arquivos 1 até R e inicializa a entrada do arquivo 1. Concorrentemente, uma transação T2 quer inicializar a entrada do arquivo 2. Esta transação não pode criar uma nova página de diretório mas sim, deve usar a mesma página criada pela transação T1. Este problema é resolvido da seguinte forma: o componente de armazenamento mantem uma cópia de todas as páginas de acesso que estão sendo atualizadas. As cópias são atualizadas no local e, para cada uma destas atualizações, o componente cria ações na lista de intenções da transação que originou estas atualizações para atualizar as páginas de acesso originais. Note que, apesar de não haver acesso concorrente sobre as páginas de ponteiros de cada arquivo, é necessário manter uma cópia destas páginas já que as páginas de ponteiros originais refletem o último estado de integridade do arquivo e só devem ser modificadas caso a transação termine corretamente.

Quando o sistema é ativado, uma cópia da página um é trazida para a memória principal bem como uma cópia de todas as páginas que armazenam o vetor MAP, que são armazenadas em um vetor chamado MAP_CORRENTE.

A figura 3 ilustra a estratégia de atualização através do seguinte exemplo. Considere uma atualização sobre a página lógica i do arquivo k. Como já mencionamos antes, as páginas de dados são atualizadas usando-se páginas físicas livres para construir a nova versão da página. Para cada página de dados modificada por uma transação, o componente de armazenamento precisa incluir

duas ações na lista de intenções da transação. Uma para desligar o bit no ve - tor MAP correspondente a página física que continha a versão original da página de dados e a outra para ligar o bit no vetor MAP correspondente a página física utilizada para construir a nova versão. Na figura 3, esta nova versão está na página física d. A fim de refletir a nova situação, a página de ponteiros que contem o endereço da página logica i deve ser atualizada para apontar para a nova página de dados.

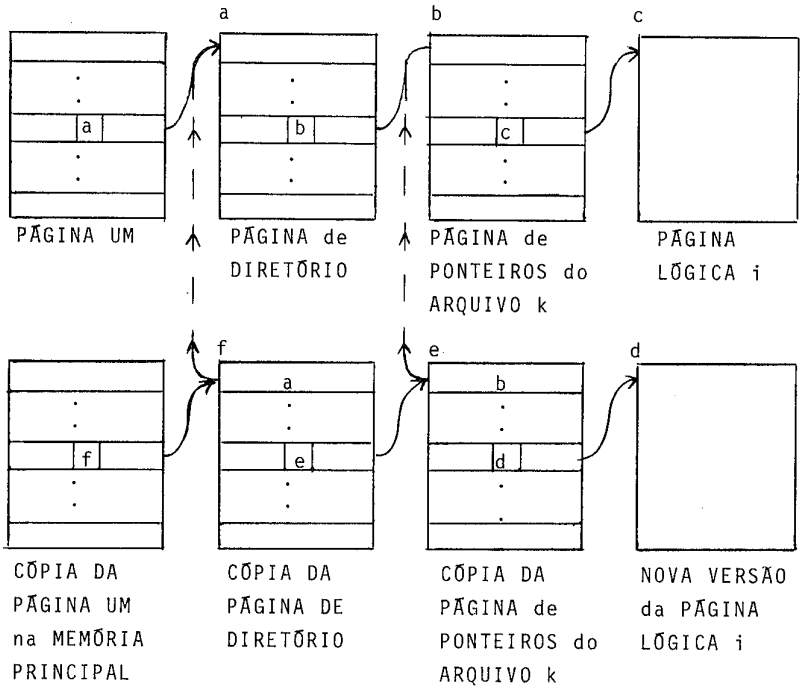


Figura 3 - Estratégia de Atualização

Dado que a página de ponteiros original faz parte de um estado consistente do arquivo k, a página de ponteiros original é copiada para uma página livre e o ponteiro para a página lógica i é atualizado na cópia para apontar para a nova versão da página de dados. A fim de atualizar a página de ponteiros original, uma ação precisa ser incluida na lista de intenções da transação em questão. Esta ação é da forma: "escreva o valor d na palavra x da página física h", onde x é a palavra que contem o endereço da página lógica i. Note que a página de arquivos original que aponta para a página de ponteiros original deve ser atuali-

zada para apontar para a página de ponteiros duplicada. Neste instante, o mesmo processo descrito acima se repete. Em outras palavras, a página de diretório original é copiada para uma página física livre e o endereço da página de ponteiros é atualizado na cópia. O endereço da página de diretório duplicada precisa ser atualizado na página um. Como uma cópia da página um está sempre na memória principal, esta cópia é atualizada para apontar para a página de diretório duplicada.

É importante enfatizar que depois que uma página de ponteiros é duplicada uma vez, não é necessário duplicá-la novamente. Todas as modificações subsequentes podem ser feitas diretamente nas cópias.

Quando uma página de acesso é duplicada, o bit correspondente no vetor MAP_CORRENTE é ligado para indicar que a página física que armazena a cópia não está livre.

O acesso a qualquer página de dados é feito através da página um da memória principal. Portanto, qualquer transação acessará sempre as cópias das páginas de acesso em vez das originais. Isto garante que todas as transações vejam sempre o estado corrente do sistema de arquivos.

Periodicamente, as páginas de acesso duplicadas devem ser liberadas. Deve-se tomar cuidado para preservar a integridade do sistema de arquivos. Por isto, o SGA precisa ter certeza que não existem transações em progresso e que não existem listas de intenções pendentes para serem executadas, dado que é através da execução das listas de intenções que as atualizações são refletidas nas páginas de acesso originais. Vamos definir uma instante de reorganização como sendo o instante no tempo no qual as condições acima se verificam e quando o SGA solicita ao componente para liberar todas as páginas de acesso duplicadas. Como consequencia, as páginas de acesso duplicadas existem entre dois instantes de reorganização. Então, quando uma página de acesso vai ser atualizada, o componente verifica se a página já foi duplicada desde o último instante de reorganização. No caso afirmativo o componente atualiza a página duplicada, caso contrário o componente duplica a página antes de atualizá-la.

A fim de introduzirmos outros aspectos da estratégia de atualização, é necessário mostrar, com mais detalhes, a estrutura interna de uma página de acesso (página de diretório e página de ponteiros). Como é ilustrado na figura 4, a primeira palavra de uma página de acesso contem um bit chamado de bit de criação, cuja utilização será explicada mais adiante, e um ponteiro. Este ponteiro é nulo para as páginas de acesso originais e ele aponta para a página de acesso original correspondente no caso das páginas de acesso duplicadas. É consultando o valor deste ponteiro, que o componente sabe se uma página de acesso qualquer é uma cópia ou uma original (veja as linhas tracejadas na figura 3).

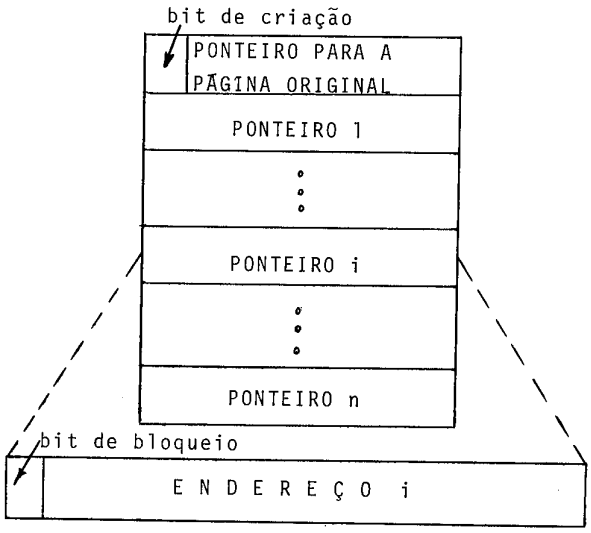


Figura 4 - Estrutura Interna de uma Página de Acesso

As palavras restantes de uma página de acesso formam uma lista de ponteiros para outras páginas de acesso ou para páginas de dados. Cada um destes ponteiros é implementado como sendo um par (bit de bloqueio, endereço). Para as páginas de arquivo, o bit de bloqueio é usado para indicar que a página apontada pela parte do endereço do ponteiro está sendo acessada por alguma transação. Este mecanismo de bloqueio é necessário para evitar que duas ou mais transações concorrentes dupliquem mais de uma vez a mesma página de acesso. Para as páginas de ponteiros este bit de bloqueio não tem utilidade já que estas páginas nunca são acessadas concorrentemente.

Vamos rever agora, o processo de atualização de uma página de diretório P. A página um contem um ponteiro para P. Se o bit de bloqueio deste ponteiro é igual a zero então, ele é feito igual a um. A página P é acessada e a parte de endereço da primeira palavra é verificada. Se a página já é uma cópia, então o bit de bloqueio é zerado, isto é, a página é desbloqueada. Caso contrário, a página é duplicada, o ponteiro P na página um é atualizada e a página P é desbloqueada (bit de bloqueio igual a zero).

Vejamos agora, a utilização do bit de criação. Quando uma página de diretório é criada, uma página física livre é encontrada, inicializada com zeros, e uma ação é incluída na lista de intenções para atualizar o ponteiro na página original para apontar para a nova página criada. É incluída outra ação na lista

de intenções para ligar o bit no vetor MAP correspondente a página física usada para armazenar a página de diretório criada. Além disso, a página é duplicada como se já existisse. A primeira palavra da cópia é atualizada para apontar para a página original e o bit de criação da cópia é feito igual a um para indicar que a página está sendo criada. O bit de criação não tem significado para as páginas originais.

Quando uma transação T acessa uma página de diretório que está sendo criada por uma transação T', o componente de armazenamento precisa incluir na lista de intenções da transação T uma ação para atualizar o ponteiro da página que está sendo criada. Este procedimento garante que mesmo que a transação T' seja cancelada, todas as outras transações que usaram a página de diretório criada não sejam afetadas. Quando uma destas transações terminar com sucesso, o sistema refletirá a criação da página.

O componente de armazenamento precisa incluir também, na lista de intenções de cada transação que utiliza uma página de diretório que estiver sendo criada, uma ação para desligar o bit de criação.

A fim de liberar as páginas de acesso duplicadas em um instante de reorganização, o SGA precisa requerer que o componente de armazenamento execute a função REINICIE (veja seção 3). Esta função traz para a memória principal a página um original armazenada na memória secundária. Desta forma, o acesso a todas as páginas duplicadas é perdido. A função REINICIE também copia o vetor MAP armazenado na memória secundária para o vetor MAP_CORRENTE mantido em memória principal, a fim de liberar as páginas físicas alocadas para armazenar as páginas duplicadas. Note que o vetor MAP nunca refletiu a existência destas páginas.

A recuperação de falhas de sistema e de transação é uma tarefa bastante simples levando-se em consideração a estratégia de atualização descrita nesta seção. Quando uma falha de sistema ocorre, o SGA pede ao componente que execute a função REINICIE. Neste caso, todas as transações que estavam em progresso no momento da ocorrência da falha precisam ser executadas novamente. Se uma falha de transação ocorre, é suficiente deletar a identificação da transação da tabela de transações ativas, dado que, sua lista de intenções ainda não foi escrita em memória estável. É importante recordar, que uma transação somente atualiza os arquivos do sistema quando sua lista de intenções é executada.

Em resumo, as páginas que armazenam o vetor MAP, todas as páginas de diretório, páginas de ponteiros e a página um são atualizadas somente, pelas ações das listas de intenções. Todas as cópias destas páginas são atualizadas no local no momento da execução da transação.

Funções Oferecidas pelo Componente de Armazenamento. É responsabilidade do componente lógico do SGA traduzir uma transação do usuário em uma sequência de

comandos que podem ser diretamente executados pelo componente de armazenamento. Esta seção descreve as funções que são oferecidas pelo componente de armazenamento. Na descrição que se segue, IT é a identificação da transação que origi - nou a chamada da função.

A tabela 1 apresenta as seis funções relativas à manipulação de arquivos. Sempre que a execução de uma destas funções implicar na modificação de uma página de acesso - páginas de diretório, de ponteiros, do vetor MAP e página um - serão criados comandos, com esta finalidade, na lista de intenções correspondente à transação IT.

TABELA 1

FUNÇÃO	DESCRIÇÃO
CRIEARQ (IT, ARQ)	Cria uma entrada para o arquivo ARQ no diretório,
DELETEARQ (IT, ARQ)	Deleta a entrada correspondente ao arquivo ARQ do diretório, liberando todas as páginas de ponteiros e de dados do arquivo.
LEIAPAGINA (IT, ARQ, PAG, ENDBUF)	Transfere a página PAG para um *buffer* cujo endereço é ENDBUF.
CRIEPAGINA (IT, ARQ, PAG, ENDBUF)	A página lógica PAG do arquivo ARQ é criada com conteúdo igual ao do *buffer* cujo endereço é ENDBUF
DELETEPAGINA (IT, ARQ, PAG)	A página física que armazena o conteúdo da página lógica PAG é dele - tada do arquivo ARQ.
ATUALIZEPAGINA (IT, ARQ, PAG, ENDBUF)	Traz para o *buffer* cujo o endereço é ENDBUF a página do arquivo ARQ. O bit modificação do *buffer* é setado.

TABELA 1 - Funções Relativas a Arquivos

Como exemplo vamos detalhar a seguir o funcionamento lógico da função CRIEPAGINA. O componente de armazenamento encontra uma página física livre e associa seu endereço a um *buffer* livre. O bit de modificação do *buffer* é feito igual a um, seu contador de bloqueio é incrementado e seu endereço é devolvido em ENDBUF. Antes do *buffer* ser reutilizado, seu conteúdo será transferido para a memória secundária. A palavra apropriada na página de ponteiros apropriada do arquivo ARQ é atualizada para apontar para a página física livre encontrada,

como resultado de uma ação criada na lista de intenções da transação IT. Outra ação na lista de intenções é criada para ligar no vetor MAP o bit correspondente a página física usada para armazenar a nova página lógica.

Dado que uma transação precisa ser considerada como uma ação atômica, as ações da transação são delimitadas por um comando INICIE_TRANSAÇÃO(IT) que cria uma entrada na tabela de transações ativas e por um comando FINALIZE_TRANSAÇÃO(IT) que grava a lista de intenções em memória estável.

A lista de intenções constitui o mecanismo básico para implementar transações atômicas. A execução das ações das listas de intenções incluem as atualizações no diretório, nas tabelas de páginas, na página um e no vetor MAP.

Quando as funções relativas a arquivos são executadas, são criadas ações na lista de intenções da transação em questão. Como já foi dito, as funções CRIEARQ e DELETEARQ geram ações para atualizar a tabela de arquivos e o vetor MAP. As funções CRIEPÁGINA, DELETEPÁGINA e ATUALIZEPÁGINA geram ações para atualizar as tabelas de páginas e o vetor MAP.

A consistência dos arquivos do sistema é preservada pelo SGA e pelo componente de armazenamento seguindo-se as regras abaixo:

Regra 1: o SGA não permite atualizações concorrentes em arquivos (páginas de ponteiros e páginas de dados).

Regra 2: todos os recursos bloqueados por uma transação (inclusive arquivos) são mantidos até que a lista de intenções da transação seja escrita em memória estável em todos os nós envolvidos com a transação.

Regra 3: em cada nó, as listas de intenções são executadas na ordem em que foram criadas. Note que esta restrição não implica que as transações precisam ser executadas serialmente. Assim que a lista de intenções da transação T é escrita em memória estável, seus recursos são liberados e usados por outras transações antes que a lista de intenções da transação T seja executada.

Como resultado, quando a transação T2 usa um arquivo previamente modificado por uma transação T1, o sistema garante que a lista de intenções de T1 será executada, em cada nó, antes da lista de T2.

Dado que o SGA implementa o mecanismo de controle de concorrência, também é de sua responsabilidade controlar que o componente armazenamento execute as listas de intenções na ordem correta. O componente de armazenamento provê funções para manipular listas de intenções.

Antes que uma lista de intenções seja criada, o componente de armazenamento precisa ter certeza de que o conteúdo de todos os *buffers* que contem páginas de dados modificadas pela transação já foram transferidos para a memória secundária. Estes *buffers* podem ser facilmente reconhecidos através da identificação

da transação associada ao *buffer* (veja seção 3). Esta estratégia oferece segurança a falhas de sistemas. Sendo assim, depois que uma lista de intenções de uma determinada transação já foi escrita em memória estável em todos os servidores, é sempre possível recuperar-se de qualquer tipo de falha.

A função EXECUTELISTA(IT) executa a lista de intenções e a função DELETELISTA(IT) deleta a lista de intenções caso algum servidor não consiga escrever a sua lista de intenções.

A Tabela 2 apresenta as funções relativas aos diferentes tipos de falhas.

TABELA 2

TIPO DE FALHA	FUNÇÃO	DESCRIÇÃO
TR	CANCELE-TRANSAÇÃO (IT)	É deletada a entrada correspondente a transação IT da tabela de transações ativas. Todas as páginas físicas alocadas pela transação são liberadas.
ST	REINICIE	Restaura o último estado de integridade representado pelo vetor MAP e pela página um da memória secundária. Executa todas as listas de intenções pendentes e finalmente, restaura o novo estado de integridade decorrente da execução das listas.
MS	INICIE-DUMP(IT)	Inicia o processo de dump dinâmico de todos os arquivos do sistema [7].
	RECUPERE (PONTPAG)	Recupera, seletivamente, os arquivos do sistema. O parâmetro PONTPAG aponta para a lista de páginas(criada pelo SGA) que devem ser recuperadas [8].

TABELA 2 - Funções Relativas a Falhas
(TR = Falha de Transação
ST = Falha de Sistema e
MS = Falha de Memória Secundária)

4. CONCLUSÕES

O componente de armazenamento é a porção de um sistema de arquivos responsável pelo mapeamento dos arquivos na memória secundária, pelo controle da transferência destes arquivos entre a memória secundária e a memória principal e pela implementação dos mecanismos de recuperação. Este artigo apresentou uma descrição informal, mas completa, de um componente de armazenamento robusto a falhas de transação, de sistema e de memória secundária, específico para um sistema de arquivos distribuído. A estratégia de atuallização utilizada no projeto do componente integra algumas técnicas já existentes com alguma idéias novas. A estratégia básica consiste em manter a qualquer instante um estado consistente dos arquivos. Este estado consistente é modificado somente através da execução de alguma lista de intenções. O estado corrente do sistema também existe e é atualizado em tempo de execução das transações.

O nível de detalhes da descrição apresentada aqui está bem próximo de uma possível implementação do componente.

5. REFERENCIAS

[1] LAMPSON, B. e STURGIS, H.E. "Crash Recovery in a Distributed Data Storage System", Xerox Palo Alto Research Center, Palo Alto, California, USA. 1976 (aceito para publicação em CACM).

[2] GRAY, J.N., "Notes on Data Base Operating Systems", Capítulo 3. F do livro *Operating Systems An Advanced Course*, Springer-Verlarg, 1978.

[3] LORIE, R.A., "Physical Integrity in a Large Segmented Database", *ACM Transactions on Database Systems*, Vol 2, No 1, Março 1977.

[4] REUTER, A., "Minimizing the I/O - Operations for Undo-Logging in Data base Systems", *Proceedings of the Fifth International Conference on Very Large Data Bases*, Rio de Janeiro, Brasil, outubro 3-5, 1979, pags. 164-172.

[5] VERHOFSTAD, J.S.M., "Recovery Techniques for Database Systems", *Computer Surveys*, Vol 10, No 2, Junho 1978.

[6] LANDES, O.E., "Recuperação de Falhas em Bancos de Dados Distribuídos: Especificação de um Componente de Armazenamento Confiável", Tese de Mestrado , Departamento de Informática, PUC/RJ, Rio de Janeiro, Brasil, Janeiro 1980.

[7] ROZENKRANTZ, D.J., "Dynamic Database Dumping", *Proceedings of the 1978 ACM/SIGMOD Conference*, Austin, Texas, Maio 31 - Junho 2, 1978, pags. 3-8.

[8] MENASCÉ, D.A., "Selective Reloading of Very Large Databases", *Proceedings of the Third International Computer Software and Applications Conference*, Chicago, Illinois, Novembro 6-8, 1979, pags. 583-587.

[9] GRAY, J.N. et al., "The Recovery Manager of a Data Management Sys - tem", IBM Research Laboratory Report RJ2623(33801), San Jose, California,

15 de Agosto, 1979.

[10] ISRAEL, J.E., MITCHELL, J.G., STURGIS, H.E., "Separating Data from Function in a Distributed File System", Xerox Corporation, Palo Alto, Research Center, Palo Alto, California, USA, 1978.

[11] LANDES, O.E., MENASCÉ, D.A., "Um Estudo sobre Técnicas de Recuperação de Erros em Bancos de Dados", *Revista Brasileira de Computação, Vol 1, Nº 1,* Julho de 1981, pp. 55 a 71.

[12] MENASCÉ, D.A., LANDES, O.E., "On the Design of a Reliable Storage Component for Distributed Database Management Systems", *Proceedings of the 6th Very Large Database Conference,* Montreal, Canada outubro de 1980.