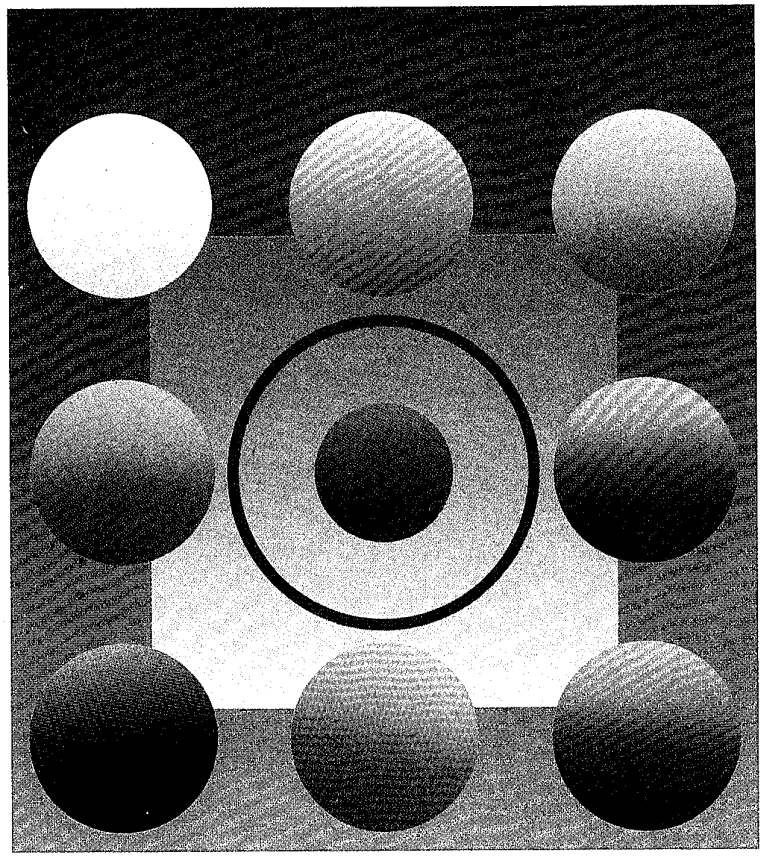
REVISTA
BRASILEIRA
DE COMPUTAÇÃO

RBC

Uma publicação da Sociedade Brasileira de Computação

1981, Vol. 1, Nº 1



NESTE NÚMERO

EDITORA CAMPUS

RBC, Rio de Janeiro, V. 1. nº 1, 1981, pp. 55 a 71

UM ESTUDO SOBRE TÉCNICAS DE RECUPERAÇÃO DE ERROS EM BANCOS DE DADOS

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

Este trabalho apresenta um estudo de diversas técnicas usadas para recuperação de falhas em SGBDs e sistemas de arquivos modernos. Estas técnicas restauram o BD para um estado consistente após a ocorrência de uma falha. Diferentes mecanismos de recuperação devem ser usados para diferentes tipos de falhas. As técnicas também diferem com relação ao estado de integridade do BD após a ocorrencia da falha. Algumas delas são tolerantes a falhas, outras restauram o BD para algum estado consistente anterior e outras restauram o BD para o estado consistente mais recente. Finalmente, são apresentadas algumas técnicas que implementam a propriedade atomica para transações.

ABSTRACT

This work is a survey of several different techniques used for crash recovery purposes in modern DBMSs and file systems. These techniques restore the DB to a consistent state after a failure occurs. Different recovery mechanisms have to be used for different kinds of failures. These techniques also differ with respect to the state of integrity of the DB after the occurrence of a crash. Some of them are fault tolerant, others restore the DB to any previous consistent state, and others restore the DB to the most recent consistent state. Finally, we present some techniques which implement the atomic ("all or none") property for transaction.

1. INTRODUÇÃO

Este trabalho tem por objetivo apresentar as diferentes técnicas e mecanismos de recuperação de erros utilizadas hoje em dia pelos diferentes sistemas de arquivos e de bancos de dados. A intenção deste trabalho não é a de exaurir o assunto, mas sim a de apresentar de forma simples e clara as diferentes idéias utilizadas para restaurar os dados danificados após a ocorrência de uma falha.

Com o propósito de tolerar a ocorrência de falhas, componentes e algorítmos adicionais podem ser incluidos no sistema. Esses componentes e algorítmos tentam garantir que a ocorrência de estados errados não resultem em futuras falhas de sistema. Em termos ideais, os componentes e algorítmos removem os erros e restauram o BD para um estado correto, a partir do qual o processamento normal pode continuar. Esses componentes e algorítmos adicionais constituem as chamadas técnicas de recuperação [1].

As técnicas de recuperação são utilizadas para restaurar os dados de um sistema para um estado aceitável pelos usuários desse sistema. A noção de aceitável é diferente para diferentes ambientes. Em geral, aceitável significa correto, válido ou consistente.

Existem vários tipos de falhas (de sistema, de transação e de memória secundária) e em consequência vários tipos de recuperação. Para Verhofstad [1], existe sempre um limite para o tipo de recuperação que pode ser providenciado. Se a falha não apenas destrói os dados ordinários, mas também os dados de recuperação (dados redundantes mantidos pelo sistema para tornar a recuperação possível), a recuperação completa pode se tornar impossível.

Diferentes tipos de falhas são, em geral, tratados por mecanismos de recuperação distintos. É possível ainda, que o sistema não inclua mecanismos para certos tipos de falhas tais como: falhas raras, falhas que implicam em um elevado custo de recuperação (maior do que o valor da informação perdida), ou ainda, falhas não identificadas como possíveis durante o projeto do sistema.

A fim de que a recuperação de falhas seja possível torna-se necessário armazenar dados que permitam restaurar o estado em que o BD se encontrava por ocasião da ocorrência da falha. Estes dados, chamados de dados de recuperação, por sua vez também podem ser danificados devido a ocorrência de falhas. Por exemplo, uma falha em um cabeçote de leitura/gravação em disco pode destruir

não apenas os dados ordinários mas também os dados de recuperação. Seria preferível manter os dados de recuperação em outros dispositivo de entrada/saída. Todavia, existem outras falhas que podem afetar esse dispositivo, por exemplo, falhas no mecanismo que grava os dados de recuperação para aquela memória auxiliar.

Os dados de recuperação, por sua vez, também podem ser protegidos de falhas pela manutenção de outros dados de recuperação que permitam restaurar os dados de recuperação originais no caso de sua destruição. Essa progressão poderia seguir indefinidamente. Na prática, é claro, deve existir a aceitação de que dados de recuperação não podem ser totalmente dignos de confiança. Deve-se partir de alguns dados de recuperação básico, aos quais atribui-se uma probabilidade pequena de destruição, para então, construir-se o mecanismo de recuperação.

Neste trabalho, são descritas técnicas que podem ser usadas para recuperação, para tolerância a falhas e para a manutenção da consistência depois da ocorrência de uma falha. A tolerância à falhas é conseguida se os algorítmos normais do sistema operam com os dados de tal maneira que após a ocorrência de certa falha o sistema estará sempre em um estado correto, isto é, o estado em que o sistema estava antes da ultima operação realizada sobre os dados (ou possivelmente a ultima série de operações).

Para Verhofstad [1], um BD está em um estado correto se a informação que o BD armazena consiste das cópias mais recentes dos dados introduzidas no BD pelos usuários e não contem dados deletados pelos usuários. Um BD está em um estado válido se suas informações são parte das informações existentes em um estado correto. E por ultimo, um BD está em um estado consistente se ele está em um estado válido, e as informações que o BD mantem satisfazem as restrições de integridade dos usuários.

Verhofstad [1] faz a seguinte classificação dos processos de recuperação, com relação ao seu tipo:

1) recuperação para um estado correto.
2) recuperação para um estado correto que existia em algum momento no passado.
3) recuperação para um possível estado prévio, isto é, restauração de um conjunto de estados de arquivos previamente existentes que podiam não ter existido simultaneamente antes.

4) recuperação para um estado válido.

5) recuperação para um estado consistente.

6) resistência contra falhas.

A tolerância a falhas difere dos outros tipos de recuperação. Enquanto os outros tipos de recuperação restauram explícitamente estados, a tolerância a falhas mantem estados corretos pela maneira como os dados são manipulados e mantidos durante o processamento normal. Pode-se dizer que a tolerância a falhas restaura implícitamente estados. No final desse trabalho é feita uma relação entre as diferentes técnicas que serão apresentadas e os diferentes tipos de recuperação.

Dentro de um único sistema, podem existir diferentes mecanismos de recuperação correspondendo a diferentes tipos de falhas. Cada técnica de recuperação possui suas vantagens e desvantagens, mas cada uma delas possibilita ao sistema tolerar a ocorrência de diferentes tipos de falhas em diferentes ambientes. Isso implica em dizer, que todas as técnicas tem a sua validade e não podemos dizer que esta é melhor que aquela sem considerarmos o ambiente em que estão atuando.

Para alguns sistemas, a transação é considerada a unidade de recuperação; ela portanto aparece para o usuário como uma ação atômica, isto é, ou todas ou nenhuma das modificações de uma transação devem ser executadas. As técnicas consideradas nesse trabalho podem ser usadas para implementar essa propriedade atômica.

2. TÉCNICAS DE RECUPERAÇÃO

As diferentes técnicas de recuperação conhecidas e usadas no presente que serão descritas a seguir, estão agrupadas em 7 categorias [1]: Programa Restaurador, *Dump*, Trilha Auditora, Cópia e Versão Corrente, Duplicação, Arquivos Diferenciais e Substituição Cuidadosa.

Programa Restaurador. Um programa restaurador é executado após a ocorrência de uma falha para recuperar o sistema para um estado válido. Esse programa não utiliza dados de recuperação. (Essa é a unica técnica considerada aqui que não utiliza dados de recuperação). O programa restaurador é utilizado depois da ocorrência de uma falha se outras técnicas de recuperação, que utilizam dados de recuperação, falham, ou não são usadas, ou ainda se o sistema não é tolerante a falhas. Esse programa examina cuidadosamente o banco de dados após a ocorrência de uma falha, para avaliar

os danos ocorridos e para restaurar o BD para algum estado válido. O programa restaurador tem o objetivo de salvar as informações que se mantem reconhecíveis após a falha.

O programa restaurador examina as estruturas de dados e tenta reconstruir o BD ou restaurar a consistência, possivelmente ao custo da deleção de alguns arquivos ou dados.

Se o sistema não possui resitência contra falhas, após a ocorrência de uma falha a memória secundária pode tornar-se inconsistente caso ocorra perda das informações armazenadas nos *buffers* da memória principal. Um programa restaurador pode ser utilizado então, para tentar salvar o conteúdo dos *buffers*. De qualquer forma, se o conteúdo da memória principal é perdido, a restauração para um estado correto não é possível.

Dump. Uma das técnicas mais utilizadas para prover a recuperação do BD face a ocorrência de uma falha é o *dump*. Um *dump* é uma cópia da versão corrente de um BD armazenada em memória *on-line* para uma memória secundária removível (geralmente fital). Juntamente com o *dump* deve-se manter um diário à medida que as transações executam e atualizam o BD. O diário contem informações suficientes para determinar as mudanças realizadas pelas transações no BD.

Para alguns sistemas, o diário contem a identificação de cada transação e os dados de entrada utilizados pela execução daquela transação. Para outros sistemas, o diário contem para cada atualização de uma entidade do BD, o novo valor da entidade (e algumas vezes o valor antigo). Em caso da ocorrência de uma falha, a mais recente cópia do BD é obtida, e as modificações indicadas pelo diário são então realizadas sobre essa cópia do BD. Se o diário contem a identificação e os dados de entrada para cada transação, as transações são reexecutadas para reconstruir o BD como ele existia antes da falha. Se o diário contem as atualizações feitas pelas transações, essas atualizações são usadas para modificar a versão copiada do BD.

Às vezes ao gerar um *dump* aproveita-se para reorganizar o BD e recarregá-lo na sua nova organização (Ex: consolidação do espaço livre deixado pela deleção de registros).

O processo de criar um *dump* pode ser Estático ou Dinâmico.

Dump Estático. Usualmente, o *dump* do BD é criado em uma maneira estática. Enquanto o *dump* está sendo criado, transações que atualizam o BD não são permitidas de executarem. Com isso, o sistema

de BD não fica disponível para uso (por transações que atualizam) durante a criação do *dump*. A criação de *dumps* frequentes aumenta a quantidade de tempo em que o sistema de BD não pode ser utilizado por transações de atualização. No entanto, à medida que o tempo entre a geração de *dumps* aumenta, também aumenta o tempo necessário para reconstruir o BD depois da ocorrência de uma falha. Isto se deve ao fato de que o tempo para reconstruir o BD depende do tamanho do diário desde a ultima criação do *dump*.

As seguintes estratégias podem ser utilizadas para geração de um *dump* estático:

1) Copiar sequencialmente todo o BD.
2) Permitir cópias paralelas de partes do BD, tal que muitas partes do BD sejam copiadas ao mesmo tempo.
3) Permitir que apenas algumas partes do BD sejam copiadas a cada vez. Essa estratégia é conhecida como *dumping* incremental e é utilizada para copiar as entidades do BD que foram atualizadas, criando pontos de cheque para essas entidades. Um ponto de cheque representa um estado em que não há nenhuma transação em progresso atualizando a entidade.

Dump Dinâmico. Não é necessário que o sistema de BD não possa estar disponível durante a criação do *dump*.O *dump* pode ser criado dinamicamente, enquanto transações são executadas e estão atualizando o BD. O *dump* criado dinamicamente deve representar um estado consistente do BD em algum instante de tempo. Para isso o *dump* precisa incluir todas as atualizações feitas pelas transações que tenham terminado em um tempo t, e nenhuma atualização feita por transações que terminaram depois daquele tempo t [2].

Existem tres tipos de métodos de geração de *dumps* dinâmicos [2]:

1) *Dump* da versão inicial: o *dump* criado representa o BD que existia no começo do processo de geração do *dump*.
2) *Dump* da versão final: o *dump* criado representa o BD que existe no momento do término do processo de geração do *dump*.
3) *Dump* da versão intermediária: o *dump* criado representa a versão do BD que existia em algum tempo t durante a execução do processo de geração do *dump*.

Recuperar um banco de dados muito grande, a partir de um *dump* pode levar muito tempo. Neste caso, é conveniente recarregar ape-

nas as porções relevantes do BD. A fim de preservar a consistência interna do BD a escolha das porções a serem recarregadas deve levar em conta as transações que atualizam o BD. Um algorítmo que permite selecionar as porções do BD que devem ser recarregadas a fim de obter um BD num estado consistente está apresentado em [9].

Trilha Auditora. A trilha auditora, também conhecida como *log* ou diário, é um arquivo sequencial que contem um histórico das atividades desenvolvidas no sistema (sequência de ações executadas nas entidades).

A trilha auditora pode ser usada para diferentes propósitos, tais como: corrigir erros e reconstruir o BD, desfazer atualizações executadas no BD e garantir a segurança e a integridade do sistema.

As informações mantidas na trilha auditora podem ser classificadas, com relação a sua utilização como técnica de recuperação, em informações relativas a transações, modificações dos dados do BD e pontos de cheque.

1) Informações sobre transações: A trilha auditora precisa manter para cada transação os seguintes dados: número sequencial da transação (identificação única da transação), identificação do terminal fonte, tipo da transação, data e hora do início da transação e o texto da transação.

2) Informações sobre modificações dos dados: A trilha auditora precisa manter as seguintes informações sobre cada atualização efetuada no BD: número sequencial da transação, identificação do terminal fonte, tipo da transação, data e hora do início da transação, valor inicial da(s) entidade(s) que serão atualizadas e valor final da(s) entidade(s) atualizadas.

3) Informações sobre pontos de cheque: Um ponto de cheque representa um estado do sistema em que não há nenhuma transação em progresso. Dentre as informações que compõem um ponto de cheque podemos citar: informações sobre terminais ativos; ponteiros para filas de mensagens por tipos de mensagem, terminal e prioridade; conteúdo das filas em memória principal; tabelas de alocação de recursos e de bloqueios, e informações sobre o BD: alocação de espaço, cabeças de cadeias, posições de arquivos sequenciais, etc...

É interessante salientar que os sistemas que utilizam essa técnica de recuperação devem implementar o "PROTOCOLO DA GRAVAÇÃO ADIANTADA DA TRILHA AUDITORA" [4]: o registro da trilha que contem

as informações relativas às modificações feitas em uma entidade do BD, deve ser escrito no arquivo sequencial que contem as informações da trilha (geralmente fita) antes que a entidade seja escrita em memória secundária (geralmente disco). Isto se deve ao fato que a memória principal (memória de semicondutores) é volátil enquanto a memória secundaria é não-volátil. Assim, se a entidade é escrita em memória não-volátil antes do registro da trilha auditora correspondente, uma falha pode tornar impossível desfazer uma ação.

Gray [4] salienta também, que para sistemas que necessitam de grande confiabilidade, é importante duplicar a gravação da trilha auditora, isto é, manter dois arquivos sequenciais em unidades de entrada/saída distintas. Se a trilha auditora não é duplicada, uma falha de memória secundária na unidade da trilha pode ocasionar a perda de alguns dados de recuperação.

Cópia da Versão Corrente. Essa técnica consiste em manter cópias do BD a fim de poder realizar uma possível restauração do BD para um estado prévio. Nessa técnica as transações são combinadas com a versão corrente do BD para produzir uma nova versão. A partir daí, a versão anterior do BD e as transações que acessaram os dados dessa versão são mantidos como cópias. No caso da ocorrência de uma falha, o estado atual do BD pode ser recuperado através da cópia (versão anterior) e das transações que acessaram essa cópia.

A técnica de *dump* pode ser utilizada para manter uma versão copiada do BD. Cópias de arquivos de alterações podem ser usadas para atualizar a versão copiada e obter a nova versão.

Duplicação. A técnica de duplicação consiste em aplicar cada atualização ao BD, paralelamente a duas cópias do mesmo. A implementação dessa técnica é feita através da manutenção de duas cópias com *flags* para indicar "atualização em progresso". Uma cópia inconsistente é sempre reconhecida como inconsistente por causa dos *flags* utilizados.

Com a técnica de duplicação, as duas cópias do BD sempre possuem o mesmo valor exceto durante a atualização dos dados. Quando uma das cópias está sendo atualizada, o *flag* associado a essa cópia é setado para indicar que a atualização dessa cópia está em progresso. Se o sistema falha durante a atualização o *flag* continuará setado após a ocorrência da falha. Nesse sentido, a técnica

de duplicação tem a característica de prover resistência a falhas. Uma cópia consistente sempre pode ser recuperada depois do reinício das atividades do sistema após a ocorrência de uma falha. Essa cópia terá, ou o valor que ela tinha antes da atualização que estava em progresso durante a falha, ou o novo valor. Sendo assim, em caso de falha de uma das duas cópias a outra poderá manter o sistema em funcionamento. A cópia inconsistente pode sempre ser reconhecida e ignorada.

Essa técnica de duplicação é utilizada em sistemas que exigem uma alta disponibilidade, tais como sistemas de reservas de passagens aéreas.

Arquivos Diferenciais. Na técnica de arquivos diferenciais, os arquivos principais se mantêm inalterados até a reorganização dos mesmos. Todas as atualizações que seriam feitas em um arquivo principal são registradas em um arquivo diferencial [5]. O arquivo diferencial será sempre pesquisado primeiro. Se o dado não for encontrado no arquivo diferencial ele deve ser pesquisado no arquivo principal.

Essa técnica apresenta as seguintes desvantagens [6]:

a) requer dois acessos ao BD por consulta.
b) eventualmente o arquivo diferencial deve ser fundido com o arquivo principal (uma operação que pode ser lenta)
c) tendo em vista que uma atualização pode afetar um dado já modificado, os arquivos diferenciais devem ser convenientemente organizados.

A fim de reduzir o número de acessos ao arquivo diferencial, Severance e Lohman [5] propõem uma técnica de filtros, descrita a seguir.

Elementos do filtro:

. Vetor de bits; B[1...M]
  Quando o arquivo diferencial está vazio, todos os bits de B são iguais a zero.
. X funções de *hash*
  Cada uma destas funções gera, a partir de uma identificação de um registro, um endereço no vetor B.

A figura 1 ilustra o funcionamento do filtro no armazenamento de um registro no arquivo diferencial.

Aplica-se X funções de *hash* à identificação do registro. Obtem-se os endereços bi1, bi2,...bix. Faz-se B[bij]=1 para j=1,2,....,X.

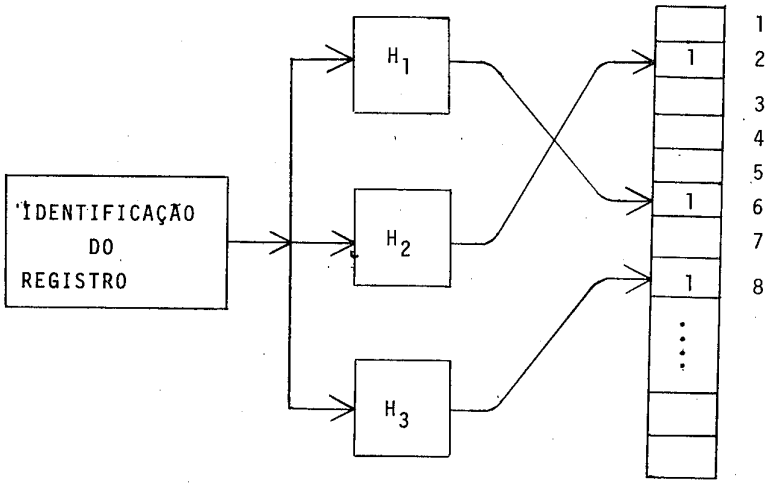


Figura 1 - Filtro com Funções *HASH*

Ao recuperar-se um registro R, pode-se de imediato saber se o registro não está no arquivo diferencial. Para isto, aplica-se as X funções à identificação de R e faz-se a operação de conjunção lógica (*AND*) com os X bits resultantes de B. Se o resultado for Ø o registro não está no arquivo diferencial e se for 1 talvez esteja .

Assim, a recuperação de um registro R é ilustrada pelo diagrama da figura 2.

Severance [5] descreve detalhadamente as vantagens da técnica de arquivos diferenciais. Dentre elas podemos citar: redução do custo de *dumping*, facilita *dumping* incremental, permite *dumping* dinâmico, acelera recuperação de perda de dados e simplifica o desenvolvimento de *software*.

Substituição Cuidadosa. O objetivo da substituição cuidadosa é permitir a atualização das estruturas de dados "no local". A atualização é executada em uma cópia da entidade que está sendo modificada, a qual substituirá a entidade original somente no caso da atualização ter sido executada com sucesso. A cópia é mantida até que a substituição de entidade tenha ocorrido completamente.

Na técnica da substituição cuidadosa as duas cópias "virtuais" da mesma entidade, só são mantidas durante a atualização. Isso torna a atualização ou a sequência de atualizações tão confiável quanto possível. Essa técnica elimina as desvantagens da técnica de

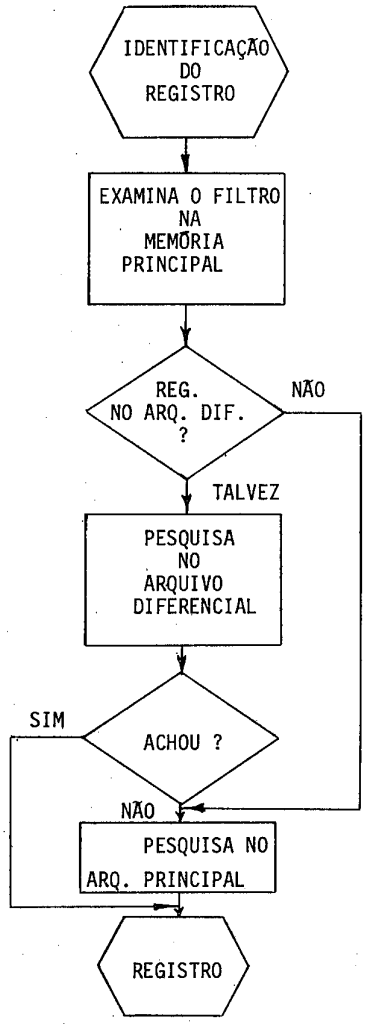


Figura 2 - Recuperação de um Registro

arquivos diferenciais mencionadas no item anterior. Entretanto ela implica em custos adicionais decorrentes de um *overhead* de acessos a disco. Por esta razão os arquivos e as estruturas de dados necessárias para a recuperação devem ser cuidadosamente projetadas a fim de não comprometer o desempenho do sistema.

Antes de descrevermos esta técnica, vamos introduzir o conceito de lista de intenções [7]. Uma lista de intenções é a lista de ações necessárias para efetivar os comandos de atualização de uma transação. Esta lista é armazenada em uma ou mais páginas físicas com propriedade atômica forte, ou seja, no momento da escrita de uma página física que armazena parte de uma lista de intenções associa-se duas páginas físicas livres, digamos i e j. A página é escrita na página física i e caso esta escrita ocorra sem erros, a página física i é copiada sobre a página física j. A lista de intenções deve ser idempotente, isto é, o resultado da execução ininterrupta da lista de intenções é equivalente ao resultado da execução da concatenação de listas parciais com uma lista completa. Assim, se L = A1, A2,A3,A4 é uma lista de intenções, temos que:

A1,A2,A3,A4 <=> A1,A2,A3,A1,A2,A1,A1,A2,A3,A4.

A fim de que as operações ou ações que compõem uma lista de intenções possam ser repetidas um número qualquer de vezes sem que o resultado da execução da lista de intenções seja alterado, é sufi-

ciente que cada ação seja do tipo:ESCREVA UM VALOR V NO ENDEREÇO E. De uma maneira geral os seguintes passos devem ser seguidos para executar uma transação:

P1: para cada transação escreva páginas de dados modificadas usando páginas livres, à medida que comandos de atualização são recebidos.

P2: ao término da transação

P2.1: crie a lista de intenções

P2.2: escreva a lista de intenções em memória estável com propriedade atômica forte

P2.3: execute a lista de intenções

P2.4: apague a lista de intenções

Observe que após o passo P2.2 a transação pode ser recuperada mesmo que ocorram falhas. Note também, que até a execução do passo P2.3 o BD está refletindo o estado imediatamente anterior ao início da transação.

Em um ambiente de banco de dados distribuído cada nó da rede só executa uma lista de intenções relativa a uma determinada transação se e somente se todos os nós envolvidos com essa transação já tiverem criado suas listas de intenções e se já tiverem escrito essas listas em memória estável com propriedade atômica forte. Seguindo-se essa estratégia garante-se a consistência do banco de dados distribuido, ou seja, ou todas ou nenhuma atualização será executada nos nós da rede envolvidos com a transação, fazendo com que todas as cópias dos dados distribuídas pela rede possuam o mesmo valor.

Para Lampson e Sturgis [7] o BD consiste de um conjunto de arquivos. Um arquivo é composto de uma página de ponteiros e de um conjunto de páginas de dados. A página de ponteiros contém os endereços das páginas de dados que compõem o arquivo.

Os arquivos são mantidos em memória secundária. A memória estável está dividida em blocos de tamanho fixo chamados de páginas que são unidade de alocação e transferência entre a memória principal e a memória secundária. Diz-se que uma memória tem propriedade atômica forte quando a transferência de uma página da memória principal para a memória secundária resulta em um dos dois possíveis resultados:

1. a página não é transferida.
2. a página é transferida com sucesso.

Cada página física armazena o conteúdo de alguma página lógica (cada página física tem o mesmo tamanho que uma página lógica). A alocação das páginas pode ser estática ou dinâmica. A alocação estática significa que uma página lógica sempre é armazenada na mesma página física, mesmo após a ocorrência de alguma modificação da página lógica. Já, a alocação dinâmica, significa que uma página lógica é armazenada em qualquer página física do sistema. Neste caso, existe um ponteiro (armazenado em alguma física) que mapeia o armazenamento desta página lógica na página física.

Para conseguir-se implementar memória com propriedade atômica forte, dependendo do tipo de alocação, existem duas estratégias:

a. alocação estática: em tempo de atualização de uma página, associa-se uma nova página física que armazenará o conteúdo modificado da página física original. Se a escrita desta nova página ocorrer sem erros, então o valor desta página é copiado sobre a página original.

b. alocação dinâmica: em tempo de atualização de uma página que está sendo referenciada por algum ponteiro, associa-se uma nova página física que armazenará o conteúdo modificado da página física original. Se a escrita desta nova página ocorrer sem erros, então o valor do ponteiro é modificado para refletir a nova situação.

Em ambos os casos, consegue-se a propriedade atômica forte, associando-se, durante a atualização, duas página físicas a cada página lógica. A figura 3 ilustra as duas estratégias descritas acima.

Na figura 3.a, a página lógica i está sempre armazenada na página física j. Durante a atualização utiliza-se uma nova página física k para criar a nova versão da página lógica i. Se a escrita da página física k ocorrer sem erros, a página física k é copiada sobre a página física original (página física j). O efeito de uma falha sobre os possíveis valores das páginas físicas j e k pode ser um dos seguintes:

1. ambas as páginas contêm seus valores originais (a falha ocorreu antes de escrever qualquer uma das páginas).
2. a página k é detetada como incorreta e a outra possui o valor original (a falha ocorreu durante a escrita da página k).
3. nenhuma página contêm erro, mas a página j contêm o valor

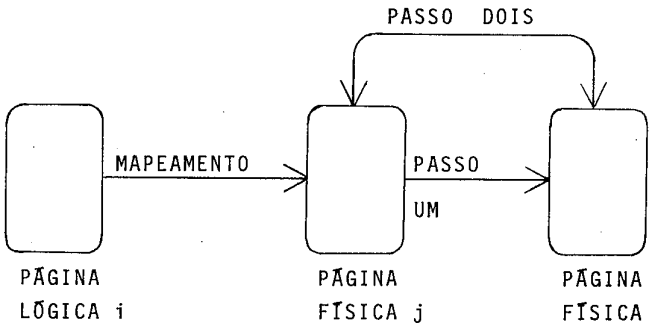


Figura 3a - Implementação da Propriedade Atômica Forte para Alocação Estática

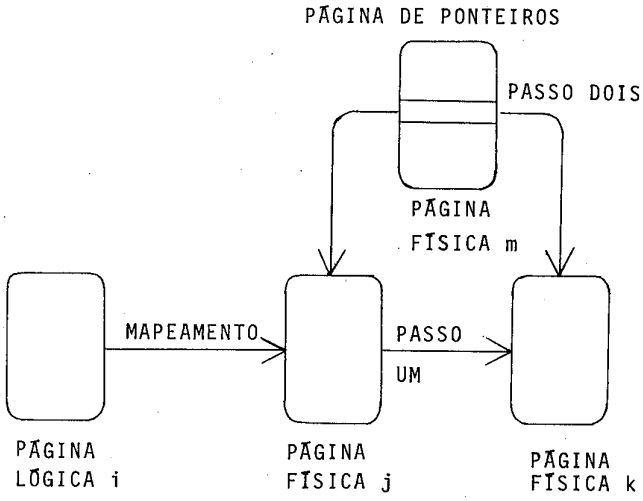


Figura 3b - Implementação da Propriedade Atômica Forte para Alocação Dinâmica

original e a outra contém o novo valor da página lógica i (a falha ocorreu imediatamente após a escrita da página física k).

4. a página física k contém o novo valor e a outra contém um valor incorreto (a falha ocorreu durante a escrita da página física j).
5. ambas as páginas contém o novo valor da página lógica i (a falha ocorreu depois da escrita da página física j).

Nos casos 1 e 2, o algorítmo de recuperação recomeçará o pro -

cesso de atualização a partir do primeiro passo. Nos casos 3 e 4, o algorítmo recomeçará o processo a partir do segundo passo. No caso 5, o processo de atualização já havia concluído quando a falha ocorreu (nada é feito).

Na figura 3.b, a página lógica i está sendo armazenada na página física j. Existe uma página de ponteiros que indica este mapeamento. Durante a atualização utiliza-se uma página física k para criar a nova versão da página lógica i. Se a escrita da página física k ocorrer sem erros, a página física m (página de ponteiros) é atualizada para refletir o novo mapeamento.

Se a página de ponteiros é dinamicamente alocada, isto é, a página é referenciada por algum ponteiro armazenado em alguma página física, devemos repetir o processo de atualização descrito na parte b) da figura 3. Esta recursividade termina quando encontra-se uma página fixa. Utiliza-se então, a primeira estratégia de atualização.

Idéias semelhantes às acima mencionadas foram implementadas no sistema R [6].

3. CONCLUSÃO

Para finalisar este trabalho apresentaremos a seguir algumas relações que podem ser obtidas entre as diferentes técnicas a partir da descrição realizada.

A figura 4 mostra referência cruzada entre as 7 diferentes categorias que agrupam as técnicas de recuperação e os diferentes tipos de recuperação [1].

Eis aqui algumas relações aparentes:

- A técnica de arquivos diferenciais torna muito fácil a implementação de *dump* incremental.
- A trilha auditora é uma alternativa para as técnicas de arquivos diferenciais, substituição cuidadosa ou duplicação.
- As técnicas de duplicação e substituição cuidadosa podem ser usadas alternativamente ou como complementares. Ambas provêm resistência contra falhas do mesmo tipo.
- As técnicas de *dump*, da trilha auditora, da cópia e versão corrente e de arquivos diferenciais podem ser uitilizadas para prover recuperação de falhas particulares ou para complementarem umas as outras a fim de recuperar o BD após a ocorrência de uma falha.
- O programa restaurador como técnica de recuperação é um úl-

	ESTADO CORRETO	ESTADO CORRETO ANTERIOR	POSSÍVEL ESTADO ANTERIOR	ESTADO VÁLIDO	ESTADO CONSIS-TENTE	RESISTENCIA CONTRA FALHAS
PROGRAMA RESTAURADOR				*	*	
DUMP			*	*		
TRILHA AUDITORA	*	*	*			
CÓPIA E VERSÃO CORRENTE		*	*			
DUPLICAÇÃO						*
ARQUIVOS DIFERENCIAIS		*	*			*
SUBSTITUIÇÃO CUIDADOSA		*				*

Figura 4 - Referência Cruzada Indicando quais os Propósitos das Diferentes Técnicas de Recuperação

timo recurso, usado quando todas as outras técnicas falham. O programa restaurador não pode trazer o BD a um estado prévio. Ele meramente "recolhe" o que restou após a ocorrência de uma falha.

4. REFERENCIAS

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

[2] ROSENKRANTZ, D.J., "Dynamic Database Dumping", *ACM/SIGMOD International Conference on the Management of Data* 31 de maio a 2 de junho de 1978, Austin, Texas.

[3] MENASCÉ, D.A., MUNTZ, R.R., "Locking and Deadlock Detection in Distributed Databases", *IEEE Transactions on Software Engineering*, Maio 1979, pags. 195-201.

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

[5] SEVERANCE, D.N., LOHMAN, G.M., "Differential Files: Their Application to the Maintenance of Large Databases", *ACM Transactions on Database Systems*, Vol 1, No 3, Setembro 1976.

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

[7] LAMPSON, B. STURGIS, H.E., "Crash Recovery in a Distributed Data Storage System", Xerox Palo Alto Research Center, Palo Alto, Califórnia, USA, 1978 (a ser publicado na CACM).

[8] DAVENPORT, R.A., "On-line Data Base Integrity", London School of Economics e CACI Inc-International, 1977.

[9] MENASCÉ, D.A., "Selective Reloading of Very Large Data-bases,"*IEEE Third International Computer Software and Applications Conference*, November de 1979, Chicago, U.S.A.