VICENTE DE PAULA E SILVA NETO

UM ESTUDO TEÓRICO SOBRE REPLICAÇÃO EM SISTEMAS DE BANCOS DE DADOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS - BRASIL
2005

VICENTE DE PAULA E SILVA NETO

UM ESTUDO TEÓRICO SOBRE REPLICAÇÃO EM SISTEMAS DE BANCOS DE DADOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de concentração:

Banco de Dados

Orientadora:

Olinda Nogueira Paes Cardoso

LAVRAS
MINAS GERAIS - BRASIL
2005

Ficha Catalográfica preparada pela Divisão de Processos Técnico da Biblioteca Central da UFLA

Neto, Vicente de Paula e Silva

Um estudo teórico sobre Replicação em Sistemas de Bancos de Dados / Vicente de Paula e Silva Neto Lavras – Minas Gerais, 2005. 85p : il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Informática. 2. Banco de Dados. 3. Replicação. I. NETO, V. P. S. II. Universidade Federal de Lavras. III. Título.

VICENTE DE PAULA E SILVA NETO

UM ESTUDO TEÓRICO SOBRE REPLICAÇÃO EM SISTEMAS DE BANCOS DE DADOS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 7 de Julho de 2005

_____

Prof. Dr. Guilherme Bastos Alvarenga

_____

Prof. Mário Luiz Rodrigues Oliveira

_____

Prof. Olinda Nogueira Paes Cardoso

(Orientadora)

LAVRAS
MINAS GERAIS - BRASIL

Resumo

A necessidade da disponibilidade de informações em ambiente compartilhado torna importante que os dados não estejam apenas num computador central, armazenando dados críticos em diferentes *sites* (diferentes locais ou nós de uma rede) um sistema ficaria disponível mesmo se alguns falharem. Outra razão é aumentar a performance, já que existe mais de uma cópia de cada item de dados, uma transação pode encontrar o dado que ela precisa em um local mais próximo. Neste trabalho será apresentado um estudo teórico sobre técnicas de replicação de dados, abordando algumas técnicas de replicação imediata e alguns algoritmos utilizando o esquema de replicação atrasada, objetivando ressaltar as vantagens e os custos de cada método.

Abstract

The necessity of the availability of information in shared environment becomes important that the data are not only in a central computer, storing given critical in different sites (different places or us of a net) a system would be available same if some to fail. Another reason is to increase the performance, since a copy of each item of data exists more than, a transaction can find the data that it needs in a next place. In this work a theoretical study on techniques of response of data will be presented, approaching some techniques of immediate response and some algorithms using the project of delayed response, objectifying to stand out the advantages and the costs of each method.

Sumário

Sumário vi
Lista de Figuras vii
1. Introdução 1
2. Referencial Teórico 2
2.1. Arquitetura de Sistemas Distribuídos 2
2.2. Modelos de Replicação 3
2.3. Algoritmos de Replicação Imediata 6
2.3.1. Técnica Write-all 6
2.3.2. Técnica Write-all-Available 7
2.3.3. Um Algoritmo de Cópias Disponíveis 7
2.3.4. Algoritmo de Cópias Disponíveis Orientado a Diretório 8
2.3.5. Algoritmo Usando Consenso do Quorum 9
2.3.6. Algoritmo de Partição Virtual 11
2.3.7. Algoritmo de Replicação de Dados Adaptativa 13
2.4. Algoritmos de Replicação Atrasada 17
2.4.1. Protocolos de Acesso Multivisão para Replicação em Larga Escala 17
2.4.2. Replicação Two-Tier 27
2.4.3. Protocolos Usando Grafos de Replicação 30
3. Conclusões 34
4. Referencial Bibliográfico 35

Lista de Figuras

Figura 2.1 – Replicação imediata e atrasada 4
Figura 2.2 – Atualização em grupo ou feita pelo mestre 5
Figura 2.3 – Exemplo de uma rede com dados replicados. 14
Figura 2.4 – Exemplo de hierarquia de *clusters* 22
Figura 2.5 – Exemplo de estrutura de cluster 24
Figura 2.6 – Diagrama de transição de estados das transações 31

1.Introdução

A principal razão para utilizar réplicas é aumentar a disponibilidade do sistema de banco de dados. Armazenando dados críticos em múltiplos *sites*, o sistema fica disponível mesmo se alguns deles estiverem em falha.

Outra razão é aumentar o desempenho. Já que existem muitas cópias de cada item de dados, uma transação pode encontrar o dado que ela precisa em um *site* mais próximo, quando comparado a um banco de dados com uma única cópia da informação.

A replicação de dados tem provado ser uma técnica útil em muitas aplicações aonde o acesso rápido e confiável é imperativo. Porém esses benefícios vêm em conjunto com a necessidade de atualizar todas as cópias dos itens de dados. Com isso, operações de leitura podem ser executadas mais rapidamente, mas operações de escrita são executadas mais lentamente.

Um banco de dados replicado é um banco de dados distribuído [CaMo85] no qual cópias múltiplas de alguns itens de dados são armazenadas em *sites* diferentes, sendo que o sistema de banco de dados deve esconder os aspectos de replicação de dados das transações dos usuários [BeHG87].

Neste trabalho será apresentado um estudo teórico sobre técnicas de replicação de dados, abordando algumas técnicas de replicação imediata e alguns algoritmos utilizando o esquema de replicação atrasada, objetivando ressaltar as vantagens e os custos de cada método.

2. Referencial Teórico

2.1. Arquitetura de Sistemas Distribuídos

Considere um banco de dados distribuído, onde cada *site* possui um Gerente de Dados (GD) e um Gerente de Transações (GT) que gerencia dados e transações no *site*.

O GD processa operações de leitura e escrita em cópias locais dos itens de dados. Ele tem um escalonador associado para controle de concorrência, baseado em uma das técnicas conhecidas como *Two-Phase Locking* (2PL), *Timestamp Ordering* (TO) ou *Serialization Graph Testing* (SGT) [BeHG87]. O escalonador em um site é sensível a conflitos em operações na mesma cópia de um item de dado.

O GT é a interface entre as transações de usuários e o Sistema de Banco de Dados (SBD). Ele traduz operações de leitura e escrita em itens de dados em operações em cópias desses itens de dados. Ele envia essas operações às cópias nos sites apropriados, onde são processadas pelos escalonadores e GDs locais.

O GT também utiliza um protocolo de *commit* atômico (*Atomic Commitment Protocol – ACP*) para terminar consistentemente uma transação que acessou dados em mais de um site. Assume-se que o GD e o escalonador em cada site participam desse ACP para qualquer transação ativa no site.

Para realizar essas funções, o GT deve determinar quais sites possuem cópias de cada item de dado. Ele usa diretórios para esse propósito. Pode haver apenas um diretório que informa onde todas as cópias dos itens de dados estão armazenadas, ou alternativamente, cada diretório pode informar somente a localização das cópias de alguns itens de dados.

Nesse caso, cada diretório é normalmente armazenado somente em sites que acessam freqüentemente as informações que ele contém. Para encontrar os diretórios restantes, o GT precisa de um diretório mestre que informe onde as cópias de cada diretório estão localizadas.

2.2. Modelos de Replicação

Quando uma transação solicita uma operação de escrita, o SBD é responsável por atualizar um conjunto de cópias de *X* (o conteúdo do conjunto depende do algoritmo utilizado para gerenciar dados replicados). O SBD pode distribuir as operações de escrita imediatamente ou pode deferir as operações de escrita em cópias replicadas até que a transação termine. Com isso, existem dois modelos básicos de replicação: replicação imediata (*eager replication*) e replicação atrasada (*lazy replication*) [BeHG87, GHOS96]. A Figura 1 mostra as duas formas de propagar atualizações para as réplicas. A replicação imediata mantém todas as réplicas exatamente sincronizadas em todos os nós atualizando todas elas como parte da transação original. Esse tipo de replicação fornece execução serializável, pois não existem anomalias de concorrência. Entretanto, a replicação imediata reduz a performance de atualização e aumenta o tempo de resposta das transações porque atualizações e mensagens extras são acrescentadas à transação.

A replicação imediata não é uma boa opção para aplicações móveis, onde a maioria dos nós está desconectada. As aplicações móveis requerem algoritmos de replicação atrasada que propagam as atualizações em réplicas de maneira assíncrona para os outros nós depois que a transação de atualização é comprometida (commit). Com isso, uma réplica é atualizada pela transação original e as atualizações em outras réplicas são propagadas assincronamente, normalmente com uma transação separada para cada nó. Alguns sistemas conectados permanentemente utilizam a replicação atrasada para aumentar o tempo de resposta.

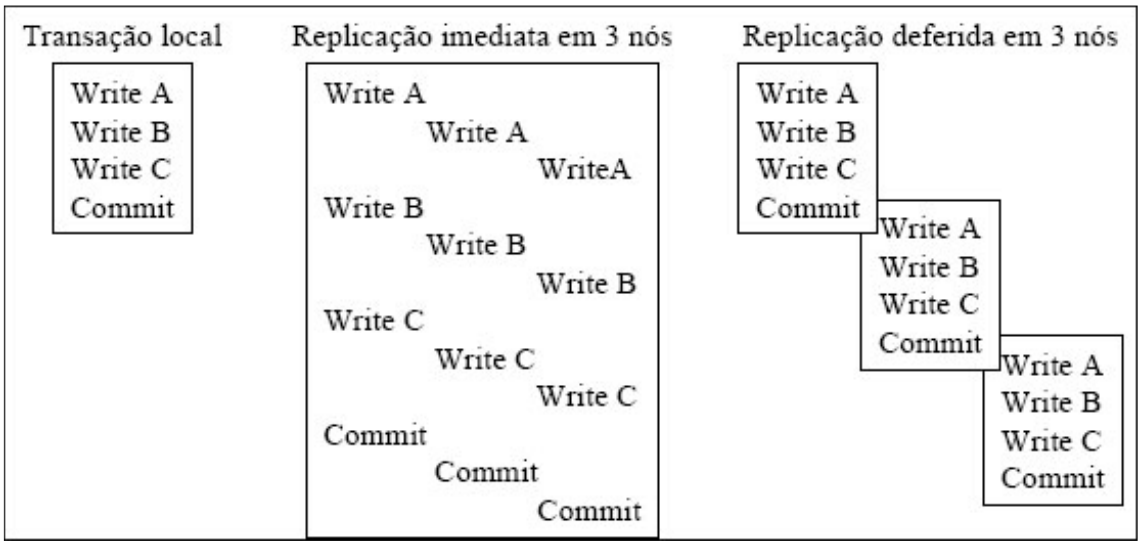

Figura 2.1 – Replicação imediata e atrasada

Com escrita atrasada, o SBD usa uma visão não replicada do banco de dados enquanto a transação executa. Isto é, para cada item de dado que a transação lê ou escreve, somente uma cópia do dado é acessada pelo SBD. Diferentes transações podem utilizar diferentes cópias. O SBD atrasa a distribuição das operações de escrita nas outras cópias até que a transação termine e esteja pronta para comprometer. Com isso, pode utilizar a votação da primeira fase do protocolo de comprometimento para enviar as operações para os sites correspondentes.

Uma desvantagem do método deferido é que ele pode atrasar o comprometimento de uma transação, pois a primeira fase do protocolo de comprometimento deve processar um número potencialmente grande de operações de escrita antes de responder a mensagem de votação. Com a técnica imediata os sites participantes podem processar operações de escrita enquanto a transação ainda está executando, evitando o atraso no momento do comprometimento.

A replicação imediata normalmente utiliza um esquema de bloqueios para detectar e controlar a execução concorrente. A replicação atrasada normalmente usa um esquema de concorrência com múltiplas versões para detectar comportamento não-serializável [BeHG87]. Com isso, a replicação atrasada pode permitir que uma transação acesse um valor desatualizado.

Outra desvantagem do método deferido é que ele tende a atrasar a detecção de conflitos entre as operações, o que só acontece no fim da execução das transações. Por exemplo, duas transações T_1 e T_2 executando concorrentemente e ambas escrevendo em *X*. Suponha que o

SBD utilize a cópia X_A para executar T_1 e a cópia X_B para executar T_2. Até que o SBD distribua a operação de escrita de T_1 para as outras cópias de x ou a operação de escrita de T_2, nenhum escalonador detecta o conflito.

Essa desvantagem do método deferido pode ser resolvida com o uso de uma mesma cópia do item de dados, chamada de cópia primária, para executar todas as transações. Nesse caso, tanto o método imediato quanto o deferido detectam o conflito no mesmo ponto de execução da transação.

Trabalhos mais antigos sobre replicação focavam em replicação imediata [BeHG87], enquanto trabalhos mais recentes focam em replicação atrasada [ABKW98].

As atualizações podem ser controladas de duas formas: atualização em grupo (*group*) ou atualização feita pelo mestre (*master*). A Figura 2.2 ilustra ambas as formas.

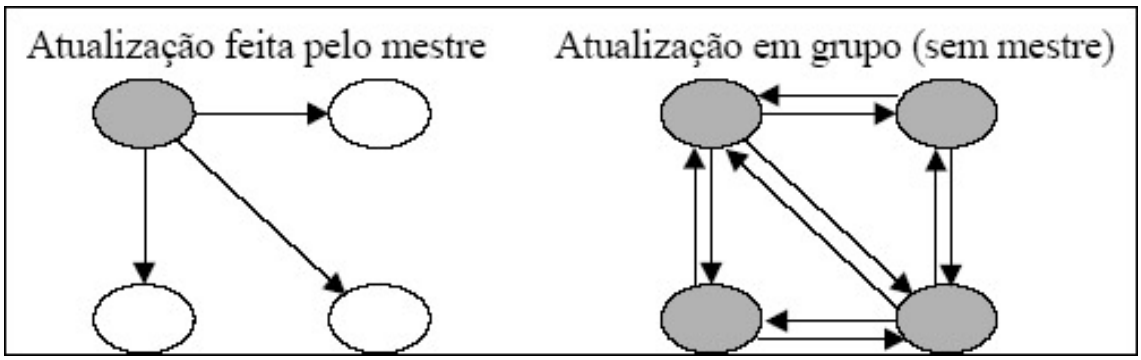

Figura 2.2 – Atualização em grupo ou feita pelo mestre

Na atualização em grupo, qualquer nó que possua uma cópia do item de dado pode atualizá-la. Isso é chamado de atualização em qualquer lugar (*update anywhere*). Esse tipo de atualização tem mais chance de ocasionar conflitos.

Na atualização feita pelo mestre, cada objeto tem um nó mestre e somente ele pode atualizar a cópia primária do objeto. Todas as outras réplicas são para leitura somente. Outros nós querendo atualizar o objeto devem requisitar a atualização ao mestre.

Na replicação atrasada com atualização em grupo (*lazy group replication*), qualquer nó pode atualizar qualquer item de dado local. Quando a transação é comprometida, uma transação é enviada para cada nó para fazer as atualizações da transação raiz em cada cópia do item de dado. É possível que dois nós atualizem o mesmo objeto e enviem suas atualizações para os outros nós. Nesse caso, o mecanismo de replicação deve detectar o problema e reconciliar as transações para que as atualizações não sejam perdidas.

Na replicação atrasada com atualização feita pelo mestre (*lazy master replication*), cada objeto possui o seu dono. As atualizações devem ser feitas pelo dono e depois propagadas para outras réplicas. Diferentes objetos têm diferentes donos. Esse tipo de replicação também não é apropriada para aplicações móveis, pois um nó querendo atualizar um item de dado deve estar conectado e participar em uma transação atômica com o dono do objeto. Da mesma forma que na replicação imediata, o esquema deferido com mestre não possui problemas de reconciliação, ao invés disso, os conflitos são resolvidos por espera ou por deadlocks.

2.3. Algoritmos de Replicação Imediata

Neste seção, serão abordadas algumas técnicas de replicação imediata [BeHG87, WoJH97], onde as operações de escrita nas réplicas são distribuídas de forma síncrona.

2.3.1. Técnica Write-all

Em um mundo ideal onde os sites nunca falham, um SBD pode gerenciar facilmente dados replicados. Ele traduz cada operação de leitura Read(X) em Read(X_A), onde X_A é qualquer cópia de X (X_A denota a cópia de X no site A). E traduz cada operação de escrita Write(X) em {Write(X_{A1}), ..., Write(X_{Am})}, onde { X_{A1}, ..., X_{Am}} são todas as cópias de X. Essa técnica para manipular dados replicados chama-se *write-all*.

Infelizmente, o mundo não é ideal, pois os sites podem falhar e se recuperar. Esse é um problema para a técnica *write-all*, porque requer que o SBD processe cada operação de escrita escrevendo em todas as cópias de um item de dado, mesmo se algumas delas estiverem em falha. Nesse caso, o SBD teria que atrasar o processamento de operações de escrita até que pudesse escrever em todas as cópias do item de dado. Esse atraso é ruim para as transações de atualização, pois quanto maior a replicação dos dados, menor é a disponibilidade do sistema para as transações de atualização. Por essa razão, a técnica de *write-all* não é satisfatória [BeHG87].

2.3.2. Técnica Write-all-Available

Usando a técnica *write-all-available*, o SBD deve escrever em todas as cópias de um item de dados que estejam disponíveis, isto é, ele pode ignorar as cópias em sites que estão em falha. Isso resolve o problema de disponibilidade, mas pode causar problemas com relação à corretude.

Usando essa técnica, existirão situações onde algumas cópias de um item de dado não refletem o seu valor mais atualizado. Com isso, uma transação pode ler uma cópia desatualizada, causando uma execução incorreta. Existe uma série de algoritmos para evitar essas execuções incorretas [BeHG87], alguns deles serão comentados a seguir.

2.3.3. Um Algoritmo de Cópias Disponíveis

Os algoritmos baseados em cópias disponíveis tratam dados replicados usando variações da técnica *write-all-available* [BeHG87]. Isto é, toda operação de leitura é traduzida em uma operação de leitura em qualquer cópia do item de dado e toda operação de escrita é traduzida em operações de escrita em todas as cópias disponíveis do item de dado em questão. Esses algoritmos tratam falhas nos sites, mas não tratam falhas de comunicação.

Para os algoritmos baseados em cópias disponíveis, assume-se que o escalonador usa 2PL, então, depois de um transação T_i ter lido ou escrito em uma cópia de X_A, nenhum outra transação pode acessar X_A de modo conflitante até que T_i tenha sido comprometida ou abortada.

Assume-se que existe um conjunto fixo de cópias para cada item de dados, conhecido por todos os sites e que este conjunto não muda dinamicamente.

Quando uma transação T_i requisita uma operação de leitura em X, o GT do site original de T_i (site onde T_i começou sua execução) seleciona alguma cópia X_A de X e submete uma operação de leitura de X_A para o site A. Tipicamente, a cópia selecionada será a mais próxima ao site original, com o intuito de minimizar o custo de comunicação causado pela operação.

O site A considera X_A como inicializada se ele já processou uma operação de escrita em X_A. Nesse caso, a operação de leitura será processada pelo escalonador e pelo GD do site A. Se X_A foi inicializada, mas a transação T_j que a inicializou ainda não foi comprometida, então,

usando 2PL, o escalonador atrasa a operação de leitura em X_A até que T_j seja comprometida ou aborte.

Se a operação de leitura for rejeitada, um reconhecimento negativo será enviado e T_i será abortada. Se a operação de leitura for aceita, o valor é retornado ao GT do site original de T_i. Entretanto, se o site A estiver fora ou X_A não tiver sido inicializada, o limite de tempo para chegada do reconhecimento da operação no site original de T_i vai ser ultrapassado (*time out*). Nesse caso, o GT pode abortar T_i ou enviar outro pedido de leitura para outro site B que possua outra cópia de *X*. Se alguma das cópias de x puder ser lida, a operação de leitura é bem sucedida, caso contrário, T_i deve ser abortada.

Quando uma transação T_i requisita uma operação de escrita em *X*, seu GT envia operações de escrita a todos os site que mantêm uma cópia de *X*. Depois de enviar as operações de escrita, o GT de T_i aguarda as respostas. Ele pode receber rejeições de alguns sites, respostas de outros (significando que a operação de escrita foi aceitada e realizada) e nenhuma reposta de outros (aqueles que falharam ou então não inicializaram suas cópias de x). Operações de escrita cujas respostas não foram recebidas são chamadas de *missing writes* [BeHG87]. Se qualquer rejeição for recebida ou se todas as operações de escrita forem *missing writes*, a operação de escrita é rejeitada e a transação T_i deve abortar. Caso contrário, a operação de escrita é bem sucedida.

2.3.4. Algoritmo de Cópias Disponíveis Orientado a Diretório

Ao invés da distribuição das cópias nos sites ser estática, como no algoritmo anterior com cópias disponíveis, o algoritmo de cópias disponíveis orientado a diretório [BeHG87] utiliza diretórios para definir o conjunto de sites que armazenam cópias de um item de dado em certo momento. Mais precisamente, para cada item de dado *X*, existe um diretório d(x) que lista o conjunto de cópias de *X*. Como um item de dado, o diretório também pode estar replicado, ou seja, pode ser implementado como um conjunto de cópias de diretórios armazenadas em diferentes sites. O diretórios são atualizados por duas transações especiais, *Include* (IN) para criar novas cópias de itens de dados e *Exclude* (EX) para destruir cópias de itens de dados.

O diretório de X em um site U, denotado por $d_U(X)$, contém uma lista de cópias de X e uma lista de cópias de diretórios de X que o site U acredita estarem disponíveis.

Quando um site A se recupera de uma falha ou quando A quer criar uma nova cópia de X, o SBD executa uma transação $IN(X_A)$. $IN(X_A)$ atualiza o valor de X_A da seguinte forma:

- encontrando uma cópia de diretório $dU(X)$, usando uma cópia local que ele sabe que existe, ou lendo um diretório mestre que lista as cópias de $d(X)$ ou perguntando a outros sites;
- lendo $d_U(X)$ para encontrar uma cópia disponível X_B de X;
- lendo X_B;
- copiando o valor de X_B para X_A.

Depois declara X_A como disponível em todas as cópias disponíveis de $d(X)$.

Quando um site entra em falha, os SBDs que tentaram acessar dados nesse site percebem a falha e então executam uma transação EX para cada cópia de item de dado armazenado no site em falha. $EX(X_A)$ declara X_A como não disponível removendo-a de toda cópia disponível de $d(X)$. Isto é, a transação $EX(X_A)$:

- lê alguma cópia de diretório $d_U(X)$;
- remove X_A da lista de cópias disponíveis que leu de $d_U(X)$;
- escreve a lista atualizada em cada cópia de diretório listada em $d_U(X)$.

Para processar uma operação de escrita em X, o SBD lê uma cópia de $d(X)$, $d_U(X)$, e requisita uma operação de escrita para toda cópia X_A que $d_U(X)$ diz estar disponível. Se o SBD descobrir que, na verdade, alguma cópia X_A não está disponível, ele deve abortar a transação T_i que pediu a operação de escrita e executar uma transação $EX(X_A)$. Quando esta transação comitar, ele pode tentar executar a transação T_i novamente.

2.3.5. Algoritmo Usando Consenso do Quorum

No algoritmo usando consenso de quorum (QC – *Quorum Consensus*), atribui-se um peso não-negativo para cada cópia X_A de X. Depois define-se um limiar de leitura RT (*Read Threshold*) e um limiar de escrita WT (*Write Threshold*) para X, tal que ambos 2WT e (RT+WT) são maiores que o peso total de todas as cópias de X. Um quorum de leitura (ou escrita) de X é qualquer conjunto de cópias de X com peso igual a pelo menos RT (ou WT).

Uma observação importante é que todo quorum de escrita de *X* tem pelo menos uma cópia em comum com todo quorum de leitura e todo quorum de escrita de *X*.

Usando esse algoritmo, um GT traduz cada operação de escrita em *X* em um conjunto de operações de escrita em cada cópia de algum quorum de escrita de *X*. Ele traduz cada operação de leitura em um conjunto de operações de leitura em cada cópia de algum quorum de leitura de *X* e retorna a cópia lida que for mais atual.

Para ajudar o GT a descobrir a cópia mais atual, cada cópia recebe um número de versão, que é igual a zero inicialmente. Quando o GT processa uma operação de escrita, ele determina o maior número de versão de todas as cópias em que ele vai escrever, acrescenta um, e atribui a cada cópia em que ele escreve o novo número de versão. Claramente, isso requer a leitura de todas as cópias no quorum de escrita antes de escrever em qualquer uma delas. Isso pode ser feito obrigando uma operação de escrita em uma cópia X_A a retornar o seu número de versão ao GT. O GT pode enviar o novo valor e o novo número de versão de carona no primeiro ciclo de mensagens do protocolo de commit atômico.

O propósito dos quoruns é garantir que operações de leitura e de escrita que acessam o mesmo item de dado também acessem pelo menos uma cópia em comum desse item de dado. Todo par de operações conflitantes será sempre sincronizado por algum escalonador, aquele que controla o acesso à cópia pertencente à interseção dos quoruns.

Uma característica interessante desse algoritmo é que a recuperação de cópias não requer nenhum tratamento especial. Um cópia de *X* que estava em falha e por isso perdeu algumas operações de escrita não vai ter o maior número de versão em nenhum quorum de escrita do qual é membro. Então, depois que ela se recupera, ela não será lida até que tenha sido escrita pelo menos uma vez. Isto é, as transações vão ignorar automaticamente o seu valor até que seja atualizado.

Infelizmente, esse algoritmo também tem desvantagens. Uma transação deve acessar cópias múltiplas de cada item de dado que ela quer ler, até mesmo quando existe uma cópia do dado em seu site original. Em muitas aplicações, as transações lêem mais de que escrevem. Nesse caso, a performance desse algoritmo não é boa.

Outro problema com esse algoritmo é que todas as cópias de um item de dado devem ser conhecidas a princípio. Uma cópia de *X* pode se recuperar de uma falha, mas uma nova cópia de *X* não pode ser criada porque alteraria a definição dos quoruns de *X*. Os pesos dos

sites poderiam ser modificados (e também a definição dos quoruns), mas isso necessitaria de um procedimento de sincronização especial [BeHG87].

2.3.6. Algoritmo de Partição Virtual

O algoritmo de partição virtual (VP – *Virtual Partition*) foi projetado de tal forma que uma transação nunca precise acessar mais de uma cópia de um item de dados para leitura [BeHG87]. Com isso, a cópia disponível que estiver mais perto pode ser usada para leitura.

Como no algoritmo usando consenso de quorum, cada cópia de *X* tem um peso nãonegativo e cada item de dado tem limiares de leitura e escrita (RT e WT respectivamente). Quoruns de leitura e escrita são definidos como antes. Entretanto, eles servem para um propósito diferente nesse algoritmo.

A idéia básica do VP para cada site A é manter uma visão, que consiste de sites com os quais A pode se comunicar. Denota-se esse conjunto por v(A). Como sempre, cada transação T tem um site original (home(T)), onde ela foi iniciada. A visão de uma transação, v(T), é a visão do seu site original v(home(T)), no momento em que T começou sua execução. Se v(home(T)) for modificado antes de T comitar, T será abortada. T pode ser reiniciada e tentar executar considerando a nova visão do site.

Uma transação T executa se a rede consiste somente dos sites em sua visão. Entretanto, para o SBD processar uma operação de leitura ou escrita de T em *X*, v(T) deve conter um quorum de leitura ou escrita de *X*. Se esse não for o caso, o SBD deve abortar T. Note que o SBD pode determinar se ele pode processar operações de leitura ou escrita em *X* a partir de informações disponíveis no site original de T. Ele não precisa tentar acessar um quorum de leitura ou escrita das cópias de *X*.

Dentro da visão na qual T executa, o SBD usa a técnica *write-all*. Isto é, ele traduz uma operação de escrita em *X* em operações de escrita em todas as cópias de *X* em v(T), e traduz uma operação de leitura em *X* em uma operação de leitura em qualquer cópia de *X* em v(T).

Manter essas visões de maneira consistente não é um problema simples, pois falhas nos sites e na comunicação ocorrem espontaneamente. O algoritmo VP usa uma transação especial chamada Atualização de Visão (*View Update*) para superar essa dificuldade, que é requisitada pelo próprio SBD. Essa transação é parecida com as transações IN e EX usadas no algoritmo

de cópias disponíveis orientado a diretório para manter uma visão consistente entre todos os sites das cópias e diretórios que estão disponíveis.

Quando um site detecta uma diferença entre sua visão e o conjunto de sites com os quais pode se comunicar, ele inicia uma transação de Atualização de Visão, cujo propósito é:

- ajustar as visões dos sites para a nova visão;
- atualizar todas as cópias de todos os itens de dados para os quais existe um quorum de leitura na nova visão.

Para ajustar as visões dos sites para a nova visão, as transações de Atualização de Visão devem ser coordenadas e para isso, o algoritmo VP usa um protocolo de formação de visão. Associado a cada visão, existe um identificador de visão (VID). Quando um site A deseja formar uma nova visão v, ele gera um VID, chamado novo_VID, maior que o seu VID corrente. Então, o site A inicia um protocolo de duas fases para estabelecer a nova visão:

- O site A envia uma mensagem JOIN-VIEW para cada site em v e espera por reconhecimentos. Essa mensagem contém o novo_VID.
- O recebimento de JOIN-VIEW pelo site B se constitui de um convite para B se juntar à nova visão sendo criada por A. B aceita o convite desde que seu VID seja menor que o novo_VID contido na mensagem; caso contrário, ele rejeita o convite. Aceitar o convite significa enviar um reconhecimento positivo e rejeitá-lo significa enviar um reconhecimento negativo ou nenhum reconhecimento.
- Depois de receber os reconhecimentos, A pode prosseguir de uma das duas formas seguintes. Ele pode abortar a criação da nova visão e pode tentar reiniciar o protocolo usando um novo_VID maior ainda para tentar convencer mais sites a aceitarem o convite. Alternativamente, A pode ignorar qualquer convite rejeitado e prosseguir formando uma nova visão contendo somente os site v' (v' $\subseteq$ v) que aceitaram o convite. Ele faz isso enviando uma mensagem VIEW-FORMED para cada site em v' anexando o conjunto v' à mensagem. Qualquer site que receba a mensagem adota v' como sua nova visão e o novo_VID (que foi recebido na mensagem JOIN-VIEW) como seu VID corrente. O site não precisa reconhecer o recebimento da mensagem VIEW-FORMED.

Para garantir unicidade, VIDs são pares da forma (c,s) onde s é um identificador de site e c é um contador em s que é incrementado cada vez que s tenta criar uma nova visão. Com

isso, VIDs criados por sites diferentes diferem no segundo componente, enquanto VIDs criados pelo mesmo site diferem no primeiro componente. Os pares são comparados de maneira usual: (c,s) < (c',s') se c < c', ou c = c' e s < s'.

Quando uma nova visão é criada, todas as cópias dos itens de dados para os quais existe um quorum de leitura na nova visão devem ser atualizadas. Isto porque, na nova visão, qualquer uma das cópias pode ser lida. A tarefa de atualizar tais cópias também é feita pela transação de Atualização de Visão. Quando um site A cria uma nova visão v, ele lê um quorum de leitura para cada item de dado *X* (para o qual existe um quorum de leitura em v). E então escreve em todas as cópias de *X* em v, usando o valor lido que for mais recente. Números de versões podem ser utilizados para determinar o valor mais recente, como no algoritmo QC.

No algoritmo VP descrito, o conjunto de cópias de cada item de dado é fixo. Adicionar novas cópias para um item de dado *X* altera o quorum de *X*. Por isso, da mesma forma que no algoritmo QC, isso requer um tipo especial de sincronização [BeHG87].

2.3.7. Algoritmo de Replicação de Dados Adaptativa

Em [WoJH97], é proposto um algoritmo de replicação de dados chamado de ADR – *Adaptive Data Replication* – que usa uma abordagem adaptativa, ou seja, ele modifica o esquema de replicação dos objetos (conjunto de sites onde os objetos estão replicados) de acordo com as modificações no padrão de leitura e escrita dos objetos (número de operações de leitura e escrita submetidas por cada site). O algoritmo modifica continuamente o esquema de replicação convergindo para um esquema ótimo.

Normalmente, o esquema de replicação de um banco de dados distribuído é estabelecido de maneira estática quando o banco de dados é projetado. E permanece estático até que o projetista intervenha e modifique o número de réplicas ou sua localização. Se os padrões de leitura e escrita dos objetos são fixos e conhecidos a priori, essa é uma solução razoável. Entretanto, se os padrões de leitura e escrita mudam dinamicamente, de forma imprevisível, um esquema de replicação estático pode significar problemas sérios de performance.

O algoritmo ADR modifica o esquema de replicação dinamicamente conforme mudanças nos padrões de leitura e escrita dos objetos. Essa técnica utiliza a proposta *read-onewrite- all* [BeHG87] e pode ser combinada com 2PL ou outro algoritmo de controle de concorrência para garantir serialização. Quando o padrão de leitura e escrita se torna regular, o algoritmo ADR modifica o esquema de replicação em direção ao caso ótimo e quando o atinge, o algoritmo estabiliza o esquema.

Em [WoJH97], é feita uma descrição do algoritmo ADR para redes com topologia em árvore, estendendo-a para o caso de topologia genérica em grafo.

Uma operação de leitura é executada pela réplica mais próxima na rede e uma operação de escrita é propagada ao longo dos arcos da subárvore que contém o escritor e os sites que fazem parte do esquema de replicação. Por exemplo, considere a rede de comunicação da Figura 2.3 e suponha que o esquema de replicação do item de dado *X* engloba os nós 3, 7 e 8 (R = {3,7,8}). Quando o nó 2 envia uma operação de escrita em *X*, ela é enviada para o nó 1, que por sua vez envia ao nó 3, que por sua vez envia aos nós 7 e 8 simultaneamente.

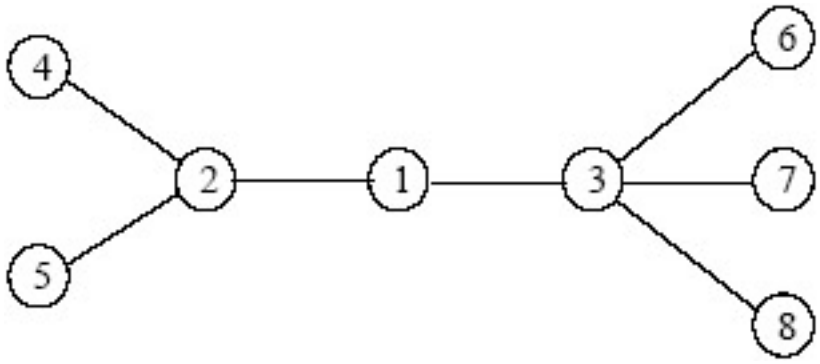

Figura 2.3 – Exemplo de uma rede com dados replicados.

Considere um esquema de replicação R. Um site A é um *R-neighbor* se ele pertence a R, mas possui um vizinho B que não pertence a R. B envia pedidos de leitura e escrita ao site A, originados em B ou em outro site C, tal que o menor caminho entre C e A seja através de B. Na Figura 3, se R = {3,7,8}, o site 3 é o *R-neighbor*, pois possui os sites vizinhos 1 e 6 que não pertencem a R.

Cada site executando o algoritmo ADR recebe a priori um parâmetro t que significa um período de tempo (por exemplo, 5 minutos), ou ainda pode significar um número máximo de operações de leitura e escrita. Modificações no esquema de replicação são executadas por

alguns sites componentes do esquema de replicação no final de cada período t ou quando o número máximo de operações de leitura e escrita for atingido.

A necessidade de mudanças no esquema de replicação é determinada de acordo com três testes denominados: teste de expansão, teste de contração e teste de troca.

O teste de expansão é executado por cada site *R-neighbor*. Se R não é um conjunto unitário, define-se um site *R-fringe* com o sendo uma folha do subgrafo induzido por R. Na Figura 3, se R = {1,3,7,8}, os sites 1, 7 e 8 são *R-fringe*. Cada site *R-fringe* executa o teste da contração. Se um site for tanto *R-neighbor* como *R-fringe*, ele executa primeiro o teste de expansão, e se esse teste falhar, ele executa o teste de contração. Se R for um conjunto unitário, o site pertencente a R executa o teste de expansão, e se esse teste falhar, ele executa o teste da troca.

Os testes funcionam da seguinte forma:

- *Teste da Expansão* (executado por cada site *R-neighbor* i) – para cada vizinho j não pertencente a R, compare dois inteiros x e y. x é o número de operações de leitura que i recebeu de j durante o último período t; y é o número total de operações de escrita que i recebeu no último período t dele mesmo, ou de outro vizinho diferente de j. Se x > y, então i envia a j uma cópia do objeto com uma indicação para salvar a cópia em seu banco de dados local. Com isso, j é inserido em R. O teste é bem sucedido se algum vizinho for incluído em R, ou falha, caso contrário.
- *Teste da Contração* (executado por cada site *R-fringe* i que possui exatamente um vizinho j pertencente a R) – compare dois inteiros x e y. x é o número de operações de escrita que i recebeu de j durante o último período t; y é o número total de operações de leitura que i recebeu no último período t dele mesmo, ou de outro vizinho diferente de j. Se x > y, então i pede permissão a j para ser excluído de R, isto é, para deixar de manter uma cópia do objeto. Se o pedido for aceito, j não envia mais operações de escrita para i e operações de leitura que chegam em i são enviadas a j.
- *Teste da Troca* (executado pelo único site i pertencente a R) – para cada vizinho n compare dois inteiros x e y. x é o número de operações que i recebeu de n durante o último período t; y é o número de todas as outras operações que i recebeu durante o último período t. Se x >

y, então i envia uma cópia do objeto para n com uma indicação de que se torna o novo único site pertencente a R, e i descarta sua cópia.

O algoritmo ADR requer que cada site do esquema de replicação saiba se é um *Rneighbor* ou um *R-fringe* ou o único site pertencente ao esquema. Saber isso, significa que cada site conhece a identidade dos seus vizinhos e sabe para cada vizinho se ele faz parte do esquema de replicação. Os sites que não pertencem ao esquema de replicação não participam do algoritmo e nem sites que não sejam *R-fringe* e que cujos vizinhos possuem uma réplica.

Cada site j mantém um registro de diretório para cada objeto. O registro indica se j faz parte do esquema de replicação; se fizer, então o registro também indica quais dos vizinhos de j estão em R; se não fizer, o registro indica a direção de R. Quando j expande, contrai ou troca, ele executa uma transação de mudança de esquema de replicação que modifica o registro de diretório para refletir as mudanças ocorridas [WoJH97].

Exemplo:

Suponha a rede da Figura 2.3 com R = {1}. Suponha que cada site, exceto o site 8, solicite 4 operações de leitura e 2 operações de escrita em cada período de tempo. O site 8 solicita 20 operações de leitura e 12 de escrita em cada período de tempo.

No fim do primeiro período de tempo, o site 1 (como um *R-neighbor* dos sites 2 e 3) executa o teste de expansão. Como o número de operações de leitura solicitadas pelo site 2 é 12 e o número de operações de escrita solicitadas pelos sites 1 e 3 é 20, o site 2 não entra no esquema de replicação. O número de operações de leitura solicitadas pelo site 3 é 32 e o número de operações de escrita solicitadas pelos sites 1 e 2 é 8, por isso, o site 3 entra no esquema de replicação, fazendo o conjunto R = {1,3}.

No fim do segundo período de tempo, os sites 1 e 3 vão executar os seguintes testes. Primeiro, o site 1 executa o teste de expansão com o site 2 e falha. Segundo, o site 1 executa o teste da contração e é bem sucedido, já que ele recebe 18 operações de escrita do site 3, e 16 operações de leitura dos sites 1 e 2. Ao mesmo tempo, o site 3 executa o teste da expansão, e inclui o site 8 no esquema de replicação, já que o número de operações de leitura enviadas pelo site 8 é 20, e o número de operações de escrita enviadas pelos outros sites é 14.

Como resultado desses testes, o esquema de replicação se torna R = {3,8} depois do segundo período. A partir do terceiro período, o esquema de replicação é estabilizado em

{3,8} e não muda mais se o padrão de leitura e escrita permanecer o mesmo. A técnica de ler uma cópia e escrever em todas impede operações de escrita de executarem quando ocorrem falhas no sistema. Para permitir essas operações de escrita mesmo em caso de falha, [WoJH97] propõe um protocolo chamado de *Primary Missing Writes* que pode ser combinado com a replicação dinâmica. Para maiores detalhes e para a generalização do algoritmo para topologia em grafos genérica, deve-se consultar [WoJH97].

2.4. Algoritmos de Replicação Atrasada

Neste seção, serão abordadas algumas técnicas de replicação atrasada [LiHD98, GHOS96, BrKo97, ABKW98] onde as operações de escrita nas réplicas são distribuídas assincronamente depois que a transação original terminar.

2.4.1. Protocolos de Acesso Multivisão para Replicação em Larga Escala

A replicação de dados não é só útil em aplicações de pequena escala aonde o número de sites é pequeno (por exemplo: < 10), mas também pode ser usada em muitas aplicações de larga escala. Com o advento das WAN e da Internet tem sido possível replicar dados para 100 ou mesmo 1000 localizações diferentes por exemplo.

Um importante ponto na gerência da replicação de dados é assegurar o acesso correto ao dado no que tange a concorrência, a falhas e a propagação das atualizações. O critério mais usado para a gerência da replicação de dados é o 1SR (*One-Copy Serializability*) [BeHG87].

O modo mais freqüente de garantir o correto acesso a dados replicados é usar protocolos de controle de concorrência para assegurar a consistência dos dados; protocolos de controle da replicação para assegurar a consistência mútua; e protocolos para estabelecer pontos de sincronismo (*commit*) para assegurar que as mudanças, ora para uma, ora para todas as cópias do dado replicado sejam feitas de forma permanente.

A maioria dos protocolos utilizados são: *Two-Phase Locking* (2PL) para controle de concorrência, *Read-One-Write-All* (ROWA) para controle da replicação e *Two-Phase Commit* (2PC) para estabelecer pontos de comprometimento. Estes três protocolos juntos asseguram o

1SR e garantem que cada transação enxergue as atualizações feitas através da serialização prévia das transações.

O critério 1SR para gerência de replicações trabalha bem em muitos sistemas replicados de pequena escala, porém não suporta eficientemente sistemas muito grandes, isto devido a natureza centralizada dos protocolos 2PL, 2PC e ROWA. Por exemplo, o protocolo 2PC requer uma mensagem trafegando entre o site master onde a atualização é iniciada e os demais sites envolvidos na atualização. Isto não somente introduz um delay significativo para o sincronismo das atualizações em todos os sites, como também resulta em uma alta taxa de transações abortadas. Acontece que em muitas situações onde os dados são replicados, principalmente em sistemas distribuídos de larga escala, nem sempre a atualização precisa ser efetivada em todos os sites de forma síncrona e ao mesmo tempo.

Para ilustrar este cenário, vamos considerar a atualização do componente de um software que sofreu uma correção. Este componente pode ser replicado para milhares de sites e este processo de replicação ser então executado em cada servidor envolvido. Neste exemplo, o protocolo 2PC abortará todo o processo de atualização se alguma operação da transação falhar devido à falhas nos sites (queda nos *links* da rede ou até mesmo por ocorrerem *deadlocks* locais).

Esta seção apresenta e discute uma série de algoritmos alternativos para tratar este problema em função da baixo desempenho dos protocolos 2PL e 2PC quando utilizado na gerência de dados replicados em sistemas distribuídos de larga escala.

[LiHD98] propõe uma estratégia de acesso multivisão (MVA) com o objetivo de diminuir a alta taxa de transações abortadas e de melhorar os problemas de baixa performance em um sistema de banco de dados que utiliza a replicação em larga escala. A idéia principal é a de reduzir o número de sites que executam um protocolo de *commit* distribuído.

Este artigo propõe um *framework* que primeiro agrupa os sites em uma arquitetura cluster em árvore e então organiza as réplicas de dados em uma estrutura de multivisão hierárquica. Existe uma cópia primária para cada dado replicado no cluster em que este reside.

Esta estratégia propõe o desmembramento das transações de atualização, que envolvem um número muito grande de sites, em um conjunto de transações relacionadas, cada uma operando em um escopo menor. Todas as transações com *commits* independentes. A vantagem disso é que as transações não precisam aguardar que todas as réplicas tenham sido atualizadas.

Além disso, a falha de uma transação não afeta a outra, ou seja, somente as transações que falharem é que precisarão ser re-executadas e a estratégia ainda garante a consistência dos dados que foram replicados.

Comparado com outros protocolos similares, o protocolo proposto oferece uma melhor performance [LiHD98]. Esta melhoria é decorrente, tanto pelo ganho no *throughput* das transações, associado a redução significativa da taxa de transações abortadas, quanto na melhoria do tempo de resposta dos usuários, visto que uma transação comita independentemente da atualização das réplicas. O sistema se encarregará da propagação da atualização, mesmo para os sites que não estejam operacionais no momento da replicação.

A MVA é uma estratégia eficiente para a gerência de dados replicados, que provê diferentes visões do dado replicado em uma arquitetura escalável e bem controlada, ao contrário dos protocolos 2PL e 2PC que fornecem uma visão única dos dados e que tem uma escalabilidade limitada.

A Estratégia

A seguir será detalhada a estratégia MVA para sistemas de banco de dados replicados em larga escala. O foco será a organização dos dados e a arquitetura do sistema.

Esta estratégia assume que o sistema de banco de dados distribuído em epígrafe consiste de centenas ou mesmo de milhares de sites.

A estratégia propõe o agrupamento dos sites em clusters, e que deve ser feito através de políticas que otimizem a performance. Por exemplo: políticas de otimização que considerem o agrupamento dos sites que se comunicam eficientemente (com variações de delays de comunicação similares), ou o agrupamento baseado na afinidade das réplicas dos dados (aqueles que estão mais relacionados tendem a ser acessados pelas mesmas transações). Os clusters também podem ser formados de acordo com critérios físicos, tais como localização geográfica ou critérios lógicos, tais como a própria organização do negócio.

Cada dado replicado pode pertencer a uma ou mais organizações. Para cada organização que possui o dado, determina-se um site que é o site proprietário do dado replicado (*owner*), e que normalmente deverá ser o seu site primário. Um dado replicado é mais freqüentemente acessado pelos sites que estão dentro do mesmo cluster do seu proprietário.

Também é considerado que é possível se conhecer o perfil do ambiente transacional do sistema. As transações dos usuários normalmente possuem diferentes requisitos de concorrência e de consistência. Algumas transações requerem uma visão bem atualizada do banco de dados, enquanto outras podem utilizar uma visão que não seja tão atualizada para, em contra partida, poder ter um ganho no tempo de resposta. É de extrema importância que a estratégia para a gerência de dados replicados suporte estes diferentes requisitos dos usuários em um mesmo *framework*.

Exemplo de Utilização

Para um melhor entendimento desta proposta, será apresentado um exemplo que ilustra bem a estratégia MVA.

Grandes companhias, como por exemplo a *Hewllet-Packard*, tem utilizado a sua intranet para a gerência da distribuição de componentes de seus softwares, como também para a gerência das configurações usadas pelos desktops da companhia. Um exemplo disso é o *COE (Common Operating Environment*), que é um sistema interno da HP responsável por permitir que mais de 100.000 usuários de *desktop* possam carregar aplicações padrões de seus softwares e configurá-las através da rede. Este sistema reduz significantemente o custo administrativo da manutenção dos 100.000 *desktops* porque o software é gerenciado de forma centralizada o que aumenta bastante a produtividade individual.

O COE é um sistema distribuído de larga escala consistindo de mais de 200 servidores locais dispersos geograficamente e que suporta mais de 100.000 *desktops* de usuários em todo o mundo. Através deste sistema, os componentes de software são replicados para os servidores locais de modo que possam ser carregados de forma mais eficiente pelos *desktops*.

A maioria dos *requests* destes usuários é *read-only*, ou seja, eles só carregam e executam os componentes de software em seus *desktops*. Os componentes de software podem ser carregados através de qualquer um dos servidores locais e precisam ser atualizados em todos os servidores onde estes estejam replicados.

Observe que a atualização de um software na maioria das vezes pode envolver vários componentes, e que certas versões de componentes de software só funcionam com específicas versões de outros softwares, logo é fundamental que toda atualização seja atômica. Muitas vezes é aceitável um atraso na atualização do software, desde que os componentes estejam

agrupados de forma a sempre garantir que este *delay* não cause uma incompatibilidade no software local. Na maioria dos casos, é mais importante usar uma versão consistente do componente do software, do que a versão mais nova.

A Arquitetura do Sistema e a Organização dos Dados

A estratégia MVA assume uma arquitetura de sistema organizada em uma árvore estruturada aonde a raiz representa todo o sistema de banco de dados replicado e distribuído, enquanto as folhas representam sites individuais. Os nós internos representam *clusters* que são formados através do agrupamento dos sites.

A Figura 4 mostra uma possível hierarquia de clusters para o COE da HP. O sistema representado pelo cluster HP de nível raiz, pode ser estruturado em cinco *clusters* de primeiro nível, um para cada área de operação da empresa. Cada *cluster* de primeiro nível, por sua vez, consiste de vários *clusters* de segundo nível. Os *clusters* de segundo nível novamente consistem de *clusters* de terceiro nível que, no caso, representam os servidores locais.

A idéia básica desta arquitetura em árvore baseada em *clusters* é de usar a técnica "**dividir para conquistar**" de modo a dar foco nos aspectos complexos da gerência de dados replicados.

A formação dos *clusters* deve seguir o princípio de recurso limitado [LiHD98]: ***A demanda*** de serviço através de qualquer componente do sistema deve ser limitado por uma constante que independe do número de nós no sistema. A formação atual dos clusters é dependente do sistema, mas as regras genéricas a seguir devem ser seguidas sempre que possível:

- número de site em cada *cluster* deve ser pequeno (ex: < 10);
- custo de comunicação é menor entre sites dentro de um cluster;
- Os sites em um *cluster* dever estar relacionados entre si, de tal modo que, os *requests* dos usuários de um *cluster* sejam locais a este cluster.

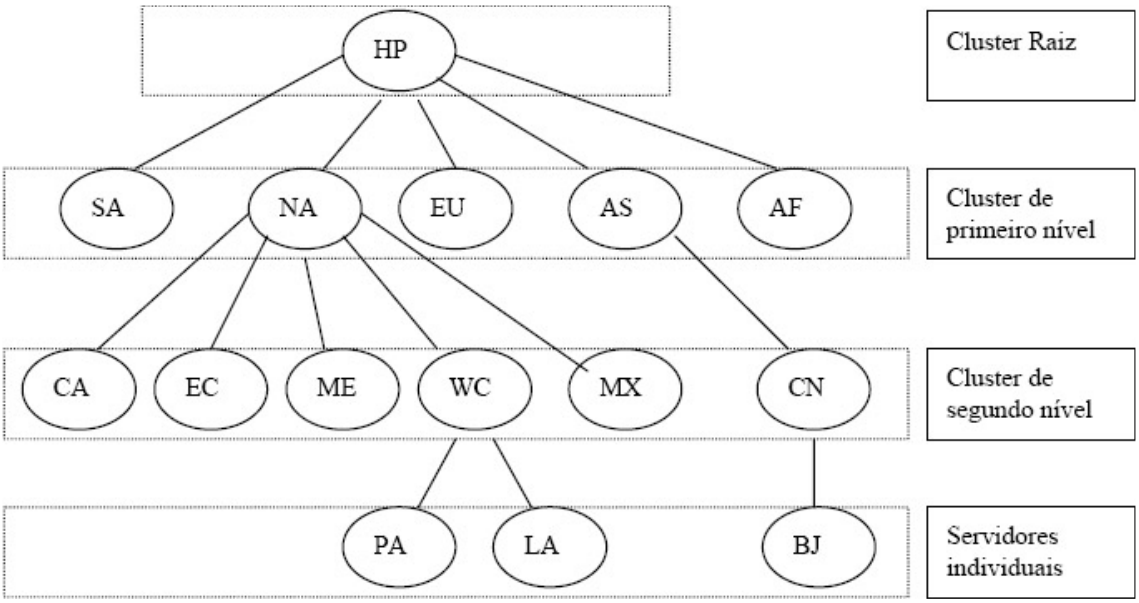

Figura 2.4 – Exemplo de hierarquia de *clusters*

A estratégia MVA contempla três diferentes tipos de sites para a replicação de dados: **o site primário, o site coordenador e os sites comuns.**

Site primário: Para cada dado replicado existe um site primário em cada *cluster* que o mesmo reside. As cópias dos dados replicados nos sites primários são chamadas de cópias primárias. Embora diferentes dados possam ter diferentes sites primários em um *cluster*, a maior parte dos dados tendem a compartilhar o mesmo site primário em aplicações reais.

No exemplo que está sendo apresentado, considere que, por exemplo: o MS Word é de propriedade do servidor PA para a organização mundial da HP, enquanto o software Chinese Star é de propriedade do servidor BJ para a HP chinesa. O servidor PA deve ser o site primário para o MS Word no cluster NA (*cluster* de primeiro nível), como também para o cluster WC (*cluster* de segundo nível). O servidor BJ pode ser o site primário para o MS Word no cluster CN (*cluster* de segundo nível) para a China. O servidor BJ também deve ser o site primário para o *software* Chinese Star no *cluster* CN.

Considerando que os dados replicados são atualizados por transações independentes e *clusters* diferentes, os sites primários em níveis diferentes de *clusters* podem ter diferentes visões do mesmo dado replicado. Em um cluster, a cópia primária e cópias não-primárias de um dado também podem ter diferentes valores.

Se um determinado site é um site primário para mais de um *cluster*, este manterá visões separadas do mesmo dado para cada *cluster*. Pelo ponto de vista da gerência de transações,

elas são diferentes cópias dos dados que foram atualizadas por diferentes transações. Por exemplo: o servidor PA é o site primário para o MS Word nos clusters WC, NA e HP. Este servidor mantém três cópias lógicas do software para os três clusters. Na maior parte das vezes, as três cópias referem-se a mesma versão do MS Word, mas é possível ter diferentes cópias, como por exemplo, no meio de um processo de *upgrade*, daí a multivisão da estratégia.

Os usuários podem executar uma transação a partir de qualquer um dos sites da rede. Uma transação *read-only* pode ser processada em qualquer destes sites, desde que contenham as cópias de dados que atendem aos requisitos de consistência da transação. Uma transação de *update*, contudo, tem que ser direcionada ao cluster que contém todas as cópias dos dados. Por exemplo, um usuário em Beijing pode carregar o MS Word do seu servidor BJ. O software, entretanto, tem que estar atualizado por uma transação global para o cluster HP. Chinese Star, por outro lado, pode ser atualizado por transações locais ao cluster CN.

Uma transação em um sistema cluster de nível *L* é processada em *L* etapas, atualizando as cópias de dados em clusters de nível *L*, respectivamente. Por exemplo, MS Word pode ser atualizado como nas seguintes etapas. Primeiro passo, uma transação é iniciada no cluster HP, que atualiza as cópias primárias nos clusters NA, SA, EU, AF e AS. Então, no segundo passo, são geradas cinco transações de propagação independentes em cada cluster de primeiro nível, que por sua vez, atualizam as cópias primárias para os clusters de segundo nível. Finalmente, no terceiro passo, são iniciadas transações de propagação para cada cluster de terceiro nível, que atualizam cópias não-primárias no cluster.

Site coordenador: Existe também o conceito de um site coordenador em cada cluster. O site coordenador atualiza as cópias primárias no cluster. Observe que cada transação normalmente atualiza vários dados replicados que podem ter diferentes sites primários. Em geral, o cluster *master*, que é aquele aonde a transação se iniciou, precisa enviar mensagens de atualização para cada site primário do dado replicado em cada um dos clusters existentes, o que normalmente são inúmeros sites. Usando sites coordenadores, somente será necessário uma mensagem entre o cluster *master* e cada cluster *slave*. A comunicação entre os clusters é significativamente reduzida em função do número de clusters ser muito menor do que o número de sites. O site coordenador coordenará as operações de atualização das cópias primárias no cluster com os coordenadores dos outros clusters. Estas operações seguem o

protocolo 2PC para garantir que as cópias primárias de diferentes sites e de diferentes clusters sejam atualizadas atomicamente.

Além disso, o site coordenador em um cluster é também responsável por gerar as transações de propagação para atualizar as cópias dos dados nos sites membros ou nos clusters de nível mais baixo. A transação é executada e comitada independentemente da sua transação original. O site coordenador é responsável por reemitir a transação se esta falhou anteriormente. É importante notar que o site coordenador pode ser independente quanto aos dados. Em geral, existe somente um site coordenador por cluster, representando todos os dados replicados no cluster. O site coordenador de um cluster deve idealmente ser um site *gateway*, tal que a comunicação entre ele e os outros sites no cluster seja eficiente e confiável.

A Figura 2.5 apresenta uma estrutura de cluster que ilustra o exemplo do sistema de gerência de distribuição de software da HP.

Esta arquitetura pode ser facilmente reconfigurada em tempo de execução. Quando um cluster torna-se muito grande, devido a inclusão de novos sites, e não pode ser eficientemente gerenciado, pode-se dividi-lo em dois ou mais clusters. Novos sites primários e coordenadores serão selecionados para cada um dos novos clusters. De forma similar, quando dois ou mais clusters no mesmo cluster pai tornam-se muito pequenos, devido a exclusão de membros, eles podem ser combinados em um único cluster no mesmo cluster pai. Quando o número de sites em um cluster pai torna-se muito pequeno, este por sua vez, pode ser combinado com outro cluster de mesmo nível.

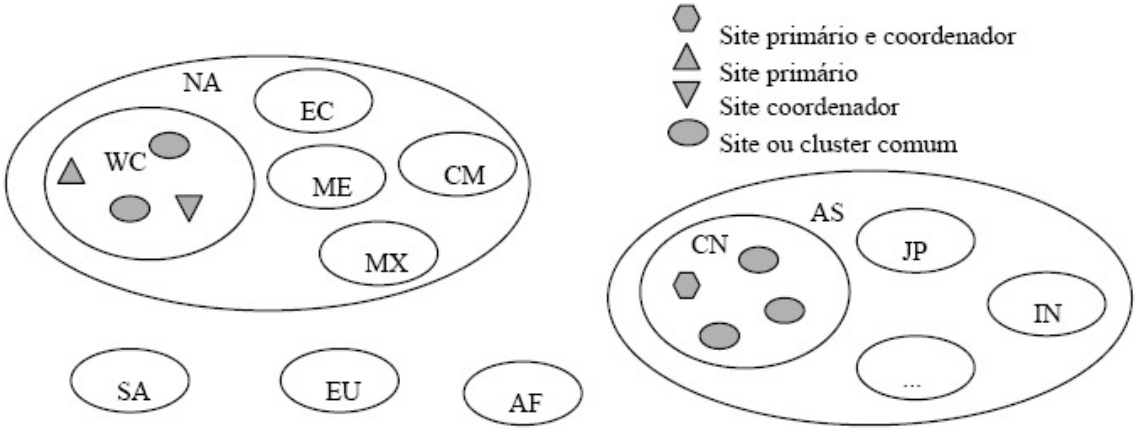

Figura 2.5 – Exemplo de estrutura de cluster

Esta arquitetura também permite a especificação de sites primários e coordenadores de backup. Eles funcionam como sites comuns quando em operação normal e substituem os sites

primários e coordenadores (originais) quando estes tornam-se não-funcionais. O propósito é aumentar a disponibilidade do sistema em caso de falhas.

Apenas para ilustrar com outros exemplos, esta estratégia também pode ser usada em muitas outras aplicações, tais como: redes de vendas eletrônicas, controle de estoque internacional, sistemas de reserva de vôo, rede de roteamento de telefones celulares, entre outras.

Protocolo Básico

A idéia básica da estratégia MVA é de propagar as atualizações dos dados replicados em um modelo *cluster-by-cluster* desmembrando as transações originais em um conjunto de transações independentes e comitadas e que estejam relacionadas entre si. O processo de atualização inicia através do cluster de mais alto nível que contém todas as cópias dos dados replicados, que deverão ser atualizados e propagados para os clusters de nível mais baixo. Em cada cluster, o protocolo primeiro atualiza as cópias de dados dos sites membros e as cópias primárias dos clusters membros. A seguir, gera transações para os clusters membros, que continuam o processo de atualização até que todas as cópias tenham sido atualizadas.

De forma diferente das transações de atualização, as transações de leitura podem ser processadas com mais flexibilidade. Elas podem escolher a leitura das cópias dos dados ao longo dos diferentes níveis de clusters aonde a cópia exista, decidindo isso a partir de seus requerimentos correntes, de performance e consistência.

Neste protocolo, será feita a distinção de três tipos diferentes de sites existentes em um cluster: o *site mestre*, aonde a transação é iniciada, o *site coordenador*, aonde as atualizações dos dados replicados são coordenadas, e os *sites escravos*, aonde as operações de leitura e escrita são realmente implementadas. O protocolo consiste de algoritmos para estes três tipos de sites.

Nos algoritmos que serão apresentados, cada transação consiste de cinco operações básicas. Uma transação inicia com a operação *begin*, termina com, ou a operação de *commit*, ou com a operação de *abort*. E inclui uma seqüência de operações de *read* e *write* intercaladas. Será usado r_I[d] e w_I[d] para denotar operações de leitura e escrita em um dado $d \in D$ em uma transação T_i, respectivamente. E b_I, c_I, e a_I para denotar as operações de *begin*, *commit* e *abort* da transação T_i respectivamente.

Algoritmo do site mestre: Quando uma transação chega, o site mestre verifica o seu tipo, assim como os seus requerimentos correntes e de consistência, e direciona as operações de leitura e escrita aos apropriados sites coordenador e escravos. Uma transação é enviada ao site mestre sempre com três parâmetros: consistência (*consistency*), corrente (*currency*) e *original*. O requerimento de *consistência* pode ser , ou YES (requer uma visão consistente do banco de dados), ou NO. Para um sistema de cluster nível-L, o requerimento *corrente* pode ser qualquer inteiro entre 0 e $L - 1$, onde o 0 representa a visão do banco de dados do cluster de nível raiz. O parâmetro *original* é um *flag* que indica se a transação é uma transação oriunda do usuário, ou uma transação gerada a partir de alguma propagação.

Uma transação é executa em um dos diferentes níveis-L. No algoritmo, será usado *level* para indicar o nível de execução que pode ser um inteiro entre 0 e $L - 1$. A seguir são apresentadas as regras gerais que determinam o nível de execução de uma transação.

- Uma transação original de *update* é sempre executada no nível 0 de execução.
- Uma transação *read-only* pode ser executada em um nível *l* de execução se:
  1. consistency = = YES;
  2. currency ≥ l; e
  3. existe um cluster nível l, que contém pelo menos uma cópia de cada dado que a transação está lendo.
- Uma transação *read-only* pode ser executada em qualquer nível se *consistency* = = NO.

No algoritmo, o nível de execução é inicialmente determinado em tempo de submissão da transação (parte do processo da operação *begin*). O nível de execução pode ser modificado posteriormente em função do tipo da transação e de mudanças no escopo. O algoritmo apresentado no Apêndice A aceita e processa uma transação T_i com requerimento de consistência: *consistency* e corrente: *currency* para o site $s_j \in S$.

Algoritmo do site coordenador: O site coordenador é responsável por propagar as atualizações para os sites escravos e clusters membros e, como óbvio, isto é necessário somente para as transações de *update*. Ele também é responsável por iniciar e coordenar o protocolo 2PC para garantir a execução das transações de forma atômica. Novamente, operações de escrita são direcionadas para todos os sites membros relevantes e para os sites primários de clusters membros. Após as atualizações terem sido comprometidas pelo *cluster*

corrente, o site coordenador gerará transações para cada cluster membro para que se possa continuar o processo de propagação. O algoritmo é apresentado no Apêndice B.

Algoritmo do site escravo: O algoritmo do site escravo processa operações de leitura e escrita de transação T_i em um site S_j. No algoritmo, cada site primário S_j (em um cluster de nível-l) mantém até $L - 1$ cópias de um dado replicado x: x_j^l, x_j^{l+1}, . . . , x_j^{L-1}. A cópia do dado $x_j^{l'}$ ($l \leq l' \leq L - 1$) é para transações no nível de execução l'. No algoritmo apresentado no Apêndice C, foi assumido que as operações de leitura e escrita são processadas seguindo os padrões dos protocolos 2PL e ROWA.

Considerações sobre Consistência

Transações são programas executados atomicamente. Elas têm que satisfazer as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) [CaMo85].

Serializability tem sido usada como o critério correto na execução de transações [BeHG87] para assegurar a consistência do banco de dados e o isolamento. Em muitas aplicações, entretanto, a serialização estrita é muito forte. [LiHD98] comenta a consistência no acesso a transações *read-only*. Uma fraca noção de consistência denominada de *consistência insular* tem sido proposta. Uma execução E assegura a consistência insular se qualquer subexecução de E que contém todas as transações de *update* em E e uma transação *read-only* em E é serializável . Uma execução insular consistente preserva os dados consistentes do banco de dados, devido as transações de *update* serem executadas no modo serializável. O isolamento da execução também é garantido devido ao fato de que cada transação vê uma visão do banco de dados resultante da execução serializável das transações.

O protocolo apresentado [LiHD98] considerou os seguintes requerimentos de tipos de consistência:

1. consistência estrita ou *one-copy serializability* (1SR);
2. consistência insular; e
3. relaxada ou sem consistência.

2.4.2. Replicação Two-Tier

Em [GHOS96], é feita uma análise das técnicas de replicação, chegando-se às seguintes conclusões:

- Usando replicação imediata, o número de deadlocks aumenta com a terceira potência do número de nós na rede e com a quinta potência do tamanho das transações (número operações a serem executadas). Se for usada replicação imediata com atualização através do nó mestre, isso ajuda a evitar deadlocks.
- Replicação atrasada com mestre é um pouco menos susceptível a deadlocks que replicação imediata em grupo, basicamente porque as transações têm menor duração.

Devido a esses problemas, [GHOS96] propõe uma outra técnica chamada de *two-tier replication*. O esquema de replicação *two-tier* assume que existem dois tipos de nós:

- Nós móveis que estão desconectados a maior parte do tempo. Eles armazenam uma réplica do banco de dados e podem originar transações experimentais. Um nó móvel pode ser o mestre de alguns itens de dados;
- Nós base que estão sempre conectados. Eles armazenam uma réplica do banco de dados e a maioria dos itens de dados têm seus mestres em nós base.

Os itens de dados replicados têm duas versões nos nós móveis:

- A versão mestre que detém o valor mais recente recebido do mestre do objeto. A versão no nó mestre é a versão mestre, mas nós desconectados ou replicados de maneira atrasada podem ter versões antigas;
- Versões experimentais: o objeto local pode ser atualizado por transações experimentais. O valor mais recente atualizado localmente é mantido como uma versão experimental.

De forma similar, existem dois tipos de transações:

- Transações base que só acessam dados mestre e produzem novos dados mestre. Elas envolvem no máximo um nó móvel conectado e podem envolver diversos nós base.
- Transações experimentais que acessam dados experimentais. Elas produzem novas versões experimentais e também produzem uma transação base a ser executada posteriormente em nós base.

Os usuários estão cientes de que todas as atualizações são experimentais até que a transação se torne uma transação base e se esta falhar, o usuário deve revisar e resubmeter a transação.

A transação base gerada por uma transação experimental pode falhar ou produzir resultados distintos. A transação base tem critérios de aceitação, que são testes pelos quais seus resultados devem passar para serem aceitos. Se uma transação experimental falhar, o nó que a originou é informado do motivo por que falhou. Falhas de aceitação são equivalentes ao mecanismo de reconciliação de transações dos esquemas que utilizam replicação atrasada em grupo [GHOS96]. As diferenças são:

- O banco de dados mestre é sempre atualizado (*converged*), pois não existem enganos (*delusion*) no sistema;
- Um nó origem só precisa contactar um nó base para descobrir se uma transação experimental é aceitável.

Quando um nó móvel se conecta a um nó base, ele executa os seguintes procedimentos:

- descarta suas versões experimentais já que elas serão atualizadas pelos mestres;
- envia atualizações nas réplicas dos objetos cujo mestre é o nó móvel para o nó base conectado;
- envia todas as transações experimentais (e seus parâmetros de entrada) ao nó base para serem executadas na ordem em que comitaram no nó móvel;
- aceita atualizações nas réplicas vindas do nó base (essa é a replicação padrão atrasada com mestre);
- aceita a resposta de sucesso ou fracasso de cada transação experimental.

O nó base na outra ponta da conexão, quando contactado por um nó móvel, executa os seguintes procedimentos:

- envia as transações de atualização atrasadas ao nó móvel;
- aceita transações de atualização atrasadas dos objetos cujos nós mestre são o nó móvel;
- aceita a lista de transações experimentais, suas mensagens de entrada e seus critérios de aceitação. Executa cada transação experimental de acordo com a ordem em que elas comitaram no nó móvel. Se a transação base falhar nos critérios de aceitação, ela é abortada e uma mensagem de diagnóstico é enviada ao nó móvel;
- depois que o nó base compromete uma transação base, ele propaga as atualizações de forma atrasada para todos os outros nós, usando o mecanismo deferido com mestre;

- quando todas as transações experimentais forem reprocessadas como transações base, o estado nó móvel converge para o estado do nó base.

As propriedades-chave do esquema usando replicação *two-tier* são:

- Nós móveis podem fazer atualizações experimentais no banco de dados;
- Transações base executam com serialização usando um única cópia, então o estado do sistema base mestre é o resultado de uma execução serializável;
- Uma transação se torna durável quando a transação base completa sua execução;
- Réplicas em todos os nós conectados convergem para o estado do sistema base;
- Se todas as transações comutam (o banco de dados termina no mesmo estado independente da ordem de execução das transações), não existem reconciliações.

Essas propriedades se aproximam dos quatro objetivos que devem ser atingidos por um esquema de replicação ideal, de acordo com [GHOS96]:

- Disponibilidade e escalabilidade: fornecer alta disponibilidade e escalabilidade através da replicação, evitando instabilidade.
- Mobilidade: permitir que nós móveis possam ler e atualizar o banco de dados enquanto estiverem desconectados da rede.
- Serialização: fornecer execução de transação serializáveis com uma única cópia.
- Convergência: fornecer convergência para evitar enganos no sistema.

2.4.3. Protocolos Usando Grafos de Replicação

Em [ABKW98, BrKo97] são descritos dois protocolos de replicação baseados em grafos usando o modelo de replicação atrasada. O modelo considerado tem as seguintes características:

- Para cada item de dado, existe uma cópia primária em um site primário que é o responsável por atualizações nesse item de dado. As outras cópias são chamadas cópias secundárias;
- banco de dados local em cada site garante as propriedades ACID para as transações e gerência *deadlocks* locais;
- o site no qual uma transação T_i é submetida é chamado de site original de T_i. Essa transação só pode atualizar um dado *d* se ela foi originada no site primário de *d*;

- Quando a cópia primária de um dado d é atualizada, essa operação deve ser propagada para as cópias secundárias de d, depois que a transação original for comprometida;
- Uma transação T_i pode estar em um dos seguintes estados de acordo com o diagrama de transição de estados exibido na Figura 6:
  1.1. ativa: uma transação T_i está ativa em seu site original se ela começou e ainda não comprometeu ou abortou;
  1.2. compromete: se uma transação T_i comprometeu em seu *site* original;
  1.3. abortada: se uma transação T_i abortou em seu site original;
  1.4. completada: se em cada site que a transação T_i executou, T_i já comprometeu e não é precedida por nenhuma transação ainda não comprometida na serialização local.

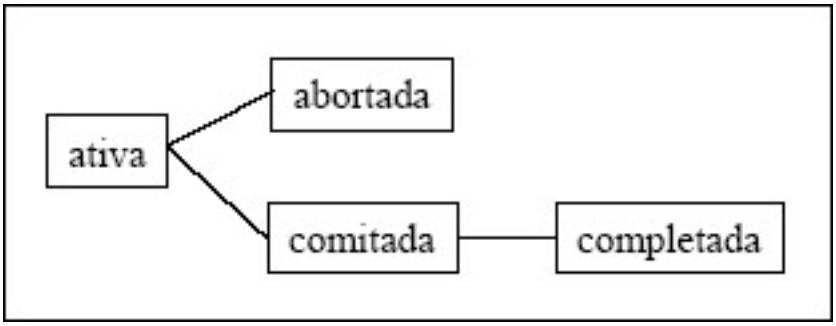

Figura 2.6 – Diagrama de transição de estados das transações

Observe que durante a execução de uma transação comprometida em um site secundário, a subtransação pode ser abortada pelo SGBD local. Nesse caso, ela será reiniciada e executada novamente nesse site.

Os protocolos descritos em [ABKW98, BrKo97] utilizam dois conceitos fundamentais: sites virtuais e grafo de replicação.

Sites virtuais são conjuntos de itens de dados em um site físico. Cada transação tem um site virtual associado em cada site físico onde ela executa. Esse site virtual existe a partir do momento em que a transação começa até o momento em que ela não é mais considerada pelo protocolo. O site virtual da transação T_i no site físico s_j é denotado por $VS_{j\,i}$. Mais de uma transação pode compartilhar um site virtual. Então, para transações T_i e T_k, pode existir o caso em que seus sites virtuais são idênticos, denotados por $VS_{j\,i} = VS_{j\,k}$.

O conjunto de sites virtuais é construído e mantido baseado em três regras:

- *Regra da localidade*: é necessário que cada transação local execute precisamente em um site virtual. Uma transação de atualização global pode ter diversos sites virtuais, um em cada site físico em que executa. Em cada momento do tempo, $VS_{j\,i}$ deve conter o conjunto de todos os itens de dados no site físico s_j que a transação T_i acessou até aquele momento.
- *Regra da união*: se duas transações T_i e T_k acessam um item de dado comum *d* em um site s_j, os seus sites virtuais no site s_j devem ser reunidos caso aconteça o seguinte:
  1.5. *d* seja um item de dado primário e as transações conflitem em *d*;
  1.6. *d* seja um item de dado secundário e as transações estejam em conflito rw_1 ou wr_2 em *d*.

Reunir os sites virtuais significa que eles se tornam idênticos, devendo conter os mesmos elementos acessados até o momento por T_i ou T_k.

- *Regra da divisão*: quando o site físico s_j determina que um transação T_i entrou no estado abortado ou completado, qualquer item de dados acessado exclusivamente por T_i é removido do seu site virtual e o protocolo não precisa mais considerar essa transação. Se não existe transação T_k compartilhando o site virtual com T_i, o site virtual de T_i é removido. Caso contrário, os sites virtuais de todas as transações que compartilham o site virtual com T_i são recomputados usando as regras de localidade e união.

Enquanto as regras de localidade e união são importantes para corretude, a regra da divisão é importante para melhoras na performance.

Um grafo de replicação é associado com a execução de um conjunto de transações para representar conflitos que surgem de atualizações em dados replicados. Existe um único grafo de replicação global para todo o sistema distribuído.

Suponha que T é o conjunto de transações globais, T' é o conjunto de transações locais, e V é o conjunto de sites virtuais onde as transações de $T \cup T'$ estão executando. Diz-se que $RG = <T \cup V, E>$ é um grafo de replicação se RG é um grafo não-dirigido bipartido.

Para cada transação T_i em T, existe um arco $(T_i, VS_{j\,i})$ em E para cada site físico onde T_i tem um site virtual $VS_{j\,i}$.

Dado um grafo de replicação e um conjunto de operações, define-se um teste, chamado de RGtest, que aplica as regras de localidade e união ao grafo, para o conjunto de operações. RGtest é bem sucedido se nenhum ciclo for resultante do teste, caso contrário, ele fracassa.

Tipicamente, se o RGtest for bem sucedido, o conjunto de operações é permitido e as atualizações experimentais ao grafo de replicação são tornadas permanentes, caso contrário, a transação envolvida pode ser forçada a esperar ou abortar.

Em [ABKW98], são descritos dois protocolos que usam esses conceitos, um protocolo pessimista e um otimista[3], que serão abordados a seguir.

Protocolo Pessimista

O protocolo pessimista [ABKW98] é baseado no protocolo descrito em [BrKo97], com algumas modificações na definição de sites virtuais e no uso da regra de Thomas Write [BeHG87] para tratar conflitos ww[4].

O protocolo funciona da seguinte forma:

1. Se Ti submete sua primeira operação, atribui a Ti um timestamp.
2. Se Ti submete uma operação de leitura ou escrita em seu site original, aplica o RGtest:
   1.7. Se o teste for bem sucedido, a operação recebe permissão de execução (aplicando a regra de Thomas Write se for operação de escrita) e as atualizações experimentais no grafo de replicação são tornadas permanentes;
   1.8. Se o teste falhar e T_i for local, T_i submete a operação de abortar;
   1.9. Se o teste falhar e T_i for global, testa se algum ciclo no grafo de replicação inclui uma transação comitada. Se for o caso, T_i submete a operação de abortar, caso contrário, T_i espera.

3. Se Ti submete uma operação de escrita em um site diferente do seu original, aplica a regra de Thomas Write.
4. Se Ti submeter uma operação de commit, prossegue com a execução. Se Ti estiver no estado completada, remove-a, tirando-a do grafo de replicação e aplicando a regra da divisão. Checa se alguma transação em espera pode ser ativada ou abortada como resultado.
5. Se Ti submeter a operação de abortar em seu site original, remove-a do grafo de replicação e remove suas subtransações de qualquer fila de execução onde estejam.

Aplica a regra da divisão e checa se outras transações em espera podem ser ativadas.

3. Conclusões

Este trabalho procurou realizar uma pesquisa na literatura de banco de dados a respeito das técnicas de replicação utilizadas em sistemas de banco de dados distribuídos. Os objetivos principais do mecanismo de replicação são aumentar a disponibilidade do banco de dados e a performance das aplicações. Entretanto, esse mecanismo tem um preço decorrente da necessidade de atualizar as réplicas dos dados em diferentes sites do banco de dados distribuído.

Foram abordadas algumas técnicas de replicação imediata [BeHG87, WoJH97] e alguns algoritmos utilizando o esquema de replicação atrasada [LiHD98, GHOS96, BrKo97, ABKW98]. A escolha da técnica mais indicada depende das características das aplicações em questão.

Na verdade, o problema da replicação pode ser considerado bem dominado na área de banco de dados. Os trabalhos mais recentes focam em replicação atrasada enquanto os mais antigos focavam em replicação imediata [ABKW98]. O enfoque atual está bastante direcionado para aplicações em larga escala [LiHD98], onde o sistema distribuído é composto de muitos sites, o que inviabiliza a utilização de replicação imediata e propicia a utilização de replicação atrasada.

4. Referencial Bibliográfico

[ABKW98] T. Anderson, Y. Breitbart, H. F. Korth, A. Wool. Replication, Consistency, and Practicality: Are these Mutually Exclusive?, *Proceedings of the ACM SIGMOD'98*, Seattle, USA, 1998.

[BrKo97] Y. Breitbart, H. F. Korth. Replication and Consistency: Being Lazy Helps Sometimes, *Proceedings of the ACM PODS'97*, Tucson, Arizona, USA, 1997.

[BeHG87] P. A. Bernstein, V. Hadsilacos, N. Goodman. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing Company, 1987.

[Bure97] M. Buretta, *Data Replication*, John Wiley & Sons, 1997.

[CaMo85] M. A. Casanova, A. V. Moura. *Princípios de Sistemas de Gerência de Banco de Dados Distribuídos*, Editora Campus, 1985.

[CoHa99] J. Cook, R. Harbus. *The DB2 replication*, Prentice Hall, 1999.

[GHOS96] J. Gray, P. Helland, P. O'Neil, D. Shasha. The Dangers of Replication and a Solution, *Proceedings of the ACM SIGMOD'96*, Montreal, Canadá, Junho 1996.

[LiHD98] X. Liu, A. Helal, W. Du. Multiview Access Protocols for Large-Scale Replication, ACM Transactions on Database Systems, Vol. 23, No. 2, Junho 1998.

[WoJH97] O. Wolfson, S. Jajodia, Y. Huang. An Adaptive Data Replication Algorithm, *ACM Transactions on Database Systems*, Vol. 22, No. 2, Junho 1997.

Resumo Estendido

A principal razão para utilizar réplicas é aumentar a disponibilidade do sistema de banco de dados. Armazenando dados críticos em múltiplos *sites*, o sistema fica disponível mesmo se alguns deles estiverem em falha.

Outra razão é aumentar o desempenho. Já que existem muitas cópias de cada item de dados, uma transação pode encontrar o dado que ela precisa em um *site* mais próximo, quando comparado a um banco de dados com uma única cópia da informação.

A replicação de dados tem provado ser uma técnica útil em muitas aplicações aonde o acesso rápido e confiável é imperativo. Porém esses benefícios vêm em conjunto com a necessidade de atualizar todas as cópias dos itens de dados. Com isso, operações de leitura podem ser executadas mais rapidamente, mas operações de escrita são executadas mais lentamente.

O problema da replicação pode ser considerado bem dominado na área de banco de dados. Os trabalhos mais recentes focam em replicação atrasada enquanto os mais antigos focavam em replicação imediata [ABKW98]. O enfoque atual está bastante direcionado para aplicações em larga escala [LiHD98].

Neste trabalho será apresentado um estudo teórico sobre técnicas de replicação de dados, abordando algumas técnicas de replicação imediata e alguns algoritmos utilizando o esquema de replicação atrasada, objetivando ressaltar as vantagens e os custos de cada método.