



UNITAU
Universidade de Taubaté





Banco de Datos Distribuído

The background features an abstract graphic design. It includes several bright blue, three-dimensional rectangular blocks of varying sizes. Interspersed among these blocks are several white, glossy spheres of different diameters. The entire composition is set against a solid black background. In the lower right area, there are wavy, concentric lines in a light blue or cyan color, creating a sense of motion or data flow.

Banco de Dados Distribuídos



A tecnologia de sistemas de banco de dados distribuídos é a união dos sistemas de banco de dados com as redes de computadores.

Os sistemas de banco de dados permitem que as aplicações tenham independência da organização física e lógica dos dados. Com grande desenvolvimento das tecnologias de redes e comunicação de dados e o avanço na computação distribuída, os sistemas de banco de dados distribuídos trouxeram um melhor aproveitamento dos recursos computacionais e permitem que os dados e o processamento sejam distribuídos entre diferentes sites e/ou servidores proporcionando maior velocidade e disponibilidade no acesso a esses dados.

Banco de Dados Distribuídos



Um sistema de computação distribuída consiste em em diversos elementos autônomos de processamento (não necessariamente homogêneos) que são interconectados por uma rede e que cooperam entre si na execução de suas tarefas. Esses sistemas permitem distribuir a lógica de processamento, as funções, os dados e o controle de execução.

A aplicação desse conceito aos bancos de dados permite que tenhamos sistemas onde o acesso aos dados seja distribuído entre diversos servidores e/ou que os dados sejam distribuídos entre diversos sites.

Banco de Dados Distribuídos



Temos então duas linhas:

Banco de Dados Paralelos: Procura melhorar o desempenho por meio da execução em paralelo de várias operações. O armazenamento dos dados pode ser realizado de forma distribuída ou não.

Banco de Dados Distribuídos: Os dados são armazenados em sites fisicamente distintos e cada site é gerenciado por um SGBD capaz de executar suas funções independente dos demais sites. O armazenamento distribuído privilegia o tempo de acesso a dados que podem estar armazenados localmente onde são mais acessados e a disponibilidade, possibilitando que os demais SGBDs continuem a responder mesmo quando algum nó não estiver funcionando.

Banco de Dados Paralelos



O processamento paralelo explora os computadores de vários processadores para executar programas utilizando diversos processadores de modo cooperativo, a fim de melhorar o desempenho. A ideia básica dos bancos de dados paralelos é executar etapas de avaliação em paralelo sempre que possível.

Arquitetura



Três arquiteturas principais foram propostas para SGBDs paralelos:

Memória Compartilhada: várias CPUs compartilham a mesma memória principal, essa arquitetura também é chamada de “tudo compartilhado”

Disco Compartilhado: cada CPU possui sua própria memória principal e os discos são compartilhados pelas CPUs

Nada Compartilhado: cada CPU tem sua própria memória principal e discos, toda comunicação entre as CPUs se dá pela rede

Arquitetura



A arquitetura de memória compartilhada é a mais próxima de um computador convencional e portar um SGBD para essa arquitetura é mais fácil. Embora essa arquitetura possa obter melhor desempenho em escala moderada de paralelismo, quando o número de CPUs aumenta muito, passa a existir uma competição entre as CPUs pelo acesso à memória. O mesmo problema ocorre com o aumento de CPUs em arquiteturas de disco compartilhado. A partir de um certo ponto, a velocidade de cada CPU diminui a cada CPU adicionada devido à competição pelo acesso à memória e aos discos.

Portar um SGBD para a arquitetura nada compartilhado é mais difícil, porém essa arquitetura tem um aumento de desempenho linear à medida que mais CPUs são adicionadas.

Avaliação de Consultas



O plano de execução de uma consulta relacional é um grafo de operadores algébricos e esses operadores podem ser executados em paralelo. Se um operador consome a saída de outro operador, temos o paralelismo em pipeline (a saída do segundo operador é utilizada pelo primeiro operador assim que é gerada). O paralelismo em pipeline é limitado pela presença de operadores que bloqueiam (operadores que não produzem nenhuma saída até consumir todas as entradas) como por exemplo ordenação e agregação.

Avaliação de Consultas



Outra forma de paralelização das consultas é a avaliação em paralelo de dados particionados, que permite a execução de um operador em paralelo ao dividir o processamento desse operador em diferentes partições. Um dos motivos de sucesso dos banco de dados paralelos em arquiteturas nada compartilhado é que a avaliação de consultas relacionais é muito receptiva à avaliação em paralelo de dados particionados.

Banco de Dados Distribuídos



Um banco de dados distribuídos é uma coleção de vários bancos de dados logicamente inter-relacionados, distribuídos por uma rede de computadores. Um sistema de gerenciamento de banco de dados distribuídos é um sistema de software que permite o gerenciamento de bancos de dados distribuídos e que torna essa distribuição transparente para os usuários.

Banco de Dados Distribuídos



As formas de distribuição dos dados entre os nós pode ser feita por:

Fragmentação: os dados de uma relação são divididos entre diferentes sites de armazenamento, a principal característica da fragmentação é permitir a manipulação mais rápida de dados que tem maior relevância em sites de armazenamento específicos.

Replicação: os dados de uma relação são copiados para diferentes sites de armazenamento, um dos benefícios da replicação é permitir a disponibilidade dos dados replicados caso o nó responsável por esses dados deixe de funcionar.

Vantagens



As principais vantagens dos bancos de dados distribuídos são:

Desempenho: a distribuição dos dados possibilita que esses dados estejam armazenados nos sites onde serão utilizados, evitando os atrasos do acesso remoto, além disso, as consultas podem ser paralelizadas entre os diversos nós do banco dados

Disponibilidade: a existência de diversos nós permite que o banco de dados continue disponível mesmo com a queda de um dos nós, com a replicação dos dados, é possível que todo o banco seja capaz de continuar disponível mesmo com a queda de um ou mais nós

Vantagens



Expansão: expandir um banco de dados distribuídos composto de servidores de pequeno ou médio porte, acrescentando mais nós, pode ser muito mais simples do que expandir um servidor de banco de dados centralizado de grande porte

Propriedades



A visão clássica de um sistema de banco de dados distribuídos é que o sistema deve tornar o impacto da distribuição dos dados transparente. Em particular as seguintes propriedades são consideradas desejáveis:

Propriedades



Independência dos dados distribuídos: Os usuários devem ser capazes de fazer consultas sem especificar onde as relações (ou cópias ou fragmentos das relações) estejam localizadas, esse princípio é uma extensão natural da independência física e lógica. As consultas devem ser otimizadas considerando os custos de comunicação e de computação dos diversos nós. Esses princípios levam a transparência de rede, de replicação e de fragmentação.

Propriedades



Transparência de rede: Os usuários devem ser protegidos contra os detalhes operacionais da rede, é desejável ocultar até mesmo a existência da rede, se possível. A transparência de rede pode ser dividida em transparência de localização, que se refere ao fato de que o comando usado para executar uma tarefa é independente tanto da localização dos dados quanto do sistema em que a operação é executada, e a transparência de nomenclatura, que significa que os nomes dos objetos do banco de dados devem ser únicos e sem dependência da localização do objeto.

Propriedades



Transparência de replicação: Não deve ser necessário que os usuários saibam da existência de cópias dos dados e se estão acessando os dados originais ou uma das cópias. O sistema deve tratar do gerenciamento de cópias e o usuário deve agir como se houvesse uma única cópia dos dados.

Propriedades



Transparência de fragmentação: Quando os objetos do banco de dados estão fragmentados, as consultas dos usuários que foram especificadas sobre relações inteiras têm que ser executadas em sub relações. As estratégias de processamento das consultas devem ser determinadas pelo sistema sem a participação ou conhecimento do usuário .

Propriedades



Atomicidade da transação distribuída: Os usuários devem ser capazes de escrever transações que acessam e atualizam dados em diversos sites como se estivessem armazenados localmente. As transações distribuídas devem seguir o mesmo princípio de atomicidade das transações locais

Propriedades



Embora essas propriedades sejam desejáveis, em certas situações essas propriedades não são eficientemente atingíveis (como por exemplo quando os nós são conectados por uma rede lenta), O custo para se atingir essas propriedades pode tornar necessário flexibilizar ou ignorar algumas dessas questões.

Tipos de Bancos de Dados Distribuídos



Se os dados são distribuídos, mas todos servidores executam o mesmo SGBD, temos um **banco de dados distribuído homogêneo**. Se diferentes sites executam diferentes SGBDs e são conectados de forma que permita o acesso aos dados a partir de vários sites, temos um sistema de **banco de dados distribuídos heterogêneo** ou sistema de múltiplos banco de dados.

Tipos de Bancos de Dados Distribuídos



Para integrar diferentes SGBDs em um sistema heterogêneo é necessário um protocolo de gateway, que é uma API que expõe funcionalidades do SGBD para aplicativos externos (por exemplo ODBC ou JDBC). O uso de protocolos de gateway mascaram as diferenças entre os SGBDs. Porém nem todas as diferenças podem ser mascaradas, um determinado SGBD pode não implementar alguma característica.

O gerenciamento de dados distribuídos tem um custo significativo em termos de desempenho e complexidade. Os protocolos de gateway acrescentam uma camada a mais de processamento tornando esse custo ainda maior.

Fatores de Complicação



Os problemas encontrados em sistemas de banco de dados tornam-se ainda mais complexos em um ambiente distribuído.

- **A replicação dos dados em diferentes sites faz com que o sistema seja responsável por escolher uma das cópias dos dados para acesso em caso de recuperação e também por assegurar que o efeito de uma alteração se refletirá em toda e qualquer cópia desse item de dados.**
- **Se houver falha em alguns sites ou se alguns links de comunicação falharem enquanto uma atualização estiver sendo executada, o sistema deverá assegurar que os efeitos da alteração se refletirão nos dados residentes nesses sites tão logo o sistema possa se recuperar da falha.**

Fatores de Complicação



- Como cada site não pode ter informações instantâneas sobre ações que estão sendo realizadas no momento em outros sites, a sincronização de transações em vários sites é consideravelmente mais difícil do que em um sistema centralizado.
- Os sistemas distribuídos exigem hardware e meios de comunicação adicionais. Quando os servidores são instalados em sites distintos, torna-se necessário empregar mais profissionais para manter essas instalações. Assim, benefícios da utilização de um sistema distribuído devem ser pesados contra o aumento de custos dessa distribuição.

Fatores de Complicação



- **A distribuição dos dados entre diversos sites e o uso intensivo de redes, geralmente redes de longa distância, para trafegar esses dados aumenta muito a dificuldade de manter a segurança em sistemas distribuídos.**

Áreas de Problemas



Além dos problemas já existentes nos banco de dados centralizados, as complicações do ambiente distribuído levam a problemas adicionais que devem ser considerados na implementação de banco de dados distribuídos.

Projeto de Banco de Dados Distribuídos



A questão é como o banco de dados e os aplicativos devem ser distribuídos pelos sites. Há duas alternativas básicas para posicionar os dados: particionados e replicados. Projetos replicados podem ser totalmente replicados, nos quais o banco de dados inteiro é armazenado em todos os sites ou parcialmente replicados, onde cada partição do banco de dados é armazenado em mais de um site mas não em todos eles. A determinação da forma como o banco de dados será fragmentado e a distribuição ótima desses fragmentos deve ser feita de modo a minimizar o custo combinado de armazenar o banco de dados, processar as transações e a comunicação e ainda assim atender as expectativas de desempenho e disponibilidade esperadas.

Processamento Distribuído de Consultas



O processamento de consultas lida com algoritmos que analisam as consultas e as convertem em uma série de operações de manipulação de dados. O problema é como decidir sobre uma estratégia de execução através da rede da forma mais econômica, seja qual for o custo definido. Os fatores a serem considerados são a distribuição de dados, os custos de comunicação e a falta de informações suficientes disponíveis no site. O objetivo é otimizar o paralelismo para melhorar o desempenho.

Gerenciamento de Diretórios Distribuídos



Um diretório contém informações sobre itens do banco de dados (como por exemplo descrição e localização). Os problemas relacionados ao gerenciamento de diretórios são semelhantes aos problemas da distribuição dos dados. Um diretório pode ser global para o SBDD inteiro ou local para cada site. Pode estar centralizado em um único site ou distribuído entre vários sites. Pode haver uma única cópia ou várias cópias .

Controle Distribuído de Concorrência



O controle de concorrência envolve a sincronização de acessos ao banco de dados distribuídos, de tal forma que a integridade do banco de dados seja mantida. O problema de controle de concorrência em um contexto distribuído é diferente do que surge em uma estrutura centralizada. É necessário não apenas se preocupar com a integridade em um único banco de dados, mas também com a consistência de várias cópias do banco de dados. A condição que estabelece que todos os valores de várias cópias de cada item de dados têm de convergir para o mesmo valor é chamada consistência mútua. O controle de concorrência pode ser feito de forma pessimista, que sincroniza a execução das solicitações do usuário antes de iniciar a execução, e o otimista, que executa as solicitações e depois verifica se a consistência compromete a integridade dos dados.

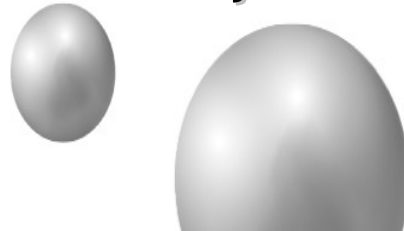
Gerenciamento Distribuído de Impasses



O problema de impasses em SBDDs tem natureza semelhante ao encontrado em sistemas operacionais. A competição entre usuários pelo acesso a um conjunto de recursos pode resultar em um impasse se o mecanismo de sincronização se basear em bloqueios.

Confiabilidade em Bancos de Dados Distribuídos

É importante que sejam fornecidos mecanismos para assegurar a consistência do banco de dados e também para detectar defeitos e se recuperar deles. A implicação em SBDDs é que quando ocorre um defeito e vários sites se tornam inoperantes ou inacessíveis, os bancos de dados nos sites que permanecem operacionais continuam consistentes e atualizados. Além disso, quando o sistema do computador ou da rede se recuperar da falha, o SBDD deve ser capaz de recuperar e manter atualizados os bancos de dados nos sites onde ocorreu a falha. Isso pode ser especialmente difícil no caso de particionamento de redes, em que sites são divididos em dois ou mais grupos sem nenhuma comunicação entre eles.



Suporte do Sistema Operacional



A implementação de sistemas de banco de dados distribuídos sobre sistemas operacionais convencionais sofre as consequências de gargalos de desempenho. O suporte oferecido pelos sistemas operacionais para operações de banco de dados não corresponde propriamente aos requisitos do software de gerenciamento de banco de dados. Os principais problemas são o gerenciamento de memória, sistemas de arquivos e os métodos de acesso, a recuperação de falhas e o gerenciamento de processos. Em ambientes distribuídos surge o problema adicional de lidar com várias camadas de software de rede.

Bancos de Dados Heterogêneos



Quando não existe nenhuma homogeneidade entre os bancos de dados em vários sites, seja em termos do modo como os dados estão estruturados logicamente (modelo de dados) ou em termos dos mecanismos fornecidos para acessá-los (linguagem de dados), torna-se necessário oferecer um mecanismo de conversão entre sistemas de banco de dados. Em geral, esse mecanismo de conversão envolve uma forma canônica para facilitar a conversão de dados, bem como modelos de programas para converter instruções de manipulação de dados.

Arquitetura



Três estratégias alternativas são utilizadas para separar funcionalidades entre diferentes processos relacionados ao SGBD, essas arquiteturas de SGBD distribuído são chamadas cliente-servidor, servidor colaborador e middleware.

Sistemas Cliente Servidor



O termo cliente/servidor aplicado aos banco de dados distribuídos tem um significado semelhante ao utilizado na computação cliente/servidor. A ideia geral é separar as funcionalidades que precisam ser oferecidas em duas classes, funções de servidor e funções de cliente. Essa separação cria uma arquitetura em dois níveis que simplifica a implementação e o gerenciamento dos SGBDs. Porém, essa separação não se refere apenas a processos mas sim a máquinas reais. Onde teremos então máquinas clientes e máquinas servidoras.

O servidor é responsável pela maior parte do trabalho de gerenciamento dos dados. Todo o processamento e otimização de consultas, o gerenciamento de transações e gerenciamento de armazenamento é feito pelo servidor.

Sistemas Cliente Servidor



O cliente é responsável pela interface com o usuário e pelo gerenciamento de dados que podem ser colocados no cache do SGBD cliente.

Essa arquitetura é bastante comum em sistemas relacionais, nos quais a comunicação entre os clientes e o(s) servidor(es) é feita por instruções SQL.

Sistemas de Servidor Colaborador



A arquitetura cliente-servidor não permite que uma única consulta abranja vários servidores, pois o cliente teria de ser capaz de subdividir tal consulta nas subconsultas apropriadas, para serem executadas em diferentes sites e, depois, reunir as respostas das subconsultas. Portanto, o processo cliente seria muito complexo e seus recursos começariam a se sobrepor aos do servidor. Uma alternativa a essa situação é um sistema de servidores colaboradores. Podemos ter uma coleção de servidores de banco de dados, cada um capaz de executar transações com os dados locais, os quais executam cooperativamente as transações que abrangem vários servidores.

Sistemas de Servidor Colaborador



Quando um servidor recebe uma consulta que exige acesso aos dados que estão em outros servidores, ele gera subconsultas apropriadas para serem executadas pelos outros servidores e reúne os resultados para computar as respostas da consulta original. De preferência, a decomposição da consulta deve ser feita usando-se otimização baseada em custo levando em conta o custo de comunicação na rede, assim como os custos de processamento local.

Sistemas de Middleware



A arquitetura de middleware é projetada para permitir que uma única consulta abranja vários servidores, sem exigir que todos os servidores de banco de dados sejam capazes de gerenciar tais estratégias de execução em vários sites. Ela é particularmente atraente ao se tentar integrar vários sistemas legados, cujos recursos básicos não podem ser estendidos.

A ideia é de que precisamos de apenas um servidor de banco de dados capaz de gerenciar consultas e transações que abranjam vários servidores. Os servidores restantes só precisam manipular consultas e transações locais.

Sistemas de Middleware



Podemos considerar esse servidor especial como uma camada de software que coordena a execução de consultas e transações em um ou mais servidores do banco de dados independentes. Tal software é frequentemente chamado de middleware. A camada de middleware é capaz de executar junções e outras operações relacionais em dados obtidos de outros servidores, mas, normalmente ela própria não mantém dados.

Armazenagem de Dados



Em um SGBD distribuído, as relações são armazenadas em vários sites. O acesso a uma relação armazenada em um site remoto acarreta custos de passagem de mensagens. Para reduzir essa sobrecarga, uma relação pode ser particionada ou fragmentada em vários sites, com os fragmentos armazenados nos sites onde são mais frequentemente acessados ou replicados em cada site onde a relação tem alta demanda.

Fragmentação



A fragmentação consiste em subdividir uma relação em relações menores ou fragmentos e armazenar os fragmentos (em vez da relação em si), possivelmente em diferentes sites. Na fragmentação horizontal, cada fragmento consiste em um subconjunto de linhas da relação original. Na fragmentação vertical, cada fragmento consiste em um subconjunto de colunas da relação original.

Fragmentação



Normalmente, as tuplas pertencentes a determinado fragmento horizontal são identificadas por uma consulta de seleção. Por exemplo, as tuplas de funcionários podem ser organizadas em fragmentos por cidade, com todos os funcionários de determinada cidade atribuídos ao mesmo fragmento. Armazenando fragmentos no banco de dados na cidade correspondente, obtemos a localidade de referencia, os dados provavelmente serão atualizados e consultados a partir da cidade onde estão localizados, reduzindo o custo de comunicação para a maioria das consultas.

Analogamente, as tuplas de determinado fragmento vertical são identificadas por uma consulta de projeção.

Fragmentação



Quando uma relação é fragmentada, devemos ser capazes de recuperar a relação original a partir dos fragmentos:

Fragmentação horizontal: a união dos fragmentos horizontais deve ser igual à relação original. Normalmente, os fragmentos também são obrigados a ser disjuntos.

Fragmentação vertical: a coleção de fragmentos verticais deve ser uma decomposição sem perda de junção.

Fragmentação



Para garantir que uma fragmentação vertical seja sem perda de junção, os sistemas frequentemente atribuem um único id a cada tupla da relação original e anexam esse id na projeção da tupla em cada fragmento. Se considerarmos a relação original contendo um campo id de tupla adicional que é uma chave, esse campo será adicionado a cada fragmento vertical. É garantido que tal decomposição é sem perda de junção.

Uma alternativa ao uso do id de tupla é incluir a chave primária da relação em cada fragmento vertical.

Em geral uma relação pode ser fragmentada, horizontal ou verticalmente, e cada fragmento pode ser fragmentado novamente, possibilitando uma fragmentação híbrida.

Replicação



Replicação consiste em armazenar várias cópias de uma relação ou fragmento de relação. Uma relação inteira pode ser replicada em um ou mais sites. Analogamente, um ou mais fragmentos de uma relação podem ser replicados em outros sites. Por exemplo, se uma relação R é fragmentada em R1, R2 e R3, poderia haver apenas uma cópia de R1, enquanto R2 é replicada em dois outros sites e R3 é replicada em todos os sites.

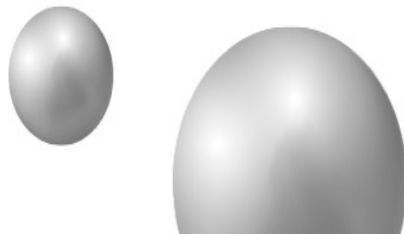
Replicação



As motivações para replicação são:

- **Maior disponibilidade dos dados:** se um site que contém uma réplica fica inativo, podemos encontrar os mesmos dados em outros sites. Analogamente, se cópias locais de relações remotas estão disponíveis, ficamos menos vulneráveis às falhas de comunicação.
- **Avaliação de consultas mais rápido:** as consultas podem ser executadas mais rapidamente usando uma cópia local de uma relação, em vez de ir até um site remoto.

A replicação pode ser síncrona, onde todas as cópias são atualizadas na mesma transação ou assíncrona, onde as cópias são atualizadas apenas periodicamente.



Atualização de Dados



A visão clássica de um SGBD distribuído é a de que, do ponto de vista do usuário, ele deve se comportar exatamente como um SGBD centralizado. Os problemas que surgem com a distribuição dos dados devem ser transparentes para o usuário.

Com relação às atualizações, essa visão significa que as transações devem continuar a ser ações atômicas, independentes da fragmentação e da replicação de dados. Em particular, todas as cópias de uma relação modificada devem ser atualizadas antes que a transação modificadora seja efetivada. Referimo-nos à replicação, como replicação síncrona, ou seja, antes que uma transação de atualização seja efetivada, ela sincroniza todas as cópias dos dados modificados.

Atualização de Dados



Uma estratégia alternativa, chamada replicação assíncrona, tornou-se amplamente usada nos SGBDs distribuídos comerciais. Nessa estratégia, as cópias de uma relação modificada são atualizadas apenas periodicamente e uma transação que leia cópias diferentes da mesma relação pode obter valores diferentes. Assim, a replicação assíncrona compromete a independência dos dados distribuídos, mas pode ser implementada mais eficientemente do que a replicação síncrona.

Replicação Síncrona



Existem duas técnicas básicas para garantir que as transações vejam o mesmo valor, independentemente da cópia de um objeto que acessam. Na primeira técnica, chamada de **votação**, uma transação deve gravar a maioria das cópias para modificar um objeto e ler pelo menos cópias suficientes para garantir que uma das cópias esteja atualizada. Por exemplo, se existirem 10 cópias e 7 são gravadas por transações de atualização, então pelo menos 4 cópias devem ser lidas. Cada cópia tem um número de versão e a cópia com o número de versão mais alto está atualizada. Essa técnica não é atraente na maioria das situações, pois ler um objeto exige ler várias cópias. Na maioria das aplicações, os objetos são lidos muito mais frequentemente do que são atualizados e um desempenho eficiente nas leituras é muito importante.

Replicação Síncrona



Na segunda técnica chamada **ler qualquer uma, gravar todas**, para ler um objeto, uma transação pode ler qualquer cópia, mas para gravar um objeto, ela deve gravar todas as cópias. As leituras são rápidas, especialmente se tivermos uma cópia local, mas as gravações são mais lentas em relação a primeira técnica. Essa técnica é mais atraente quando as leituras são muito mais frequentes do que as gravações e é normalmente adotada nas implementações de replicação síncrona.

Replicação Síncrona

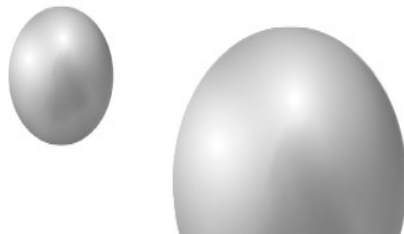


A replicação síncrona tem um custo significativo. Antes que uma transação de atualização possa ser efetivada, ela deve obter bloqueios exclusivos sobre todas as cópias dos dados modificados, supondo que seja usada a técnica do ler qualquer uma, gravar todas. Talvez a transação precise enviar pedidos de bloqueio para sites remotos, esperar que os bloqueios sejam concedidos e, durante esse período potencialmente longo, continuar a manter todos os seus outros bloqueios. Se os sites ou links de comunicação falharem, a transação não poderá ser efetivada até que todos os sites em que ela tenha modificado dados se recuperem e possam ser acessados. Finalmente, mesmo que todos os bloqueios possam ser prontamente e não haja nenhuma falha, a efetivação de uma transação exige o envio de várias mensagens adicionais, como parte de um protocolo de efetivação.

Replicação Assíncrona



Devido ao custo e complicações, nem sempre é desejável ou possível utilizar a replicação síncrona. A replicação assíncrona está ganhando popularidade, mesmo permitindo que diferentes cópias do mesmo objeto tenham valores diferentes por curto período de tempo. Essa situação viola o princípio da independência de dados distribuídos, os usuários precisam saber qual cópia estão acessando, reconhecer que cópias são atualizadas apenas periodicamente e conviver com esse nível de consistência de dados reduzido.



Replicação Assíncrona



Existem dois tipos de replicação assíncrona:

Replicação assíncrona de site primário - a cópia de uma relação é designada como primária ou mestra. Réplicas da relação inteira ou fragmentos da relação podem ser criados em outros sites, essas são cópias secundárias e, ao contrário da cópia primária, não podem ser atualizadas.

Replicação assíncrona ponto a ponto - mais de uma cópia (embora, talvez, não todas) pode ser designada como atualizável, ou seja, uma cópia mestra.

Replicação Assíncrona Ponto a Ponto



Na replicação assíncrona ponto a ponto, além de propagar as alterações, uma estratégia de solução de conflitos deve ser usada para tratar as alterações conflitantes feitas em diferentes sites. Por exemplo, a idade de um funcionário pode ser alterada para 35 em um site e para 38 em outro.

Replicação Assíncrona Ponto a Ponto



Algumas situações especiais, nas quais a replicação ponto a ponto não leva a conflitos, surgem muito frequentemente e, nesse caso, ela é utilizada em sua melhor forma. Por exemplo:

- **Cada mestre pode atualizar apenas um fragmento (normalmente um fragmento horizontal) da relação e quaisquer fragmentos atualizáveis por diferentes sites são disjuntos. Por exemplo, pode ser que o salário dos funcionários alemães seja atualizável apenas em Frankfurt e os salários dos funcionários indianos atualizáveis apenas em Madras, mesmo que a relação inteira esteja armazenada tanto em Frankfurt quanto em Madras.**

Replicação Assíncrona Ponto a Ponto



- Os direitos de atualização são mantidos por apenas um mestre por vez. Por exemplo, um site é designado como backup de outro site. As alterações feitas no site mestre são propagadas para outros e não são permitidas atualizações em outros sites (incluindo o backup). Mas se o site mestre falhar, o site de backup assumirá o controle, e agora, as atualizações serão permitidas apenas nesse site.

Replicação Assíncrona de Site Primário



O principal problema na implementação da replicação de site primário é determinar como as alterações feitas na cópia primária são propagadas para as cópias secundárias. Normalmente as alterações são propagadas em duas etapas, chamadas **Captura** e **Aplicação**.

As alterações feitas por transações efetivadas na cópia primária são identificadas de algum modo na etapa de Captura e, subsequentemente, propagadas para as cópias secundárias durante a etapa de Aplicação.

Em contraste com a replicação síncrona, uma transação que modifique diretamente uma relação replicada bloqueia e altera apenas a cópia primária. Normalmente ela é efetivada muito antes que a etapa de Aplicação seja executada.

Replicação Assíncrona de Site Primário



A etapa de Captura pode ser implementada utilizando duas estratégias.

- Na **Captura baseada em log**, o log mantido para propósito de recuperação é utilizado para manter um registro da atualizações. Esse log é gravado em uma Tabela de Alteração de Dados (**TAD**) e a etapa de Aplicação utiliza a TAD para propagar as alterações para os demais sites.
- Na **Captura procedural**, um procedimento ativado automaticamente pelo SGBD ou por programa aplicativo inicia a etapa de Captura, o qual normalmente consiste em tirar um snapshot da cópia primária.

Replicação Assíncrona de Site Primário



A Captura baseada em log tem uma sobrecarga menor do que a Captura procedural, porém, exige um entendimento detalhado do log, que é muito específico para cada sistema. Portanto, um fabricante não pode implementar facilmente um mecanismo de Captura baseado em log que capture alterações feitas nos dados de um SGBD de outro fabricante.

Replicação Assíncrona de Site Primário



A etapa de Aplicação pega as alterações coletadas pela etapa de Captura que podem estar na TAD ou em um snapshot, e as propaga para os sites secundários. Isso pode ser feito fazendo o site primário enviar continuamente a TAD ou com os sites secundários solicitando periodicamente a TAD ou o snapshot para o site primário. Normalmente, cada site secundário executa uma cópia do processo de Aplicação e “puxa” uma cópia da TAD ou do snapshot usando solicitações periódicas. O intervalo entre essas solicitações pode ser fixo ou controlado por um programa aplicativo do usuário.

Replicação Assíncrona de Site Primário



A Captura baseada em log, em conjunto com a Aplicação contínua, minimiza o atraso na propagação das alterações. Essa é a melhor combinação em situações em que as replicas devem ser o mais sincronizadas possível com a copia primária. Basicamente, a Captura baseada em log e a Aplicação contínua é uma alternativa menos dispendiosa para a replicação síncrona.

A Captura procedural e a Aplicação dirigida por aplicativo oferecem a maior flexibilidade. Frequentemente essa flexibilidade é útil em aplicações de Data Warehousing, onde a capacidade de filtrar dados recuperados é mais importante do que o fato da réplica ser atualizada.

Processamento de Consultas



O sucesso da tecnologia de banco de dados relacionais se deve, em parte, à disponibilidade de linguagens não procedurais como SQL, que podem melhorar de forma significativa o desenvolvimento de aplicações e a produtividade do usuário final. Ao ocultar os detalhes de baixo nível a respeito da organização física dos dados, as linguagens de banco de dados relacionais permitem a expressão de consultas complexas de modo conciso e simples. Em particular, para construir a resposta à consulta, o usuário não especifica o procedimento a ser seguido.

Processamento de Consultas



Esse procedimento é, na realidade, projetado por um módulo do banco do SGBD, em geral chamado de processador de consultas. Isso libera o usuário da necessidade de realizar a otimização da consulta, uma tarefa demorada que é realizada melhor pelo processador de consultas, pois ele é capaz de explorar uma grande quantidade de informações sobre o banco de dados.

Como se trata de uma questão crítica de desempenho, o processamento de consultas recebe considerável atenção no contexto dos SGBDs centralizados e distribuídos. Contudo, o problema de processamento de consultas é muito mais difícil em ambientes distribuídos, pois um número maior de parâmetros afeta o desempenho de consultas distribuídas.

Processamento de Consultas



Em um contexto centralizado, as estratégias de execução de consultas podem ser expressas em uma extensão da álgebra relacional. A principal função de um processador de consultas centralizado é escolher, para uma dada consulta, a melhor consulta de álgebra relacional entre todas as equivalentes. Devido a complexidade desse problema principalmente quando a consulta envolve um grande número de relações, em geral ele é reduzido à escolha de uma solução próxima da solução ótima.

Processamento de Consultas



Em um sistema distribuído, a álgebra relacional não é suficiente para expressar as estratégias de execução. Ela deve ser complementada com operações para intercâmbio de dados entre sites. Além de ordenar as operações da álgebra relacional, o processador de consultas distribuídas também deve considerar a distribuição dos dados pelos diferentes servidores. Isso aumenta muito o espaço de solução a partir do qual é escolhida a estratégia de execução distribuída, tornando o processamento de consultas distribuídas significativamente mais difícil.

Processamento de Consultas



A consulta algébrica escolhida deve alcançar tanto a correção quanto a eficiência. Ela será correta se a consulta de baixo nível tiver a mesma semântica da consulta original, ou seja, se ambas as consultas produzirem o mesmo resultado. O mapeamento bem definido do cálculo relacional para a álgebra relacional facilita a questão da correção. Porém a produção de uma estratégia de execução eficiente é mais complicada. Uma consulta de cálculo relacional pode ter muitas transformações equivalentes e corretas para a álgebra relacional. Tendo em vista que cada estratégia de execução pode levar a consumos muito diferentes de recursos computacionais como operações de E/S de disco, CPUs e redes de comunicação, a principal dificuldade é selecionar a estratégia de execução que minimize o consumo de recursos.

Processamento de Consultas



O projeto de banco de dados distribuídos é de grande importância para o processamento de consultas, pois a definição de fragmentos se baseia no objetivo de aumentar a localidade de referência e, às vezes, a execução em paralelo das consultas mais importantes.

Transação Distribuída



Em um SGBD distribuído, determinada transação é submetida em algum site, mas também pode acessar dados em outros sites. Quando uma transação é submetida em algum site, o gerenciador de transações desse site a subdivide em um conjunto de uma ou mais subtransações que são executadas em diferentes sites, submete-as aos gerenciadores de transações nos outros sites e coordena suas atividades.

O gerenciamento de transações distribuídas leva a questões sobre o controle de concorrência distribuído e a recuperação distribuída.

Transação Distribuída



Controle de concorrência distribuída – os algoritmos de controle de concorrência podem ser divididos em algoritmos baseados em bloqueios e algoritmos baseados na ordenação das transações, é necessário gerenciar o bloqueio dos objetos armazenados em vários sites e tanto para algoritmos baseados em bloqueio, quanto em algoritmos baseados em ordenação das transações, também é necessário detectar impasses em bancos de dados distribuídos

Recuperação distribuída – a atomicidade da transação deve ser garantida, quando uma transação é efetivada, todas as suas ações, em todos os sites nos quais ela é executada, devem persistir. Analogamente, quando uma transação é cancelada, nenhuma das suas ações deve persistir.

Controle de Concorrência Distribuído



O controle de concorrência trata das propriedades de isolamento e consistência das transações. O mecanismo de controle distribuído da concorrência de um SGDB distribuído assegura que a consistência do banco de dados seja mantida em um ambiente distribuído multiusuário.

Se as transações são consistentes internamente (isto é, não violam nenhuma das restrições de consistência), a maneira mais simples de alcançar esse objetivo é executar cada transação sozinha, uma após a outra. É obvio que essa alternativa não pode ser implementada em um sistema prático, pois ela minimiza o throughput (também denominado vazão) do sistema.

Controle de Concorrência Distribuído



O nível de concorrência (ou seja, o número de transações concorrentes) talvez seja o parâmetro mais importante em sistemas distribuídos. Por essa razão, o mecanismo de controle de concorrência tenta encontrar um equilíbrio adequado entre a manutenção da consistência do banco de dados e a manutenção de um nível elevado de concorrência.

Controle de Concorrência Distribuído



Um dos mecanismos utilizados para controle de concorrência é o bloqueio dos objetos do banco de dados. Quando os bloqueios são obtidos e liberados, é determinado pelo protocolo de controle de concorrência.

O gerenciamento de bloqueios pode ser distribuído entre os sites de muitas maneiras:

Centralizado – um único site é responsável por tratar dos pedidos de bloqueio e desbloqueio de todos os objetos.

Cópia primária – uma cópia de cada objeto é designada como cópia primária. Todos os pedidos de bloqueio e desbloqueio de uma cópia desse objeto são manipulados pelo gerenciador de bloqueios do site onde a cópia primária esta armazenada, independente de onde a cópia esta armazenada.

Controle de Concorrência Distribuído



Totalmente distribuído – os pedidos de bloqueio e desbloqueio de uma cópia de um objeto armazenado em um site são manipulados pelo gerenciador de bloqueios do site onde a cópia esta armazenada.

Controle de Concorrência Distribuído



O esquema centralizado é vulnerável à falha do site único que controla o bloqueio.

O esquema da cópia principal evita esse problema, mas, em geral, ler um objeto exige a comunicação com dois sites: o site onde a cópia primária reside e o site onde reside a cópia a ser lida.

Esse problema é evitado no esquema totalmente distribuído, pois o bloqueio é feito onde a cópia a ser lida reside. Entretanto, durante a gravação, bloqueios devem ser estabelecidos em todos os sites onde as cópias são modificadas, enquanto nos outros esquemas, os bloqueios só precisam ser estabelecidos em um site. O esquema de bloqueio totalmente distribuído é o mais atraente, se leituras são muito mais frequentes do que gravações, como normalmente acontece.

Impasse Distribuído



Uma questão que exige atenção especial ao se usar cópia primária ou bloqueio totalmente distribuído é a detecção de impasses. Assim como em um SGBD centralizado, os impasses devem ser detectados e solucionados por meio do cancelamento de alguma transação que esteja em impasse.

Cada site mantém um grafo de “espera por” e um ciclo em um grafo local indica um impasse. Entretanto pode haver um impasse mesmo que nenhum grafo local contenha um ciclo. Por exemplo, podemos ter uma transação T1 esperando por um bloqueio obtido por uma transação T2 no site A e a transação T2 esperando por um bloqueio obtido pela transação T1 no site B. Dessa forma, nem o site A quanto o site B é capaz de identificar o impasse.

Impasse Distribuído



Para detectar impasses distribuídos, podemos usar três algoritmos:

O primeiro algoritmo, que é centralizado, consiste em enviar periodicamente todos os grafos de “espera por” locais para um único site, que é responsável pela detecção de impasse global. Nesse site, é gerado o grafo de “espera por” global, pela combinação de todos os grafos locais.

Impasse Distribuído



O segundo algoritmo é hierárquico, agrupa os sites em uma árvore. Periodicamente, cada site envia os grafos de “espera por” da sua subárvore para o nó hierarquicamente superior. Dessa forma, os impasses são identificados em um nível hierárquico inferior ou chegará até o grafo de “espera por” global. Esse esquema é baseado na observação de que mais impasses são prováveis entre sites intimamente relacionados do que entre sites não relacionados, e se esforça mais em detectar impasses entre sites relacionados.

Impasse Distribuído



O terceiro algoritmo é simples: se uma transação espera mais tempo do que algum intervalo de tempo de espera escolhido, ela é cancelada. Embora esse algoritmo possa causar muitos reinícios desnecessários, a sobrecarga de detecção de impasses é baixa e, em um banco de dados distribuídos heterogêneo, se os sites participantes não puderem cooperar para ampliar o compartilhamento de seus grafos de “espera por”, ele pode ser a única opção.

Impasse Distribuído



Um ponto sutil a observar com relação à detecção de impasses distribuídos é que os atrasos na propagação de informações locais poderiam fazer o algoritmo de detecção de impasses identificar “impasses” inexistentes. Tais situações, chamadas de impasses fantasmas, levam a cancelamentos desnecessários.