Base de Dados Distribuídas

- Bases de Dados Heterogéneas e Homogéneas
- Armazenagem Distribuída de Dados
- Transacções Distribuídas
- Protocolos de Commit
- Processamento Distribuído de Consultas
- Controlo de Concorrência em Bases de Dados Distribuídas
- Bases de Dados Distribuídas Heterogéneas

Sistema de Base de Dados Distribuída

- Um sistema de bases de dados distribuída consiste num conjunto de sites ligados por uma rede convencional que não partilham nenhum componente físico.
- Os sistemas de bases de dados que correm em cada site são independentes uns dos outros.
- As transacções podem aceder a dados em um ou mais sites.

Bases de Dados Distribuídas Homogéneas

- Numa base de dados distribuída homogénea
  - Todos os sites têm software idêntico
  - Estão conscientes uns dos outros e acordam em cooperar no processamento dos pedidos dos utilizadores.
  - Cada site aceita perder parte da sua autonomia em termos de direitos de mudança de esquema e software.
  - Ao utilizador parece um sistema único.
- Numa base de dados distribuída heterogénea
  - Sites diferentes podem usar esquemas e software diferentes
    - Diferenças nos esquemas constituem problemas complicados no processamento de consultas
    - Diferenças no software constituem problemas complicados no processamento de transacções
  - Os sites podem não estar conscientes uns dos outros e poderão fornecer apenas capacidades limitadas para cooperação no processamento de transacções

Armazenamento de dados distribuído

- Assumir o modelo relacional
- Replicação
  - O sistema mantém cópias múltiplas de dados, guardados em sites diferentes para recolha mais rápida e tolerância a falhas.
- Fragmentação
  - Uma relação é particionada em vários fragmentos guardados em sites distintos
- Replicação e fragmentação podem ser combinadas
  - Uma relação é particionada em vários fragmentos: o sistema mantém várias cópias idênticas de cada um destes fragmentos.

Replicação de dados

- Uma relação ou fragmento de uma relação está **replicado** se está guardado redundantemente em dois ou mais sites.
- Replicação total de uma relação acontece quando uma relação está guardada em todos os sites.
- Bases de dados com redundância total são aquelas onde cada site tem uma cópia de toda a base de dados.

Replicação de dados (Cont.)

- Vantagens da Replicação
  - **Disponibilidade**: a falha de um site contendo a relação *r* não resulta na indisponibilidade de *r* se existirem réplicas.
  - **Paralelismo**: consultas sobre *r* podem ser processadas por vários nós em paralelo.
  - **Redução dos dados a transferir**: uma relação *r* está disponível localmente em cada site que contenha uma réplica de *r*.
- Desvantagens da Replicação
  - Aumento do custo de actualizações: cada réplica de uma relação *r* tem de ser actualizadade.
  - Aumento da complexidade do controlo de concorrência: actualizações concorrentes em réplicas distintas poderão levar a dados inconsistentes a não ser que mecanismos especiais de controlo de concorrência sejam implementados.
    - Uma solução: escolher uma cópia como **cópia primária** e aplicar operações de controlo de concorrência sobre a cópia primária.

Fragmentação de Dados

- Dividir uma relação r em fragmentos r_1, r_2, …, r_n que contêm informação suficiente para reconstruir a relação r.
- **Fragmentação horizontal**: cada tuplo de *r* é atribuído a um ou mais fragmentos
- **Fragmentação vertical**: o esquema de uma relação *r* é dividido em vários esquemas mais pequenos
  - Todos os esquema têm que conter um chave candidata comum (ou superchave) para preservar a propriedade de junção sem perdas.
  - Um atributo especial, o atributo identificador de tuplos, pode ser acrescentado ao esquema de forma a servir de chave candidata.
- Exemplo : relação account com o esquema seguinte:
- *Account-schema* = (*branch-name*, *account-number, balance*)

Fragmentação Horizontal da Relação *account*

branch-name	*account-number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$account_1 = \sigma_{branch\text{-}name=\text{“}Hillside\text{”}}(account)$

branch-name	*account-number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_2 = \sigma_{branch\text{-}name=\text{“}Valleyview\text{”}}(account)$

Fragmentação Vertical da Relação *employee-info*

branch-name	*customer-name*	*tuple-id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\text{-}name,\ customer\text{-}name,\ tuple\text{-}id}(employee\text{-}info)$

account number	*balance*	*tuple-id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\text{-}number,\ balance,\ tuple\text{-}id}(employee\text{-}info)$

Vantagens da Fragmentação

- Horizontal:
  - Permite o processamento paralelo de fragmentos de uma relação
  - Permite que uma relação seja dividida de forma a que os tuplos estejam onde são mais frequentemente acedidos
- Vertical:
  - Permite que os tuplos sejam divididos de forma a que cada parte do tuplo esteja guardada onde é mais frequentemente acedida
  - O atributo de identificação de tuplo permite a junção eficiente de fragmentos verticais
  - Permite o processamento paralelo de uma relação
- A fragmentação vertical e horizontal podem ser combinadas.
  - Os fragmentos podem ser sucessivamente fragmentados até uma determinada profundidade.

Transparência de Dados

- **Transparência de dados**: Nível até ao qual o utilizador do sistema permanece inconsciente dos detalhes de como e onde os dados são guardados num sistema distribuído.
- Considerar questões de transparência em relação a:
  - Transparência de fragmentação
  - Transparência de replicação
  - Transparência de localização

Nomeação de Itens de Dados - Critérios

1. Cada item de dados tem de ter um nome único em todo o sistema.
2. Deve ser possível encontrar a localização dos itens de dados eficientemente.
3. Deve ser possível mudar a localização de itens de dados transparentemente.
4. Cada site deve poder criar novos itens de dados autonomamente.

Esquema Centralizado – Servidor de Nomes

- Estrutura:
  - O servidor de nomes atribui todos os nomes
  - Cada site mantém um registo dos itens de dados locais
  - Os sites pedem ao servidor de nomes para localizar itens de dados não locais
- Vantagens:
  - Satisfaz os critérios de nomeação 1-3
- Desvantagens:
  - Não satisfaz o critério de nomeação 4
  - O servidor de nomes constitui potencialmente um bottleneck em termo de performance
  - O servidor de nomes é um ponto de falha único

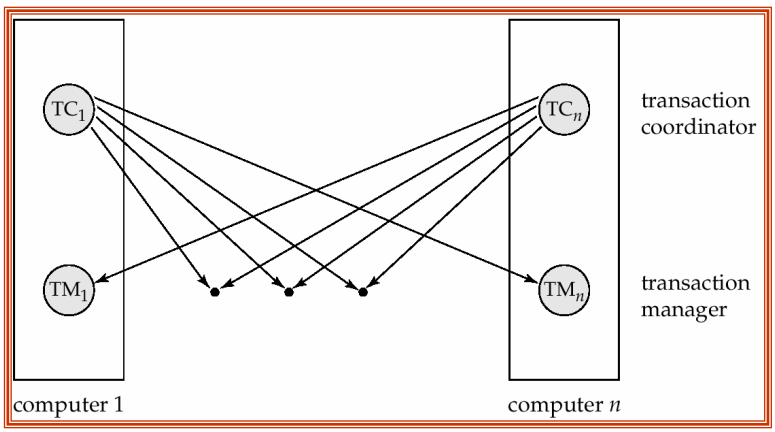
Uso de Aliases

- Como alternativa a um esquema centralizado: cada site prefixa com o seu identificador de site todos os nomes de itens de dados que gera, i.e., *site* 17.a*ccount.*
  - Satisfaz o requisito de identificador único e evita problemas relacionados com controlo central.
  - No entanto não permite obter transparência de rede.
- Solução: Criar um conjunto de **aliases** para itens de dados; Guardar a correspondência de aliases para nomes reais em cada site.
- O utilizador pode não ter consciência da localização física dos itens de dados, e não é afectada se um item de dados é transferido de um site para outro.

Transacções Distribuídas

- Uma transacção pode aceder a dados em vários sites.
- Cada site tem um gestor de transacções local responsável por:
  - Manter um log para efeitos de recuperação
  - Participar na coordenação da execução concorrente de transacções nesse site.
- Cada site tem um coordenador de transacções, que é responsável por:
  - Iniciar a execução de transacções originadas no site.
  - Distribuir subtransacções por site apropriados para execução.
  - Coordenar a conclusão de cada transacção originada no site, que pode resultar no commit da transacção em todos os sites ou na falha em todos os sites.

Arquitectura de um Sistema de Transacções



Tipos de Falhas do Sistema

- Falhas próprias de sistemas distribuídos:
  - Falha de um site.
  - Perda de mensagens
    - Tratada pelos protocolos de rede tais como TCP-IP
  - Falha de um elo de comunicação
    - Tratada pelos protocolos de rede pelo envio de mensagens por elos alternativos
  - **Partição de Rede**
    - Uma rede diz-se **particionada** quando está dividida em dois ou mais subsistemas que não possuem qualquer ligação entre eles
      - Nota: um subsistema pode consistir num só nó
- Partição de rede e falhas de site são normalmente indistinguíveis.

Protocolos de Commit

- Os protocolos de Commit são usados para assegurar a atomicidade entre sites
  - Uma transacção que executa em múltiplos sites tem que fazer o commit em todos os sites ou falhar em todos os sites.
  - Não é aceitável uma mesma transacção fazer o commit num site e falhar noutro
- O protocolo *two-phase commit* (2 *PC*) é amplamente usado
- O protocolo *three-phase commit* (3 *PC*) é mais complicado e pesado, mas evita alguns problemas do protocolo two-phase commit.

Protocolo Two Phase Commit (2PC)

- Assume um modelo **falha-pára** – os sites que falham simplesmente param de trabalhar, e não provocam mais erros, tais como o envio de mensagens incorrectas para outros sites.
- A execução do protocolo é iniciada pelo coordenador após se chegar ao último passo da transacção.
- O protocolo envolve todos os sites onde a transaccao executou
- Seja *T* uma transacção iniciada no site S_i, e seja C_i o coordenador de transacção em S_i

Fase 1: Obtenção de Decisão

- O coordenador pede a todos os participantes para se *prepararem* para fazer o commit da transacção T_i.
  - C_i adiciona o registo <**prepare** *T*> ao log e escreve o log para disco
  - Envia mensagens **prepare** *T* para todos os sites nos quais *T* executou
- Quando recebe uma mensagem, o gestor de transacções no site determina se pode fazer o commit da transacção
  - Se não, adiciona um registo <**no** *T*> ao log e envia a mensagem **abort** *T* para C_i
  - Se a transacção pode fazer o commit, então:
  - Adiciona o registo <**ready** *T*> ao log
  - Força o envio de *todos os registos de T* para disco
  - Envia a mensagem **ready** *T* para C_i

Fase 2: Registo da Decisão

- Pode ser feito o commit para*T* se C_i recebeu uma mensagem **ready** *T* de todos os sites participantes: senão *T* deve ser abortada.
- O coordenador adiciona um registo de decisao, <**commit** *T*> ou <a**bort** *T*>, ao log e força o envio do registo para disco.
- O coordenador envia uma mensagem para cada site participante informando da decisao (commit ou abort)
- Os sites participantes agem localmente de forma apropriada.

Tratamento de Falhas – Falha de Sites

Quando um site S_i recupera após uma falha examina o seu log para determinar o destino das transaccoes activas aquando da falha.

- Log contém o registo <**commit** *T*>: o site executa **redo** (*T*)
- Log contém o registo <**abort** *T*>: o site executa **undo** (*T*)
- Log contém o registo <**ready** *T*>: o site deve consultar C_i para determinar o destino de *T*.
  - Se *T* fez o commit, **redo** (*T*)
  - Se *T* foi abortada, **undo** (*T*)
- Se o log não contém registos respeitantes a *T* significa que S_k falhou antes de responder à mensagem **prepare** *T* de C_i
  - O site S_k tem de abortar *T*
  - S_k executa **undo** (*T*)

Tratamento de Falhas – Falha do Coordenador

- Se o coordenador falha enquanto o protocolo de commit para *T* está a ser executado, então os sites participantes devem decidir o destino de *T*:
  1. Se um site activo contém um registo <**commit** *T*> no seu log, então tem de ser feito o commit para *T*.
  2. Se um site activo contém um registo <**abort** *T*> no seu log, então *T* tem de ser abortada.
  3. Se um site participante activo não contém um registo <**ready** *T*> no seu log, então o coordenador C_i não pode ter decidido fazer o commit de *T*. Podemos pois abortar *T*.
  4. If nenhuma das situações acima acontece, entao todos os sites activos têm de ter um registo <**ready** *T*> nos seus logs, mas mais nenhum registo de controlo adicional (tais como <**abort** *T*> ou <**commit** *T*>). Neste caso todos os sites activos devem esperar que C_i recupere para saber a decisão.
- **Blocking problem** : os sites activos podem ter de esperar pela recuperação do coordenador.

Tratamento de Falhas – Particionamento da Rede

- Se o coordenador e todos os participantes permanecem numa mesma partição, a falha não tem qualquer efeito no protocolo de commit.
- Se o coordenador e os participantes pertencem a várias partições:
  - Os site que não estão na mesma partição do coordenador pensam que este falhou e executam o protocolo de modo a tratar a falha do coordenador.
    - Não produz resultados errados, mas os sites poderão ter de esperar pela decisão do coordenador.
- O coordenador e os sites que estão na mesma partição pensam que os sites em outras partições falharam, e segue o protocolo de commit usual.
    - Mais uma vez, não são produzidos resultados errados

Recuperação e controlo de concorrência

- **Transacções em dúvida** têm um registo <**ready** *T*>, mas não têm nem um <**commit** *T*>, nem um <**abort** *T*> no log.
- O site em recuperação deve determinar o estado commit-abort dessas transacções através do contacto com outros sites; isto pode atrasar e potencialmente bloquear a recuperação.
- Os algoritmos de recuperação podem anotar informação de lock no log.
  - Em vez de <**ready** *T*>, escrever <**ready** *T*, *L*> *L* = lista de locks de *T* quando o log foi escrito (read locks podem ser omitidos).
  - Para cada transacção em dúvida *T*, todos os locks anotados no registo de log <**ready** *T*, *L*> são readquiridos.
- Após reaquisição de lock, o processamento da transacção poder ser concluído; o commit ou rollback de transacções em dúvida é feito concorrentemente com a execução das novas transacções.

Three Phase Commit (3PC)

- Assumimos que:
  - Não há particionamento da rede
  - Em qualquer ponto, pelo menos um site deve estar disponível.
  - No máximo K sites (participantes assim como coordenador) podem falhar
- Fase 1: Obtenção da Decisão Preliminar: Idêntico à fase 1 do 2PC.
  - Todos os sites estão prontos para fazer o commit se instruídos para o fazerem
- A Fase 2 do 2PC é dividida em 2 fases, Fase 2 e Fase 3 do 3PC
  - Na fase 2 o coordenador toma a decisão como no 2PC (chamada a pre-commit decision) e regista a decisão em vários (pelo menos K) sites
  - Na Fase 3, o coordenador envia uma mensagem de commit/abort para todos os sites participantes,
- No 3PC, o conhecimento da decisão pre-commit pode ser usado para fazer o commit apesar da falha do coordenador
  - Evita o blocking problem desde que < K sites falhem
- Desvantagens:
  - Maiores overheads
  - Assumpções poderão não ser satisfeitas na prática.

Processamento de Consultas Distribuídas

- Para sistemas centralizados, o critério primário para medir o custo de uma determinada estratégia é o número de acessos a disco.
- Num sistema distribuído, outros aspecto devem ser tidos em conta:
  - O custo da transmissão de dados pela rede.
  - O ganho potencial em performance pelo facto de ter vários sites a processar partes de uma consulta em paralelo.

Transformação de consultas

- Tradução de consultas algébricas sobre os fragmentos.
  - Tem de ser possível reconstruir uma relação *r* dos seus fragmentos
  - Substituir a relação *r* pela expressão que constroi a relação *r* dos seus fragmentos
- Considerem a fragmentação horizontal da relação *account* em

  $account_1 = \sigma_{branch\text{-}name = \text{“Hillside”}}(account)$

  $account_2 = \sigma_{branch\text{-}name = \text{“Valleyview”}}(account)$
- A consulta query $\sigma_{branch\text{-}name = \text{“Hillside”}}(account)$ fica

  $\sigma_{\text{branch-name = “Hillside”}}(account_1 \cup account_2)$

  que é optimizada para

  $\sigma_{branch\text{-}name = \text{“Hillside”}}(account_1) \cup \sigma_{\text{branch-name = “Hillside”}}(account_2)$

Consulta Exemplo (Cont.)

- Uma vez que $account_1$ tem apenas tuplos respeitantes à agência Hillside, podemos eliminar a operação de selecção.
- Aplicar a definição de $account_2$ para obter

  σ *branch-name* = "Hillside" (σ *branch-name* = "Valleyview" (*account*)
- Esta expression retorna o conjunto vazio independentemente do conteúdo da relação *account*.
- A estratégia final é a de o site de Hillside retornar $account_1$ como resultado da consulta.

Processamento de junção simples

- Considere a seguinte expressão da álgebra relacional onde as 3 relações não se encontram replicadas nem fragmentadas

  $account \bowtie depositor \bowtie branch$
- *account* está guardada no site S_1
- *depositor* em S_2
- *branch* em S_3
- Para uma consulta emitida em S_I, o sistema tem que produzir o resultado no site S_I

Possíveis estratégias de processamento de consultas

- Enviar cópias das três relações para o site S_I e escolher uma estratégia para processar a consulta inteira localmente em S_I.
- Enviar uma cópia da relação account para S_2 e calcular $temp_1$ = *account* ⋈ *depositor em* S_2. Enviar $temp_1$ de S_2 para S_3, e calcular $temp_2$ = $temp_1$ ⋈ *branch* at S_3. Enviar o resultado $temp_2$ para S_I.
- É possível definir estratégias similares mudando os papéis de S_1, S_2, S_3
- É necessário considerar os seguintes factores:
  - Quantidade de dados a enviar
  - Custo de transmissão de dados entre sites
  - Velocidade relativa de processamento em cada site

Estratégia de Semijoin

- Seja r_1 uma relação com esquema R_1 guardada no site S_1
- Seja r_2 uma relação com esquema R_2 guardada no site S_2
- Avaliar a expressão $r_1 \bowtie r_2$ e obter o resultado em S_1.
- 1. Calcular $temp_1 \leftarrow \prod_{R1 \cap R2}$ (r1) em $S1$.
- 2. Enviar $temp_1$ de S_1 para S_2.
- 3. Calcular $temp_2 \leftarrow r_2 \bowtie$ temp1 em S_2
- 4. Enviar $temp_2$ de S_2 para S_1.
- 5. Calcular $r_1 \bowtie temp_2$ em S_1. Isto é o mesmo que $r_1 \bowtie r_2$.

Definição formal

- O **semijoin** de r_1 com r_2, é denotado por:

  $r_1 \ltimes r_2$

- É definido por:
- $\prod_{R1} (r_1 \bowtie r_2)$
- Ou seja, $r_1 \ltimes r_2$ selecciona os tuplos de r_1 que contribuíram para $r_1 \bowtie r_2$.
- No passo 3 anterior, $temp_2 = r_2 \ltimes r_1$.
- Para joins de várias relações, a estratégia acima pode ser extendida para uma série de passos de semijoins.

Estratégias de join que exploram paralelismo

- Considerem $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ onde as relacoes *r*i estao guardadas no site S_i. O resultado deve ser apresentado no site S_1.
- r_1 é enviado para S_2 e $r_1 \bowtie r_2$ é calculado em S_2: simultaneamente r_3 é enviado para S_4 e $r_3 \bowtie r_4$ é calculado em S_4
- S_2 envia tuplos de $(r_1 \bowtie r_2)$ para S_1 assim que são produzidos; S_4 envia tuplos de $(r_3 \bowtie r_4)$ para S_1
- Assim que tuplos de $(r_1 \bowtie r_2)$ e $(r_3 \bowtie r_4)$ chegam a S_1 $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$ é calculado em paralelo com a computacao de $(r_1 \bowtie r_2)$ em S_2 e a computacao de $(r_3 \bowtie r_4)$ em S_4.

Controlo de Concorrência

- Modificação dos esquemas de controlo de concorrência para uso em ambientes distribuídos.
- Assumimos que cada site participa na execução do protocolo de commit para assegurar a atomicidade global de transacções.
- Assumimos que todas as réplica de qualquer item estão actualizadas
  - Veremos mais tarde como relaxar esta condição no caso de um site falhar

Abordagem Single-Lock-Manager

- O sistema mantém um *único* lock manager que reside num *único* site escolhido, S_i
- Quando uma transacção precisa de fazer o lock de um item de dados, envia o pedido de lock para S_i e o lock manager determina se pode conceder o lock imediatamente
  - Se sim, o lock manager envia uma mensagem ao site que iniciou o pedido
  - Se não, o pedido é suspenso até que possa ser concedido, altura em que uma mensagem é enviada para o site de início

Abordagem Single-Lock-Manager (Cont.)

- A transacção pode ler o item de dados de *qualquer* um dos sites onde uma réplica do item de dados resida.
- As escritas devem ser efectuadas em todas as réplicas do item de dados
- Vantagens deste esquema:
  - Implementação simples
  - Tratamento de deadlocks simples
- Desvantagens deste esquema:
  - Bottleneck: o site lock manager torna-se um bottleneck
  - Vulnerabilidade: o sistema é vulnerável a falhas do site do lock manager.

Lock Manager Distribuído

- Nesta abordagem, a funcionalidade de locking é implementada por lock managers em cada site
  - Os lock managers controlam o accesso a itens de dados locais
    - Protocolos especiais devem ser usados para réplicas
- Vantagem: o trabalho é distribuído e torna-se robusto a falhas
- Desvantagem: a detecção de deadlocks é mais complicada
  - Os lock managers cooperam para a detecção de deadlocks
    - Mais sobre isto mais à frente
- Diversas variantes desta abordagem
  - Primary copy
  - Majority protocol
  - Biased protocol
  - Quorum consensus

Primary Copy

- Escolher uma réplica de do item de dados para ser a **primary copy**.
  - O site contendo a réplica é chamado o **primary site** para aquele item de dados
  - Itens de dados diferentes podem ter primary sites diferentes
- Quando uma transacção precisa de fazer o lock de um item de dados *Q*, pede o lock no primary site de *Q*.
  - Implicitamente, obtém o lock em todas as réplicas do item de dados
- Vantagem
  - Controlo de concorrência para dados replicados é tratado de forma similar a itens de dados não replicados – implementação simples.
- Desvantagem
  - Se o primary site de *Q* falha, *Q* fica inacessível apesar de outros sites contendo réplicas permanecerem acessíveis.

Majority Protocol

- Os lock managers locais em cada site administram os pedidos de lock e unlock para itens de dados guardados no seu site.
- Quando uma transacção deseja obter o lock de um item de dados não replicado *Q* residente no site S_i, envia uma mensagem para o lock manager de S_i .
  - Se *Q* está sob um lock em modo incompatível, então o pedido é suspenso até que possa ser concedido.
  - Quando o pedido de lock pode ser atribuído, o lock manager envia uma mensagem para o site de início indicando que o pedido de lock foi concedido.

Majority Protocol (Cont.)

- No caso de dados replicados
  - Se *Q* está replicado em n sites, então uma mensagem deve ser enviada para mais de metade dos n sites nos quais *Q* está guardada.
  - A transacção não opera sobre *Q* até que tenha obtido um lock na maioria das réplicas de *Q*.
  - Aquando da escrita do item de dados, a transacção executa a escrita em *todas* as réplicas.
- Vantagem
  - Pode ser usado mesmo que alguns sites não estejam disponíveis
    - Detalhes sobre o tratamento de escrita na presença de uma falha de um site mais à frente
- Desvantagem
  - Requere 2(*n*/2 + 1) mensagens para tratamento de pedidos de locks, e (*n*/2 + 1) mensagens para tratamento de pedidos de unlock.
  - Há potencial para deadlocks mesmo para um item único - e.g., cada 3 transacções podem ter locks em 1/3 das réplicas dos itens de dados.

Biased Protocol

- Um lock manager local em cada site tal como no majority protocol, mas agora os pedidos para para locks partilhados são tratados de forma diferente dos pedidos de locks exclusivos.
- **Locks Partilhados**. Quando uma transacção precisa de fazer um lock sobre o item de dados *Q*, pede simplesmente um lock em *Q* ao lock manager num site contendo uma réplica de *Q*.
- **Locks Exclusivos**. Quando uma transacção precisa de fazer um lock sobre o item de dados *Q*, pede um lock em *Q* ao lock manager de todos os sites contendo uma réplica de *Q*.
- Vantagem – resulta num overhead menor para operações de leitura.
- Desvantagem - overhead adicional nas operações de escrita

Quorum Consensus Protocol

- É uma generalização dos protocolos majority e biased
- A cada site é atribuído um peso.
  - Seja S o total dos pesos de um site
- Escolher dois valores read quorum Q_r e write quorum Q_w
  - Tais que $Q_r + Q_w > S$ and $2 * Q_w > S$
  - Os quorums podem ser escolhidos (e S calculado) separadamente para cada item de dados
- Cada operação de read tem de fazer o lock em réplicas suficientes tais que a soma dos pesos dos sites seja >= Q_r
- Cada operação de write tem de fazer o lock em réplicas suficientes tais que a soma dos pesos dos sites seja >= Q_w
- Por agora assumimos que todas as réplicas são escritas
  - Extensões para permitir que alguns sites estejam indisponíveis serão descritas mais tarde

Tratamento de Deadlocks

Considerem as duas transacções seguintes e o historial, com o item X e a transacção T_1 no site 1, e o item Y e a transacção T_2 no site 2:

T_1:	write (X)	T_2:	write (Y)
	write (Y)		write (X)

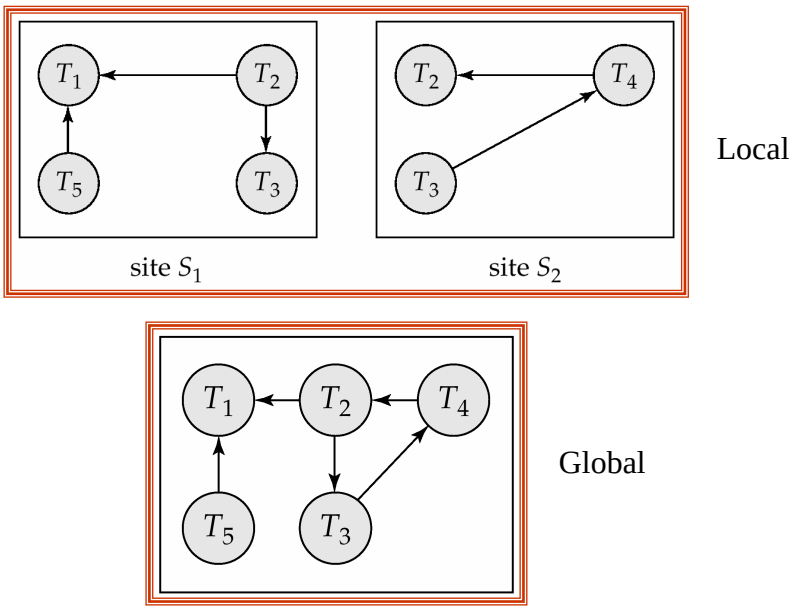
X-lock on X write (X)	
	X-lock on Y write (Y) wait for X-lock on X
Wait for X-lock on Y	

Resultado: deadlock que não pode ser detectado localmente nos sites

Abordagem Centralizada

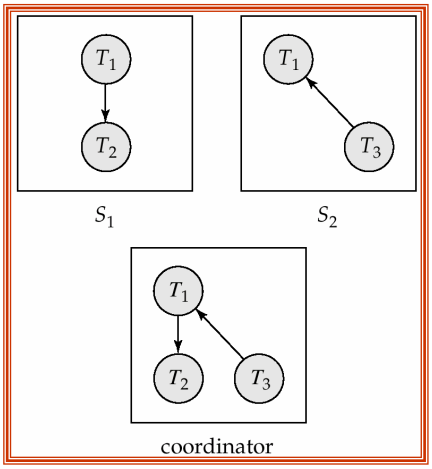
- Um grafo global de espera é construído e mantido num *único* site; o coordenador de deadlock-detection
  - *Grafo real* : Estado do real sistema, desconhecido.
  - *Grafo construído*: Aproximação gerada pelo controlador durante a execução do seu algoritmo.
- O grafo de espera global pode ser construído quando:
  - Um novo ramo é inserido ou removido dos grafos de espera locais.
  - Um número de alterações ocorreram num grafo de espera local.
  - O coordenador necessita de invocar a detecção de ciclos.
- Se o coordenador encontra um ciclo, selecciona uma vítima e notifica todos os sites. Os sites abortam a transacção vítima.

Grafos de espera locais e globais



Exemplo de um ciclo falso de espera

Estado inicial:



Ciclos Falsos (Cont.)

- Suponha que com início no estado ilustrado na figura anterior,
  1. T_2 liberta os recursos em S_1
     - Resultando na remoção da mensagem $T_1 \rightarrow T_2$ do gestor de transacções no site S_1 no coordinator)
  2. E depois T_2 requere um recurso de T_3 no site S_2
     - Resultando na inserção da mensagem $T_2 \rightarrow T_3$ de S_2 no coordenador
- Suponhamos ainda que a mensagem de insert chega antes damensagem de **delete**
  - Pode acontecer por motivos de atrasos de rede
- O coordenador detectaria um ciclo falso

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$$

- Este ciclo falso nunca existiria na realidade.
- Ciclos falsos não ocorrem nunca com locking a duas fases.

Rollbacks desnecessários

- Rollbacks desnecessários podem resultar quando deadlock ocorreu de facto e foi escolhida uma vítima, mas entretanto uma das transacções foi abortada por razões diferentes das do deadlock.
- Rollbacks desnecessários podem resultar de ciclos falsos no grafo de global de espera; no entanto a probabilidade de ciclos falsos é baixa.

Bases de Dados Distribuídas Heterogéneas

- Muitas aplicações de bases de dados requerem dados oriundos de várias bases de dados pré-existentes localizadas num conjunto heterogéneo de plataformas de hardware e software
- Os modelos de dados podem ser diferentes (hierárquico, relacional , etc.)
- Os protocolos de commit de transacções podem ser incompatíveis
- O controlo de concorrência pode ser baseado em técnicas diferentes (locking, timestamping, etc.)
- Detalhes aos nível do sistema são quase sempre totalmente incompatíveis.
- Um **sistema multi-bases-de-dados** é uma camada de software sobre um conjunto de sistemas de bases de dados, que é concebido para manipular informação em bases de dados heterogéneas
  - Cria uma ilusão de uma integração lógica de bases de dados sem qualquer integração física de bases de dados

Vantagens

- Preserva-se o investimento existente em:
  - Hardware
  - Software de sistema
  - Aplicações
- Autonomia local e controlo administrativo
- Permite o uso de SGBD's dedicados
- É uma passo no sentido de uma unificação homogénea de SGBD's
  - Dificuldades da integração total em SGBD's homogéneos
    - Dificuldades técnicas e custo de conversão
    - Dificuldade organizacionais/políticas
      - As organizações não querem perder o controlo dos seus dados
      - As bases de dados locais querem manter um elevado grau de **autonomia**

Visão Unificada de Dados

- Acordo num modelo comum de dados
  - Tipicamente o modelo relacional
- Acordo num esquema conceptual comum
  - Nomes diferentes para a mesma relação/atributo
  - Os mesmos nomes para relações/atributos significam coisas diferentes
- Acordo numa representação única de dados partilhados
  - E.g. tipos de dados, precisão,
  - Códigos de caracteres
    - ASCII vs EBCDIC
    - Variações na ordem de ordenação
- Acordo em unidades de medida
- Variações nos nomes
  - E.g. Köln vs Cologne, Mumbai vs Bombay

Processamento de Consultas

- Existem várias questões no processamento de consultas numa base de dados heterogénea
- Tradução do esquema
  - Escrever um **tradutor** para cada origem de dados para traduzir os dados para uma esquema global
  - Os tradutores devem também traduzir as actualizações no esquema global para actualizações nos esquemas locais
- Capacidades limitadas de consultas
  - Algumas origens de dados permitem apenas formas restritas de selecções
    - E.g. web forms, flat files
  - As consultas têm de ser partidas e processadas em parte na origem e noutra parte num site diferente
- Remoção de informação duplicada quando os sites têm informação repetida
  - Decidir que sites executam a consulta
- Optimização global de consultas

Sistemas Mediadores

- **Sistemas mediadores** são sistemas que integram múltiplas fontes de dados heterogéneas pelo fornecimento de uma visão global integrada, e fornecendo capacidades de consulta na visão global
  - Ao contrário de sistemas completos multi-bases-de-dados, os mediadores normalmentes não se preocupam com o processamento de transacções
  - No entanto os nomes de mediador e multi-bases-de-dados são muitas vezes usados como sinónimos
  - O termo **Bases de Dados Virtual** é também usado para referenciar sistemas mediador/multi-bases-de-dados