Capítulo 7

Controlo Distribuído da Concorrência

Motivação:

-Transação: Uma transação T_i é uma ordenação parcial sobre as suas operações e a condição de término.

- Transações Planas: Possui um único ponto de início (Begin_Transaction) e um único ponto de término (End_Transaction)
- Transações Aninhadas: Uma transação inclui outras transações entre seus pontos de início e consolidação. Transações embutidas em outras costumam ser chamadas de subtransações

- Operação: Denota-se $O_{ij}(x)$ alguma operação O_j da transação Ti sobre uma entidade da base de dados x.

- Operação em conflito: Duas operações $O_i(x)$ e $O_j(x)$ são conflitantes se:

1. Pertencem a transações diferentes
2. Ambas acessam o mesmo item de dados
3. Pelo menos uma delas é uma operação Write_Item

- Controlo de concorrência num SGBD Distribuído assegura a consistência num ambiente distribuído e multi-utilizador.
- Tem o objetivo de encontrar um equilíbrio adequado entre a consistência da Base de Dados e um nível elevado de concorrência.

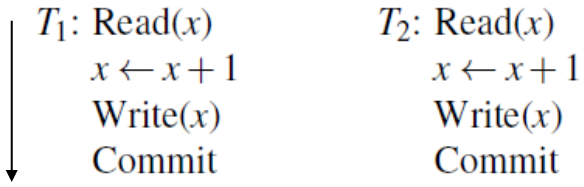
Serializabilidade

- A serializabilização de escalonamentos é usado para identificar quais escalonamentos estão corretos quando as execuções da transação tiverem intercalação das suas operações nos escalonamentos.
- Se num escalonamento S as operações não estão intercaladas (ou seja, as operações de cada transação ocorrem consecutivamente) dizemos que é um escalonamento é serial.
- A execução serial de um conjunto de transações mantém a consistência da base de dados, porém pode acarretar estados de inatividade da CPU, desperdiçando processamento.
- A execução concorrente de transações deve deixar a base de dados num estado que possa ser alcançado por uma execução sequencial em alguma ordem.
- Caso essa situação seja alcançada serão resolvidos problemas como os de atualizações perdidas (assegurando o isolamento e não permitindo que os resultados incompletos sejam acessados por outras transações)
- Um escalonamento S, ou Schedule (também chamado de Histórico) é definido sobre um conjunto de transações $T = \{T_1, T_2, ..., T_n\}$ e específica uma ordem intercalada de execução dessas operações de transações.
- Um escalonamento pode ser especificado como uma ordem parcial sobre T.

- Formalizando um escalonamento completo:
1. O domínio da relação será a união dos domínios individuais
2. A relação de ordenação é um superconjunto das relações de ordenação de transações individuais
3. Manter a ordem de execução entre operações conflitantes

EXEMPLO

Considere



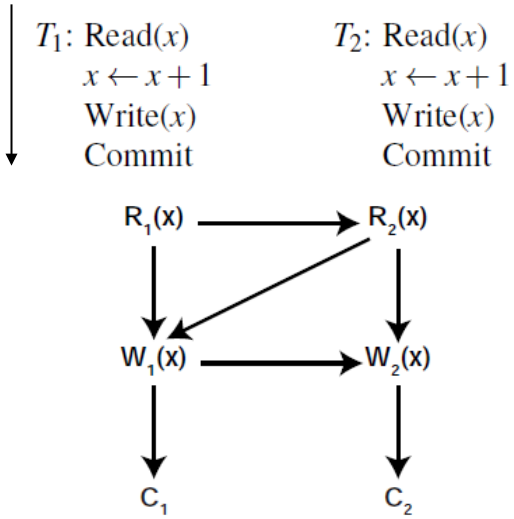
$$\Sigma_1 = \{R_1(x), W_1(x), C_1\}$$
$$\Sigma_2 = \{R_2(x), W_2(x), C_2\}$$

Thus

$$\Sigma_T = \Sigma_1 \cup \Sigma_2 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$$

and

$$\prec_H = \{(R_1,R_2),(R_1,W_1),(R_1,C_1),(R_1,W_2),(R_1,C_2),(R_2,W_1),(R_2,C_1),(R_2,W_2),$$
$$(R_2,C_2),(W_1,C_1),(W_1,W_2),(W_1,C_2),(C_1,W_2),(C_1,C_2),(W_2,C_2)\}$$



$$H_T^c = \{R_1(x), R_2(x), W_1(x), C_1, W_2(x), C_2\}$$

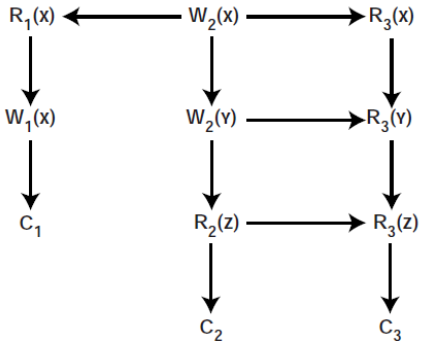
- Um escalonamento também pode ser definido como um prefixo de um escalonamento completo.
- Utilizar esse prefixo permite lidar com escalonamentos incompletos.
- A serializabilidade lida apenas com operações de transações que entram em conflito, e não com todas as operações.
- Ao introduzir falhas, deve-se ser capaz de lidar com escalonamentos incompletos, e é isso que o prefixo permite fazer.

EXEMPLO de Escalonamento Incompleto

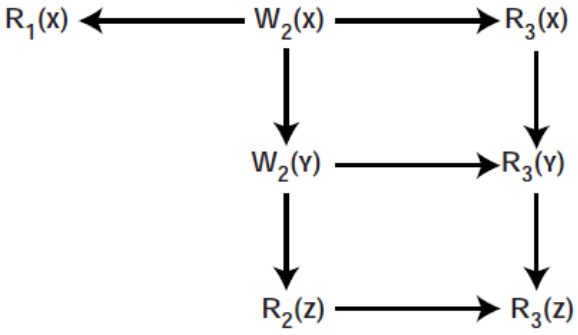
T_1: Read(x) Write(x) Commit

T_2: Write(x) Write(y) Read(z) Commit

T_3: Read(x) Read(y) Read(z) Commit



A Complete History



Prefix of Complete History

Considerando as três transações do exemplo anterior, temos o escalonamento a seguir:

$$H = \{\underbrace{W_2(x), W_2(y), R_2(z)}_{T_2}, \underbrace{R_1(x), W_1(x)}_{T_1}, \underbrace{R_3(x), R_3(y), R_3(z)}_{T_3}\}$$

É serial pois todas as operações de T2 são executadas antes das de T1, e todas de T1 são executadas antes de T3.

$$T_2 \rightarrow T_1 \rightarrow T_3 \qquad T_2 \prec_H T_1 \prec_H T_3.$$

Equivalência de Conflitos:
Dois escalonamentos definidos em cima de um mesmo conjuntos de transações T são ditos equivalentes se para cada par de operações conflitantes, uma operação O_{ij} que será executada numa T_1 é a mesma que será executada numa T_2 que ocorre após T_1. Essa situação é o que chamamos de equivalência de conflito.

Formalizando a serializabilidade:
Um escalonamento S é serializável se, e somente se, ele é equivalente de conflitos a um escalonamento serial.

Diferenças entre Escalonamento Serial e o Serializável:

- A serializabilidade estende-se de maneira direta às base de dados distribuídas não replicados (ou particionados).
- Escalonamento em cada site -> escalonamento local
- Se a BD não for replicada e cada escalonamento local for serializável, sua união, chamada escalonamento global, também será serializável.

EXEMPLO

Suponha que antes das transações o valor de (x) seja 1, qual será o valor final nos dois sites?

T1: Read(x)	T2: Read(x)
x<- x + 5	x <- x * 10
Write(x)	Write(x)
Commit	Commit

$S1 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$ **60**

$S2 = \{R_2(x), W_2(x), C_2, R_1(x), W_1(x), C_1\}$ **15**

O exemplo anterior viola a consistência mútua das duas bases de dados locais.
A consistência mútua exige que todos os valores de todos os itens de dados replicados sejam idênticos.

Para resolver esse problema:

1. Cada escalonamento local deve ser serializável
2. Duas operações conflitantes devem estar na mesma ordem relativa em todos os escalonamentos locais que aparecem juntas

Taxonomia dos Mecanismos de Concorrência

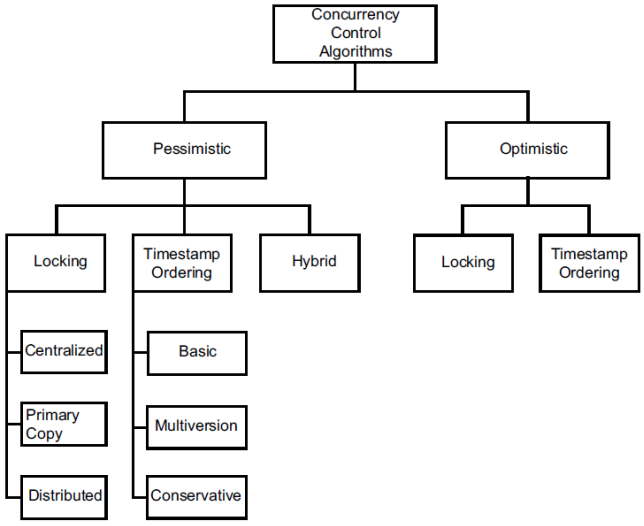
Existem diversos modos de abordagens de controlo de concorrência.

- Distribuição da Base de Dados - total ou parcialmente replicada
- Topologia da Rede - estrela, circular, com capacidade de difusão (*broadcasting*)
- Primitiva de sincronização - *timestamp ordering* ou *locking-based*

As primitivas de sincronização podem ser usadas em algoritmos com dois pontos de vista:

- Pessimistas - Muitas transações entrarão em conflito, portanto a sincronização da execução concorrente ocorre mais cedo no seu ciclo de execução

- Otimistas - Poucas transações entrarão em conflito, portanto a sincronização de execução concorrente ocorre até ao seu término.



Algoritmos de Controlo de Concorrência

Locking-Based

- Controlo de concorrência "baseado no bloqueio" assegura que os dados compartilhados por operações conflitantes sejam acessados por uma única operação de cada vez
- Isso é conseguido pela associação de um "bloqueio" a cada unidade de bloqueio
- Esse bloqueio é definido por uma transação antes de ser acesado e é redefinido no final do seu uso
- Existem dois modos de bloqueio:
  - Bloqueio de leitura (RL - read lock)
  - Bloqueio de gravação (WR - write lock)

	$RL_i(X)$	$WL_i(X)$
$RL_j(X)$	compatible	not compatible
$WL_j(X)$	not compatible	not compatible

Comunicação:

- Em sistemas baseados no bloqueio, o escalonador é um gestor de bloqueio (LM - Lock Manager).
- O gestor de transações repassa ao gestor de bloqueio a operação da base de dados (leitura ou escrita) e as informações associadas (item acessado, identificador de transação).
- O gestor de bloqueio verifica se a unidade de bloqueio que contém o item de dados já está bloqueada. Se já estiver, e se o modo de bloqueio existente for incompatível com o da transação atual, a operação será adiada. Caso contrário, o bloqueio será estabelecido no modo desejado e a operação da BD será repassada ao processador de dados para acesso à BD real.

Bloqueio de 2 Fases (Two-Phase Locking):

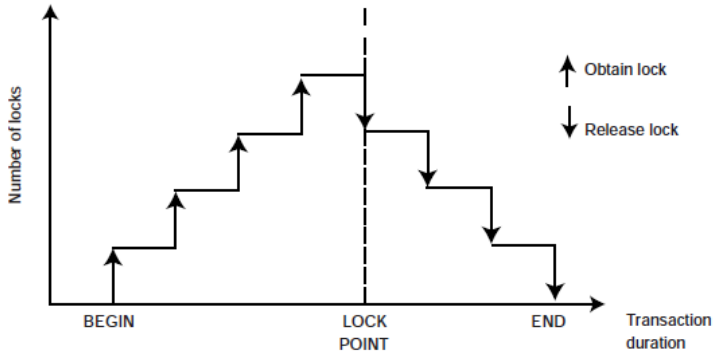
EXEMPLO

Considere a seguinte transação e o seu escalonamento H

T_1:	T_2:
Read(x)	Read(x)
$x \leftarrow x+1$	$x \leftarrow x*2$
Write(x)	Write(x)
Read(y)	Read(y)
$y \leftarrow y-1$	$y \leftarrow y*2$
Write(y)	Write(y)
Commit	Commit

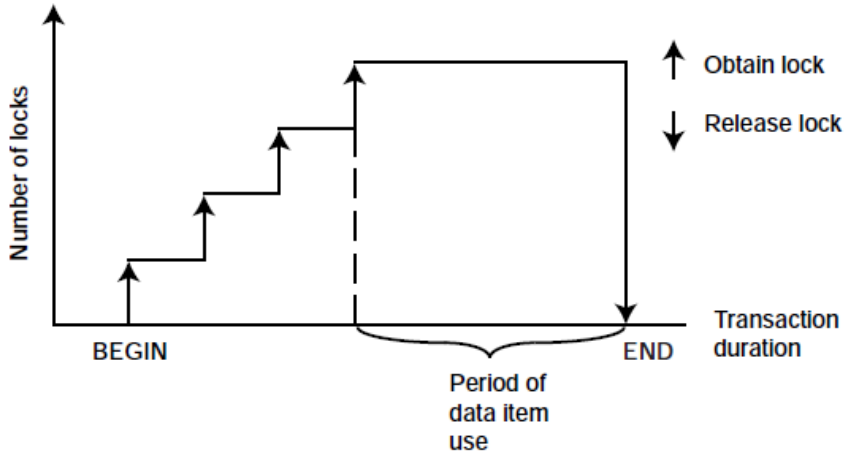
$$H = \{wl_1(x), R_1(x), W_1(x), lr_1(x), wl_2(x), R_2(x), w_2(x), lr_2(x), wl_2(y), R_2(y), W_2(y), lr_2(y), wl_1(y), R_1(y), W_1(y), lr_1(y)\}$$

- A regra de bloqueio de duas fases estabelece que nenhuma transação deve solicitar um bloqueio após liberar um dos seus bloqueios.
- Como alternativa, uma transação não deve liberar um bloqueio até ter certeza de que não solicitará outro bloqueio.
- Algoritmos 2PL executam transações em duas fases
- Cada transação tem uma fase de crescimento na qual ela obtém bloqueios e acessa itens de dados
- Uma fase de contração durante a qual ele libera bloqueios
- O ponto de bloqueio é o momento em que a transação conseguiu todos os seus bloqueios, mas ainda não começou a liberar nenhum deles
- Teorema: Qualquer escalonamento gerado por um algoritmo que obedece a regra 2PL é serializável

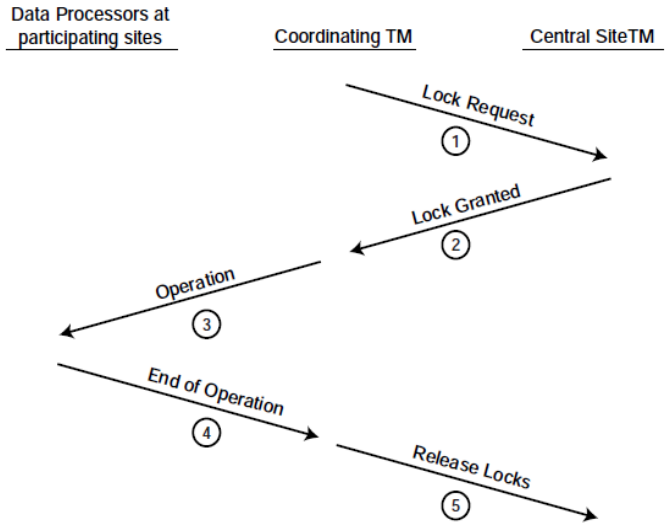


- É difícil implementar a liberação de bloqueios em cascata pois o gestor de bloqueio tem que saber que a transação obteve todos os seus bloqueios e não precisará bloquear outro item de dados
- O gestor precisa também saber que a transação não precisa mais de acesso ao item de dados em questão
- Se a transação abortar após liberar um bloqueio, pode fazer com que outras transações sejam abortadas que também tenham acessado o item de dados desbloqueado

- A maioria dos escalonadores 2PL implementam o bloqueio de 2 fases estrito (*strict two-phase locking*).



2PL Centralizado



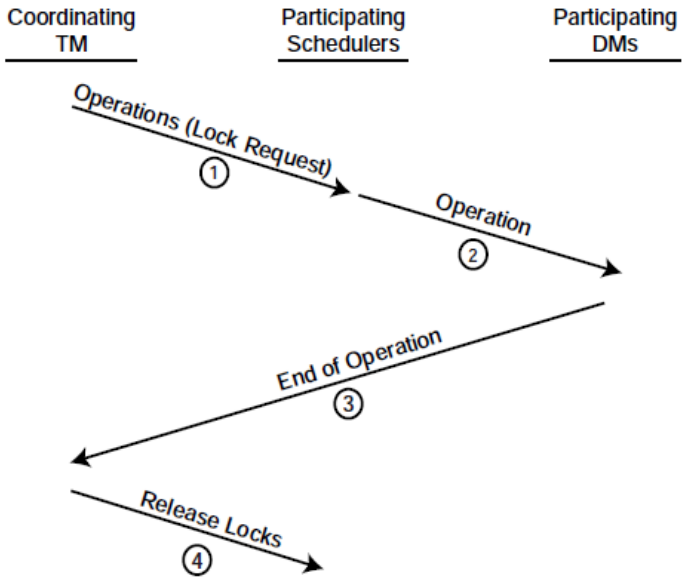
2PL de Cópia Primária

- É uma extensão direta do 2PL Centralizado
- Implementa gestores de bloqueios em vários sites, onde cada um irá adminsistrar um dado conjunto de unidade de bloqueio
- Mudanças mínimas em relaçação ao C2PL

2PL Distribuído

- Espera a disponibilidade de gestores de bloqueio em cada site
- Se a BD não for replicada, o 2PL Distribuído irá degenerar no algoritmo de 2PL de cópia primária
- Caso sejam replicados, será implementado o protocolo ROWA

Gestor de Transações



Timestamp Ordering - TO

- Algoritmos de controlo de concorrência do tipo *timestamp* selecionam uma ordem de serialização e executam as transações de acordo com ela.
- Para estabelecer essa ordem, o gestor de transações atribui a cada transação um *timestamp*.
- Gerado pelo Gestor de Transações, o *timestamp* é um identificador simples que permite a unicidade, exclusividade e o caráter monotónico de cada operação
- Cada nova operação é comparada com operações conflitantes que já tenham sido escalonadas.
- Se a nova operação for mais nova que as operações conflitantes já escalonada, será aceita. Do contrário será rejeitada obrigando a transação reiniciar com um novo *timestamp*.
- Um escalonador de TO tem a garantia de gerar escalonamentos serializáveis.
- Além do contador local, o tempo também pode ser usado para definir o *timestamp*.

Algoritmo Básico de TO

- É uma implementação direta da regra de TO.
- O gestor de transações de coordenação atribui o *timestamp* a cada transação, determina os sites em que cada item de dados está armazenado e envia as operações relevantes a esses sites.

Como as transações nunca esperam enquanto mantêm direitos de acesso aos itens de dados, o algoritmo básico de TO nunca provoca impasses. No entanto, o preço para se livrar de impasses é a reinicialização potencial de uma transação várias vezes.

Algoritmo de TO Conservador (*Conservative*)

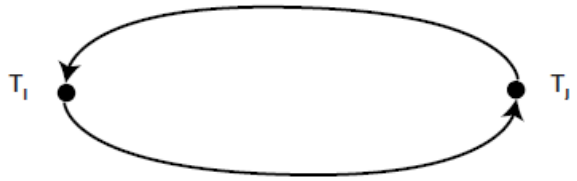
- Problema das reinicializações em sites comparativamente inativos em relação a outros.
- Ao invés de usar um contador central (dispendioso), cada gestor de transações pode enviar suas operações remotas aos gestores de transações de outros sites.
- Assim, qualquer gestor que possuir um contador inferior ao divulgado, ajusta seu próprio a um valor de uma unidade maior que o valor de entrada.
- O algoritmo básico de TO tenta executar uma operação logo que ela é aceita (agressivo / progressivo).
- Algoritmos conservadores atrasam cada operação até ter certeza que não chegará nenhuma operação com um *timestamp* menor.
- As operações de cada transação são inseridas em *buffers* até ser possível estabelecer uma relação, tal que, não seja detectada possíveis rejeições, e elas possam ser executadas nessa ordem.

Algoritmo de TO de Várias Versões (*Multiversion*)

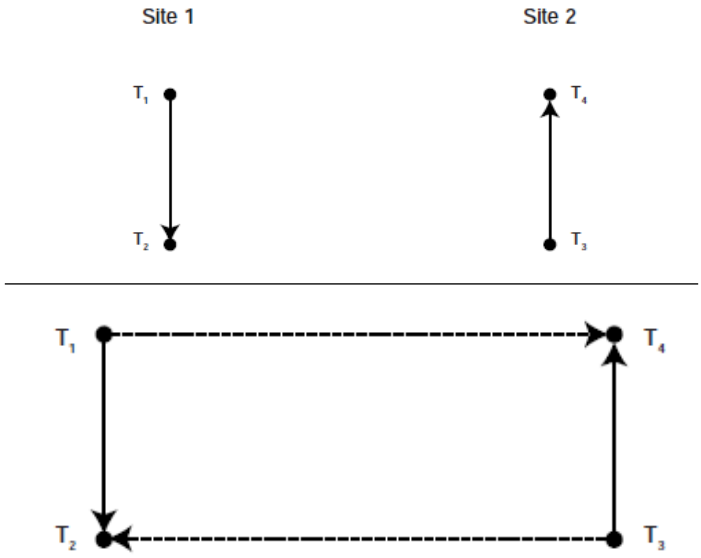
- Teve maior foco em base de dados centralizadas.
- As atualizações não modificam a base de dados. Cada operação de gravação cria uma nova versão do item de dados.
- Cada versão é marcada pelo *timestamp* da transação que o cria.
- Troca espaço de armazenamento pelo tempo.
- Para economizar espaço, as versões podem ser purgadas de tempos em tempos.

Gestão de Impasses

- Um impasse pode ocorrer porque as transações esperam uma pela outra. De modo informal, uma situação de impasse é um conjunto de solicitações que jamais poderão ser concedidas pelo mecanismo de controlo de concorrência.
- Um impasse é um fenômeno permanente, não termina a menos que ocorra uma intervenção externa.
- *Wait-for-Graph* – representa a espera entre as transações.
  - Cada nó representa uma transação concorrente.
  - O arco representa a espera entre da liberação do bloqueio sobre alguma entidade.



- A formação do WFG é mais complicada em sistemas distribuídos, pois duas transações que participam de uma condição de impasse podem estar em execução em sites diferentes (impasse global).
- LWFG - grafo de espera local em cada site
- GWFG - grafo de espera global, união de todos LWFGs



Deteção e Resolução de Impasses:

- Normalmente, a resolução é feita pela seleção de uma ou mais transações vitimas que serão apropriadas antecipadamente e abortadas para romper os ciclos no GWFG.

- Fatores a tomar em consideração:

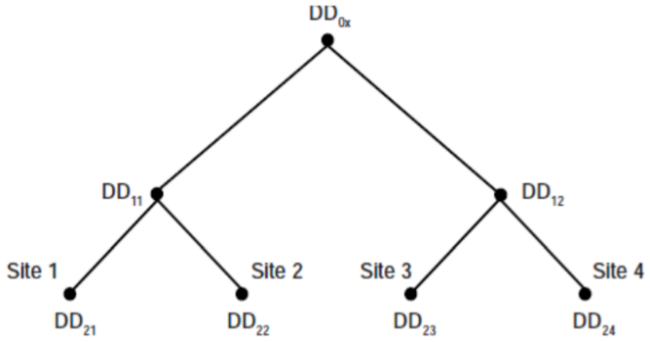
1. A quantidade de esforço já investido na transação
2. O custo de se abortar a transação
3. A quantidade de esforço necessária para concluir
4. O número de ciclos que contém a transação (melhor abortar as que possuem mais de um clico)

Deteção Centralizada de Impasses:
- Um site é designado como o detector de impasses para todo o sistema. Ele recebe todos os LWFG do sistema e monta o GWFG.

- *Qual consequência o tempo de intervalo entre os recebimentos de LWFGs pelo site responsável refletem no sistema?*

Deteção Hierárquica de Impasses:
- É construída uma hierarquia de detectores de impasses.
- Impasses locais são detectados por esse site com o uso do LWFG.
- Cada site envia seu LWFG ao detector de impasses do próximo nível.
- Portanto, quando ocorrem impasses entre sites diferentes, eles são descobertos pelo detector de nível acima .

Consequências: Reduz a dependência do site central no entanto sua implementação é consideravelmente mais complicada.



Deteção Hierárquica de Impasses:
- Delegam a responsabilidade de detectar impasses a sites individuais
- Detectores de impasses comunicam seu LWFGs entre si (apenas os ciclos de impasses potenciais são transmitidos).

O LWFG é formado e modificado da seguinte maneira:

1. Tendo em vista que cada site recebe os ciclos de impasses potenciais de outros sites, essas arestas são acrescentadas aos LWFGs.

2. Arestas no LWFG que mostram transações locais esperando por transações de outros sites são unidas com as arestas dos LWFGs que mostram que as transações remotas estão esperando pelas locais.