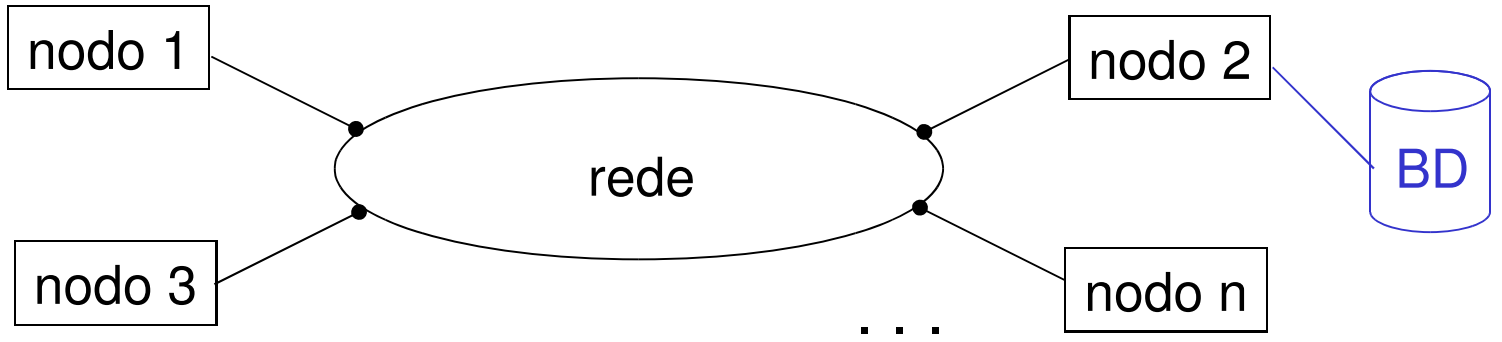
Sumário

BD Distribuído (BDD)

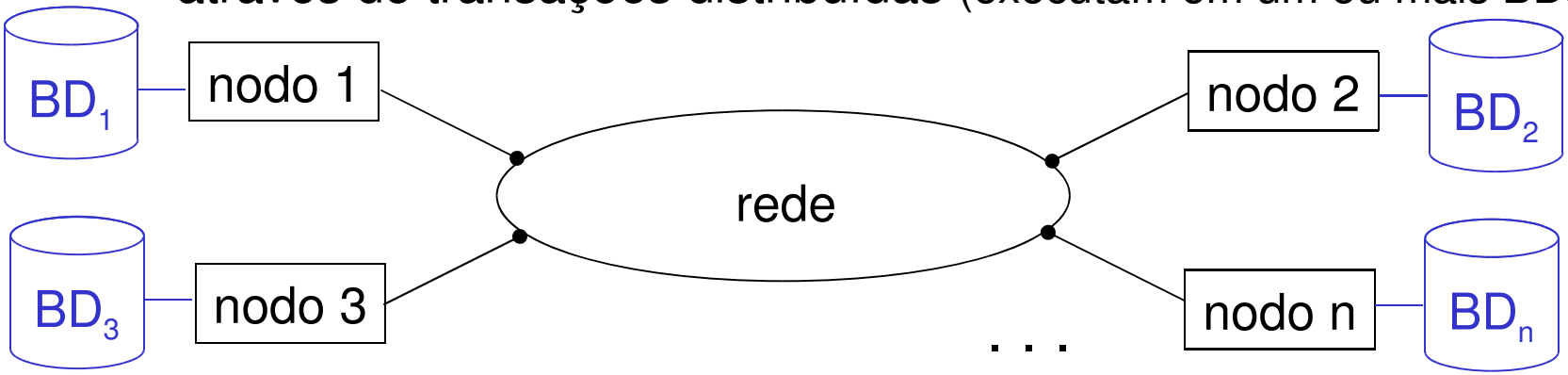
- Definição
  - coleção de múltiplos BDs logicamente inter-relacionados e dispersos sobre uma rede de computadores
- Motivação
  - organizações sofisticadas
    - estrutura geograficamente distribuída e necessidade de compartilhar dados
  - avanço da computação distribuída e das redes
    - maior eficiência de acesso e processamento paralelo
  - integração de dados
    - acesso unificado a dados heterogêneos

BDC x BDD

- BD Centralizado
  - BD único acessado por uma ou mais aplicações locais e/ou remotas através de transações centralizadas



- BD Distribuído
  - vários BDs autônomos (homogêneos ou não); aplicações o acessam através de transações distribuídas (executam em um ou mais BDs)



BDD - Vantagens

- Transparência
  - omite detalhes sobre a distribuição dos dados
    - transparência de localização e nomeação
      - uma instrução DML não se preocupa onde está o dado
      - uma vez informado um dado, ele é buscado em locais onde esteja definido, mesmo tendo nomes diferentes
    - transparência da forma de acesso
      - define-se uma consulta sem se preocupar com futuras transformações sobre ela para alcançar os dados desejados
        - » transparência de dialeto DML

          - consulta não se preocupa com a DML de cada BD
        - » transparência de fragmentação/replicação

          - decomposição da consulta
          - busca-se o nodo mais próximo onde está o dado
    - um catálogo robusto é requerido (DDD)

BDD - Vantagens

- Confiabilidade e Disponibilidade
  - se um nodo falha, outros nodos podem processar transações
  - dados são encontrados em diversos nodos
- Desempenho não é Prejudicado
  - mantém-se dados mais próximos do local onde são mais necessários
  - BDs locais são independentes e menores que um grande BDC
    - menor *overhead* para transações
    - menos transações executando do que em um BDC
  - consultas podem ser desmembradas e executadas em paralelo em diferentes nodos
- Facilidade de Expansão
  - arquitetura de um BDD permite a inclusão de novos BDs

Funções de um SGBDD

- Controle da Transparência de Armazenamento
  - uso de um DDD para controle da estrutura, RIs, replicação e fragmentação de dados nos nodos
- Processamento de Consultas Distribuídas
  - capacidade de transmissão de consultas a nodos remotos (com possível tradução)
  - planejamento de estratégias de acesso
    - quais dados de quais nodos serão acessados? qual a ordem e a sincronização para execução da consulta nestes nodos? ...
- Gerenciamento de Transações Distribuídas
  - técnicas adaptadas de controle de concorrência e *recovery*
    - consideração de falhas em nodos e falhas de comunicação, garantia de ACID distribuído, ...

Projeto de BDD

- Projeto de BDC
  - definir e estruturar dados persistentes relevantes para um domínio de aplicação
    - levantamento de requisitos, modelagem conceitual, modelagem lógica e modelagem física
- Projeto de BDD (do “zero”)
  - modelagem lógica
    - definir adicionalmente a alocação do esquema lógico nos BDs dos nodos
      - decisão sobre quais dados serão armazenados em quais nodos
      - leva em conta fragmentação e replicação de dados

Fragmentação de Dados

- Separação dos dados de uma relação para fins de armazenamento em mais de um nodo
  - definição de um esquema de fragmentação
- Tipos de fragmentação
  - horizontal, vertical e mista

Fragmentação Horizontal (FH)

- Separação de uma relação *R* em nível de tupla
- Cada fragmento horizontal fh_i de *R* ($fh_i(R)$) é definido através de uma seleção
  - $fh_i(R) = \sigma_c(R)$
- *R* é obtida através da união de todos os seus fragmentos
  - $R = fh_1(R) \cup fh_2(R) \cup \ldots \cup fh_n(R)$
- FH com fragmentação derivada
  - tuplas de uma relação secundária *S* (com referência à *R*) são também fragmentadas

FH - Exemplo

Filiais

número	cidade	endereço
1	Fpolis	rua X, 10
2	Blumenau	rua H, 55
3	Blumenau	rua E, 18
4	Fpolis	rua K, 87
5	Fpolis	rua Q, 52

Funcionários

código	nome	endereço	DN	cargo	salário	filial
1	João	rua X, 10	11/11/70	vendedor	1500,00	1
2	Maria	rua H, 55	12/04/71	caixa	1200,00	1
3	Paulo	rua E, 18	13/08/72	vendedor	1300,00	1
4	Carlos	rua K, 87	14/01/73	caixa	1000,00	2
5	Ana	rua Q, 52	15/05/74	vendedor	1300,00	2
. . .						

fh_1

fh_2 *(derivada para Funcionários)*

$\sigma_{\text{cidade = 'Blumenau'}}$ (Filiais)

$\sigma_{\text{cidade = 'Fpolis'}}$ (Filiais)

número	cidade	endereço
2	Blumenau	rua H, 55
3	Blumenau	rua E, 18

número	cidade	endereço
1	Fpolis	rua X, 10
4	Fpolis	rua K, 87
5	Fpolis	rua Q, 52

código	nome	endereço	. . .	filial
1	João	rua X, 10		1
2	Maria	rua H, 55		1
3	Paulo	rua E, 18		1
. . .				

Fragmentação Vertical (FV)

- Separação de *R* em nível de atributo
- Cada fragmento vertical fv_i de *R* ($fv_i(R)$) é definido através de uma projeção
  - $fv_i(R) = \pi_{a1, ..., aj}(R)$
- *R* é obtida através da junção natural de todos os seus fragmentos
  - $R = fv_1(R) \bowtie fv_2(R) \bowtie ... \bowtie fv_n(R)$
  - requer a mesma chave candidata em todos os fragmentos

FV - Exemplo

código	nome	endereço	DN	cargo	salário	filial
1	João	rua X, 10	11/11/70	vendedor	1500,00	1
2	Maria	rua H, 55	12/04/71	caixa	1200,00	1
3	Paulo	rua E, 18	13/08/72	vendedor	1300,00	1
4	Carlos	rua K, 87	14/01/73	caixa	1000,00	2
5	Ana	rua Q, 52	15/05/74	vendedor	1300,00	2
. . .						

Funcionários

fv_1 (dados pessoais)

$\pi_{\text{código, nome, endereço, DN}}$ (Funcionários)

código	nome	endereço	DN
1	João	rua X, 10	11/11/70
2	Maria	rua H, 55	12/04/71
3	Paulo	rua E, 18	13/08/72
4	Carlos	rua K, 87	14/01/73
5	Ana	rua Q, 52	15/05/74
. . .			

fv_2 (dados profissionais)

$\pi_{\text{código, cargo, salário, filial}}$ (Funcionários)

código	cargo	salário	filial
1	vendedor	1500,00	1
2	caixa	1200,00	1
3	vendedor	1300,00	1
4	caixa	1000,00	2
5	vendedor	1300,00	2
. . .			

Fragmentação Mista (FM)

- Separação de *R* em nível de tupla e atributo
- *R* é obtida através da execução de operações de reconstrução de fragmentos horizontais e verticais
  - exemplo
    1. Funcionários são separados por filial
    2. para cada filial, separar dados pessoais e profissionais de funcionários
    - ordem de reconstrução
      1. junção natural dos FVs de funcionários em cada filial
      2. união de dados de funcionários por filial

Esquema de Alocação

- Definição dos nodos onde serão armazenados os fragmentos (ou relações completas)
  - definição de associações fragmento – nodo
- Se associação é Fragmento [1,N] – **1** Nodo
  - não há replicação
- Se associação é Fragmento N – **M** Nodo
  - há replicação
- Possibilidades de replicação
  - total, nula ou parcial

Possibilidades de Replicação

- Total
  - desempenho bom para consultas
    - não há necessidade de acesso remoto
  - muita redundância de dados e desempenho ruim para atualizações
    - manutenção de cópias consistentes (uso de *triggers*, por exemplo)
    - *scheduler* e *recovery* mais complexos
      - bloqueios em todos os nodos
      - UNDO e REDO em todos os nodos
- Nula
  - inverte-se as vantagens e desvantagens
- Parcial
  - meio termo entre as opções anteriores

Projeto do Esquema de Alocação

- Considera basicamente
  - metas de desempenho no acesso ao BDD
    - ex.: rapidez nas atualizações (baixa distribuição) X confiabilidade (alta distribuição), ...
  - freqüência de transações em cada nodo
    - pode ser o gargalo do BDD, se distribuição foi mal definida (ex.: muitos dados concentrados em um nodo)

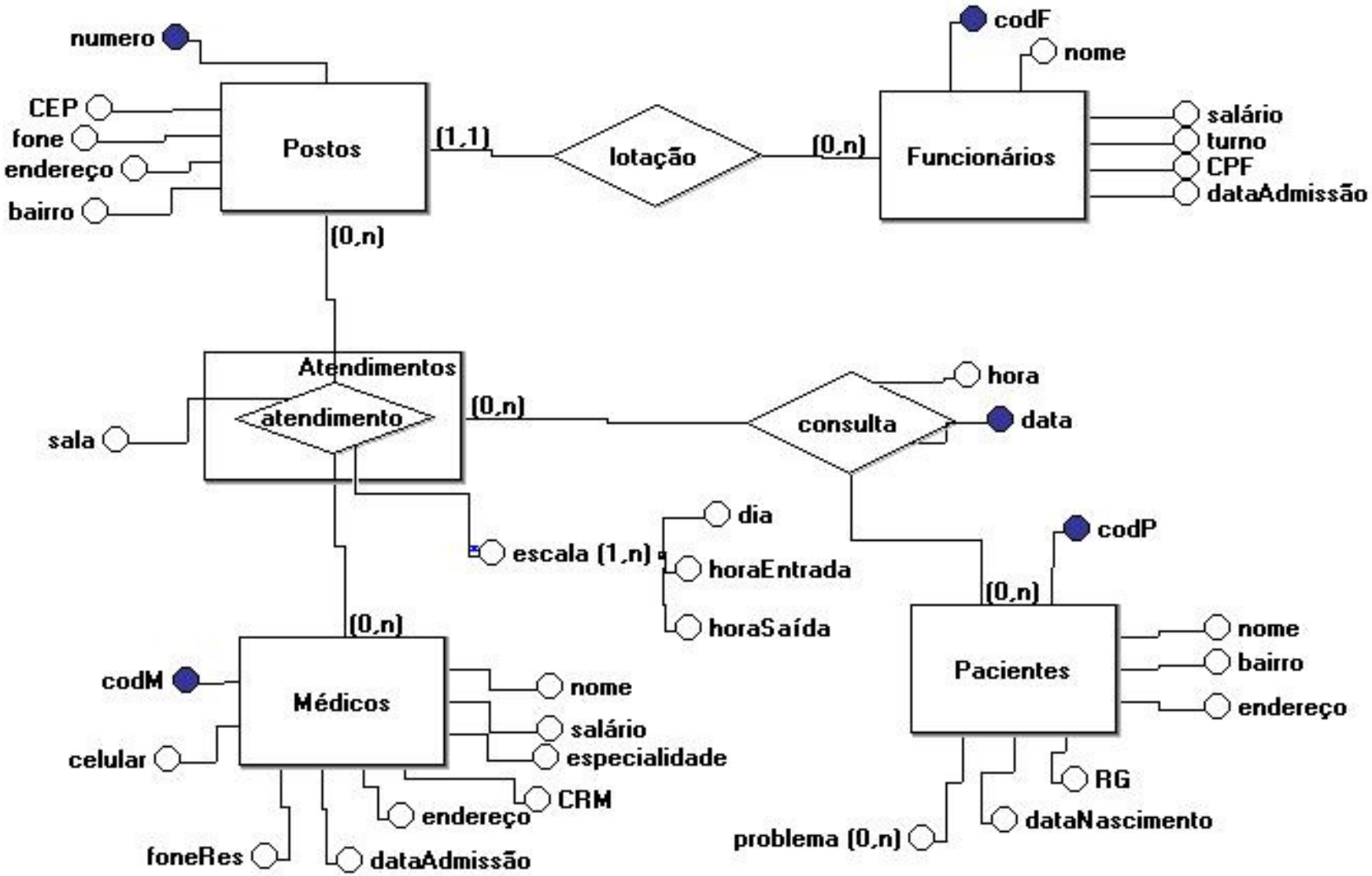
Projeto do Esquema de Alocação

- Considerações sobre replicação
  - deseja-se alta disponibilidade; transações desejam dados que podem estar em qq nodo; grande parte das transações é de leitura
    - replicação total
  - transações que acessam determinados dados partem geralmente dos mesmos nodos
    - replicação parcial destes dados nestes nodos
  - atualizações ocorrem em dados cadastrados localmente
    - replicação nula (apenas fragmentação dos dados de interesse local)

Estudo de Caso de Projeto BDD: Clínica

"Uma clínica de uma cidade possui um posto matriz no centro e outros postos em bairros. No posto matriz fica o departamento pessoal. Cada posto tem um *código, rua, número, bairro, CEP e fone*. A clínica emprega médicos e funcionários e presta serviço a pacientes através de consultas com médicos (consultas marcadas devem ser mantidas no BD). Um funcionário trabalha em um posto e possui um *código, nome, CPF, salário, função, data de admissão e turno de trabalho*. Médicos dão atendimento em um certo subconjunto de postos (com uma escala semanal de horários predefinida em cada posto, atendendo em uma sala do posto). Um médico tem *especialidade, código, CRM, nome, salário, endereço, fone residencial e celular para contato e data de admissão*. Os postos oferecem atendimento para todas as especialidades que a clínica suporta. Apenas pacientes que residem na cidade tem direito a consultar nos postos, devendo se dirigir ao posto do seu bairro. Para todo paciente cadastra-se um *código, nome, rua, número, bairro, RG, data de nascimento e eventual(is) problema(s).*"

Estudo de Caso - Esquema ER



Estudo de Caso - Esquema BDDR

Postos (nroPosto, CEP, fone, endereço, bairro)
Funcionários (codF, nome, CPF, salário, função, admissão, turno, *nroPosto*)
Pacientes (codP, nome, rua, número, bairro, RG, DN)
Problemas (*codP*, descrição)
Médicos (codM, CRM, nome, especialidade, admissão, salário, endereço, foneRes, celular)
Atendimentos (*codM*, *nroPosto*, sala)
Escalas (*codM*, *nroPosto*, dia, hora-Início, hora-Término)
Consultas (*codM*, *nroPosto*, *codP*, data, hora)

Estudo de Caso - Esquema Alocação

1. Dados de postos, médicos e seus atendimentos e escalas, funcionários, consultas, pacientes e seus problemas: fragmentados por posto (supõe-se um nodo por posto)
2. Apenas alguns dados de Médicos são necessários em cada posto: nome, codM, especialidade, endereço, celular, foneRes (idem para Funcionários: codF, nome, função, turno)
3. Dados de médicos replicados nos postos em que trabalham
4. Posto matriz mantém dados completos de médicos, funcionários e postos

Para cada Posto "X":

FragPostoX ← $\sigma_{\text{bairro = "X"}}$ (Postos)

FragPacX ← $\sigma_{\text{bairro = "X"}}$ (Pacientes)

FragProbPacX ← Problemas $\bowtie$ π_{codP} (FragPacX)

FragFuncX ← $\pi_{\text{codf,..., turno, nroPosto}}$ (Funcionários) $\bowtie$ π_{nroPosto} (FragPostoX)

FragAtendX ← Atendimentos $\bowtie$ π_{nroPosto} (FragPostoX)

FragEscalasX ← Escalas $\bowtie$ π_{nroPosto} (FragPostoX)

FragConsultasX ← Consultas $\bowtie$ π_{nroPosto} (FragPostoX)

FragMedX ← $\pi_{\text{nome, codM, ..., foneRes}}$ (Médicos) $\bowtie$ π_{codM} (FragAtendX)

Transações em BDD

- Duas categorias de transações
  - locais
    - manipulam apenas dados locais não replicados
  - globais (ou distribuídas)
    - manipulam dados de outros nodos
- Cada BD local possui um gerente de *recovery* e de *scheduler*
  - capaz de coordenar a recuperação e o escalonamento de requisições de dados de transações locais
  - pode ser capaz de realizar a mesma coordenação para transações distribuídas

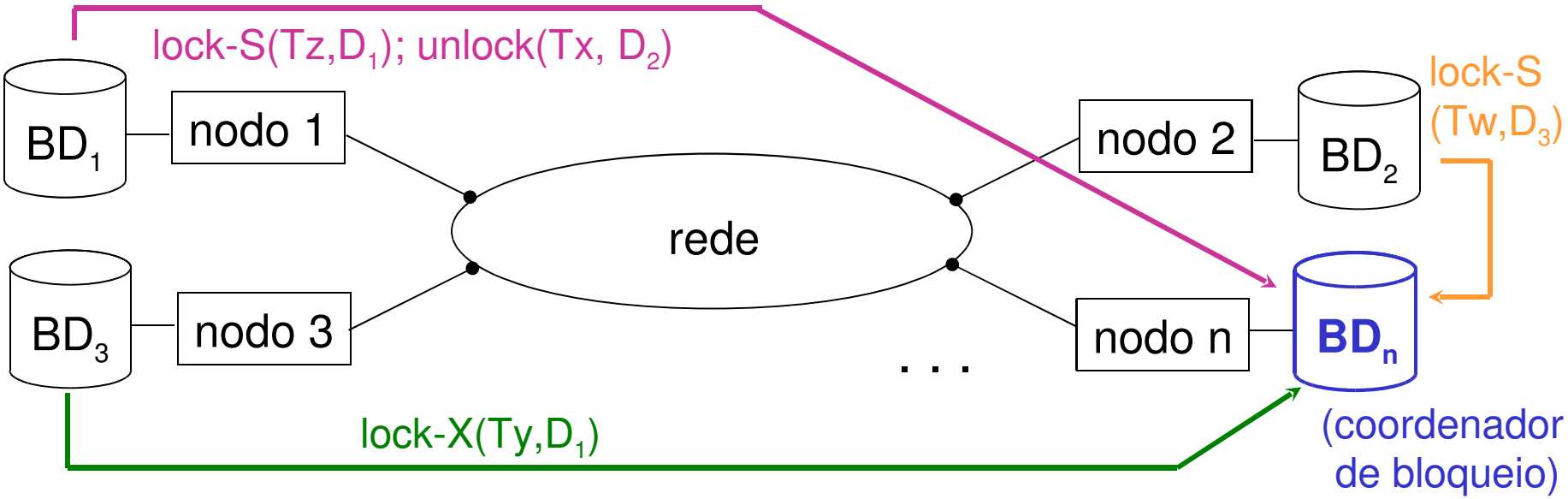
Problemas Adicionais com Gerência de Transações Distribuídas

- Replicação e Fragmentação de dados
  - *scheduler* deve manter cópias consistentes em caso de atualização
    - uma transação distribuída só encerra com sucesso se a consistência global for garantida
  - *recovery* deve manter atomicidade e durabilidade de valores de cópias e de fragmentos em caso de falha
- Novos tipos de falhas
  - falha em um nodo (ou no BD do nodo)
  - falha de comunicação (na rede ou em mensagens)

Scheduler de um BDD

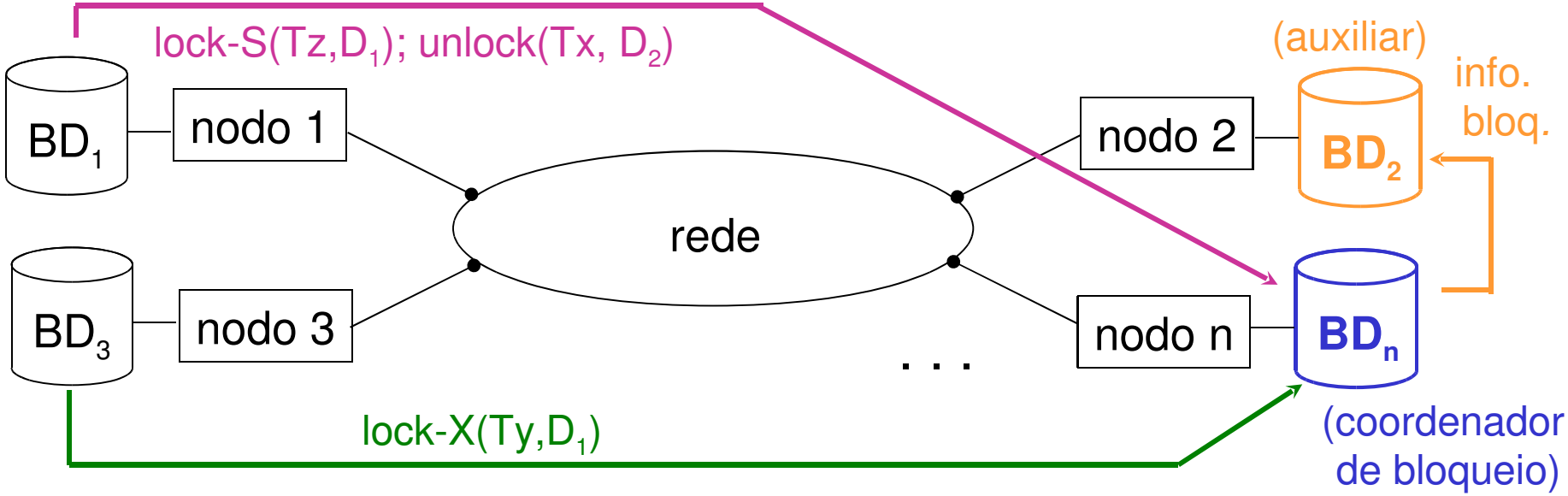
- Controles adicionais
  - divide uma transação distribuída em sub-transações para execução em outros nodos
  - coordena o *commit* ou o *abort* distribuído
- Técnicas de bloqueio são geralmente adotadas
  - bloqueios devem ocorrer em todas as cópias e fragmentos de dados desejados por transações
  - deve haver nodos responsáveis pela coordenação dos bloqueios
  - algumas técnicas
    - coordenador de bloqueio central
    - coordenador de bloqueio central com auxiliares
    - coordenação de bloqueio distribuída

Coordenador Central



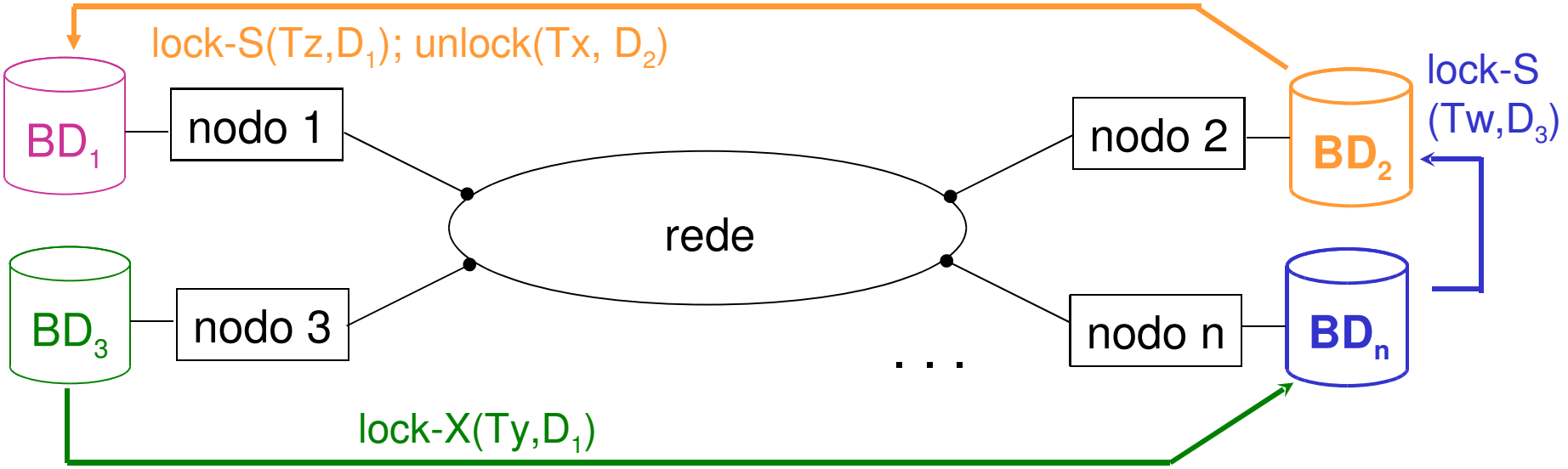
- Vantagens
  - controle de concorrência é simples (gerência em um único local, como em BDC)
  - nodos que não são coordenadores: menor *overhead* de gerenciamento de transações
- Desvantagens
  - sobrecarga de gerência de concorrência em um único nodo
  - se o coordenador falha... (!)

Coordenador Central c/ Auxiliar(es)



- Vantagens
  - controle de concorrência ainda é simples
  - se o coordenador falha, o(algum) auxiliar é eleito coordenador
- Desvantagem
  - *overhead* para sincronização das informações de bloqueio entre o coordenador e o(s) auxiliar(es)

Coordenação Distribuída



- Cada nodo coordena os bloqueios dos seus dados
- Vantagens
  - nenhum nodo fica sobrecarregado com gerência de bloqueios
  - se um nodo falha, não compromete todas as transações ativas
- Desvantagem
  - difícil prever *deadlock* distribuído
    - um nodo não tem conhecimento de todos os bloqueios no BDD
    - a menos que se use uma técnica que evite ou detecte *deadlock* (ex.: 2PL Conservador, grafo de espera global, ...)

Scheduler de um BDD

- Técnicas que usam coordenador central N_c
  - se não há auxiliares e N_c falha, transações distribuídas ativas são abortadas e um novo nodo assume o papel de coordenador
  - há a possibilidade de ocorrer eleição (caso de falha de nodo ou de comunicação com N_c)
    - um nodo N_i tenta comunicação várias vezes com N_c e seus auxiliares
    - se a comunicação foi sem sucesso, N_i assume que houve falha e tenta se eleger coordenador
      - N_i envia mensagens aos demais nodos, solicitando a sua eleição como novo coordenador
        - » se receber uma maioria de votos e nenhuma mensagem que outro nodo tornou-se coordenador, N_i é eleito

Controle de *Deadlock* em BDD

- Técnicas de prevenção baseadas em *timestamp* para BDC são geralmente adotadas
  - cada transação com um TS global
    - gerado por um único nodo coordenador de TS
    - gerado pelo nodo que iniciou a transação
      - requer relógios sincronizados
- Técnicas de detecção baseadas em grafo de espera
  - grafo de espera global centralizado
  - grafo de espera global distribuído

Grafo de Espera Global Centralizado

- Grafo mantido pelo coordenador (GEG)
  - união dos grafos de espera locais (GELs)
  - técnicas de manutenção de um GEG
    - atualizado toda vez que um GEL é gerado ou atualizado
    - atualizado periodicamente, após um certo número de modificações nos GELs
    - gerado sempre que for invocado um método de detecção de *deadlock* distribuído
      - geralmente após uma espera longa por um dado
- Problema
  - *overhead* para geração e manutenção do GEG

Grafo de Espera Global Distribuído

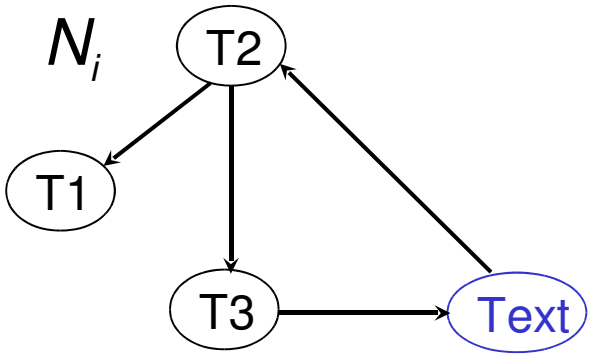
- Todo nodo mantém um GEL
  - cada GEL é uma parte do GEG
  - a técnica sempre garante que um *deadlock* é detectado em algum dos nodos
    - em algum nodo o GEG será reconstruído por completo
- Funcionamento da técnica
  - um GEL de um nodo N_i pode ter um nodo especial T_{ext}
    - T_{ext} é um nodo que representa todas as transações externas envolvidas em esperas com transações iniciadas em N_i
  - ciclo envolvendo transações locais $\Rightarrow$ *deadlock* local
    - problema resolvido em nível local
  - ciclo envolvendo T_{ext} $\Rightarrow$ possibilidade de *deadlock* global
    - invoca-se um método de detecção de *deadlock* distribuído

Detecção de *Deadlock* Distribuído

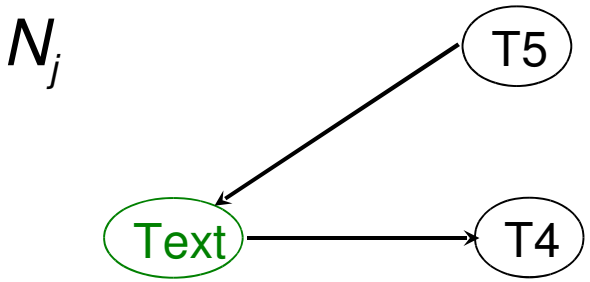
- Um nodo N_i descobre um ciclo com T_{ext}
  - significa que uma transação de N_i espera por um dado bloqueado por uma transação de um nodo externo N_j
- N_i envia o seu GEL para N_j
- N_j atualiza o seu grafo
  - caso ocorra ciclo, N_j toma uma atitude
    - alguma solução adotada por BDCs
- Problema
  - tráfego de GELs na rede

Detecção de *Deadlock* Distribuído

- Exemplo 1

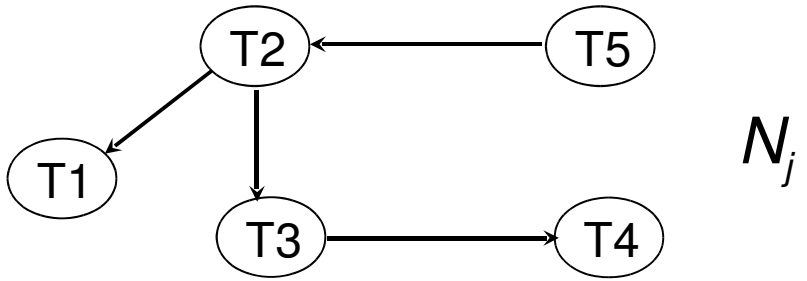


(corresponde
a T4 e T5 de N_j)



(corresponde
a T2 e T3 de N_i)

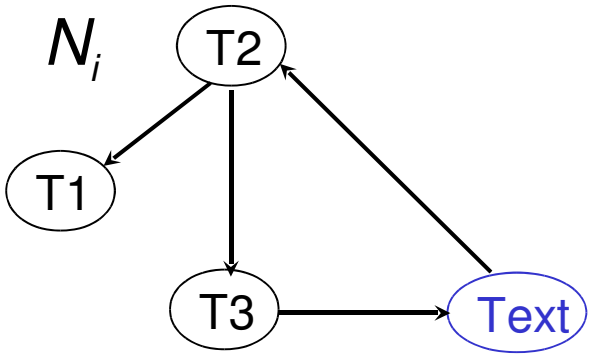
N_i envia GEL para N_j:



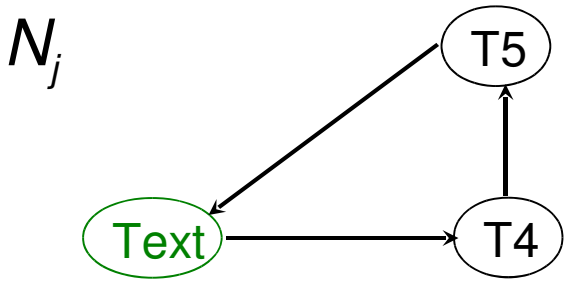
Não há *deadlock* distribuído!

Detecção de *Deadlock* Distribuído

- Exemplo 2

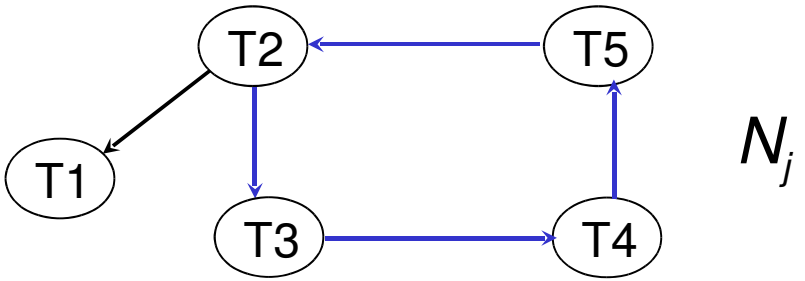


(corresponde
a T4 e T5 de N_j)



(corresponde
a T2 e T3 de N_i)

N_i envia GEL para N_j:



Há *deadlock* distribuído!

Recovery em um BDD

- Controles adicionais para transações distribuídas
  - detecção de falhas no ambiente distribuído
    - nodo ou de comunicação (falhas distribuídas)
  - tratamento de falhas em transações fica a cargo do nodo que gerou a transação
    - o nodo que gera uma transação T_x é o coordenador de *recovery* de T_x

Recovery em um BDD

- Atomicidade e Durabilidade Distribuída
  - técnica 2PC (*2-Phase Commit*)
    - considera-se uma transação distribuída T_x coordenada por um nodo C_i
    - protocolo 2PC inicia quando C_i recebe mensagens de todos os nodos (*"finish T_x"*), sinalizando que já executaram todas as ações de T_x
      - pressupõe que C_i já executou com sucesso as atualizações de T_x e está pronto para o *commit*
        - » caso contrário, C_i teria notificado "*abort T_x*" a priori
      - pressupõe que nenhum nodo participante tenha abortado T_x anteriormente ou tenha falhado sem enviar "*finish T_x*"
        - » neste caso, C_i teria notificado "*abort T_x*" a priori

Técnica 2PC

- Fase 1 – Confirmação de *Commit*
  - C_i envia *"prepare T_x"* para os nodos que executaram T_x
    - C_i grava antes **<prepare Tx>** no seu *Log*
  - cada nodo N_k informa se pode efetivar ou não T_x, enviando *"ready T_x"* ou *"abort T_x"*
    - se N_k enviou *"ready Tx"* e ela foi recebida com sucesso em C_i, N_k grava **<ready Tx>** no seu *Log*

Técnica 2PC

- Fase 2 – Decisão
  - C_i recebe (ou não) respostas e decide o que faz
  - se C_i recebe *"ready T_x"* de todos, ele realiza o *commit*
    - grava **<commit Tx>** no *Log*
    - envia *"commit T_x"* para os nodos e todos registram no *Log*
    - se C_i recebe confirmação do *commit Tx* de todos os nodos (*"acknowledge T_x"*)
      - C_i grava **<complete Tx>** no *Log*
        - » garantia de que *Tx* está realmente encerrada no BDD
        - » evita REDO(*Tx*) em caso de falha posterior
  - caso contrário
    - grava **<abort Tx>** no *Log*
    - envia *"abort T_x"* para os nodos e todos registram no *Log*
    - todos realizam UNDO(*Tx*)

Falha em um Nodo do BDD

- Pode ocorrer enquanto uma transação distribuída T_x está ativa ou aguardando o *commit* distribuído
  - pode afetar um nodo N_k ou o coordenador C_i
- Se N_k falha
  - antes de enviar *"ready T_x"*
    - C_i aborta T_x
  - depois de enviar *"ready T_x"*
    - C_i efetiva T_x
      - C_i assume que quando N_k voltar à ativa, irá se recuperar

Falha em um Nodo do BDD

- Quando N_k volta à ativa, passa por um processo de *recovery* de cada uma das suas transações distribuídas T_x (além das locais!)
  - se encontra `<commit Tx>` $\Rightarrow$ REDO(T_x)
  - se encontra `<abort Tx>` ou nenhum registro sobre o encerramento de T_x $\Rightarrow$ UNDO(T_x)
  - se encontra `<ready Tx>` $\Rightarrow$ consulta C_i para saber o destino de T_x (*"status T_x"*), realizando REDO(T_x) ou UNDO(T_x), conforme o caso

Falha no Coordenador C_i de uma T_x

- Outro nodo participante na execução de T_x assume a coordenação (C_{novo})
  - exemplo: processo de eleição ou ordenação pré-definida de nodos
- C_{novo} questiona o status de T_x em si próprio e nos demais nodos participantes (alguns casos a considerar)
  1. se alguém retorna **<commit T_x>** ou todos retornam **<ready T_x>** $\Rightarrow$ efetiva-se T_x
     - C_{novo} retoma o protocolo 2PC a partir da fase 2
  2. se alguém retorna **<abort T_x>** $\Rightarrow$ aborta-se T_x
     - C_{novo} retoma o protocolo 2PC a partir da fase 2
  3. se ninguém retorna **<commit T_x>** ou **<abort T_x>**, e alguém retorna **<ready T_x>** $\Rightarrow$ T_x está em processo de efetivação
     - C_{novo} retoma o protocolo 2PC a partir da fase 1 (re-envia *"prepare Tx"* para quem não retornou **<ready T_x>**)
  4. se ninguém retorna **<commit T_x>** ou **<abort T_x>** ou **<ready T_x>** $\Rightarrow$ T_x está em andamento
     - deve-se esperar que C_i se recupere (pois C_i pode decidir por abortar *Tx)* OU decidir-se por abortar *Tx*, caso se ultrapasse um *timeout* de espera

Exercício 1

Suponha um BDD com 4 nodos (*N1*, *N2*, *N3* e um nodo coordenador de transações *Ci*) e os seguintes *Logs*:

N1

<start T1>
<write T1,x,1,2>
<start T2>
<ready T1>
<write T2,q,5,7>
<ready T2>

N2

<start T3>
<write T3,g,0,2>
<write T3,h,4,2>
<start T2>
<write T2,r,5,7>
<ready T2>

N3

<start T2>
<write T2,w,1,8>
<start T3>
<write T3,b,1,5>
<ready T2>
<ready T3>

Ci

<start T1>
<start T2>
<start T3>
<write T1,y,8,0>
<write T1,z,1,4>
<write T3,a,5,3>
<write T2,f,6,9>
<prepare T1>
<prepare T2>
<commit T1>
<prepare T3>

a) suponha que *N2* falha. Que ações *Ci* irá tomar com relação a *T2* e *T3*?
b) suponha que *N1* voltou a ativa após uma falha. Que ações ele irá tomar?
c) suponha que o BDD se recupera por completo de uma falha de sistema. Que ações *Ci* irá tomar?
d) suponha que *Ci* falha e *N3* assume a coordenação de *T2* e *T3*. Que ações *N3* irá tomar?
e) suponha que *Ci* falha e *N1* assume a coordenação de *T1* e *T2*. Que ações *N1* irá tomar?

Processamento de Consultas em BDD

- BDC
  - estimativas de custo baseiam-se mais no número de acessos locais a disco
- BDD
  - consultas podem requisitar dados em vários nodos, logo há outros fatores a estimar
    - custo do volume de dados transmitido na rede
      - deve ser o menor possível!
    - processamento paralelo de partes da consulta em diferentes nodos
      - DDD mantém a localização e as filtragens H e V que indicam em quais nodos estão os fragmentos de dados

Processamento de Consultas em BDD

- Dada uma consulta que envolva uma relação *R*, investiga-se como *R* está armazenada no BDD
  - *R* está replicada em nodos remotos
    - escolhe-se o nodo com menor custo de transmissão
      - Obs.: esta premissa vale para qualquer acesso remoto, mesmo s/ replicação
  - *R* está fragmentada em vários nodos
    - deve-se realizar transformações na consulta
      - transformação é feita com base nas informações do DDD
  - *R* está replicada e fragmentada
    - ambas as ações devem ser consideradas

Exemplos de Dados no DDD

```
Nodo filialFpolis.empresa.br
     fragmento FFp
          relação global: Filiais
          predicado FH: cidade = 'Fpolis'
          atributos FV: *
     fragmento FuncFp
          relação global: Funcionários
          predicado FH: filial IN
                    (select código from FFp)
          atributos FV: código, nome,
                        endereço, cargo, DN
Nodo filialBlumenau.empresa.br
     fragmento FBlu
          ...
```

Transformação de Consultas

- Exemplo
  - BDD de uma empresa distribuído por filiais
    - 3 nodos de filiais: Fpolis, Blumenau e Joinville
    - 3 FHs da relação Filiais: FFp, FBlu e FJv
  - Consulta *C*: $\sigma_{\text{cidade = 'Fpolis'}}$ (Filiais)
    - transforma *C* em:

$\sigma_{\text{cidade = 'Fpolis'}}$ (FFp) $\cup$ $\sigma_{\text{cidade = 'Fpolis'}}$ (FBlu) $\cup$ $\sigma_{\text{cidade = 'Fpolis'}}$ (FJv)

predicado de *C* está contido ou é igual ao predicado do FH (resultado final da transformação de *C*)

predicado de *C* + predicado do FH retorna ϕ (esses fragmentos são desconsiderados!)

(FFp)

- se há fragmentação vertical, verificar se algum atributo desejado por *C* (em π ou σ) encontra-se no fragmento
  - se nenhum atributo é encontrado, *C* não executa no fragmento

Tratamento de Junções

- Se junções ocorrem entre dados armazenados em nodos diferentes
  - encontrar uma alternativa que implique menor volume de dados transmitidos entre nodos
- Situação Exemplo
  - Nodo1: Func (codF, nome, sobrenome, DN, salário, ..., *nroFil*)
    - estimativas: n_{Func} = 10.000 tuplas; t_{Func} = 100 bytes; t_{Func}(codF) = 9 bytes; t_{Func}(nroFil) = 4 bytes; t_{Func}(nome) = 15 bytes; t_{Func}(sobrenome) = 15 bytes
  - Nodo2: Filiais (nroFil, nomeFil, cidade, ...)
    - estimativas: $n_{Filiais}$ = 100 tuplas; $t_{Filiais}$ = 35 bytes; $t_{Filiais}$(nroFil) = 4 bytes; $t_{Filiais}$(nomeFil) = 10 bytes

Tratamento de Junções - Exemplo

- Consulta formulada em um Nodo3:
  $\pi_{nome,\ sobrenome,\ nomeFil}$ (Func ⋈ Filiais)
- Temos:
  - tam_{Func}: 10.000 * 100 = 1.000.000 bytes
  - $tam_{Filiais}$: 100 * 35 = 3.500 bytes
  - $t_{resultado}$ = 15 + 15 + 10 = 40 bytes
  - $tam_{resultado}$ = 10.000 * 40 = 400.000 bytes
- Estimando custo de transmissão
  - alternativa 1: transferir as 2 relações para nodo3 e processar a consulta
    - custo transmissão = tam_{Func} + $tam_{Filiais}$ = 1.003.500 bytes

Tratamento de Junções - Exemplo

- Estimando custo de transmissão
  - alternativa 2: transferir a relação Func para o nodo2, processar a consulta no nodo2 e enviar o resultado para o nodo3
    - custo transmissão = $tam_{Func} + tam_{resultado}$ = 1.400.000 bytes
  - alternativa 3: transferir a relação Filiais para o nodo1, processar a consulta no nodo1 e enviar o resultado para o nodo3
    - custo transmissão = $tam_{Filiais} + tam_{resultado}$ = 403.500 bytes **(Melhor!)**

Boa Estratégia para Junções em BDDs

- Consulta formulada no nodo2

  $\pi_{\text{nome, sobrenome, nomeFil}}$ (Func $\bowtie$ $\sigma_{\text{cidade = 'Fpolis'}}$(Filiais))

  - alternativa 1
    - passo1: filtra Filiais pela cidade de *Fpolis* e envia para o nodo1
      - supondo $C_{Filiais}(cidade) = 20$
      - custo transmissão = $C_{Filiais}(cidade) * t_{Filiais} = 20*35 =$ **700 bytes**
    - passo2: realiza a junção em *temp*, projeta os atributos necessários ($\pi_{\text{nome, sobrenome, nroFil}}$ (*temp*)) e envia para o nodo2
      - se $n_{Func} = 10.000$ e $n_{Filiais} = 100$, em média 100 funcionários trabalham em uma filial: $C_{Func}(nroFil) = 10.000/100 = 100$
      - se temos 20 filiais em *Fpolis*, temos 2.000 funcionários selecionados: $n_{temp} = 20 * C_{Func}(nroFil) = 2.000$
      - custo transmissão = $(t_{nome}(temp) + t_{sobrenome}(temp) + t_{nroFil}(temp)) * n_{temp} = 34 * 2.000 =$ **68.000 bytes**
    - custo total transmissão = 700 + 68.000 = 68.700 bytes

Boa Estratégia para Junções em BDDs

- Consulta formulada no nodo2

  $\pi_{\text{nome, sobrenome, nomeFil}}$ (Func $\bowtie$ $\sigma_{\text{cidade = 'Fpolis'}}$(Filiais))

  - alternativa 2
    - passo1: filtra Filiais pela cidade de *Fpolis* em *temp*, projeta *nroFilial* em *temp* e envia *temp* para nodo 1
      - custo transmissão = $C_{Filiais}(cidade)$ * $t_{Filiais}$(nroFil) = 20*4 = 80 bytes
    - passo2: idem ao passo2 da alternativa 1
      - custo transmissão = 68.000 bytes
    - custo total transmissão = 80 + 68.000 = 68.080 bytes
  - **Melhor alternativa, pois somente a chave para junção foi transmitida!**

Processamento Paralelo de Consultas

- Se uma consulta envolve dados de mais de um nodo, o plano de execução pode considerar processamento paralelo
- Exemplos

  $\sigma_{\underbrace{\text{cidade = 'Fpolis'} \wedge \text{zona = 'sul'}}_{\text{processa no nodo2}} \wedge \underbrace{\text{salário > 2000}}_{\text{processa no nodo1}}} (\text{Filiais} \bowtie \text{Func})$

  – supondo filiais fragmentadas por cidade:

  $\underbrace{(\text{FCri}) \cup (\text{FFp})}_{\text{processa no nodo de Fpolis}} \cup \underbrace{(\text{FBlu}) \cup (\text{FJv})}_{\text{processa no nodo de Blumenau}}$

Exercício 2

- A relação Func está no nodo1 e Filiais está fragmentada por cidade, estando as filiais de Fpolis no nodo2 (FFp) ($n_{FFp} = 20$) e as filiais de Blumenau no nodo3 (FBlu) ($n_{FBlu} = 10$)
- Dada a seguinte consulta no nodo2:

$$\pi_{nome,codF,Func.nroFil}(\sigma_{Func.nroFil = Filiais.nroFil}(Func \times Filiais))$$

$$\wedge (Filiais.cidade = \text{'Fpolis'} \vee Filiais.cidade = \text{'Blumenau'})$$

pode-se considerar algumas alternativas, como:

A1) filtrar os funcionários desejados (de *Fpolis* e de *Blumenau*) no nodo1 e enviar o resultado para o nodo 2

A2) trazer os atributos desejados de funcionários e os códigos das filiais de Blumenau para o nodo2 e processar a consulta no nodo2

a) Qual alternativa tem o menor custo de transmissão?

b) O que pode ser processado em paralelo em A1 e A2?