*** Bases de dados Paralelas e Distribuídas ***

Base de dados centralizado

- Os dados estão localizados num lugar (por exemplo, servidor)
- Todas as funcionalidades de DBMS são feitas pelo servidor
  - Aplicação de propriedades ACID das transações
  - Controle simultâneo, mecanismos de recuperação
  - Respondendo a consultas

Bases de dados distribuídas

- Os dados são armazenados em vários lugares (cada um está executando um SGBD)
- Nova noção de transações distribuídas
- As funcionalidades do SGBD agora estão distribuídas em várias máquinas

Porquê as bases de dados distribuídas?

- Escalabilidade
  - Se o volume de dados, a carga de leitura ou a carga de escrita crescerem mais do que uma única máquina pode manipular, pode-se espalhar potencialmente a carga em várias máquinas.
- Tolerância de falha / alta disponibilidade
  - Se a aplicação precisar de continuar a funcionar, mesmo que uma máquina (ou várias máquinas) falhem, podem-se usar várias máquinas para a redundância. Quando um falha, outro pode assumir o controle.
- Latência
  - As aplicações são, por natureza, distribuídas. Se existem utilizadores em todo o mundo, é preferível ter servidores em vários locais do mundo, para que os utilizadores possam ser atendidos a partir de um datacenter geograficamente mais próximo deles.

Por que o processamento paralelo?

- Processando 1 Terabyte
  - em 10MB / s => ~ 1.2 dias para examinar
  - 1000 x paralelo => 1,5 minutos para examinar
- Divida um grande problema em muitos mais pequenos para serem resolvidos em paralelo
- Sistemas de bases de dados paralelas em grande escala são cada vez mais utilizados para:
  - armazenar grandes volumes de informação
  - processamento de consultas demoradas para suporte à decisão
  - fornecendo alto débito para processamento de transações

Maior problema das bases de dados

- Grande volume de dados => use disco e memória grande
- Bottlenecks
  - Velocidade (disco) << velocidade (RAM) << velocidade (microprocessador)
- Previsões
  - Lei de Moore: crescimento da velocidade do processador (com multicore): 50% por ano
  - Crescimento da capacidade DRAM: 4 × a cada três anos
  - Volume do disco: 2 × nos últimos dez anos
- Maior problema: Bottlenecks do I / O
- Solução para aumentar a largura de banda de I / O
  - partição de dados

- acesso paralelo aos dados

Bases de dados paralelas

- As bases de dados paralelas melhoram o processamento e as velocidades de E / S usando vários CPUs e discos em paralelo
  - os dados podem ser divididos em vários discos
  - cada processador pode trabalhar de forma independente na sua própria partição
- Explota o paralelismo no gerenciamento dos dados para fornecer alto desempenho, alta disponibilidade e extensibilidade
  - Suporta bases de dados muito grandes com cargas muito altas
- Diferentes consultas podem ser executadas em paralelo
- O controle simultâneo cuida dos conflitos
- Problemas críticos:
  - colocação de dados
  - processamento de consulta paralela
  - balanceamento de carga
- A maioria das pesquisas foi feita no contexto do modelo relacional que fornece uma boa base para o paralelismo baseado em dados
  - operações relacionais individuais (por exemplo, classificar, juntar, agregar) podem ser executadas em paralelo

Bases de dados paralelas vs distribuídas

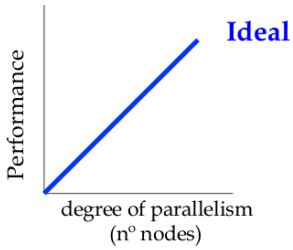
- Embora os princípios básicos do DBMS paralelas sejam os mesmos que os DBMS distribuídas, as técnicas para sistemas de base de dados paralelas são bastante diferentes …
  - DB paralelas
    - Interconexão rápida
    - Software homogêneo
    - O alto desempenho é objetivo
    - Transparência é objetivo
  - DB Distribuídas
    - Distribuiu geograficamente
    - O compartilhamento de dados é objetivo (pode ter heterogeneidade, autonomia)
    - Operação desconectada possível
- O processamento distribuído geralmente faz uso de processamento paralelo (não vice-versa)
  - pode ter processamento paralelo em uma única máquina
  - Suposições
    - Bases de dados paralelas
      - As máquinas estão fisicamente próximas uma da outra (por exemplo, a mesma sala de servidores)
      - Máquinas conectadas com LANs e switches dedicados de alta velocidade
      - O custo da comunicação é considerado pequeno
      - Arquitetura: pode ser partilhada – memória e disco
    - Bases de dados distribuídas
      - As máquinas podem estar em locais geográficos distintos
      - São conectados usando rede de propósito público, por exemplo, Internet
      - O custo e os problemas de comunicação não podem ser ignorados
      - Arquitetura: geralmente não existe partilha

SGBD paralelo - Principais objetivos

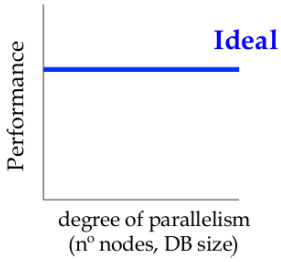
- Alto desempenho através da paralelização de várias operações
  - Alto rendimento com paralelismo inter-consulta
  - Tempo de resposta baixo com paralelismo intra-operação
  - O balanceamento de carga é a capacidade do sistema para dividir uma determinada carga de trabalho igualmente entre todos os processadores
- Alta disponibilidade, explorando a replicação de dados
- Extensibilidade com os objetivos ideais
  - Velocidade linear
  - Ampliação linear

Cenário de extensibilidade ideal

- Speed-Up
  - refere-se a um aumento linear de desempenho para uma constante do tamanho da base de dados enquanto o número de nós (isto é, processamento e energia de armazenamento) é aumentado linearmente
  - mais recursos significa proporcionalmente menos tempo para a quantidade de dados



- Scale-Up
  - refere-se a um desempenho sustentado para um aumento linear no tamanho da base de dados e no número de nós
  - se os recursos aumentaram proporcionalmente ao aumento do tamanho dos dados, o tempo é constante



Barreiras para paralelismo

- Inicialização
  - O tempo necessário para iniciar uma operação paralela pode dominar o tempo de computação real
- Interferência
  - Ao acessar recursos compartilhados, cada novo processo retarda os outros (problema de ponto quente)
- Skew
  - O tempo de resposta de um conjunto de processos paralelos é o tempo do mais lento
- As técnicas paralelas de gerenciamento de dados pretendem superar essas barreiras

Arquiteturas da base de dados

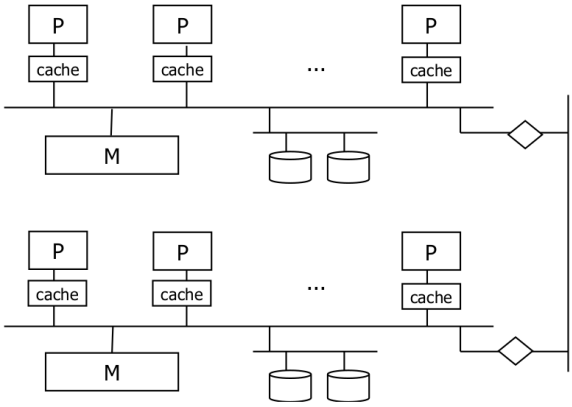
- Arquiteturas para escalar para uma carga maior ...
- Arquitetura multiprocessador
  - Shared memory (SM)
  - Shared disk (SD)
  - Shared nothing (SN)
- Arquiteturas híbridas
  - Non-Uniform Memory Architecture (NUMA)
  - Cluster

NUMA

- Memória partilhada vs. Memória Distribuída
  - mistura dois aspectos diferentes: endereçamento e memória
    - endereçamento: espaço de endereço único versus vários espaços de endereço
    - memória física: central vs distribuída
- NUMA usa espaço de endereço único em memória física distribuída
  - facilita a portabilidade das aplicações
  - extensibilidade
- Cache Coherent NUMA (CC-NUMA)
  - o mais bem sucedido

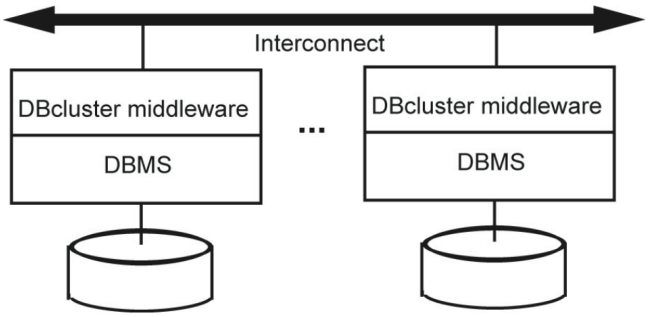
CC-NUMA

- Princípio
  - memória principal distribuída como shared-nothing
  - no entanto, qualquer processador tem acesso a todos os outros processadores da memória
    - acesso à memória remota muito eficiente, apenas algumas vezes (normalmente entre 2 e 3 vezes) o custo do acesso local
- Processadores diferentes podem acessar os mesmos dados em um modo de atualização conflituosa
  - são necessários protocolos de consistência de cache global…



Cluster

- Combina um bom balanceamento de carga de Memória partilhada com extensibilidade de Shared Nothing
- Nó do servidor: componentes disponíveis na plataforma
  - de simples componentes de PC para SMP mais poderoso
  - a melhor relação custo / desempenho
  - forma mais barata
- Conexões de rede rápidas (Gigabits / seg) e baixa latência



SN cluster vs SD cluster

- O **SN cluster** pode produzir o melhor custo / desempenho e extensibilidade
  - mas adicionar ou substituir nós de cluster requer reorganização de disco e dados
- O **SD cluster** evita essa reorganização, mas exige que os discos sejam acessados globalmente pelos nós do cluster
  - Network-attached storage (NAS)
    - protocolo de sistema de arquivos distribuído, como NFS, relativamente lento e não apropriado para gerenciamento de banco de dados
  - Storage-area network (SAN)
    - Protocolo baseado em bloco, facilitando assim a consistência do cache, eficiente, mas mais caro.

Técnicas de SGBD Paralelas e Distribuídas

- Colocação de dados
  - Posicionamento físico da base de dados em vários nós
  - Estático vs. Dinâmico
- Algoritmos de processamento de dados paralelos
  - Select é fácil
  - Join (e todas as outras operações non-select) é mais difícil
- Otimização de consulta paralela
  - Escolha dos melhores planos de execução paralela
  - Paralelização automática das consultas e balanceamento de carga
- Gerenciamento de transações distribuídas

Armazenamento de dados distribuídos

- Duas formas comuns de distribuir informação por vários nós
- **Replicação**
  - mantendo uma cópia dos mesmos dados em vários nós diferentes, potencialmente em diferentes locais
  - fornece redundância: se alguns nós não estiverem disponíveis, os dados ainda podem ser servidos dos nós restantes
  - também pode ajudar a melhorar o desempenho
- **Particionamento**
  - dividir uma grande base de dados em subconjuntos menores chamados de partições
  - diferentes partições podem ser atribuídas a diferentes nós
- Replicação e particionamento podem ser combinados

Transparência de dados

- Definição
  - Grau em que o utilizador do sistema pode permanecer inconsciente dos detalhes de como e onde os itens de dados são armazenados em um sistema distribuído
- Considere problemas de transparência em relação a:
  - Transparência de replicação
  - Particionamento da transparência
  - Transparência de localização

Paralelismo de I/O

- Reduzir o tempo necessário para recuperar as relações do disco por particionamento
- Particionamento horizontal - os tuplos de uma relação são divididas entre vários discos, de modo que cada tuplo reside num disco
- Técnicas de particionamento * (número de discos = n)
  - **Round-robin**
    - envia o i-ésimo tuplo inserido na relação com o disco: i mod n.
  - **Hash partitioning**
    - aplica uma função hash para um ou mais atributos que variam num range de 0 ... n-1
  - **Range partitioning**
    - associa um intervalo de atributo (s) chave a cada partição

* visão mais simples - uma descrição mais detalhada nos próximos capítulos

Query Paralelismo

- **Interquery** Parallelism
  - Execução paralela de várias queries geradas por transações concorrentes
- **Intraquery** Parallelism
  - Execução de uma única query em paralelo em vários processadores / discos

Interquery Parallelism

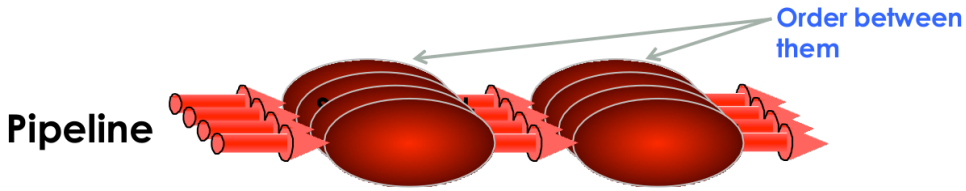
- Para aumentar a taxa de transferência transacional
  - usado principalmente para ampliar um sistema de processamento de transações para suportar um maior número de transações por segundo
- Forma mais fácil de paralelismo para suporte numa base de dados paralela da memória partilhada
- Mais complicado num shared-disk ou shared-nothing
  - bloquear e registrar deve ser coordenado passando mensagens entre processadores
  - dados num buffer local podem ter sido atualizados num outro processador
  - a coerência do cache deve ser mantida: leituras e escritas de informação no buffer devem encontrar a versão mais recente da informação

Intraquery Parallelism

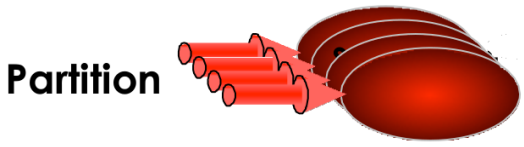
- O mesmo operador é executado por muitos processadores, cada um trabalhando num subconjunto dos dados
  - para acelerar queries longas
- Duas formas complementares de intraquery parallelism:
  - Intra-operação: paraleliza a execução de cada operação individual na consulta
  - Inter-operação: executa as diferentes operações em uma expressão de query em paralelo
- Intra-operation escala melhor com paralelismo crescente, porque o número de tuplas processadas por cada operação normalmente é mais do que o número de operações em uma consulta

Paralelismo do operador IntraQuery

- Inter-operator (Pipeline)
  - tarefas ordenadas (ou parcialmente encomendadas) e diferentes máquinas estão executando tarefas diferentes



- Intra-operator (Particionado)
  - uma tarefa dividida em todas as máquinas para executar em paralelo



Processamento de dados paralelo

- Supondo:
  - somente queries de leitura
  - arquitetura shared-nothing

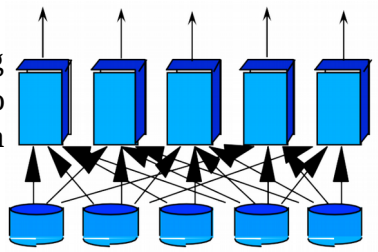
    - n processadores (P0, ..., Pn-1) e n discos (D0, ..., Dn-1) onde o disco Di está associado ao processador Pi
- Se um processador tiver vários discos, eles podem simplesmente simular um único disco Di
- Arquiteturas shared-nothing podem ser simuladas de forma eficiente em shared-memory e shared-disk systems
    - os algoritmos para sistemas shared-nothing podem, portanto, ser executados em sistemas de memória e disco partilhado
    - no entanto, algumas otimizações podem ser possíveis

Algoritmos paralelos

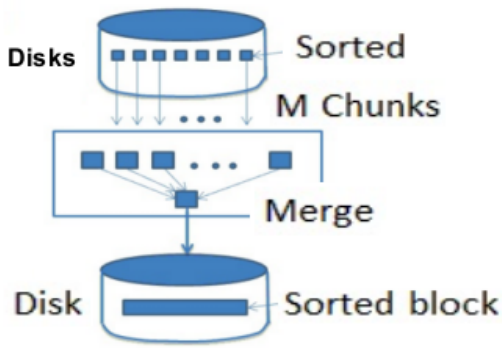
- Caso de uso:
    - Modelo Relacional
- Algoritmos paralelos para operadores de álgebra relacionais são os blocos de construção necessários para consulta paralela em processamento
- O processamento paralelo de dados deve explorar o intra-operator parallelism
- Foco em operadores sort, select e join
    - outros operadores binários (como union)) podem ser tratados em maneira similar de se juntar

Classificação paralela

- Range-Partitioning Sort
    - Escolha dos processadores P0, ..., Pm, onde m <= n -1 para ordenar
    - Re-partição R baseado em intervalos (nos atributos de classificação) em m partições
        - este passo requer I/O e sobrecarga de comunicação
    - Máquina i recebe todas as partições de todas as máquinas e classifica essa partição, sem qualquer interação com os outros
        - Pi armazena as tuplas que recebeu temporariamente no disco Di
    - A operação de merge final é trivial: range-partitioning garante que, para 1 j m, os valores das chaves no processador Pi são todos menos do que os valores-chave em Pj



    - Os dados negativos são um problema
        - os intervalos podem ser de largura diferente
        - aplicar primeiro a fase de amostragem
- Parallel External Sort-Merge
    - Suponha que a relação já tenha sido particionada entre os discos D0, ..., Dn-1 (de qualquer maneira).
    - Cada nó classifica os seus próprios dados
    - Todos os nós começam a enviar os seus dados ordenados (um bloco de cada vez) para uma única máquina
    - Esta máquina aplica a técnica de merge-sort à medida que os dados chegam



Seleção Paralela (Scan) σ_c (R)

- A relação **R** é dividida em m máquinas
    - cada partição de R é em torno de | R | / m tuplos
- Cada máquina verifica a sua própria partição e aplica a condição de seleção c
- Os dados são particionados usando

    - **round robin ou uma função hash** (em toda o tuplo)
        - a relação deve ser bem distribuída em todos os nós
        - em todas as partições será feito o scan
    - **range ou hash-based** (na coluna de seleção)
        - a relação pode ser agrupada em alguns nós
        - poucas partições precisam ser tocadas
- A projeção paralela também é direta
    - todas as partições serão tocadas
    - não é sensível a como os dados são particionados

Parallel Join

- A operação de junção requer que os pares de tuplas sejam testados para verificar se eles satisfazem a condição de join
- Se os tuplos satisfazerem a condição de join, o par é adicionado à saída do join
- Os algoritmos de Parallel join tentam dividir os pares a serem testados em vários processadores
- Cada processador então calcula parte do join localmente
- Finalmente, os resultados de cada processador podem ser juntados para produzir o resultado final

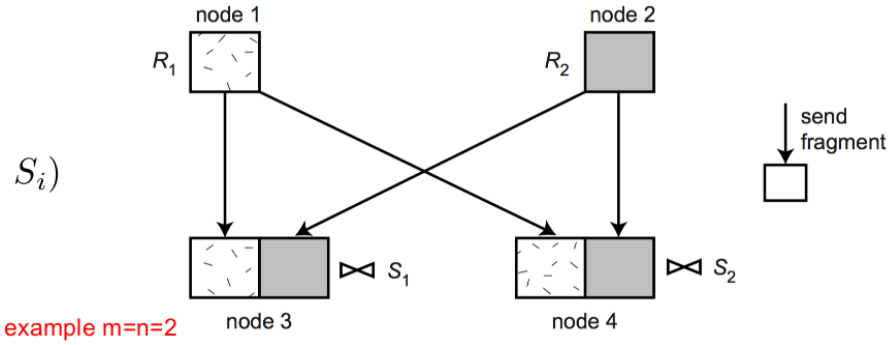
Join Algorithms

- Três algoritmos básicos de união paralela para bases de dados particionadas
    - Parallel Nested Loop (PNL)
    - Parallel Associative Join (PAJ)
    - Parallel Hash Join (PHJ)
- Todos os algoritmos anteriores são paralelismo intra-operator
- Eles também se aplicam a outros operadores complexos, como a eliminação duplicada, a união, a interseção, etc., com menor adaptação
- Exemplos seguintes:
    - join de duas relações R e S que são divididas em nós m e n, respectivamente

Parallel Nested Loop Join

- Mais simples e mais gerais. Basicamente, compõe o produto cartesiano das relações R e S em paralelo
- O algoritmo funciona em duas fases:
    1. cada fragmento de R é enviado e replicado em cada nó contendo um fragmento de S (existem n desses nós)
        - esta fase é feita em paralelo por m nós
    2. cada S-node j recebe a relação R inteiramente, e junta localmente R com o fragmento Sj.
        - Esta fase é feita em paralelo por n nós.
        - O processamento de junção pode começar assim que os dados são recebidos

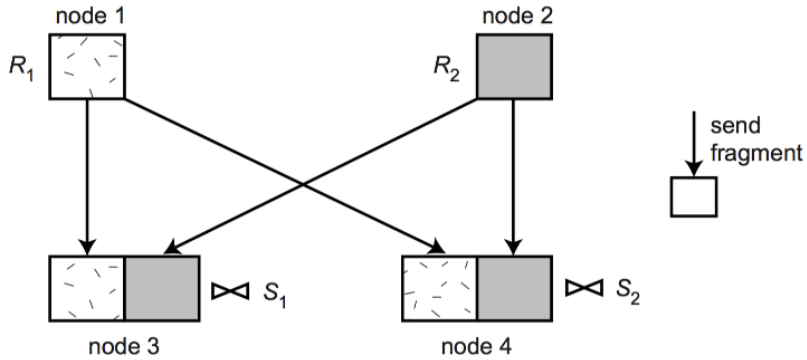
$$R \bowtie S = \bigcup_{i=1}^{n} (R \bowtie S_i)$$



Associação associativa paralela

- Aplica-se apenas a equijoin com uma das relações de operando particionadas de acordo com o atributo join
- Suponha
  - equijoin predicado está no atributo A de R e B de S
  - S é particionado de acordo com a função hash aplicada ao atributo B
    - os tuplos de S que possuem o mesmo valor h (B) são colocadas no mesmo nó
  - nenhum conhecimento de como R é particionado
- O algoritmo prossegue em duas fases:
  1. A relação R é enviada associativamente aos nós S com base na função hash h aplicada ao atributo A
  2. Cada S-node j recebe em paralelo dos diferentes nós R o subconjunto relevante de R (i.e., Rj) e o une localmente com os fragmentos Sj

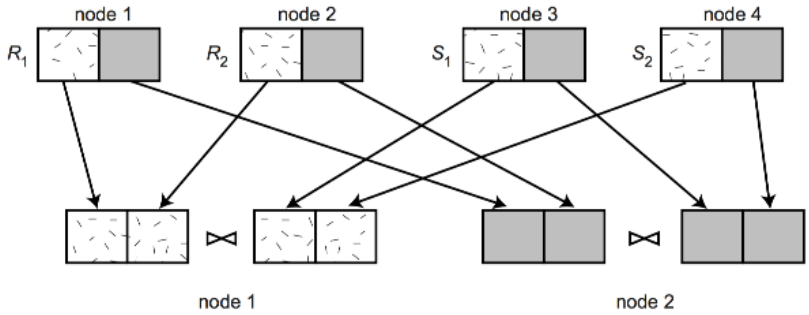
$$R \bowtie S = \bigcup_{i=1}^{n} (R \bowtie S_i)$$



Parallel Hash Join

- Generalização do algoritmo de junção associativa paralela
- Também se aplica a equijoin, mas não requer qualquer particionamento particular das relações do operando
- Idéia básica: partição de R e S no mesmo número p de conjuntos mutuamente exclusivos (fragmentos) R1, R2, ..., Rp e S1, S2, ..., Sp,
- Os nós p podem realmente ser selecionados em tempo de execução com base na carga do sistema
- O algoritmo pode ser dividido em duas fases principais:
  1. build: hashes R no atributo join, envia para o alvo p nodes que compõem uma tabela hash para as tuplas recebidas
  2. sondagem: envia S associativamente para os nós p alvo que sondam a tabela hash para cada tupla entrante

$$R \bowtie S = \bigcup_{i=1}^{p} (R_i \bowtie S_i)$$



Processamento paralelo - Custos

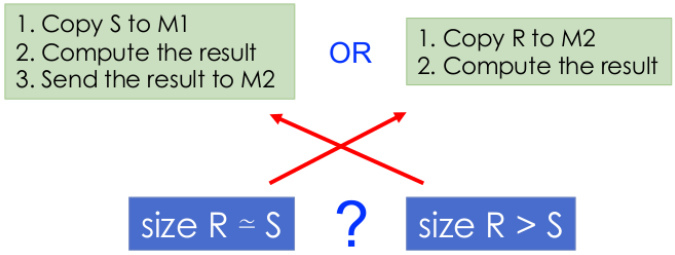
- O processo de join é conseguido com um grau de paralelismo de n ou p
- Cada algoritmo requer mover pelo menos uma das relações do operando
- Cenário ideal: sem distorção no particionamento, e sem sobrecarga devido à avaliação paralela
  - speed-up esperado: 1 / n
- Se a inclinação e as despesas gerais também devem ser levadas em consideração, o tempo gasto por uma operação paralela pode ser estimado como:
- Tcost = Tpart + Tasm + max (T0, T1, ..., Tn-1)
  Tpart - tempo para particionar as relações (incluindo custos de comunicação)
  Tasm - tempo para montar os resultados
  Ti - tempo necessário para a operação no processador Pi. Isso precisa ser estimado tendo em conta a inclinação e o tempo desperdiçado em contenções

Otimização de query paralela

- O objetivo é selecionar o "melhor" plano de execução paralelo para uma query usando os seguintes componentes:
  - Procurar espaço
    - Modelos de planos de execução alternativos como árvores de operação
    - Left-deep vs. Right-deep vs. Bushy trees
  - Estratégia de pesquisa
    - Programação dinâmica para espaço de busca pequeno
    - Randomizado para grande espaço de pesquisa
  - Modelo de custo (abstração do sistema de execução)
    - Informações do esquema físico. (particionamento, índices, etc.)
    - Estatísticas e funções de custo
- Meta: minimizar o movimento de dados entre máquinas

Melhor Plano de Execução - Exemplo

- Duas máquinas
  - M1 tem a relação R (A, B)
  - M2 tem a relação S (C, D)
- Query
- SELECIONE A, C DE R junte S em B = D;
- Resultado
  - deve estar em M2
- Opções:



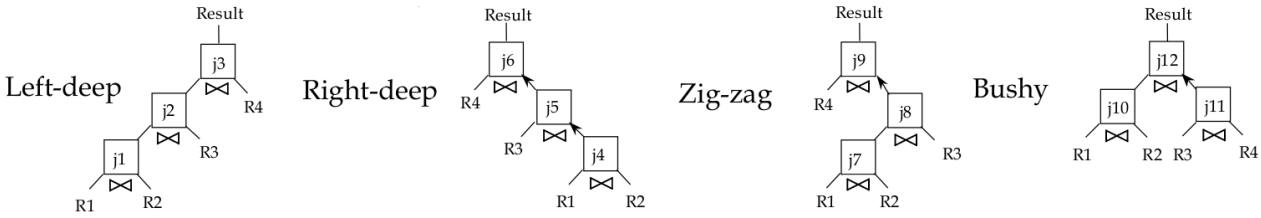
- Cenários:

- Plano de execução ainda melhor:
  1. 1. No cálculo M2
     INSERT INTO TEMP1 SELECT DISTINCT D FROM S;
  2. Copie TEMP1 para M1
  3. No cálculo M1
     INSERT INTO TEMP2
     SELECT A, B FROM R join TEMP1 on B = D;
  4. Copie TEMP2 para M2

     $R \ltimes S$

  5. No cálculo M2
     INSERT INTO ANSWER
     SELECT A, C FROM TEMP2 join S on B = D;
- TEMP2 é left semijoin de R e S
- Muito bom se TEMP1 e TEMP2 forem relativamente pequenos

Search space - Operator Trees



Balanceamento de carga

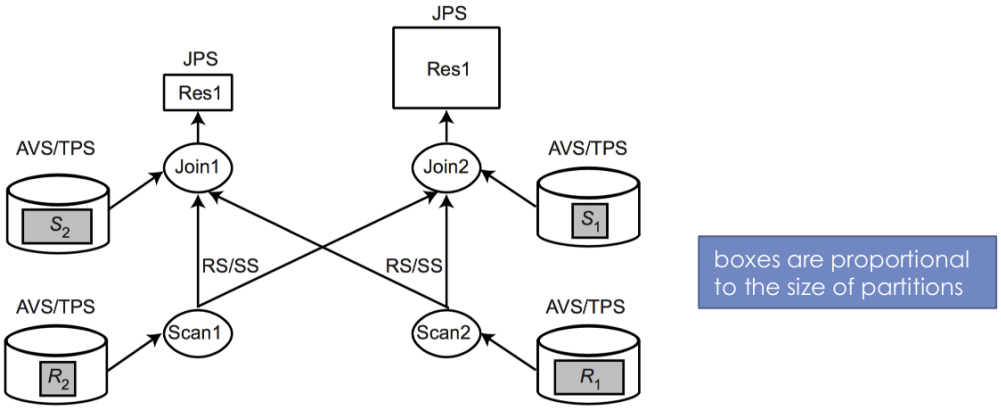
- Equilibrar o load de diferentes transações e queries entre diferentes nós é essencial para maximizar a taxa de transferência
- Surgem problemas para o intra-operator parallelism com distribuições de dados distorcidas
  - attribute data skew (AVS)
    - inerente ao conjunto de dados (por exemplo, há mais cidadãos em Paris do que em Aveiro).
  - tuple placement skew (TPS)
    - introduzido quando os dados são inicialmente particionados (por exemplo, com particionamento de intervalo)
  - selectivity skew (SS)
    - introduzido quando há variação na seletividade de predicados selecionados em cada nó
  - redistribution skew (RS)
    - ocorre no passo de redistribuição entre dois operadores (semelhante ao TPS)
  - join product skew (JPS)
    - ocorre porque a seletividade de junção pode variar entre nós
- Soluções
  - algoritmos paralelos sofisticados que lidam com distorção
  - alocação dinâmica do processador (no tempo de execução)

Data Skew - Exemplo

- Uma consulta sobre duas relações R e S que são mal particionadas

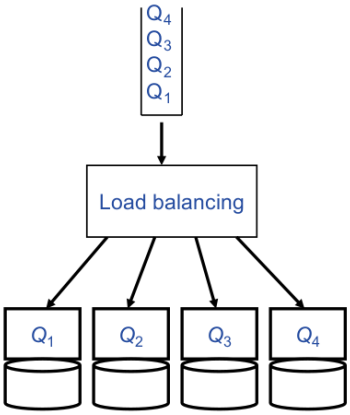
- devido a dados (AVS) ou à função de particionamento (TPS)
- tempos de processamento de instâncias (scan1 e scan2) não são iguais

- O caso do operador de join é pior
  - o número de tuplos recebidos é diferente devido à má redistribuição das partições (RS) ou seletividade variável de acordo com a partição de R processada (SS)
  - tamanho irregular das partições S (AVS / TPS) produz tempos de processamento diferentes para os tuplos enviados pelo operador de sacn. O tamanho do resultado é diferente de uma partição para outra devido ao join selectivity (JPS)



Balanceamento de carga em um cluster da base de dados

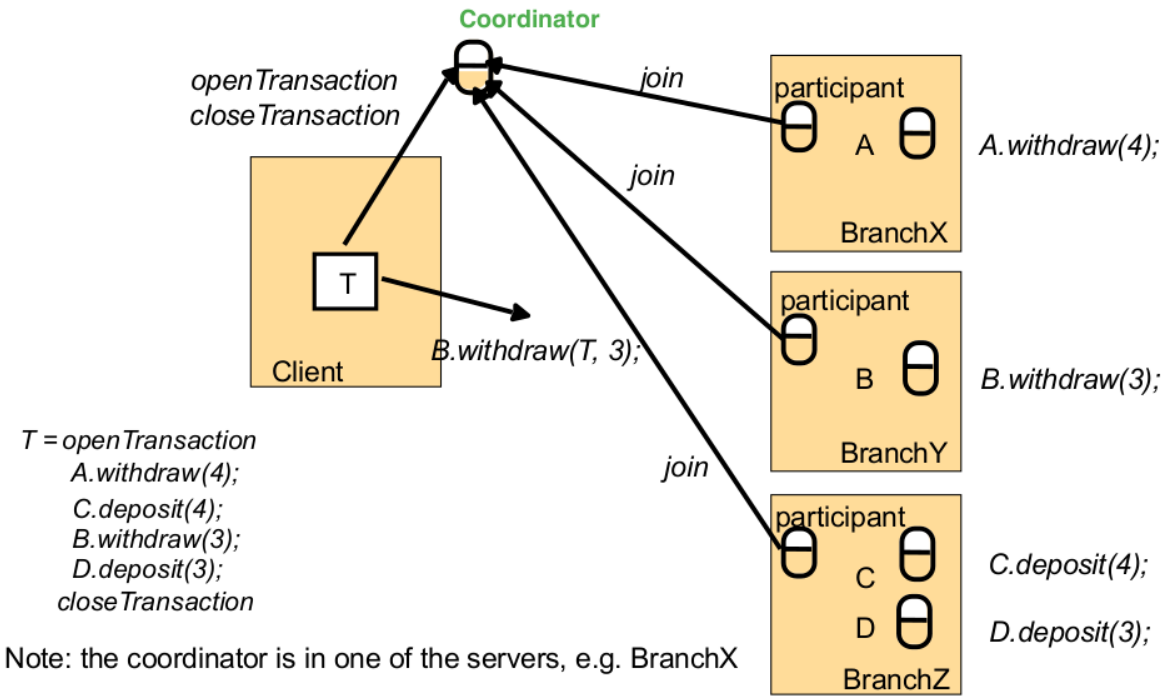
- Escolha o nó para executar Q
  - round robin
  - the least loaded
    - Precisa obter informações de carga
- Failover
  - No caso de um nó N falhar, as consultas de N são assumidas por outro nó
    - requer uma cópia dos dados de N ou SD
- Em caso de interferência
  - os dados de um nó sobrecarregado são replicados para outro nó



Transações Distribuídas

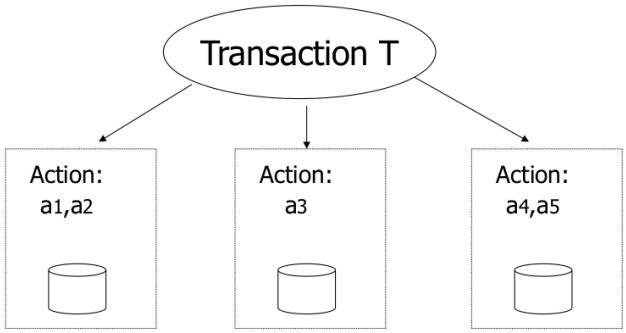
- A transação pode aceder dados em vários sites
- Cada site possui um **gerente de transações** local responsável por:
  - manter um registro para fins de recuperação
  - participando na coordenação da execução simultânea das transações que executam nesse site
- Cada site possui um **coordenador de transações**, responsável por:
  - iniciar a execução de transações originadas no site
  - distribuindo subtransações em locais apropriados para execução
  - coordenando o término de cada transação que se origina no site, o que pode resultar na transação sendo comprometida em todos os sites ou abortada em todos os sites

Distributed Banking Transaction



Distributed commit problem

- Commit deve ser atômico ...
- Como uma transação distribuída que possui componentes em vários sites pode executar atomicamente?
- Solução: Commit em duas fases (2PC), 2PC centralizado, 2PC distribuído, 2PC linear, etc.



Protocolo de commit em duas fases

- Primeira fase - coordenador coleta um voto (commit or abort) de cada participante
  - Participante armazena resultados parciais em armazenamento permanente antes de votar
- Segunda fase - o coordenador toma uma decisão
  - se todos os participantes desejam fazer o commit e ninguém crash, o coordenador envia para todos os participantes a mensagem de commit
    - todos comprometem
    - se o participante falhar, de seguida, para recuperar, pode obter mensagens de commit do coordenador
  - caso contrário, se algum participante tiver feito “crashed or aborted”, o coordenador envia uma mensagem "abortar" a todos os participantes
    - todos abortam

Resumo

- Sistemas Centralizados versus Distribuídos versus Paralelizados
- Parallel Databases
  - Conceito / Objetivos
  - Arquiteturas
  - Tipos de paralelismo
  - Técnicas de SGBD
    - Colocação de dados

- Algoritmos de processamento
- Otimização de queries
- Gerenciamento de transações