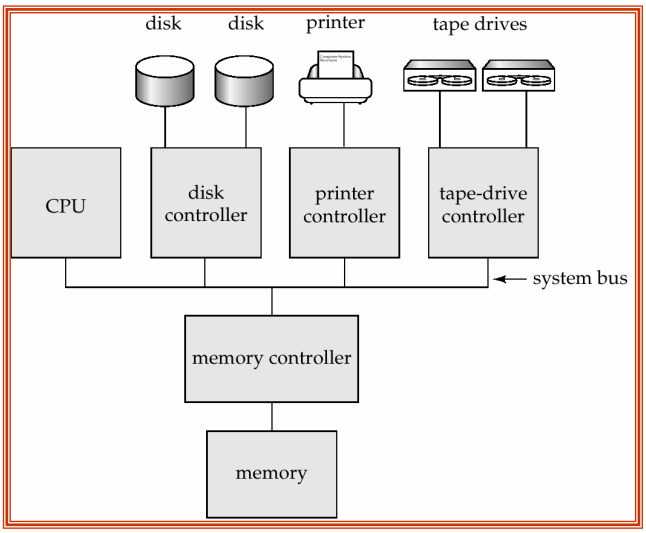
Arquitecturas de Sistemas de Base de Dados

- Sistemas Centralizados
- Sistemas Cliente-Servidor
- Sistemas Paralelos
- Sistemas Distribuídos

Sistemas Centralizados

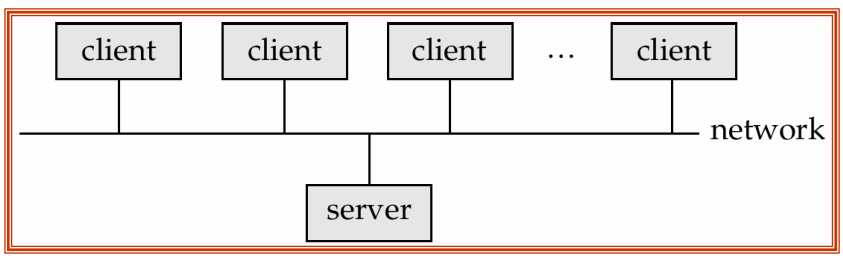
- Correm sobre um único computador e não interagem com outros sistemas.
- Sistema genérico de computador: um ou mais CPUs e um número de controladores que estão ligados por um bus comum que permite o acesso a memória partilhada.
- Sistema Single-user (e.g., computador pessoal ou workstation): unidade desk-top, single user, normalmente tem um único CPU e um ou dois discos rígidos; o SO poderá suportar apenas um utilizador.
- Sistema Multi-user: mais discos, mais memória, vários CPUs, e SO multi-user. Serve um número grande de utilizadores que estão ligados ou sistema através de terminais. São muitas vezes chamados de sistemas *servidor*.

Um Sistema Centralizado



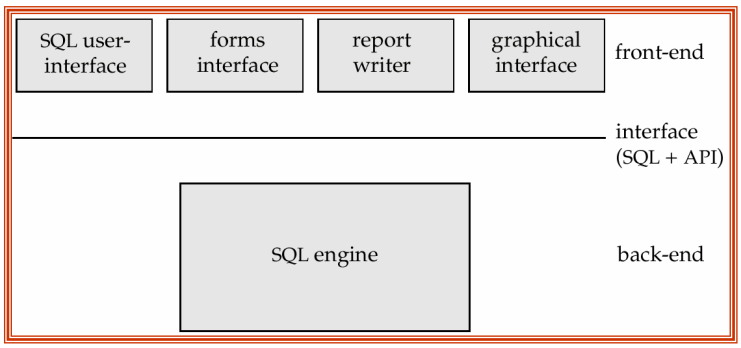
Sistemas Cliente-Servidor

- Sistemas Servidor respondem a pedidos gerados em *m* sistemas clientes, cuja estrutura genérica é mostrada abaixo:



Sistemas Cliente-Servidor (Cont.)

- As funcionalidades das BD's podem ser divididas em:
  - **Back-end**: gerem estruturas de acesso, avaliação de consultas e optimização, controlo de concorrência e recuperação.
  - **Front-end**: consiste em ferramentas tais como *formulários*, geradores de *relatório*, e interfaces gráficas do utilizador.
- A interface entre o front-end e o back-end é através de SQL ou através de uma API (application program interface).



Sistemas Cliente-Servidor (Cont.)

- Vantagens da substituição de mainframes por redes de workstations ou computadores pessoais ligados a máquinas servidoras de back-end:
  - Melhor relação funcionalidade/custo
  - Flexibilidade na atribuição de recursos e capacidades de expansão
  - Melhores interfaces de utilizador
  - Manutenção mais fácil
- Os sistemas servidores podem ser classificados em dois tipos:
  - **Servidores transaccionais** que são largamente usados em sistemas de base de dados relacionais, e
  - **Servidores de dados**, usados em sistemas de base de dados orientados por objectos.

Servidores Transaccionais

- Também chamados sistemas **servidores de consultas** ou SQL *server* systems; os clientes enviam pedidos para o sistema servidor onde as transacções são executadas, e os resultados são enviados de volta para o cliente.
- Os pedidos são especificados em SQL, e comunicados ao servidor através de um mecanismo de *remote procedure call (RPC)*.
- RPC transaccional permite que várias chamadas RPC formem colectivamente uma transacção.
- *Open Database Connectivity (ODBC)* é uma API em linguagem C da Microsoft que define um standard para ligação a um servidor, enviando pedidos SQL, e recebendo os resultados.
- JDBC é um standard similar a ODBC, para Java

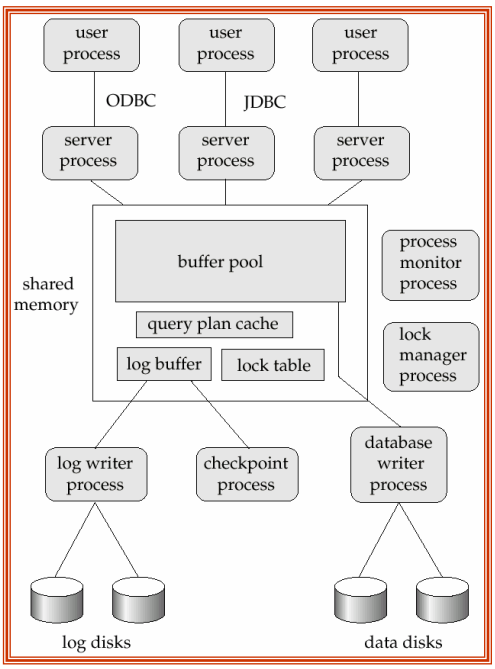
Estrutura de processos de um sistema transaccional

- Um servidor transaccional típico possui múltiplos processos a aceder a dados em memória partilhada.
- Processos Servidor
  - Recebem consultas dos utilizadores (transacções), executam-nas e enviam o resultado de volta
  - Os processos podem ser **multithreaded**, permitindo que um único processo possa executar várias consultas concorrentemente
  - Temos tipicamente múltiplos processos servidor multithreaded
- Processo de gestão de Locks
  - Veremos mais à frente
- Processo de escrita na Base de Dados
  - O buffer de output modificado é escrito em blocos no disco continuamente

Processos do Servidor Transaccional (Cont.)

- Processo de escrita de Log
  - Os processos servidor adicionam registos log ao buffer de registos log
  - O processo de escrita de Log guarda os registos de log em suporte não-volátil.
- Processo checkpoint
  - Executa checkpoints periódicos
- Processo monitor de processos
  - Monitoriza os outros processos, e toma medidas de recuperação se algum destes processos falha
    - E.g. abortar as transacções a serem executadas por um processo servidor e reiniciar o processo

Processos do Servidor Transaccional (Cont.)



Processos do Servidor Transaccional (Cont.)

- A memória partilhada contém dados partilhados
  - Buffer pool
  - Lock table
  - Log buffer
  - Consultas em cache (reusadas se a mesma consulta volta a ser submetida)
- Todos os processos da BD podem aceder à memória partilhada
- Para assegurar que dois processos não estão a aceder à mesma estrutura de dados ao mesmo tempo, os SGBD implementam **exclusão mútua** usando alternativamente:
  - Semáforos do sistema operativo
  - Instruções atómicas tais como test-and-set

Processos do Servidor Transaccional (Cont.)

- Para evitar o overhead da comunicação interprocessos para pedido/atribuição de lock, cada processo da base de dados opera directamente sobre a estrutura de dados da tabela de lock em vez de enviar pedidos para o processo de gestão de locks
  - Exclusão mútua na tabela de locks é assegurada usando semáforos, ou mais usualmente, instruções atómicas
  - Se um lock pode ser obtido, a tabela de lock é actualizada directamente na memória partilhada
  - Se um lock não pode ser obtido imediatamente, o pedido de lock é anotado na tabela de lock e o processo (ou thread) espera até o lock ser concedido
  - Quando um lock é libertado, o processo que o liberta actualiza a tabela de lock para registar a libertação do lock, e concede o lock aos pedidos em espera (se houver algum)
  - O processo/thread à espera de lock pode alternativamente:
    - Ler continuamente a tabela de lock para verificar a concessão de lock, ou
    - Usar o mecanismo de semáforos do sistema operativo para esperar num semáforo.
      - O identificador de semáforo é guardado na tabela de lock
      - Quando o lock é libertado, o processo que o libertou sinaliza o semáforo para dizer ao processo/thread para prosseguir
- O processo de gestão de locks é usado para detecção de deadlocks

Servidores de Dados

- Usados em LANs, onde existe uma conexão de velocidade muito alta entre os clientes e o servidor, as máquinas cliente tem um poder de processamento comparável ao servidor, e as tarefas a executar são computacionalmente complexas.
- Enviar dados para as máquinas cliente onde o processameneto é efectuado, e depois os resultados são enviados de volta para o servidor.
- Esta arquitectura requer uma completa funcionalidade de back-end nos clientes.
- É usado em muito OO-SGBD
- Algumas questões:
  - Page-Shipping versus Item-Shipping
  - Locking
  - Data Caching
  - Lock Caching

Servidores de Dados (Cont.)

- **Page-Shipping** versus **Item-Shipping**
  - Unidade de envio mais pequena ⇒ mais mensagens
  - Compensa o **prefetch** de items relacionados junto com o item pedido
  - Page shipping pode ser visto como uma forma de prefetching
- Locking
  - O overhead de pedir e obter locks do servidor é alto devido aos tempos das mensagens
  - Podemos atribuir locks nos items requeridos e prefetched; com page shipping, uma transacção obtem lock sobre toda a página.
  - Locks num item *prefetched* podem ser *called back* pelo servidor, e é retornado pela transacção cliente se o item *prefetched* não foi usado.
  - Locks sobre a página podem ser **desescalados** para locks sobre items na página quando existem conflitos de locks. Locks sobre items não usados podem então ser retornados ao servidor.

Servidores de Dados (Cont.)

- **Data Caching**
  - Os dados podem ser mantidos em cache no cliente mesmo entre transacções
  - No entanto, é fundamental verificar a actualidade dos dados antes da sua utilização (**cache coherency**)
  - A verificação pode ser feita aquando do pedido de lock do item
- **Lock Caching**
  - Os locks podem ser mantidos pelo sistema cliente mesmo entre transacções
  - As transacções podem obter os locks em cache localmente, sem contactar o servidor
  - O servidor pode fazer o **call back** de locks dos clientes quando recebe pedidos conflituosos de locks. O cliente devolve o lock assim que não hajam transacções locais a usá-lo.
  - Similar a *deescalation*, mas entre transacções.

Sistemas Paralelos

- Sistemas de base de dados paralelos consistem em múltiplos processadores e múltiplos discos ligados por uma rede de interconexão rápida.
- Uma máquina **paralela coarse-grain** consiste num número pequeno de procedores poderosos
- Uma máquina **massivamente paralela** ou **fine grain** utiliza milhares de processadores mais pequenos.
- Existem duas medidas de performance principais:
  - **throughput** --- o número de tarefas que podem ser completadas dados um intervalo de tempo
  - **Tempo de resposta** --- o tempo necessário para que uma tarefa submetida seja completada

Speed-Up e Scale-Up

- **Speedup**: um problema de tamanho fixo executado num sistema pequeno é dado a um sistema *N*-vezes maior.
  - Medido por:

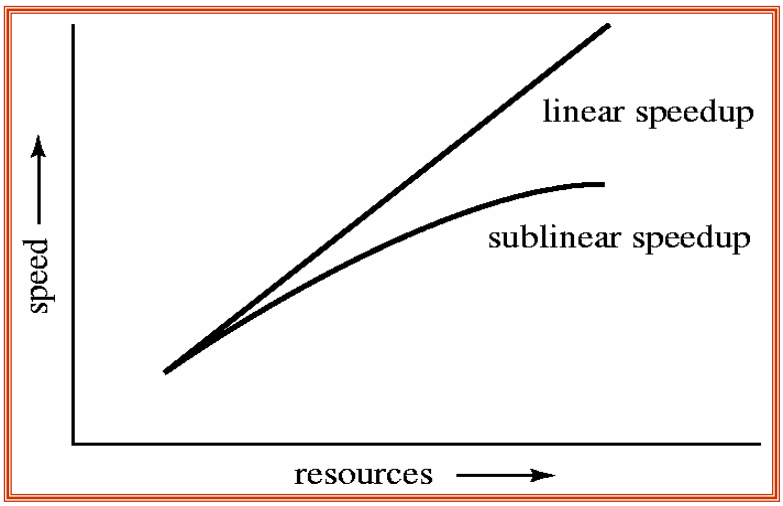
$$speedup = \frac{tempo\ no\ sistema\ pequeno}{tempo\ no\ sistema\ grande}$$

  - Speedup é **linear** se o quociente é igual a N.
- **Scaleup**: aumentar o tamanho do problema e do sistema
  - Um sistema *N*-vezes maior é usado para resolver uma tarefa *N*-vezes maior
  - Medido por:

$$scaleup = \frac{tempo\ do\ problema\ pequeno\ no\ sistema\ pequeno}{tempo\ do\ problema\ grande\ no\ sistema\ grande}$$

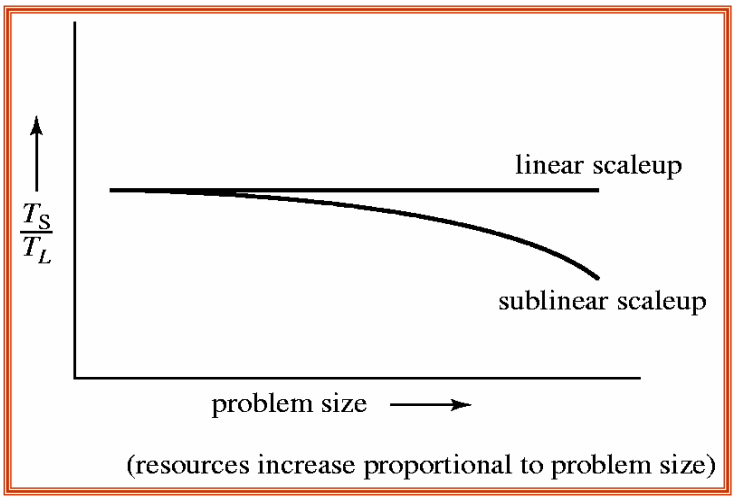
  - Scale up é **linear** se o quociente é igual a 1.

Speedup



Speedup

Scaleup



Scaleup

Scale de batch e de transações

- **Scaleup de Batch:**
  - Uma única grande tarefa; típico em muitas consultas a bases de dados e em simulação científica.
  - Usar um computador *N*-vezes maior num problema *N*-vezes maior.
- **Scaleup de transacções:**
  - Um grande número de pequenas consultas submetidas por utilizadores independentes a uma base de dados partilhada; típico em processamento de transacções e sistemas de tempo partilhado.
  - *N*-vezes mais utilizadores a submeter pedidos a uma base de dados *N*-vezes maior, sobre um computador *N*-vezes maior.
  - Apropriada para execução paralela.

Factores limitativos de Speedup e Scaleup

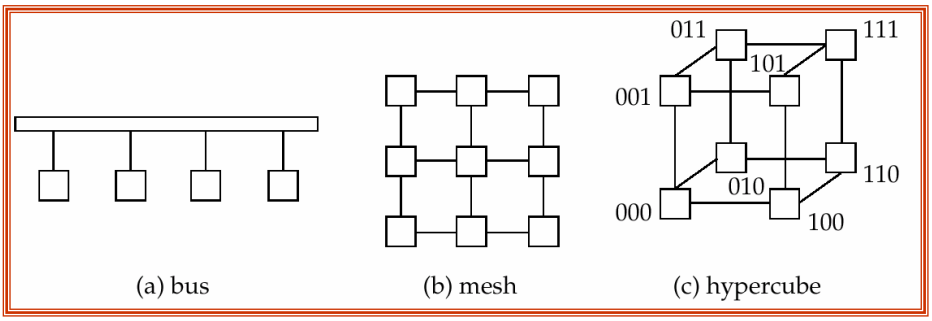
Speedup e scaleup são muitas vezes sublineares devido a:

- **Custo de inicialização**: O custo de inicializar múltiplos processos pode dominar o tempo de computação, se o grau de paralelismo for elevado.
- **Interferência**: Os processos a aceder a recursos partilhados (e.g.,bus do sistema, discos, ou locks) competem entre si, gastando tempo à espera de outros processos, em vez de executarem trabalho útil.
- **Skew**: Aumentando o grau de paralelismo aumenta-se a variação nos tempos de execução das tarefas paralelas. O tempo global de execução é determinado pela **mais demorada** das tarefas paralelas.

Arquitecturas de Redes de Interconexão

- **Bus**. As componentes do sistema enviam e recebem dados por um único bus de comunicação;
  - Dificuldades de escalabilidade com o aumento de paralelismo.
- **Mesh**. As componentes são configuradas como nós numa grelha, e cada componente é ligada a todas as componentes adjacentes
  - As ligações aumentam com o número crescente de componentes, e por isso tem melhor escalabilidade.
  - Mas pode requerer percorrer $2\sqrt{n}$ ligações para enviar uma mensagem para um nó.
- **Hipercubo**. As componentes são numeradas em binário; as componentes são ligadas umas às outras se as suas representações binárias diferem em exactamente um bit.
  - *n* componentes estão ligadas a outras *log(n)* componentes e podem chegar a qualquer outra percorrem no máximo *log(n)* ligações; reduz os tempos de comunicação.

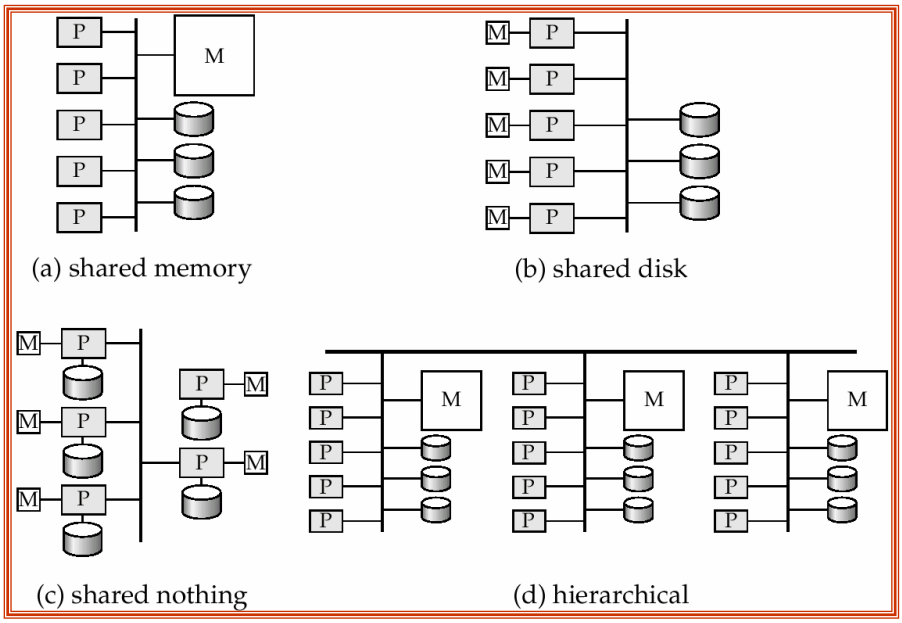
Arquitecturas de Interconexão



Arquitecturas de Bases de Dados Paralelas

- **Memória partilhada** – os processadores partilham a mesma memória
- **Disco partilhado** – os processadores partilham o mesmo disco
- **Shared nothing** – os processadores não partilham memória nem disco
- **Hierárquico** – híbrido das arquitecturas acima

Arquitecturas de Bases de Dados Paralelas



Memória Partilhada

- Os processadores e os discos tem acesso a uma memória comum, tipicamente através de um bus ou de uma rede de interconexão.
- Comunicação entre processadores extremamente eficiente – os dados na memória partilhada podem ser acedidos por qualquer processador sem terem que ser movidos por software.
- Desvantagem – a arquitectura não é escalável para além de 32 ou 64 processadores uma vez que o bus ou a rede de interconexão se tornam num bottleneck
- Bastante usada para níveis baixos de paralelismo (4 a 8).

Disco Partilhado

- Todos os processadores podem aceder directamente a todos os discos através de uma rede de interconexão, mas os processadores tem memórias privadas.
  - O bus de memória deixa de ser um bottleneck
  - A arquitectura fornece um grau de **tolerancia a falhas** — se um processador falha, os outros processadores podem assumir as sua tarefas uma vez que a base de dados reside em discos que são acessiveis de todos os processadores.
- Exemplos: IBM Sysplex e DEC clusters (agora Compaq) a correr Rdb (agora Oracle Rdb) foram utilizadores comerciais pioneiros
- Desvantagem: o bottleneck ocorre na rede de interconexão dos discos.
- Sistemas de disco-partilhado são escaláveis num número maior de processadores, mas a comunicação entre processadores é mais lenta.

Shared Nothing

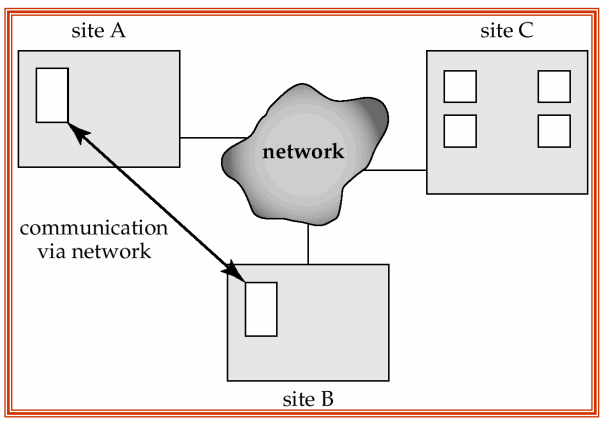
- Cada nó consiste num processador, memória e um ou mais discos. Os processadores num nó comunicam com outro processador usando a rede de interconexão. Cada nó funciona como um servidor para os dados residentes nos seus próprios discos.
- Exemplos: Teradata, Tandem, Oracle-n CUBE
- Os dados acedidos dos discos locais (e da memória local) não passam pela rede de interconexão, logo minimizando a interferencia da partilha de recursos.
- Multi-processadores Shared-nothing podem ser escalados em milhares de processadores sem interferência.
- Desvantagem principal: custo de comunicação e acesso a discos não-locais; envio de dados envolve interacção de software nos dois extremos.

Hierárquico

- Combina as características das arquitecturas de memória partilhada, disco partilhado e shared-nothing.
- O nível mais alto é uma arquitectura shared-nothing – os nós estão ligados por uma rede de interconexão, e não partilham entre si disco e memória.
- Cada nó do sistema pode ser um sistema de memória partilhada com alguns processadores.
- Alternativamente, cado nó pode ser um sistema de disco partilhado, e cada um dos sistemas partilhando um conjunto de discos pode ser um sistema de memória partilhada.
- A complexidade de programação de tais sistema é reduzida usando arquitecturas de **memória virtual distribuída**
  - Também chamadas **non-uniform memory architecture (NUMA)**

Sistemas Distribuídos

- Os dados estão espalhados por várias máquinas (também referidos por **sites** ou **nós**.)
- A rede interliga as máquinas
- Os dados são partilhados pelos utilizadores nas várias máquinas



Bases de Dados Distribuídas

- Bases de dados distribuídas homogéneas
  - O mesmo software/esquema é usados em todos os sites, os dados podem estar repartidos entre os sites
  - Objectivo: fornecer uma visão de uma única base de dados, escondendo detalhes de distribuição
- Bases de dados distribuídas heterogéneas
  - Software/esquema diferentes em diferentes sites
  - Objectivo: integrar bases de dados existentes para fornecer funcionalidades úteis
- Diferenciação entre transacção *local e global*
  - Uma transacção local acede a dados num único site onde a transacção foi iniciada.
  - Uma transacção global acede a dados que ou estão num site diferente daquele em que a transacção foi iniciada ou acede a dados em vários sites diferentes.

Trade-offs em Sistemas Distribuídos

- Partilha de dados – os utilizadores num site podem aceder a dados residentes em outros sites.
- Autonomia – cada site pode manter um nível de controlo sobre os dados guardados localmente.
- Maior disponibilidade do sistema através de redundância — os dados podem estar replicados em sites remotos, e o sistema pode funcionar mesmo que um site falhe.
- Desvantagem: a complexidade aumenta pois é necessário assegurar uma coordenação correcta entre os sites.
  - Custo de desenvolvimento de software.
  - Maior possiblidade de bugs.
  - Aumento do overhead de processamento.

Aspectos de Implementação para Bases de Dados Distribuídas

- Atomicidade é necessária mesmo para transacções que actualizam dados em mais do que um site
  - Uma transacção não pode ser committed num site e abortada noutro
- Para assegurar atomicidade é usado o two-phase commit protocol (2PC)
  - Idea básica: cada site executa a transacção até mesmo antes do commit, e deixa a decisão final para um coordenador
  - Cada site tem que seguir a decisão do coordenador: mesmo que haja uma falha enquanto se espera pela decisão do coordenador
    - Para isso os updates das transacções são registados em suportes estavéis e a transacção é marcada como “waiting”
- 2PC nem sempre é apropriado: outros modelos de transacções baseados em mensagens persistentes também são usados
- É necessário o controlo de concorrência distribuída (e de detecção de deadlocks)
- É necessária a replicação de dados para melhorar a disponibilidade de dados