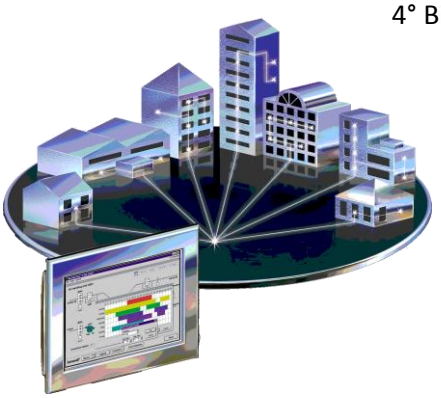
Disciplina: **Engenharia de Software**

4° Bimestre – Aula 1: **ENGENHARIA DE SOFTWARE DISTRIBUÍDO**



O que é um sistema distribuído?

_____
_____
_____
_____

Segundo Tanenbaum e Steen (2007) um sistema distribuído é uma coleção de computadores independentes que aparece para o usuário como um único sistema coerente.

QUAIS AS VANTAGENS DA UTILIZAÇÃO DE UMA ABORDAGEM DISTRIBUÍDA DE DESENVOLVIMENTO DE SISTEMAS?

Verdadeiro ou Falso?

( ) A engenharia de sistema distribuídos tem muito em comum com a engenharia de qualquer outro software. No entanto, existem questões específicas que precisam ser consideradas ao projetar esse tipo de sistema. Estas questões surgem pois os componentes do sistema podem ser executados em computadores gerenciados de forma independente e porque eles se comunicam por meio de uma rede.

( ) Os sistemas distribuídos são mais **complexos** que os sistemas centralizados, o que os torna mais difíceis para projetar, implementar e testar.

Por que os sistemas distribuídos são mais complexos?

... Em vez de o desempenho do sistema depender da velocidade de execução de um processador, ele depende da largura da banda de rede, da carga de rede e da velocidade de todos os computadores que fazem parte do sistema.

... Sistemas distribuídos tem respostas imprevisíveis.

... Mover recursos de uma parte do sistema para outra pode afetar significativamente o desempenho do sistema.

Questões importantes sobre sistemas distribuídos

Transparência — Em que medida o sistema distribuído deve aparecer para o usuário como um único sistema? Quando é útil aos usuários entender que o sistema é distribuído?

Escalabilidade — Como o sistema pode ser construído para que seja escalável? Ou seja, como todo o sistema poderia ser projetado para que sua capacidade possa ser aumentada em resposta às crescentes exigências feitas ao sistema?

Proteção — Como podem ser definidas e implementadas as políticas de proteção que se aplicam a um conjunto de sistemas gerenciados independentemente?

Qualidade de serviço — Como a qualidade do serviço que é entregue aos usuários do sistema deve ser especificada e como o sistema deve ser implementado para oferecer uma qualidade aceitável e serviço para todos os usuários?

Gerenciamento de falhas — Como as falhas do sistema podem ser detectadas, contidas (para que elas tenham efeitos mínimos em outros componentes do sistema) e reparadas?

Modelos de interação

Existem dois tipos fundamentais de interação que podem ocorrer entre os computadores em um sistema de computação distribuído: interação procedural e interação baseada em mensagens.

Interação procedural

Envolve um computador que chama um serviço conhecido oferecido por algum outro computador e (normalmente) esperando que esse serviço seja fornecido. A comunicação procedural é implementada usando-se chamadas de procedimento remoto (RPC's, do inglês *remote procedure calls*). Nas RPC's um componente chama outro componente, como se fosse um método ou procedimento local. O middleware do sistema intercepta essa chamada e transmite-a para um componente remoto. Este realiza o processamento necessário e, via middleware, retorna o resultado para o componente chamado.

Interação baseada em mensagens

Envolve o computador 'que envia' que define as informações sobre o que é requerido em uma mensagem, que são, então, enviadas para outro computador. Geralmente, as mensagens transmitem mais informações em uma única interação do que uma chamada de procedimento para outra máquina.

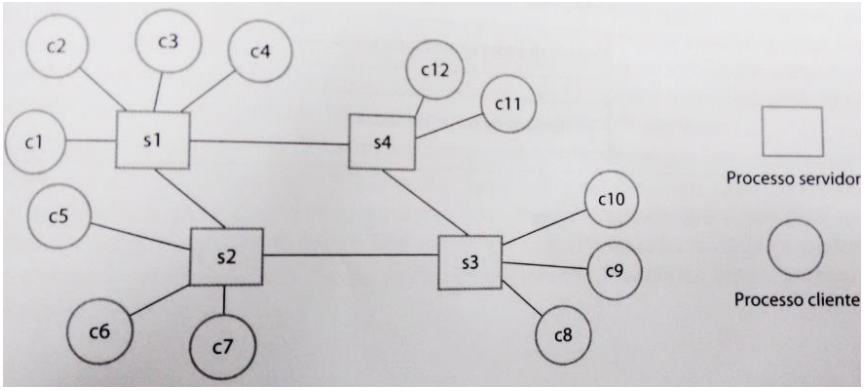
Middleware

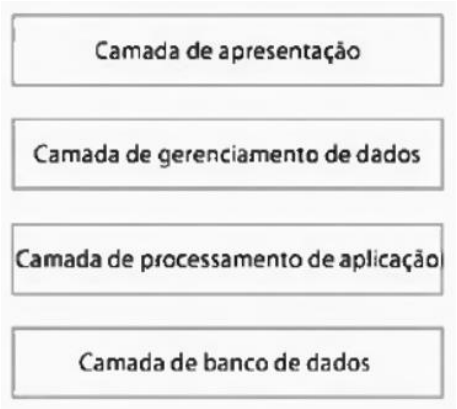
Os componentes em um sistema distribuído podem ser implementados em diferentes linguagens de programação e podem ser executados em diferentes tipos de processador. Os modelos de dados, representação de informações, protocolos para comunicação podem ser todos diferentes. Um sistema distribuído, portanto, requer um **software que possa gerenciar essas diversas partes e assegurar que elas podem se comunicar e trocar dados**.

Computação cliente-servidor

Sistemas distribuídos que são acessados pela internet normalmente são organizados como sistemas cliente-servidor. Em um sistema ciente-servidor, o usuário interage com um programa em execução em seu computador local. Este interage com outro programa em execução em um computador remoto. O computador remoto fornece serviços. Esse modelo cliente-servidor é um modelo de arquitetura geral de uma aplicação. Não está restrito a aplicações distribuídas em várias máquinas.

Em uma arquitetura cliente-servidor, uma aplicação é modelada como um conjunto de serviços que são fornecidos por servidores. Os clientes podem acessar esses serviços e apresentar os resultados para os usuários finais. Os clientes precisam estar cientes dos servidores que estão disponíveis, mas não devem saber da existência de outros clientes. Clientes e servidores são processos separados.





Padrões de arquiteturas para sistemas distribuídos

Os projetistas de sistemas distribuídos precisam organizar seus projetos de sistema para encontrar um equilíbrio entre desempenho, confiança, proteção e capacidade de gerenciamento do sistema. Não existe um modelo universal que seja apropriado para todas as circunstâncias, de modo que surgiram vários estilos de arquitetura. Ao projetar uma arquitetura distribuída, você deve escolher um estilo de arquitetura que ofereça suporte aos requisitos não funcionais críticos de seu sistema.

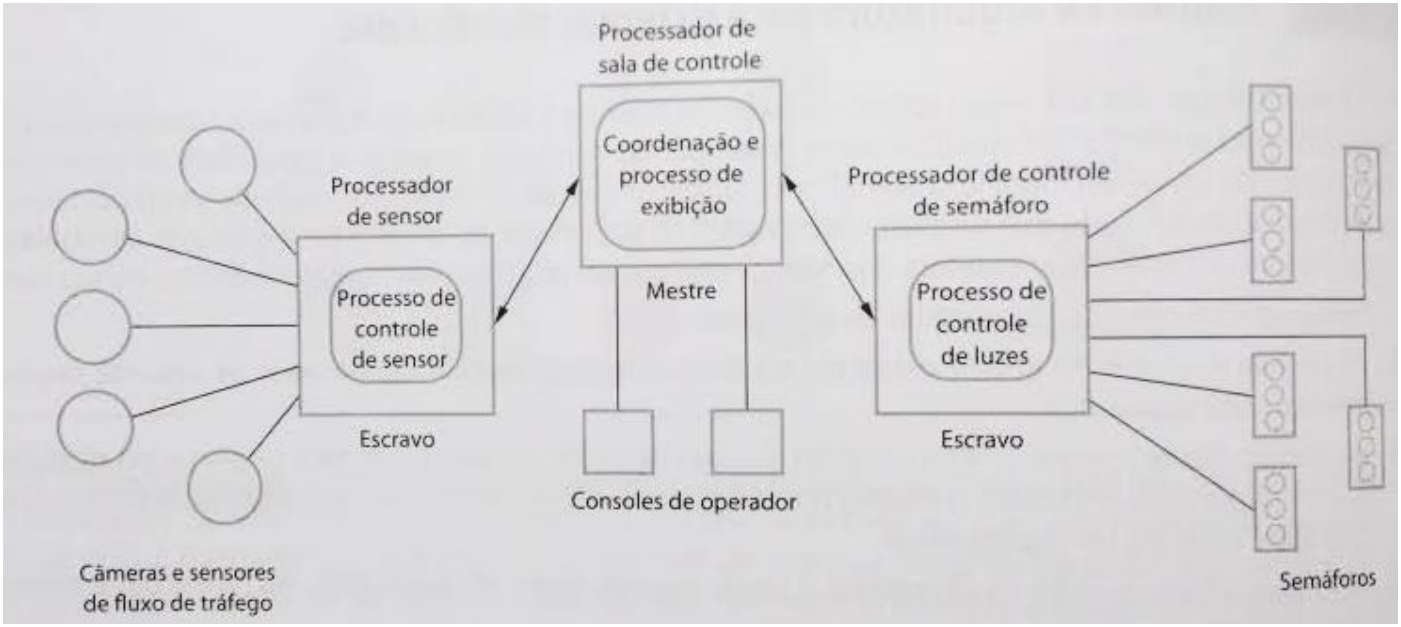
1. Arquitetura mestre-escravo
2. Arquitetura cliente-servidor de duas camadas
3. Arquitetura cliente-servidor multicamadas
4. Arquitetura distribuída de componentes
5. Arquitetura ponto-a-ponto

Arquitetura mestre-escravos

São comumente usados em sistemas de tempo real em que pode haver processadores separados associados à aquisição de dados do ambiente do sistema, processamento de dados, de gerenciamento de atuadores.

Processo mestre: responsável pelo processamento, coordenação e comunicação.

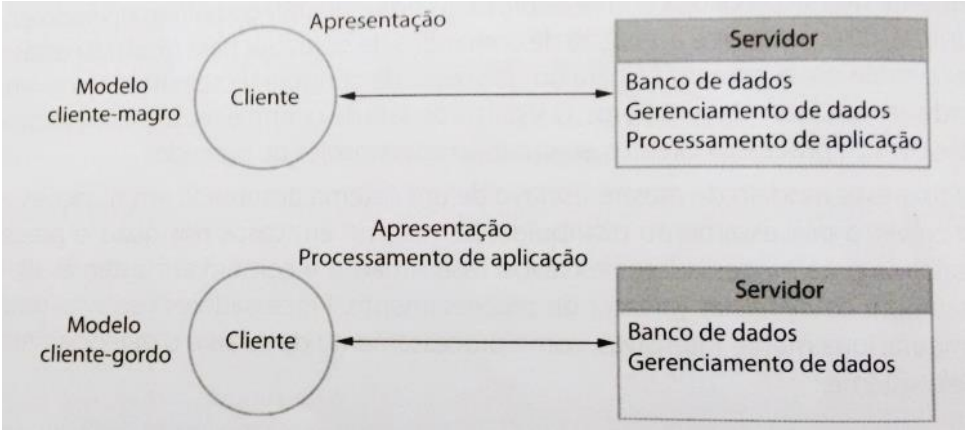
Processos escravos: dedicados a ações específicas. Além disso, podem ser usados para operações computacionalmente intensivas, como processamento de sinais e o gerenciamento de equipamentos controlados pelo sistema.



Arquitetura cliente-servidor de duas camadas

Na página 3, foi abordado de forma geral os sistemas cliente-servidor, em que parte do sistema de aplicação é executada no computador do usuário (cliente) e parte é executada em um computador remoto (servidor).

Uma arquitetura cliente-servidor de duas camadas é a forma mais simples da arquitetura cliente-servidor. O sistema é implementado como um **único servidor lógico** e, também, um **número indefinido de clientes** que usam esse servidor. Há duas formas desse modo de arquitetura (conforme mostra a figura abaixo):



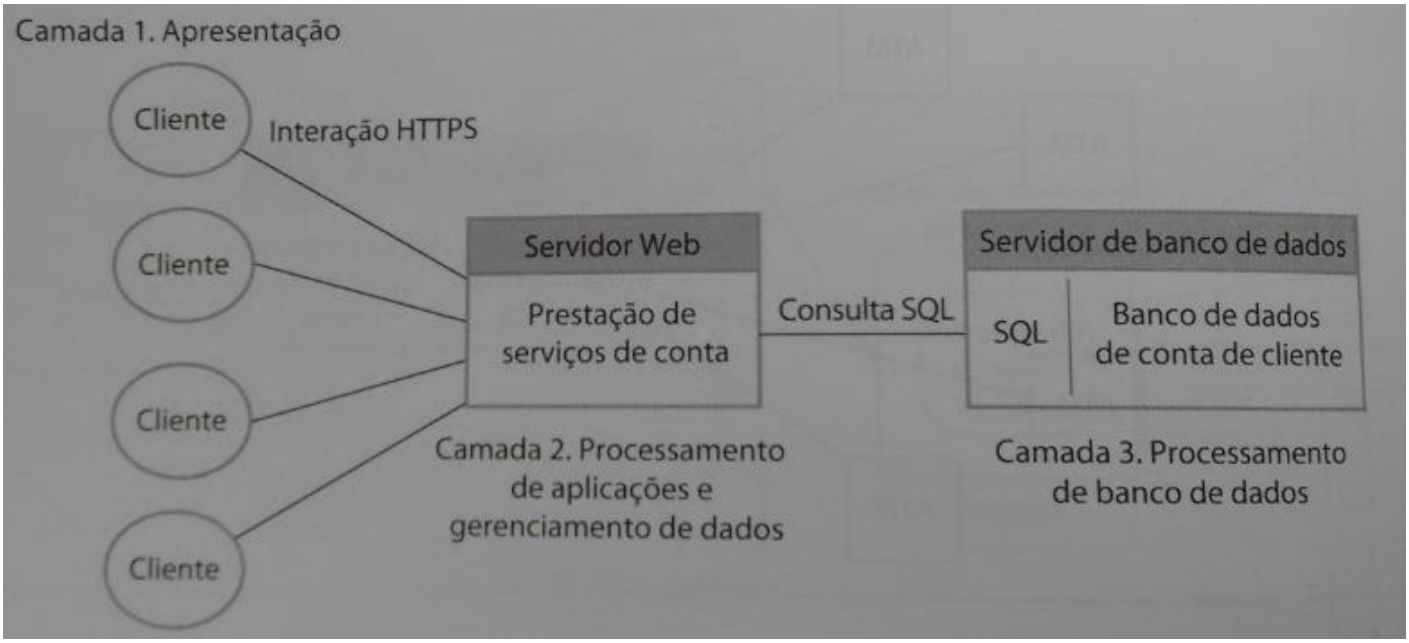
- **<u>Modelo cliente-magro</u>**: a camada de apresentação é implementada no cliente e todas as outras camadas (gerenciamento de dados, processamento de aplicação e banco de dados) são implementadas em um servidor.
    - <u>Vantagem</u>
        - A vantagem no modelo cliente-magro é a simplicidade em gerenciar clientes. Isso é um grande problema se houver um grande número de clientes, pois pode ser difícil e caro instalar novo software em todos eles. Se um browser de Web for usado como o cliente, não é necessário instalar qualquer software.
    - <u>Desvantagem</u>
        - A desvantagem da abordagem cliente-magro é poder colocar uma carga pesada de processamento no servidor e na rede. O servidor é responsável por toda a computação e isso pode levar à geração de tráfego significativo de rede entre o cliente e o servidor.

- **<u>Modelo cliente-gordo</u>**: parte ou todo o processamento de aplicação é executado no cliente e as funções de banco de dados e gerenciamento são implementadas no servidor.
    - <u>Vantagem</u>
        - Distribui o peso computacional. O servidor é essencialmente um servidor de transação que gerencia todas as transações do banco de dados. O gerenciamento de dados é simples e não é necessário haver interação entre o cliente e o sistema de processamento de aplicação.

- Desvantagem
  - O problema do modelo-gordo é o gerenciamento de sistema que é mais complexo. A funcionalidade de aplicação é espalhada por muitos computadores. Quando o software de aplicação precisa ser alterado, isso envolve a reinstalação em cada computador cliente, o que pode ter um grande custo se houve centenas de clientes no sistema.

Arquitetura cliente-servidor multicamadas

O problema fundamental com uma abordagem cliente-servidor de duas camadas é que as camadas lógicas de sistema – apresentação, processamento de aplicação, gerenciamento de dados e banco de dados - devem ser mapeadas para dois sistemas de computador: o cliente e o servidor. Pode haver problemas com escalabilidade e desempenho se o modelo cliente-magro for escolhido, ou problemas de gerenciamento de sistema se o modelo cliente-gordo for suado. Para evitar esses problemas, uma arquitetura de 'cliente-servidor multicamadas' pode ser usada. Nessa arquitetura, as diferentes camadas do sistema, apresentação, gerenciamento de dados, processamento de aplicação e aplicação de banco de dados, são processos separados que podem ser executados em diferentes processadores.



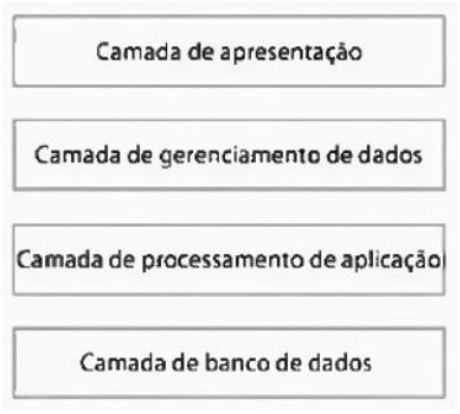
Comentários

A seguir, a tabela descreve situações em que as arquiteturas cliente-servidor podem ser adequadas:

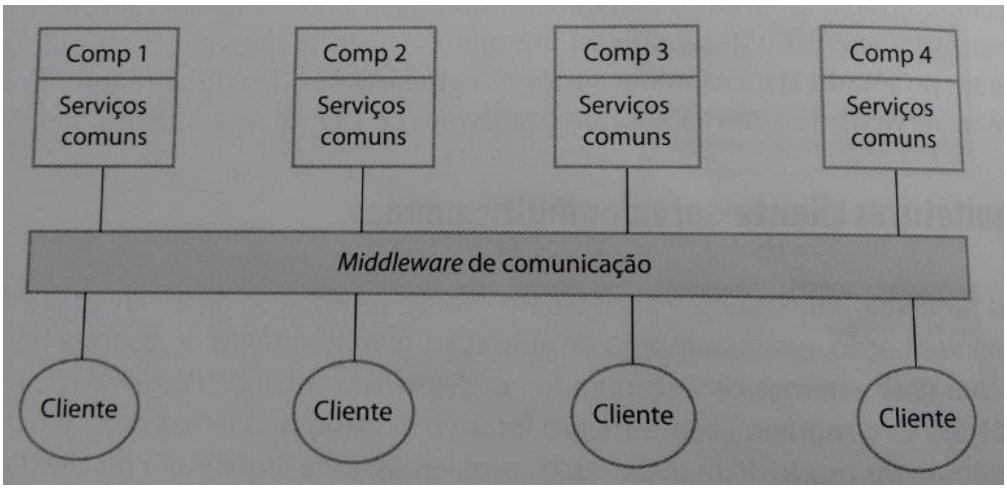
Arquitetura	**Aplicações**
Arquitetura cliente-servidor de duas camadas com clientes-magros	
Arquitetura cliente-servidor de duas camadas com clientes-gordos	
Arquitetura cliente-servidor multicamadas	

Arquitetura de componentes distribuídos

A figura abaixo mostra o processamento organizado em camadas, e cada camada de um sistema pode ser implementada como um servidor lógico separado. Esse modelo funciona bem para muitos tipos de aplicação. No entanto, ele limita a flexibilidade de projetistas de sistema, pois é necessário decidir quais serviços devem ser incluídos em cada camada. Na prática, nem sempre é claro se um serviço é um serviço de gerenciamento de dados, um serviço de aplicação ou um serviço de banco de dados. Os projetistas também devem planejar para escalabilidade e, assim, fornecer alguns meios para que os servidores sejam replicados à medida que mais clientes são adicionados ao sistema.



Uma abordagem mais geral de projeto de sistemas distribuídos é projetar o sistema como um conjunto de serviços, sem tentar alocar esses serviços nas camadas do sistema. Cada serviço, ou grupo de serviços relacionados, é implementado usando um componente separado. Em uma arquitetura de componentes distribuídos (figura abaixo), o sistema é organizado como um conjunto de componentes ou objetos interativos. Esses componentes fornecem uma interface para um conjunto de serviços que eles fornecem. Outros componentes chamam esses serviços através do middleware.



Os sistemas de componentes distribuídos são dependentes do middleware, o qual gerencia as interações de componentes, reconcilia as diferenças entre os tipos de parâmetros passados entre componentes e fornece um conjunto de serviços comuns que os componentes de aplicação podem usar.

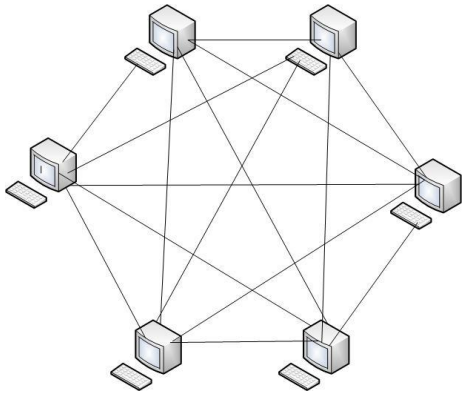
Benefício e desvantagem

Arquiteturas ponto-a-ponto (p2p, do inglês *peer-to-peer*)

O modelo de computação cliente-servidor discutido nas seções anteriores faz uma distinção clara entre servidores, que são provedores de serviços, e clientes, que são receptores de serviços. Geralmente, esse modelo leva a uma distribuição desigual de carga no sistema, em que os servidores trabalham mais que clientes. Isso pode levar as

organizações a investirem muito na capacidade de servidor, enquanto existe capacidade de processamento não usada em centenas ou milhares de PCs usados como clientes.

Os sistemas ponto-a-ponto são sistemas **descentralizados** (conforme imagem abaixo) em que os processamentos podem ser realizados por qualquer nó na rede. Em princípio, pelo menos, não existem distinções entre clientes e servidores. Em aplicações ponto-a-ponto, todo o sistema é projetado para aproveitar o poder computacional e o armazenamento disponível por meio de uma rede potencialmente enorme de computadores. As normas e protocolos que permitem a comunicação entre os nós são embutidas na própria aplicação, e cada nó deve executar uma cópia da aplicação.



Essa arquitetura apresenta alguma desvantagem?

As tecnologias ponto-a-ponto têm sido usadas, principalmente, para sistemas pessoais, mas também estão sendo usadas pelas empresas.

Quando é adequado usar um modelo de arquitetura ponto-a-ponto?

Exercícios

1) O que você entende por escalabilidade?
2) Explique por que sistemas de software distribuídos são mais complexos do que os sistemas de software centralizados, em que toda a funcionalidade de sistema é implementada em um único computador.
3) Qual a diferença fundamental entre a abordagem cliente-gordo e a abordagem cliente-magro para ar arquiteturas de sistemas cliente-servidor?
4) Pesquise: O que uma arquitetura p2p semicentralizada? Como ela funciona?