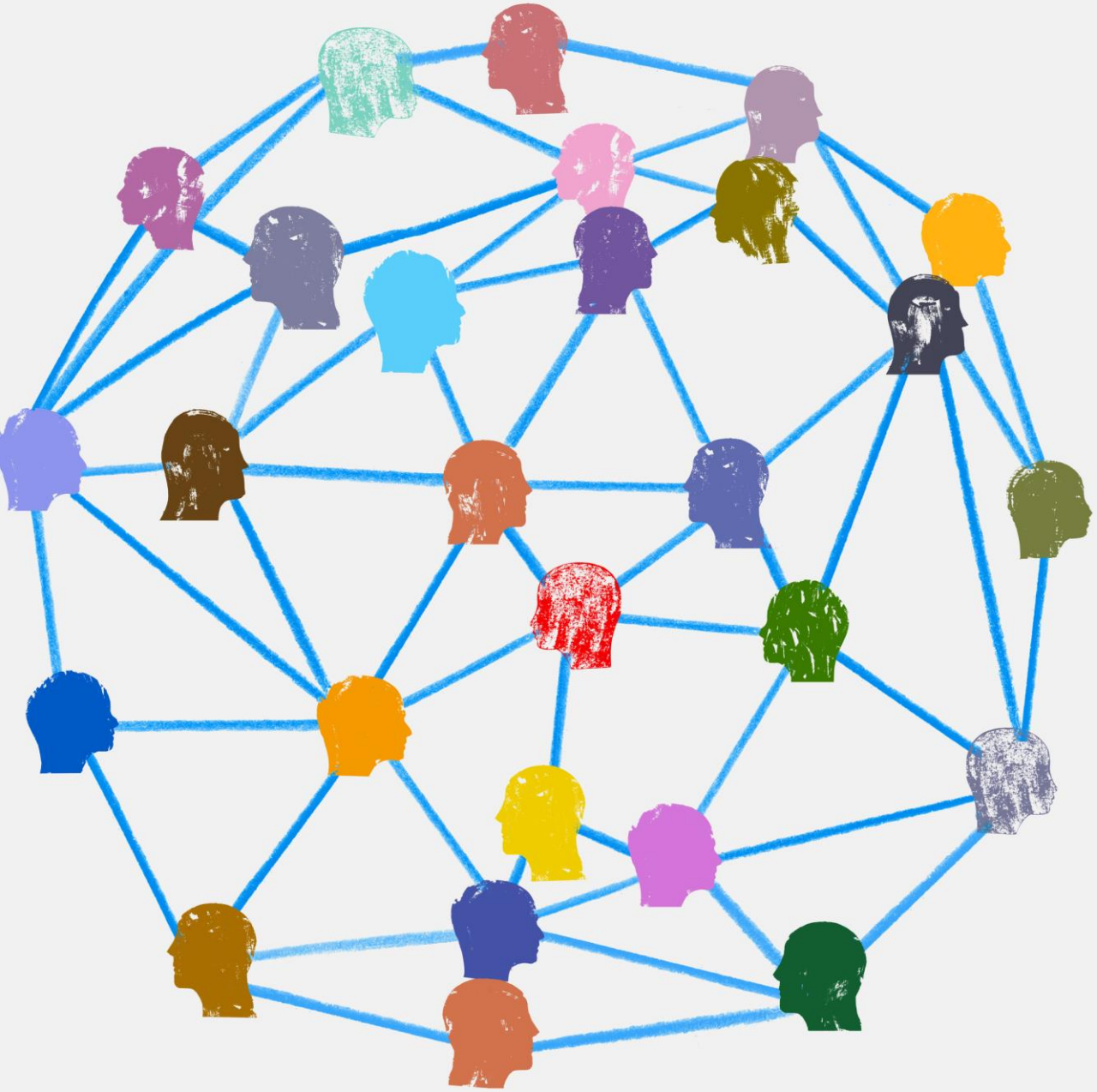
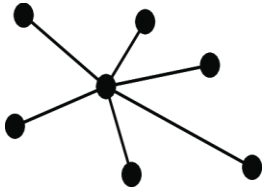


Sistemas Distribuídos

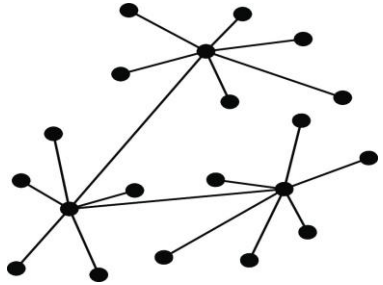
INTRODUÇÃO

Distribuído versus Descentralizado

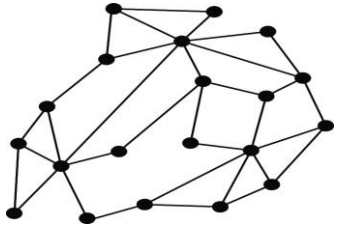
O que muitas pessoas afirmam



Centralizado



Descentralizado



Distribuído

Quando um sistema descentralizado se torna distribuído?

- Adicionar 1 ligação entre dois nós em um sistema descentralizado?
- Adicionar 2 ligações entre outros dois nós?
- Em geral: adicionar k > 0 ligações....?

Abordagem alternativa

Duas visões sobre a realização de sistemas distribuídos

- Visão integradora: conectar sistemas de computadores em rede existentes em um sistema maior.
- Visão expansiva: um sistema de computadores em rede existente é estendido com computadores adicionais.

Duas definições

- Um sistema descentralizado é um sistema de computadores em rede no qual processos e recursos são necessariamente distribuídos entre múltiplos computadores.
- Um sistema distribuído é um sistema de computadores em rede no qual processos e recursos são suficientemente distribuídos entre múltiplos computadores.

Algumas concepções errôneas comuns

Soluções centralizadas não escalonam

Faça a distinção entre centralizado logicamente e fisicamente. A raiz do Sistema de Nomes de Domínio:

- centralizada logicamente fisicamente (massivamente) distribuída descentralizada entre várias organizações

Soluções centralizadas têm um único ponto de falha

Geralmente não é verdade (por exemplo, a raiz do DNS). Um único ponto de falha é frequentemente:

- mais fácil de gerenciar
- mais fácil de tornar mais robusto

Importante

Existem muitas concepções errôneas mal fundamentadas sobre escalabilidade, tolerância a falhas, segurança, etc. Precisamos desenvolver **habilidades** para que sistemas distribuídos possam ser prontamente compreendidos para julgar essas concepções errôneas.

Perspectivas sobre sistemas distribuídos

Sistemas distribuídos são complexos: adota as perspectivas

- Arquitetura: organizações comuns
- Processo: que tipo de processos e suas relações
- Comunicação: facilidades para troca de dados
- Coordenação: algoritmos independentes de aplicação
- Nomeação: como você identifica recursos?
- Consistência e replicação: desempenho exige que os dados sejam iguais
- Tolerância a falhas: manter o funcionamento na presença de falhas parciais
- Segurança: garantir acesso autorizado aos recursos

O que queremos alcançar?

Objetivos gerais de design

Apoiar o compartilhamento de recursos

Transparência de distribuição

Abertura

Escalabilidade

Compartilhamento de recursos

Exemplos :

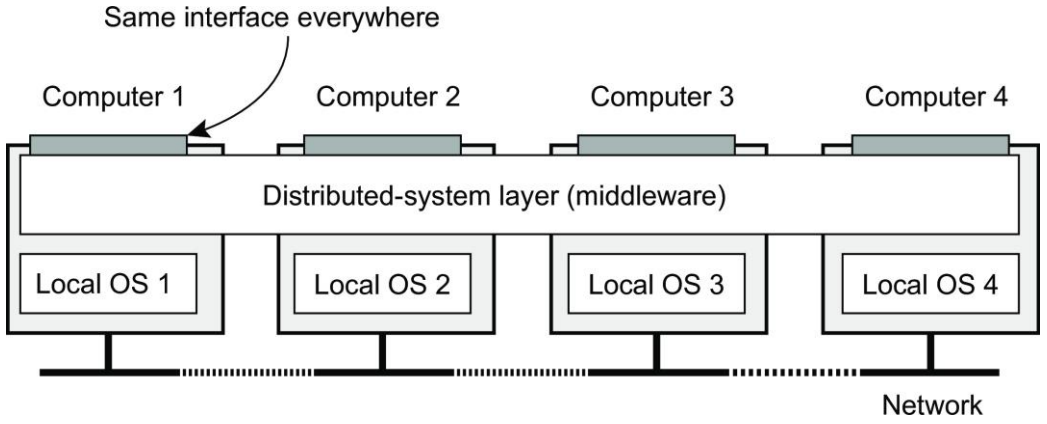
- Armazenamento e arquivos compartilhados baseados em nuvem
- Streaming de multimídia assistido por pares (peers)
- Serviços de e-mail compartilhados (pense em sistemas de e-mail terceirizados)
- Hospedagem Web compartilhada (pense em redes de distribuição de conteúdo)

Observação:

“A rede é o computador” (citação de John Gage, então na Sun Microsystems)

Transparência de distribuição

O que é transparência? O fenômeno pelo qual um sistema distribuído tenta esconder o fato de que seus processos e recursos estão fisicamente distribuídos por vários computadores, possivelmente separados por grandes distâncias.



Observação A transparência de distribuição é tratada por meio de várias técnicas diferentes em uma camada entre aplicativos e sistemas operacionais: uma camada de middleware.

Transparência de distribuição

Transparência	Descrição
Acesso	Ocultar diferenças na representação dos dados e na forma como um objeto é acessado
Localização	Ocultar onde um objeto está localizado
Realocação	Ocultar que um objeto pode ser movido para outra localização enquanto está em uso
Migração	Ocultar que um objeto pode se mover para outra localização
Replicação	Ocultar que um objeto é replicado
Concorrência	Ocultar que um objeto pode ser compartilhado por vários usuários independentes
Falhas	Ocultar a falha e a recuperação de um objeto

Grau de transparência

A busca pela transparência de distribuição total pode ser excessiva:

- Existem latências de comunicação que não podem ser ocultadas
- Ocultar completamente falhas de redes e nós é (teórica e praticamente) impossível
- Você não pode distinguir um computador lento de um que está com falha
- Você nunca pode ter certeza de que um servidor realmente realizou uma operação antes de uma falha
- A transparência total custará desempenho, expondo a distribuição do sistema
- Manter réplicas exatamente atualizadas com o mestre leva tempo
- Descartar imediatamente operações de gravação no disco para tolerância a falhas

Grau de transparência

Expor a distribuição pode ser benéfico Utilizando serviços baseados em localização (encontrando amigos próximos) Quando lidando com usuários em diferentes fusos horários Quando facilita para o usuário entender o que está acontecendo (por exemplo, se um servidor não responde por um longo período, relatá-lo como falhando).

Conclusão A transparência de distribuição é um objetivo interessante, mas alcançá-la é uma história diferente, e muitas vezes não deve ser o alvo.

Abertura dos sistemas distribuídos- Sistema distribuído aberto

Um sistema que oferece componentes que podem ser facilmente utilizados ou integrados a outros sistemas. Um sistema distribuído aberto frequentemente consistirá de componentes que se originam de outros lugares.

Sobre o que estamos falando?

Ser capaz de interagir com serviços de outros sistemas abertos, independentemente do ambiente subjacente:

- Os sistemas devem se conformar a interfaces bem definidas
- Os sistemas devem interoperar facilmente
- Os sistemas devem suportar a portabilidade de aplicações
- Os sistemas devem ser facilmente extensíveis

Políticas versus mecanismos

Implementando abertura: políticas

- Qual nível de consistência exigimos para dados em cache no cliente?
- Quais operações permitimos que o código baixado execute?
- Quais requisitos de QoS ajustamos em face de variações na largura de banda?
- Qual nível de sigilo exigimos para a comunicação?

Implementando abertura: mecanismos

- Permitir a configuração (dinâmica) de políticas de cache
- Suportar diferentes níveis de confiança para código móvel
- Fornecer parâmetros de QoS ajustáveis por fluxo de dados
- Oferecer diferentes algoritmos de criptografia

Sobre separação estrita

Observação:

Quanto mais rigorosa a separação entre política e mecanismo, mais precisamos garantir mecanismos adequados, o que pode levar a muitos parâmetros de configuração e gestão complexa.

Encontrando um equilíbrio:

Codificar políticas diretamente frequentemente simplifica a gestão e reduz a complexidade, à custa de menor flexibilidade. Não há uma solução óbvia.

Confiabilidade

Conceitos básicos

- Um componente fornece serviços aos clientes.

Para fornecer serviços, o componente pode precisar dos serviços de outros componentes ⇒ um componente pode depender de outro componente.

Especificamente

Um componente C depende de C∗ se a correção do comportamento de C depender da correção do comportamento de C∗. (Componentes são processos ou canais.)

Confiabilidade

Requisitos relacionados à confiabilidade:

Requisito	Descrição
Disponibilidade	Prontidão para uso
Confiabilidade	Continuidade na entrega de serviços
Segurança	Probabilidade muito baixa de catástrofes
Manutenibilidade	Quão fácil é reparar um sistema falhado

Confiabilidade versus Disponibilidade

Confiabilidade R(t) do componente C

Probabilidade condicional de que C tenha funcionado corretamente durante [0, t) dado que C estava funcionando corretamente no momento T = 0.

Métricas tradicionais

Tempo Médio Para Falha (*Mean Time To Failure* - MTTF): O tempo médio até que um componente falhe.

Tempo Médio Para Reparo (*Mean Time To Repair* - MTTR): O tempo médio necessário para reparar um componente.

Tempo Médio Entre Falhas (*Mean Time Between Failures*- MTBF): Simplesmente MTTF + MTTR.

Terminologia

Falha, erro, defeito

Termo	Descrição	Exemplo
Falha	Um componente não está atendendo às suas especificações	Programa travado
Erro	Parte de um componente que pode levar a uma falha	Bug de programação
Defeito	Causa de um erro	Programador descuidado

Terminologia – Tratamento de defeitos

Termo	Descrição	Exemplo
Prevenção de defeitos	Prevenir a ocorrência de um defeito	Não contratar programadores descuidados
Tolerância a defeitos	Construir um componente e fazê-lo mascarar a ocorrência de um defeito	Construir cada componente por dois programadores independentes
Remoção de defeitos	Reduzir a presença, número ou gravidade de um defeito	Eliminar programadores descuidados
Previsão de defeitos	Estimar a presença atual, a incidência futura e as consequências dos defeitos	Estimar como um recrutador está se saindo na contratação de programadores descuidados

Sobre segurança

Observação

Um sistema distribuído que não é seguro, não é confiável.

- O que precisamos
  - Confidencialidade: a informação é divulgada apenas para partes autorizadas.
  - Integridade: garantir que alterações aos ativos de um sistema possam ser feitas apenas de maneira autorizada.
  - Autorização, Autenticação, Confiança Autenticação: verificar a correção de uma identidade alegada.
  - Autorização: a entidade identificada tem direitos de acesso apropriados?
  - Confiança: uma entidade pode ter certeza de que outra realizará ações específicas de acordo com uma expectativa definida.

Escalabilidade em sistemas distribuídos

Observação

Muitos desenvolvedores de sistemas distribuídos modernos usam facilmente o adjetivo “escalável” sem deixar claro por que seu sistema realmente escala.

Pelo menos três componentes

- Número de usuários ou processos (escalabilidade em tamanho)
- Distância máxima entre nós (escalabilidade geográfica)
- Número de domínios administrativos (escalabilidade administrativa)

Observação

A maioria dos sistemas considera apenas, até certo ponto, a escalabilidade em tamanho.

Frequentemente, a solução é: múltiplos servidores poderosos operando de forma independente e paralela. Hoje, o desafio ainda reside na escalabilidade geográfica e administrativa.

Escalabilidade em tamanho

- Causas raízes para problemas de escalabilidade com soluções centralizadas
- A capacidade computacional, limitada pelos CPUs
- A capacidade de armazenamento, incluindo a taxa de transferência entre CPUs e discos
- A rede entre o usuário e o serviço centralizado

Problemas com escalabilidade geográfica

Não é possível simplesmente ir de LAN para WAN: muitos sistemas distribuídos assumem interações síncronas cliente-servidor: o cliente envia uma solicitação e espera por uma resposta. A latência pode facilmente proibir esse esquema.

Links WAN são frequentemente inerentemente não confiáveis: simplesmente mover vídeo em streaming de LAN para WAN tende a falhar.

Falta de comunicação multiponto, de modo que uma simples transmissão de busca não pode ser implementada. A solução é desenvolver serviços de nomenclatura e diretórios separados (que têm seus próprios problemas de escalabilidade).

Problemas com escalabilidade administrativa

Políticas conflitantes sobre uso (e, portanto, pagamento), gerenciamento e segurança.

Exemplos

- Computational grids: compartilhar recursos caros entre diferentes domínios.
- Equipamentos compartilhados: como controlar, gerenciar e usar um telescópio de rádio compartilhado construído como uma grande rede de sensores compartilhados?

Exceção: várias redes peer-to-peer

- Sistemas de compartilhamento de arquivos (baseados, por exemplo, no BitTorrent)
- Telefonia peer-to-peer (versões iniciais do Skype)
- Streaming de áudio assistido por pares (Spotify)

Nota: os usuários finais colaboram e não as entidades administrativas.

Técnicas para escalonamento

Ocultar latências de comunicação

Utilizar comunicação assíncrona

Ter um manipulador separado para respostas recebidas

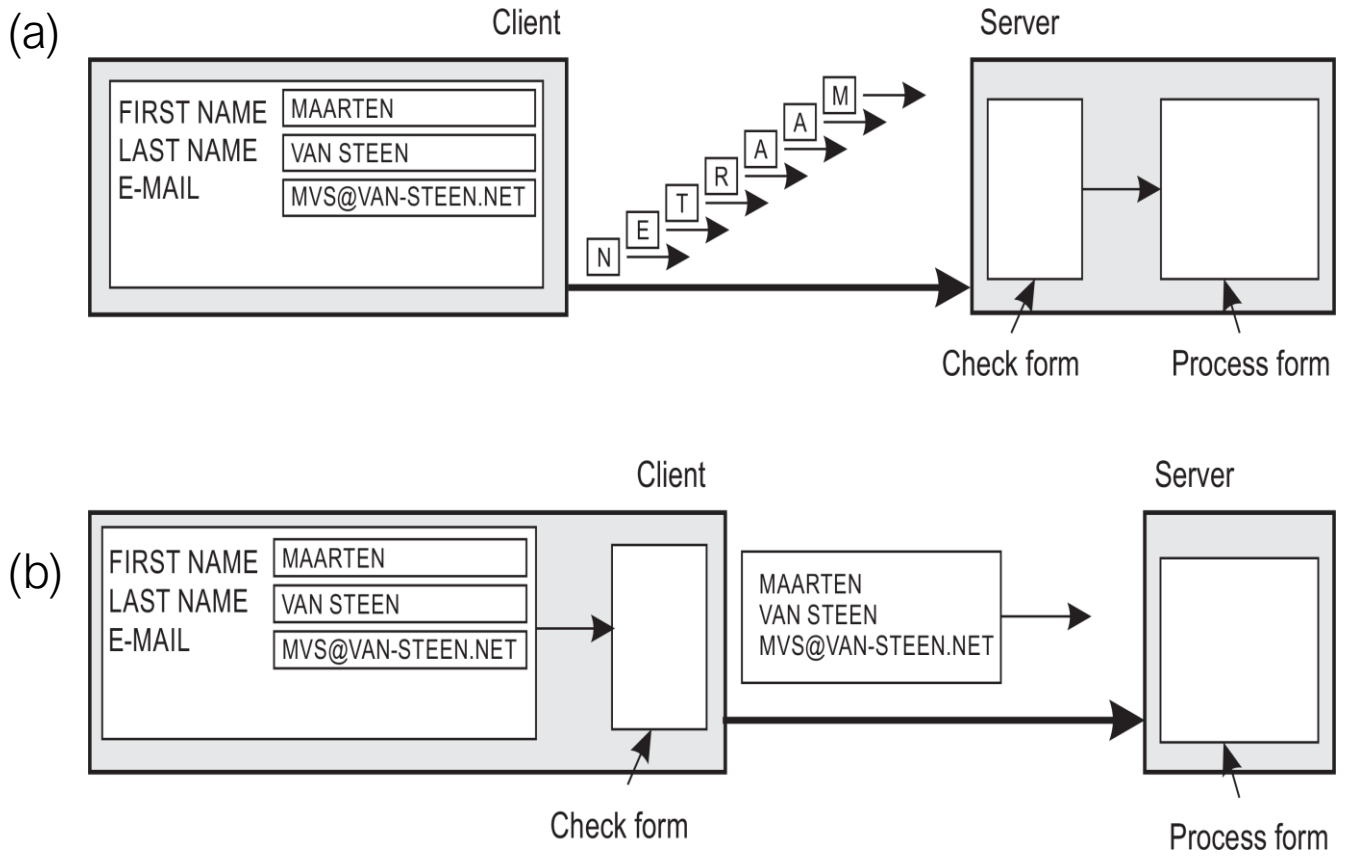
Problema: nem todas as aplicações se encaixam nesse modelo

Técnicas para escalonamento

Diferenças entre deixar um (a) um servidor ou (b) um cliente verificar formulários enquanto estão sendo preenchidos.

Escalonamento vertical – Melhora da capacidade Cpu, Memória

Escalonamento horizontal - implanta mais máquinas (Expande o SD.



Técnicas para escalonamento

- Particionar dados e computações entre várias máquinas
- Mover computações para clientes (applet Java e scripts)
- Serviços de nomenclatura descentralizados (DNS)
- Sistemas de informação descentralizados (WWW)

Técnicas para escalonamento

- Replicação e cache: Disponibilizar cópias de dados em diferentes máquinas
- Servidores de arquivos e bancos de dados replicados
- Sites espelhados
- Caches de Web (em navegadores e proxies)
- Cache de arquivos (no servidor e no cliente)

Escalonamento: O problema com a replicação

Aplicar replicação é fácil, exceto por uma coisa

- Ter múltiplas cópias (em cache ou replicadas) leva a inconsistências: modificar uma cópia faz com que essa cópia se torne diferente das demais.
- Manter sempre as cópias consistentes de maneira geral requer sincronização global a cada modificação.
- A sincronização global impede soluções em larga escala.

Observação

Se pudermos tolerar inconsistências, podemos reduzir a necessidade de sincronização global, mas tolerar inconsistências depende da aplicação.

Computação Paralela

Tipos de Computação Paralela:

- Paralelismo de Dados: Divide grandes conjuntos de dados em blocos menores e realiza operações sobre esses blocos em paralelo.
- Paralelismo de Tarefas: Divide uma tarefa em subtarefas independentes que podem ser executadas simultaneamente.
- Paralelismo de Instruções: Executa múltiplas instruções simultaneamente dentro de uma única tarefa ou processo.

Computação Paralela

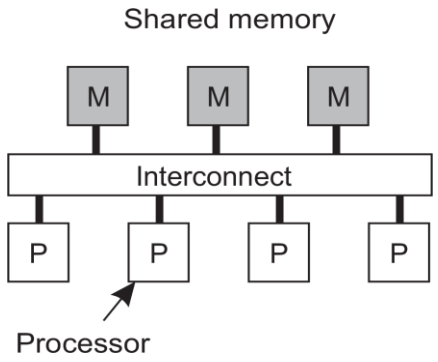
Arquiteturas de Computação Paralela

- Multiprocessadores: Sistemas que possuem múltiplos processadores em um único sistema de computador. Multinúcleos: Processadores que possuem múltiplos núcleos de processamento em um único chip.
- Multicomputadores: Podem ter coordenação mais flexível e menos integrada, permitindo maior independência entre os nós.
- Clusters de Computadores: Conjunto de computadores independentes conectados em rede, trabalhando juntos como um único sistema.
- Supercomputadores: Máquinas altamente poderosas projetadas para realizar grandes quantidades de cálculos em paralelo.

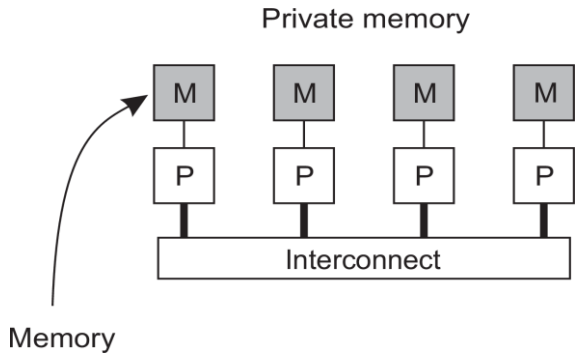
Computação Paralela

Observação

A computação distribuída de alto desempenho começou com a computação paralela



versus



Multiprocessador e Multinúcleo

Multicomputador

Sistemas de memória compartilhada distribuída

Observação

Multiprocessadores são relativamente fáceis de programar em comparação com multicomputadores, mas enfrentam problemas ao aumentar o número de processadores (ou núcleos).

Solução: Tentar implementar um modelo de memória compartilhada em cima de um multicomputador.

Exemplo: Através de técnicas de memória virtual

Mapear todas as páginas da memória principal (de diferentes processadores) em um único espaço de endereçamento virtual. Se um processo no processador A acessar uma página P localizada no processador B, o sistema operacional em A captura e busca P de B, assim como faria se P estivesse localizada no disco local.

Problema

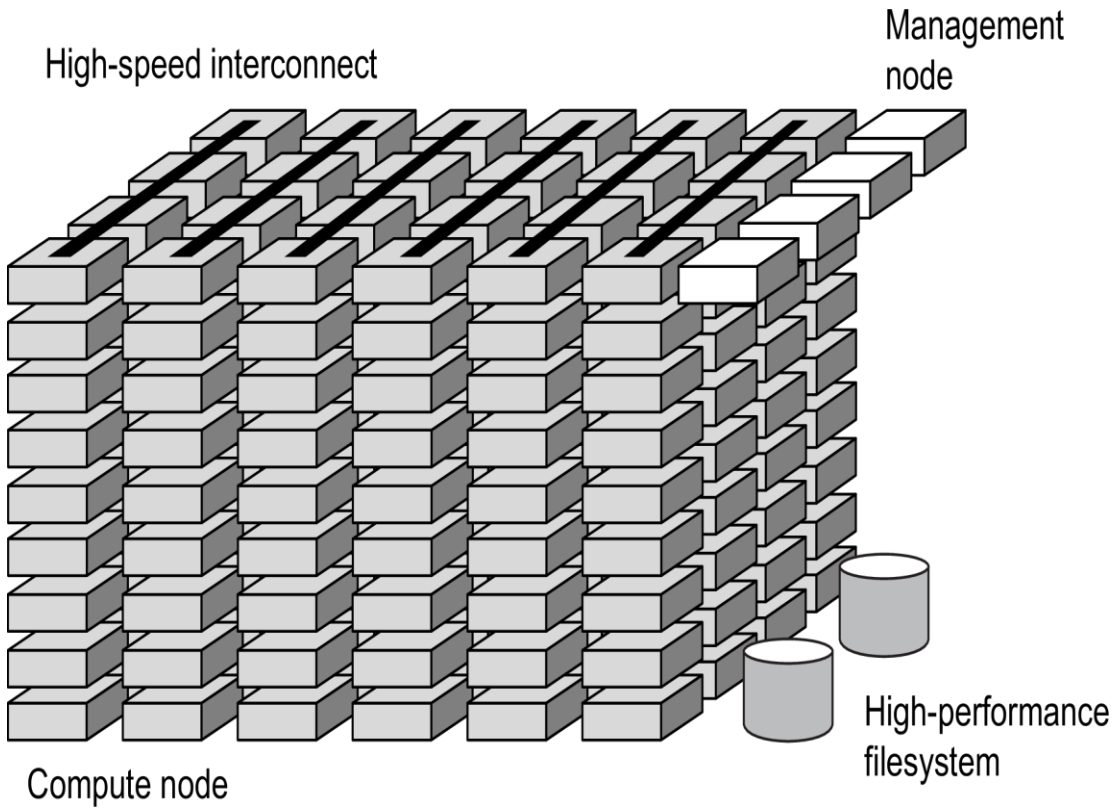
O desempenho da memória compartilhada distribuída nunca pôde competir com o dos multiprocessadores e não atendeu às expectativas dos programadores. Foi amplamente abandonado atualmente.

Computação em cluster

Essencialmente, um grupo de sistemas de alto desempenho conectados através de uma LAN

Homogêneo: mesmo sistema operacional, hardware quase idêntico

Um único nó de gerenciamento, ou nós de gerenciamento fortemente acoplados



Computação em grid

O próximo passo: muitos nós de diferentes locais

- Heterogêneo
- Disperso entre várias organizações
- Pode facilmente abranger uma rede de área ampla

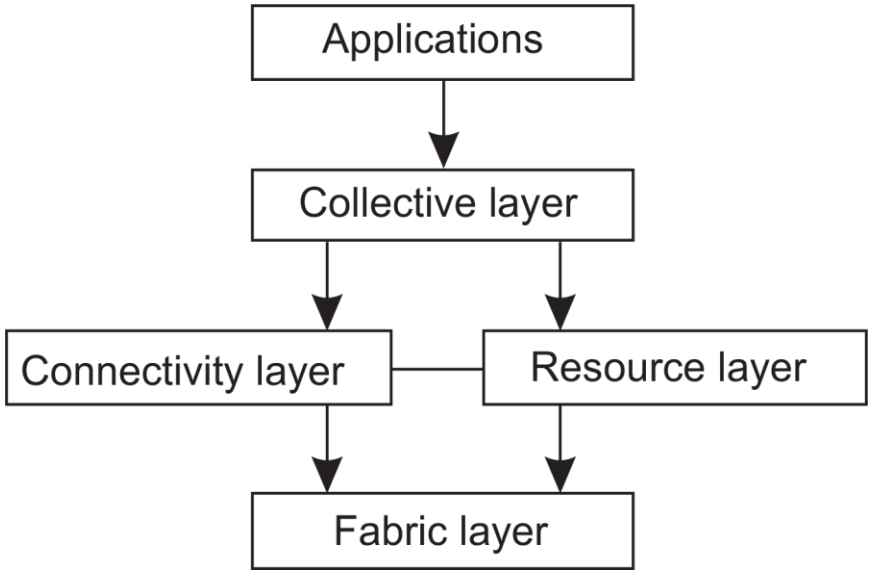
Nota

Para permitir colaborações, as grids geralmente usam **organizações virtuais**. Em essência, isso é um agrupamento de usuários (ou melhor: seus IDs) que permite a autorização na alocação de recursos.

Arquitetura para computação em grid

As camadas

- Fabric: Fornece interfaces para recursos locais (para consulta de estado e capacidades, bloqueio, etc.)
- Connectivity: Protocolos de comunicação/transação, por exemplo, para mover dados entre recursos. Também diversos protocolos de autenticação.
- Resource: Gerencia um único recurso, como a criação de processos ou a leitura de dados.
- Collective: Lida com o acesso a múltiplos recursos: descoberta, agendamento, replicação.
- Application: Contém aplicativos de grid reais em uma única organização.



Integração de aplicativos

Situação: Organizações confrontadas com muitos aplicativos em rede, mas alcançar a interoperabilidade era doloroso.

Abordagem básica

Um aplicativo em rede é aquele que roda em um servidor, disponibilizando seus serviços para clientes remotos.

Integração simples: os clientes combinam solicitações para (diferentes) aplicativos; enviam; coletam respostas e apresentam um resultado coerente ao usuário.

Próximo passo

Permitir comunicação direta entre aplicativos, levando à Integração de Aplicativos Empresariais (*Enterprise Application Integration*).

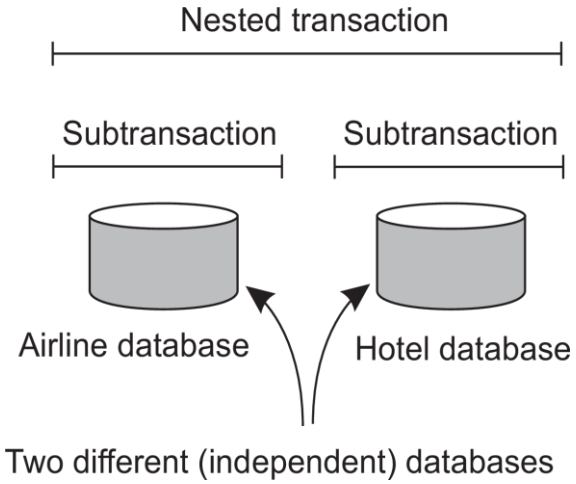
Exemplo de EAI: transações (aninhadas)

Transação

Primitiva	Descrição
BEGIN TRANSACTION	Marca o início de uma transação
END TRANSACTION	Termina a transação e tenta fazer o commit
ABORT TRANSACTION	Cancela a transação e restaura os valores antigos
READ	Lê dados de um arquivo, uma tabela ou outras fontes
WRITE	Escreve dados em um arquivo, uma tabela ou outras fontes

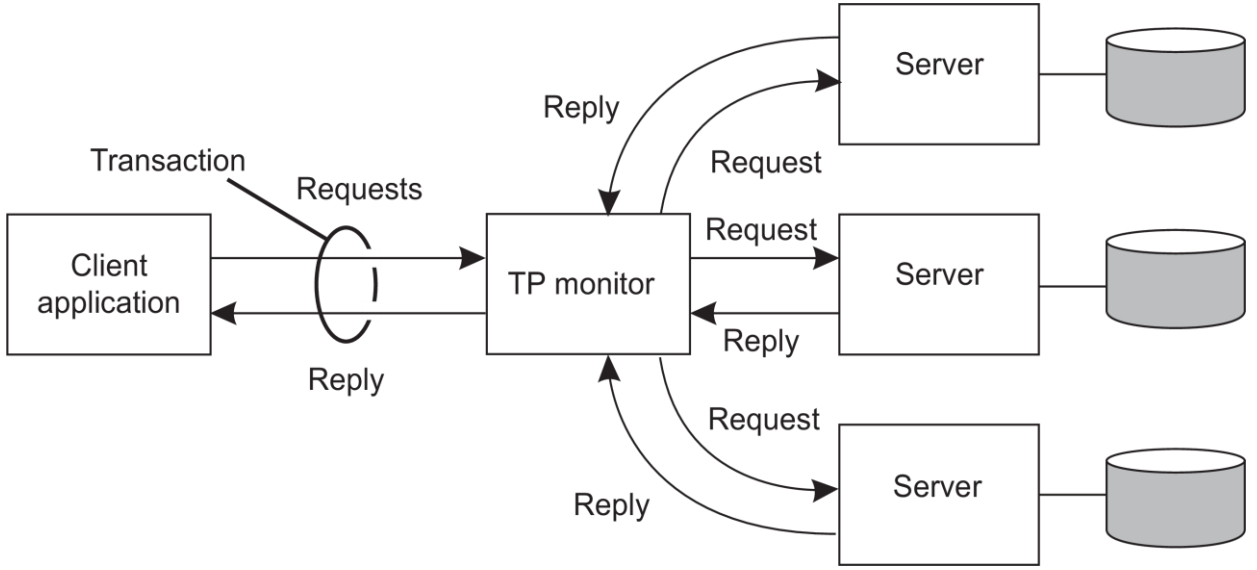
Questão: tudo ou nada

Atômica: acontece de forma indivisível (aparentemente)
Consistente: não viola as invariantes do sistema
Isolada: sem interferência mútua
Durável: commit significa que as mudanças são permanentes



Em sistemas distribuídos, transações são frequentemente construídas como um número de subtransações, formando juntas uma transação aninhada,

Transaction Processing Monitor
TPM: Monitor de Processamento de Transações

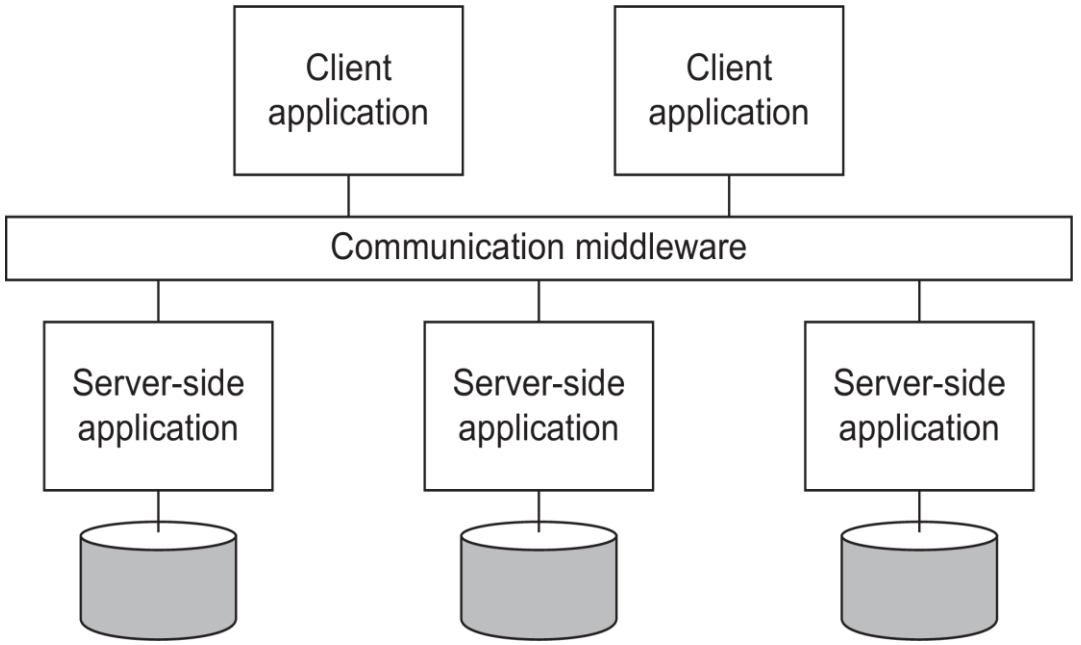


Observação

Frequentemente, os dados envolvidos em uma transação estão distribuídos por vários servidores. Um monitor de processamento de transações (TP Monitor) é **responsável por coordenar a execução de uma transação**.

Middleware e EAI (*Enterprise Application Integration*)

- Middleware oferece facilidades de comunicação para integração
- Remote Procedure Call (RPC): permite que um programa execute uma sub-rotina ou procedimento em outro endereço de espaço de memória (geralmente em uma máquina remota), como se fosse uma chamada de procedimento local. O resultado é retornado como retorno da chamada.
- Message Oriented Middleware (MOM): Mensagens são enviadas para um ponto de contato lógico (publicadas) e encaminhadas para aplicativos inscritos. Permite a troca de mensagens entre diferentes sistemas distribuídos.



Como integrar aplicativos?

Transferência de arquivos: Tecnicamente simples, mas não flexível:

- Definir o formato e o layout do arquivo
- Definir a gestão de arquivos
- Propagação de atualizações e notificações de atualização

Banco de dados compartilhado: Muito mais flexível, mas ainda requer um esquema de dados comum, além do risco de gargalo.

Chamada de procedimento remoto (RPC): Eficaz quando a execução de uma série de ações é necessária.

Mensagens: RPCs **exigem que o chamador e o chamado estejam ativos** ao mesmo tempo. A comunicação por mensagens permite o desacoplamento no tempo e no espaço.

Sistemas distribuídos pervasivos

Emergindo a próxima geração de **sistemas distribuídos em que os nós são pequenos**, móveis e frequentemente embutidos em um sistema maior, caracterizados pelo fato de que o sistema naturalmente se integra ao ambiente do usuário.

Três subtipos (sobrepostos):

- **Sistemas de computação ubíqua**: pervasivos e continuamente presentes, ou seja, há uma **interação** contínua entre o sistema e o usuário. Dispositivos computacionais estão integrados em objetos do cotidiano e ambientes, permitindo acesso à informação e serviços em qualquer lugar e a qualquer momento.
- **Sistemas de computação móvel**: pervasivos, mas com ênfase no fato de que os dispositivos são **inerentemente móveis**.
- **Redes de sensores (e atuadores)**: pervasivas, com ênfase na sensorização e atuação (colaborativa) real do ambiente.

Sistemas ubíquos

Elementos principais

- (Distribuição): Dispositivos são conectados em rede, distribuídos e acessíveis de forma transparente.
- (Interação): A interação entre usuários e dispositivos é altamente discreta.
- (Consciência de contexto): O sistema está ciente do contexto do usuário para otimizar a interação.
- (Autonomia): Dispositivos operam autonomamente sem intervenção humana e, portanto, são altamente autogerenciados.
- (Inteligência): O sistema como um todo pode lidar com uma ampla gama de ações e interações dinâmicas.

Computação móvel

Características distintivas

- Uma infinidade de dispositivos móveis diferentes (**smartphones, tablets, dispositivos GPS, controles remotos, crachás ativos**).
- Móvel implica que a localização de um dispositivo é esperada para mudar ao longo do tempo ⇒ mudança nos serviços locais, acessibilidade, etc. Palavra-chave: descoberta.
- Manter comunicação estável pode introduzir problemas sérios.
- Por um longo tempo, a pesquisa se concentrou em compartilhar recursos diretamente entre dispositivos móveis. Isso nunca se popularizou e atualmente é considerado um caminho infrutífero para a pesquisa.

Conclusão

Dispositivos móveis estabelecem conexões com servidores estacionários, essencialmente colocando a computação móvel na posição de clientes de serviços baseados em nuvem.

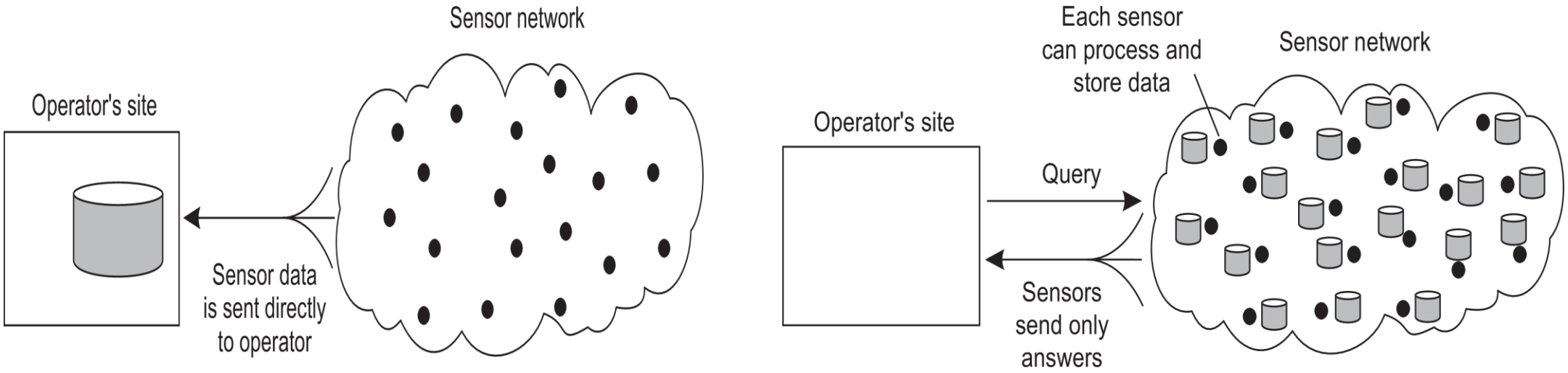
Redes de sensores (Sensor networks)

Características

- Os nós aos quais os sensores estão conectados são:
- Muitos (dezenas a milhares)
- Simples (pequena capacidade de memória/processamento/comunicação)
- Frequentemente alimentados por bateria (ou até mesmo sem bateria)

Redes de sensores como bancos de dados distribuídos

Dois extremos



Desenvolvendo sistemas distribuídos: Armadilhas

Observação

Muitos sistemas distribuídos são desnecessariamente complexos, devido a erros que exigiram correções posteriormente. Muitas suposições falsas são frequentemente feitas.

Suposições falsas (e muitas vezes ocultas)

- ✓ A rede é confiável
- ✓ A rede é segura
- ✓ A rede é homogênea
- ✓ A topologia não muda
- ✓ A latência é zero
- ✓ A largura de banda é infinita
- ✓ O custo de transporte é zero
- ✓ Existe um único administrador

Atividade

- Estudos de Caso:

- Analisar um sistema distribuído real e discutir suas características e desafios.

- Entender quais são as ferramentas disponíveis no mercado para cluster e computação distribuída.