ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

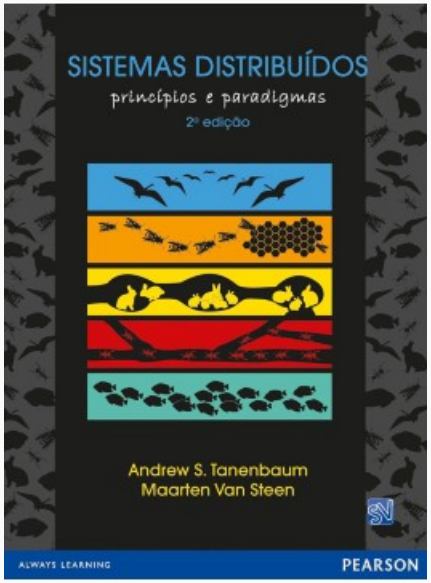
INTRODUÇÃO

Daniel Cordeiro

7 e 9 de março de 2018

Escola de Artes, Ciências e Humanidades | EACH | USP

LIVRO-TEXTO TAMBÉM EM PORTUGUÊS



Sistemas Distribuídos: Princípios e Paradigmas, 2ª edição (em português) ou 3ª edição (em inglês, gratuita)

Autores: Andrew S. Tanenbaum e Maarten van Steen

Pearson Prentice Hall

Slides

Exceto se indicado o contrário, os slides serão baseados nos slides do prof. Maarten van Steen.


SANDÁLIAS
DA
HUMILDADE

SISTEMAS DISTRIBUÍDOS: DEFINIÇÃO

Um sistema distribuído é um sistema de software que garante:

> *uma coleção de elementos de computação autônomos que são vistos pelos usuários como um sistema único e coerente*

Características importantes

- Elementos de computação autônomos, também denominados *nós* (ou *nodos*), sejam eles dispositivos de hardware ou processos de software
- Sistema único e coerente: usuários ou aplicações veem um único sistema $\Rightarrow$ nós precisam colaborar entre si

Comportamento independente
Cada nó é autônomo e, portanto, tem sua própria percepção de tempo: não há um relógio global. Leva a problemas fundamentais de sincronização e de coordenação.

Coleção de nós

- Como gerenciar *associações em grupos*
- Como saber se você realmente está se comunicando com um *(não-)membro autorizado do grupo*

Redes de overlay
Cada nós na coleção se comunica apenas com nós no sistema, seus vizinhos. O conjunto de vizinhos pode ser dinâmico, ou pode ser descoberto de forma implícita (ex: pode ser necessário procurá-lo)

Tipos de overlay
Um exemplo bem conhecido de redes de overlay: *sistemas peer-to-peer*

Estruturada cada nó tem um *conjunto bem definido de vizinhos* com os quais pode comunicar (árvore, anel)

Não estruturada cada nó tem referências a *um conjunto aleatoriamente selecionado de outros nós* do sistema

Essência
A coleção de nós opera sempre da mesma forma, não importando onde, quando ou como a interação entre um usuário e o sistema acontece

Exemplos

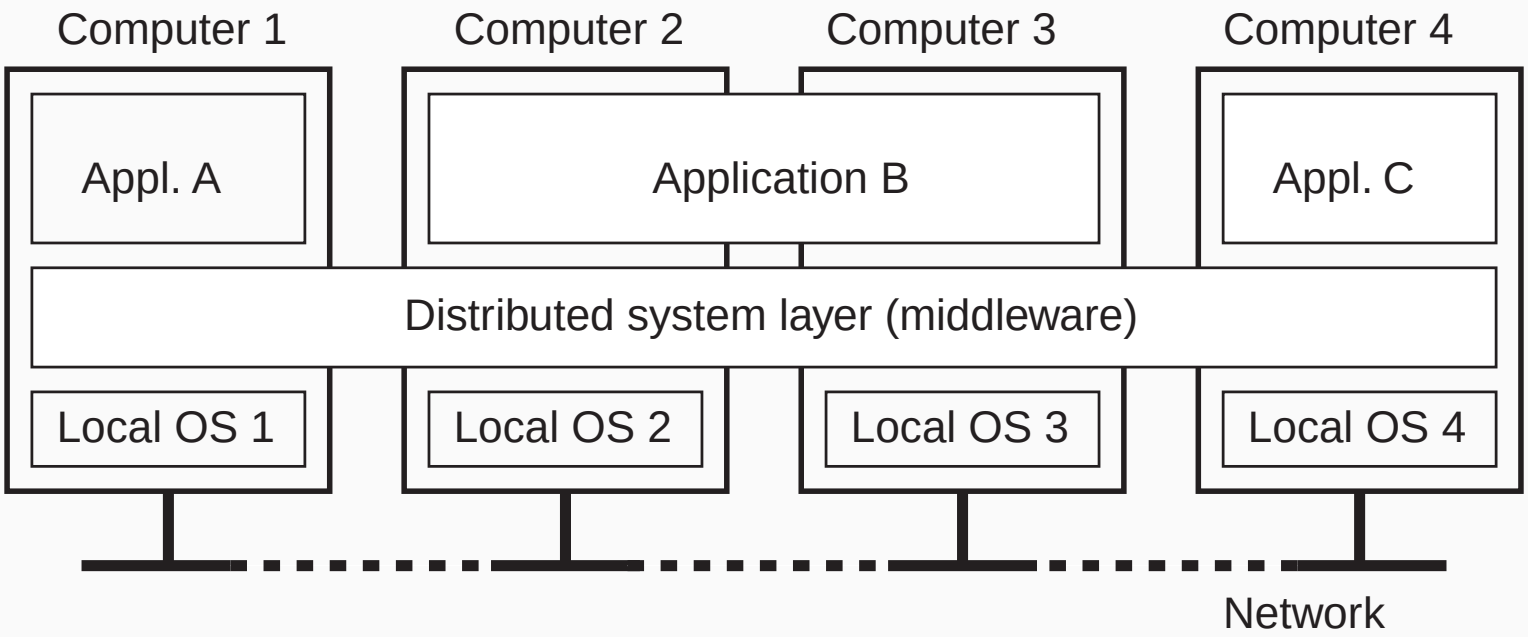
- Um usuário não consegue dizer onde a computação está acontecendo
- Onde especificamente os dados estão armazenados deveria ser irrelevante para a aplicação
- O dado ser ou não replicado deveria estar completamente escondido

A palavra chave é transparência de distribuição

O problema: falhas parciais

É inevitável que a qualquer momento um pedaço do sistema distribuído falhe. Esconder essas falhas parciais e sua recuperação normalmente é muito difícil (em geral, é impossível)

MIDDLEWARE: O SO DOS SISTEMAS DISTRIBUÍDOS



O que tem em um middleware?

Grosso modo, um conjunto de funções e componentes que não precisam ser reimplementados por cada aplicação separadamente.

OBJETIVOS DE SISTEMAS DISTRIBUÍDOS:

- Disponibilização de recursos compartilhados
- Transparência de distribuição
- Abertura
- Escalabilidade

Exemplos clássicos

- Compartilhamento de dados e arquivos na nuvem
- *Streaming* multimídia *peer-to-peer*
- Serviços de mensagens compartilhadas
- Serviços de hospedagem web compartilhados (à lá redes de distribuição de conteúdo)

A ponto de pensarmos que:

> *"A rede é o computador"*
>
> *John Gage, Sun Microsystems (1984)*

TRANSPARÊNCIA DE DISTRIBUIÇÃO

Tipos:

Transparência	Descrição
Acesso	Esconder diferenças entre as representações de dados e mecanismos de invocação
Localização	Esconder onde o objeto está localizado
Relocalização	Esconder que um objeto pode ser movido para outra localidade *enquanto está sendo utilizado*
Migração	Esconder que um objeto pode ser movido para outra localidade
Replicação	Esconder que um objeto está sendo replicado
Concorrência	Esconder que um objeto pode ser compartilhado entre diferentes usuários independentes
Falhas	Esconder falhas e a possível recuperação de um objeto

TRANSPARÊNCIA DE DISTRIBUIÇÃO

Tipos:

Transparência	Descrição
Acesso	Esconder diferenças entre as representações de dados e mecanismos de invocação
Localização	Esconder onde o objeto está localizado
Relocalização	Esconder que um objeto pode ser movido para outra localidade *enquanto está sendo utilizado*
Migração	Esconder que um objeto pode ser movido para outra localidade
Replicação	Esconder que um objeto está sendo replicado
Concorrência	Esconder que um objeto pode ser compartilhado entre diferentes usuários independentes
Falhas	Esconder falhas e a possível recuperação de um objeto

Nota

Transparência de distribuição é um objetivo nobre, mas atingi-lo são outros quinhentos...

Observação:
Tentar fazer com que a distribuição seja totalmente transparente pode ser um exagero:

Observação:
Tentar fazer com que a distribuição seja totalmente transparente pode ser um exagero:

- Usuários podem estar localizados em continentes diferentes

Observação:
Tentar fazer com que a distribuição seja totalmente transparente pode ser um exagero:

- Usuários podem estar localizados em continentes diferentes
- Esconder completamente as falhas da rede e dos nós é (teoricamente e na prática) **impossível**
  - Você não consegue distinguir um computador lento de um que está falhando
  - Você nunca consegue ter certeza de que um servidor terminou de realizar uma operação antes dele ter falhar

Observação:
Tentar fazer com que a distribuição seja totalmente transparente pode ser um exagero:

- Usuários podem estar localizados em continentes diferentes
- Esconder completamente as falhas da rede e dos nós é (teoricamente e na prática) **impossível**
  - Você não consegue distinguir um computador lento de um que está falhando
  - Você nunca consegue ter certeza de que um servidor terminou de realizar uma operação antes dele ter falhar
- Transparência completa terá um custo no desempenho, que irá expor a distribuição do sistema
  - Manter caches web *rigorosamente* atualizados com o original
  - Realizar *flush* das operações de escrita para garantir tolerância a falhas

Sistemas distribuídos abertos

São capazes de interagir com outros sistemas abertos:

- devem respeitar interfaces bem definidas
- devem ser facilmente interoperáveis
- devem permitir a portabilidade de aplicações
- devem ser fáceis de estender

Sistemas distribuídos abertos
São capazes de interagir com outros sistemas abertos:

- devem respeitar interfaces bem definidas
- devem ser facilmente interoperáveis
- devem permitir a portabilidade de aplicações
- devem ser fáceis de estender

A abertura dos sistemas distribuídos os tornam independentes de:

- hardware
- plataformas
- linguagens

A implementação de abertura requer diferentes políticas

- Qual o nível de consistência necessária para os dados no cache do cliente?
- Quais operações podem ser realizadas por programas que acabamos de baixar da Internet?
- Quais os requisitos de QoS podem ser ajustados face a variações na banda disponível?
- Qual o nível de sigilo necessário para a comunicação?

POLÍTICAS VERSUS MECANISMOS

A implementação de abertura requer diferentes políticas

- Qual o nível de consistência necessária para os dados no cache do cliente?
- Quais operações podem ser realizadas por programas que acabamos de baixar da Internet?
- Quais os requisitos de QoS podem ser ajustados face a variações na banda disponível?
- Qual o nível de sigilo necessário para a comunicação?

Idealmente, sistemas distribuídos proveem apenas mecanismos:

- Permitem a atribuição de políticas (dinâmicas) de cache
- Possuem diferentes níveis de confiança para código externo
- Proveem parâmetros de QoS ajustáveis por fluxo de dados
- Oferecem diferentes algoritmos de criptografia

Observação
Quanto mais estrita for a separação entre políticas e mecanismos, mais nós precisamos garantir o uso de mecanismos apropriados, resultando potencialmente em muitos parâmetros de configuração e um gerenciamento mais complexo

Como encontrar um equilíbrio?
Definir políticas estritas normalmente simplifica o gerenciamento e reduz a complexidade, por outro lado isso implica em menos flexibilidade. Não há uma solução simples.

Observação:
Muitos desenvolvedores de sistemas distribuídos modernos usam o adjetivo “escalável” sem deixar claro o porquê deles escalarem.

Observação:
Muitos desenvolvedores de sistemas distribuídos modernos usam o adjetivo “escalável” sem deixar claro o porquê deles escalarem.

Escalabilidade se refere a pelo menos três componentes:

- Número de usuários e/ou processos – escalabilidade de tamanho
- Distância máxima entre nós – escalabilidade geográfica
- Número de domínios administrativos – escalabilidade administrativa

Observação:
Muitos desenvolvedores de sistemas distribuídos modernos usam o adjetivo “escalável” sem deixar claro o porquê deles escalarem.

Escalabilidade se refere a pelo menos três componentes:

- Número de usuários e/ou processos – escalabilidade de tamanho
- Distância máxima entre nós – escalabilidade geográfica
- Número de domínios administrativos – escalabilidade administrativa

Observação:
A maior parte dos sistemas escalam apenas (e até certo ponto) em tamanho. A solução(?!): servidores potentes. Hoje em dia, o desafio é conseguir escalabilidade geográfica e administrativa.

DIFICULDADE PARA OBTENÇÃO DE ESCALABILIDADE

Um sistema completamente descentralizado tem as seguintes características:

- Nenhuma máquina tem informação completa sobre o estado do sistema
- Máquinas tomam decisões baseadas apenas em informação local
- Falhas em uma máquina não devem arruinar a execução do algoritmo
- Não é possível assumir a existência de um relógio global

Ideia geral: esconder latência de comunicação

Não fique esperando por respostas; faça outra coisa

- Utilize comunicação assíncrona
- Mantenha diferentes *handlers* para tratamento de mensagens recebidas
- Problema: nem toda aplicação se encaixa nesse modelo

Particionamento de dados e computação em muitas máquinas

- Mova a computação para os clientes (ex: Javascript, Applets Java, etc.)
- Serviços de nomes decentralizados (DNS)
- Sistemas de informação decentralizados (WWW)

Replicação/caching
Faça cópias dos dados e disponibilize-as em diferentes máquinas:

- Bancos de dados e sistemas de arquivos replicados
- Sites web “espelhados”
- Caches web (nos navegadores e nos *proxies*)
- Cache de arquivos (no servidor e nos clientes)

Observação
Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por um problema:

Observação
Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por um problema:

- Manter múltiplas cópias (em cache ou replicadas) leva a inconsistências: a modificação em uma cópia a torna diferente das demais

Observação
Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por um problema:

- Manter múltiplas cópias (em cache ou replicadas) leva a inconsistências: a modificação em uma cópia a torna diferente das demais
- Manter as cópias consistentes requer sincronização global em cada modificação

Observação
Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por um problema:

- Manter múltiplas cópias (em cache ou replicadas) leva a inconsistências: a modificação em uma cópia a torna diferente das demais
- Manter as cópias consistentes requer sincronização global em cada modificação
- Sincronização global impossibilita soluções escaláveis

Observação
Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por um problema:

- Manter múltiplas cópias (em cache ou replicadas) leva a inconsistências: a modificação em uma cópia a torna diferente das demais
- Manter as cópias consistentes requer sincronização global em cada modificação
- Sincronização global impossibilita soluções escaláveis

Observação:
Se nós pudermos tolerar inconsistências, nós podemos reduzir a dependência em sincronização globais, mas tolerar inconsistências é algo que depende da aplicação.

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de “consertos” ao longo do tempo. Em geral, há muitas hipóteses falsas:

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia da rede não muda

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia da rede não muda
- A latência é zero

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia da rede não muda
- A latência é zero
- Largura de banda é infinita

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia da rede não muda
- A latência é zero
- Largura de banda é infinita
- O custo de transporte é zero

Observação:
Muitos sistemas distribuídos se tornam desnecessariamente complexos por causa de "consertos" ao longo do tempo. Em geral, há muitas hipóteses falsas:

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia da rede não muda
- A latência é zero
- Largura de banda é infinita
- O custo de transporte é zero
- A rede possui um administrador

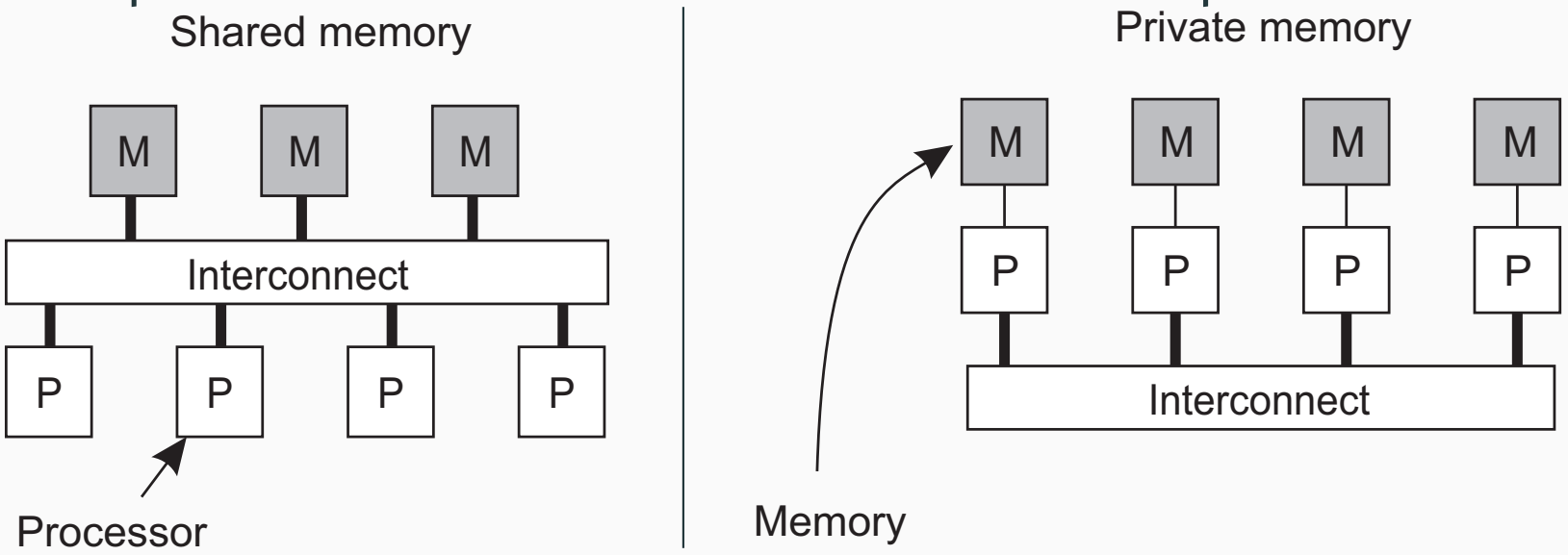
TRÊS TIPOS DE SISTEMAS DISTRIBUÍDOS

- Sistemas para computação distribuída de alto desempenho
- Sistemas de informação distribuídos
- Sistemas distribuídos para computação ubíqua

Observação

A computação distribuída de alto desempenho foi originada na computação paralela

Multiprocessadores e multicores versus multicomputadores



Observação

Multiprocessadores são relativamente fáceis de programar se comparados a multicomputadores, mas ainda assim os problemas aparecem quando o número de processadores (ou cores) aumentam. Solução: tentar implementar um modelo de memória compartilhada para multicomputadores.

Exemplo usando técnicas de memória virtual

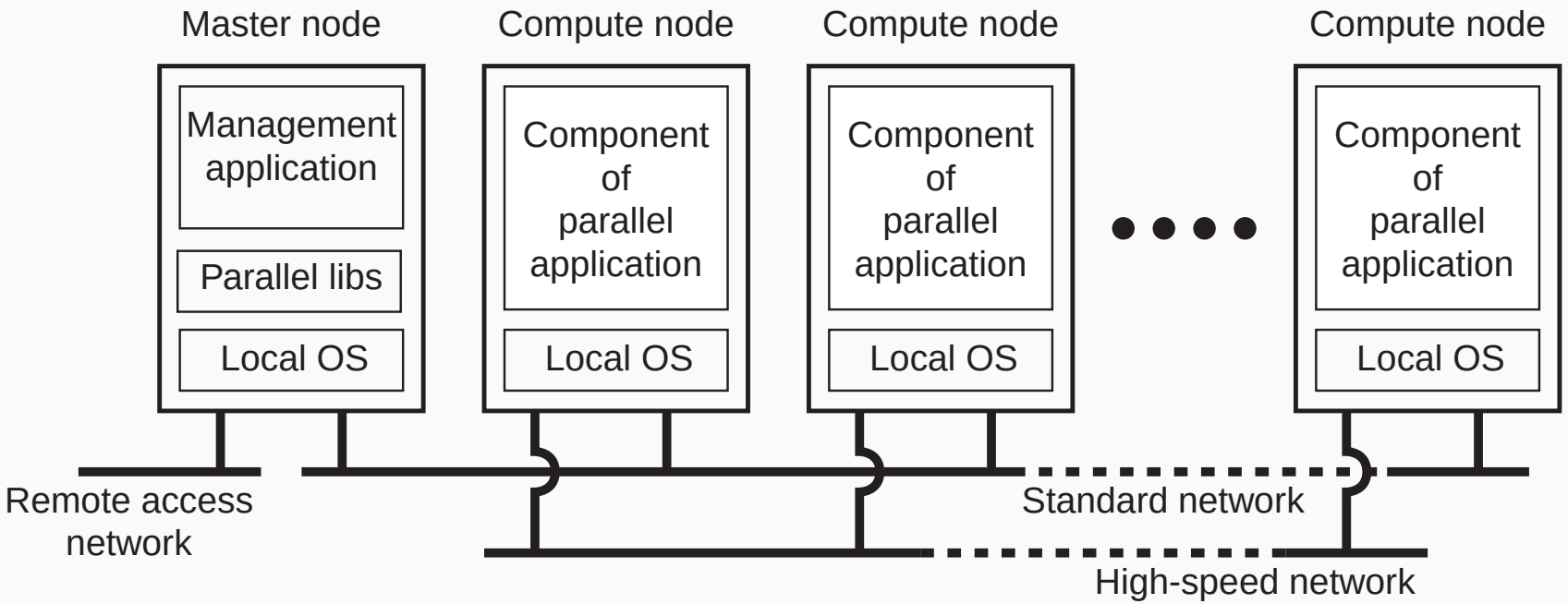
Mapear todas as páginas da memória principal (de todos os diferentes processadores) em um único espaço de endereçamento virtual. Se o processo no processador *A* referenciar uma página *P* localizada no processador *B*, o SO em *A* lança uma interrupção e recupera *P* de *B*, do mesmo modo que faria se *P* estivesse localizado no disco.

Problema

O desempenho de um sistema de memória compartilhada distribuída nunca poderia competir com o desempenho de multiprocessadores e, por isso, a ideia foi abandonada por hora.

Essencialmente um grupo de computadores de boa qualidade conectados via LAN

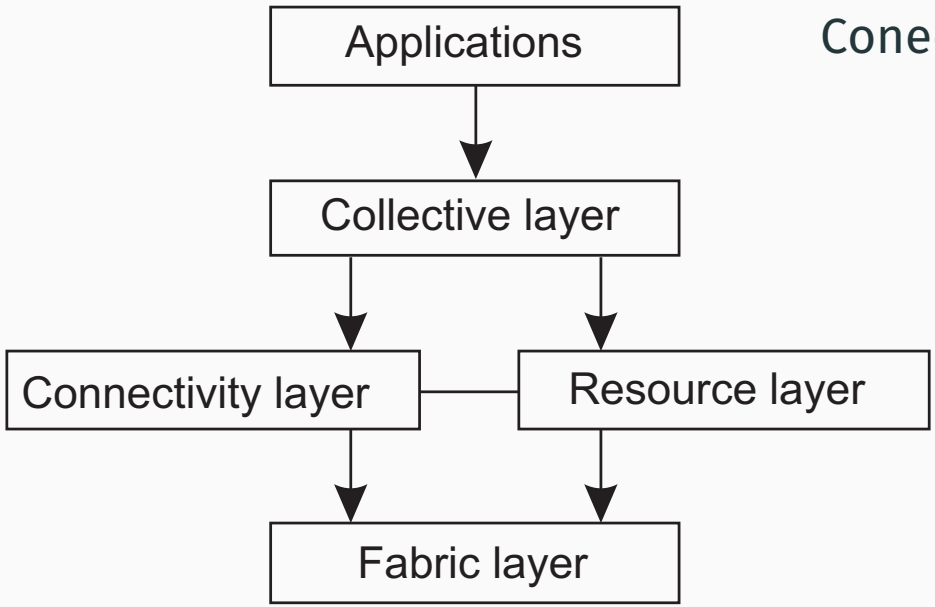
- Homogêneo: mesmo SO, hardware quase idêntico
- Um único nó gerenciador



O próximo passo: vários nós vindos de todos os cantos:

- Heterogêneos
- Espalhados entre diversas organizações
- Normalmente formam uma rede de longa distância (*wide-area network*)

Nota:
Para permitir colaborações, grades normalmente usam *organizações virtuais*. Essencialmente, isso significa que os usuários (ou melhor, seus IDs) são organizados em grupos que possuem autorização para usar alguns recursos.



Camadas

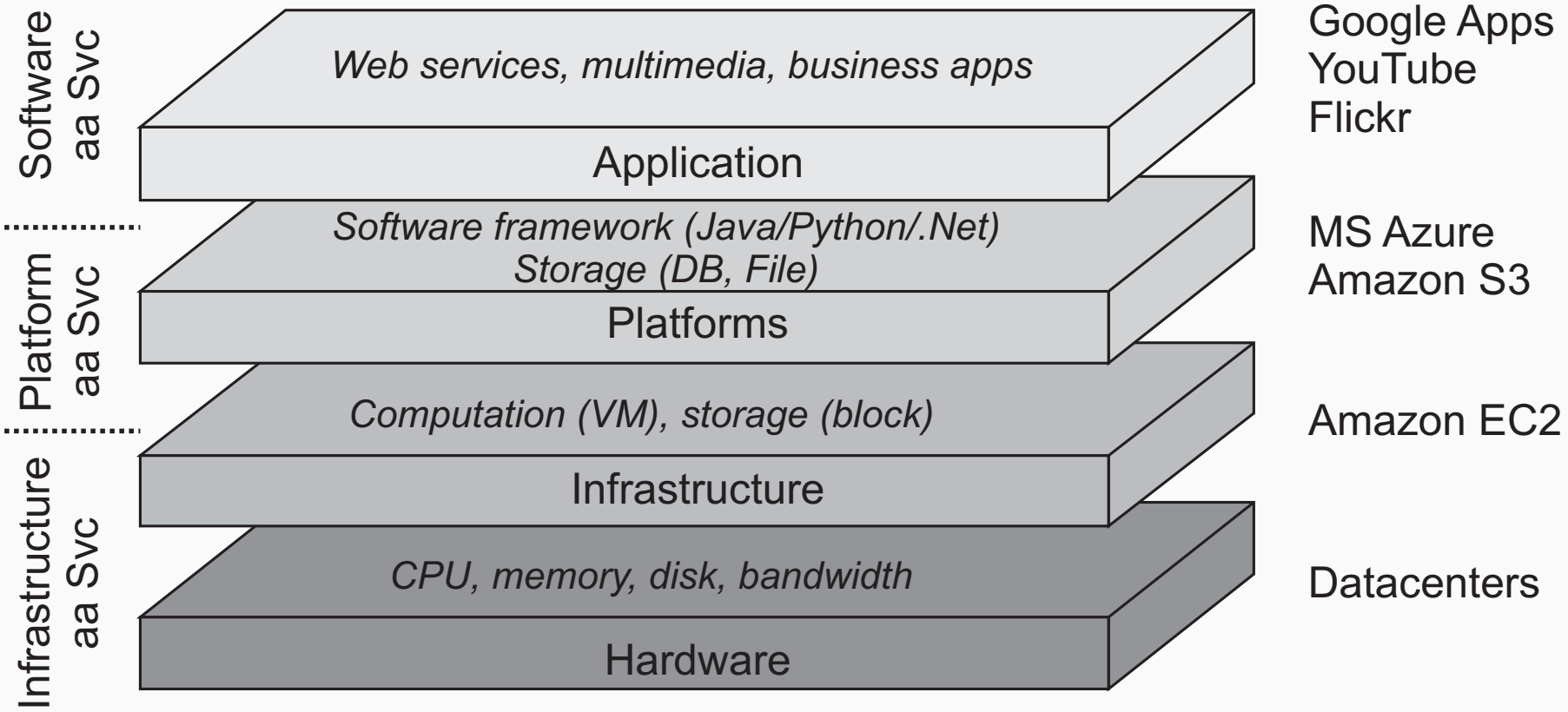
Infraestrutura provê interfaces para os recursos locais

Conectividade protocolos de comunicação/transação e autenticação

Recurso gerencia um único recurso, por exemplo criando processos ou lendo dados

Coletiva realiza acesso à múltiplos recursos: descoberta, escalonamento, replicação

Aplicação contém a aplicação real da grade em uma única organização


Software aa Svc
Platform aa Svc
Infrastructure aa Svc
Web services, multimedia, business apps
Application
Software framework (Java/Python/.Net)
Storage (DB, File)
Platforms
Computation (VM), storage (block)
Infrastructure
CPU, memory, disk, bandwidth
Hardware
Google Apps
YouTube
Flickr
MS Azure
Amazon S3
Amazon EC2
Datacenters

Computação em nuvem
Faz uma distinção entre quatro camadas:

Hardware processadores, roteadores, energia, sistemas de refrigeração

Infraestrutura Utiliza técnicas de virtualização para alocação e gerenciamento de armazenamento e servidores virtuais

Plataforma Provê abstrações de alto nível para os serviços da plataforma. Ex: Amazon S3 para armazenamento de arquivos em *buckets*

Aplicação as aplicações propriamente ditas, tais como as suítes de aplicativos para escritórios.

USAR COMPUTAÇÃO EM NUVEM É ECONOMICAMENTE VIÁVEL?

Observação

Uma razão importante para o sucesso de computação em nuvem é que ela permite que organizações terceirizem sua infraestrutura de TI: hardware e software. A pergunta é: terceirizar é mesmo mais barato?

Abordagem

- Considere *aplicações corporativas*, modeladas como uma coleção de componentes (C_i), cada qual precisando de N_i servidores
- Podemos ver a aplicação como um grafo dirigido, com um vértice representando um componente e um arco $\langle i, \vec{j} \rangle$ representando o fluxo de dados de C_i para C_j.
- Cada arco tem dois pesos associados:
  - $T_{i,j}$, o número de transações por unidade de tempo que causam o fluxo de dados de C_i para C_j
  - $S_{i,j}$, a quantidade de dados total associada a $T_{i,j}$

Plano de migração
Encontre para cada componente C_i, quantos dos n_i dentre seus N_i servidores deveriam migrar, tal que a economia no orçamento menos os cursos de comunicação via Internet sejam maximais.

Requesitos para o plano de migração

1. As restrições impostas pelas políticas devem ser respeitadas
2. Latências adicionais não violarão nenhuma das restrições
3. Todas as transações continuarão a operar corretamente; requisições ou dados não serão perdidos durante a transação

Economia no orçamento

- B_c: economias com a migração de um componente computacionalmente intensivo
- M_c número total de componentes computacionalmente intensivos
- B_s: economias com a migração de um componente intensivo em armazenamento
- M_s número total de componentes intensivos em armazenamento

A economia total, obviamente, é: $B_c \times M_c + B_s \times M_s$.

Tráfego de/para a nuvem

$$Tr_{\text{local, inet}} = \sum_{C_i} (T_{\text{usuário},i} S_{\text{usuário},i} + T_{i,\text{usuário}} S_{i,\text{usuário}})$$

- $T_{\text{usuário},i}$: transações por unidade de tempo que causam fluxo de dados do usuário para C_i
- $S_{\text{usuário},i}$ quantidade de dados associados com $T_{\text{usuário},i}$

TAXA DE TRANSAÇÕES APÓS MIGRAÇÃO

Mais notações:

- $C_{i,\text{local}}$: conjunto de servidores de C_i que continuam executando localmente
- $C_{i,\text{cloud}}$: conjunto de servidores de C_i migrados para o cloud
- Assuma que a distribuição de tráfego é a mesma para o servidor local ou no cloud.

Note que $|C_{i,\text{cloud}} = n_i|$. Seja $f_i = n_i/N_i$ e s_i um servidor de C_i.

$$T^*_{i,j} = \begin{cases} (1-f_i)\cdot(1-f_j)\cdot T_{i,j} & \text{quando } s_i \in C_{i,local} \text{ e } s_j \in C_{j,local} \\ (1-f_i)\cdot f_j \cdot T_{i,j} & \text{quando } s_i \in C_{i,local} \text{ e } s_j \in C_{j,cloud} \\ f_i \cdot (1-f_j)\cdot T_{i,j} & \text{quando } s_i \in C_{i,cloud} \text{ e } s_j \in C_{j,local} \\ f_i \cdot f_j \cdot T_{i,j} & \text{quando } s_i \in C_{i,cloud} \text{ e } s_j \in C_{j,cloud} \end{cases}$$

Observação:
Uma quantidade enorme de sistemas distribuídos em uso hoje em dia são formas de sistemas de informação tradicionais, *integrando* sistemas legados. Exemplo: sistemas de processamento de transações.

```
BEGIN_TRANSACTION(server, transaction)
READ(transaction, file-1, data)
WRITE(transaction, file-2, data)
newData := MODIFIED(data)
IF WRONG(newData) THEN
    ABORT_TRANSACTION(transaction)
ELSE
    WRITE(transaction, file-2, newData)
    END_TRANSACTION(transaction)
END IF
```

Observação:
Uma quantidade enorme de sistemas distribuídos em uso hoje em dia são formas de sistemas de informação tradicionais, *integrando* sistemas legados. Exemplo: sistemas de processamento de transações.

```
BEGIN_TRANSACTION(server, transaction)
READ(transaction, file-1, data)
WRITE(transaction, file-2, data)
newData := MODIFIED(data)
IF WRONG(newData) THEN
    ABORT_TRANSACTION(transaction)
ELSE
    WRITE(transaction, file-2, newData)
    END_TRANSACTION(transaction)
END IF
```

Nota:
Transações formam uma operação atômica.

Uma transação é um conjunto de operações sobre o estado de um objeto (banco de dados, composição de objetos, etc.) que satisfazem as seguintes propriedades (ACID):

Atomicidade ou todas as operações são bem sucedidas, ou todas falham. Quando uma transação falha, o estado do objeto permanecerá inalterado.

Uma transação é um conjunto de operações sobre o estado de um objeto (banco de dados, composição de objetos, etc.) que satisfazem as seguintes propriedades (ACID):

Atomicidade ou todas as operações são bem sucedidas, ou todas falham. Quando uma transação falha, o estado do objeto permanecerá inalterado.

Consistência uma transação estabelece um estado de transição válido. Isto não exclui a existência de estados intermediários inválidos durante sua execução.

Uma transação é um conjunto de operações sobre o estado de um objeto (banco de dados, composição de objetos, etc.) que satisfazem as seguintes propriedades (ACID):

Atomicidade ou todas as operações são bem sucedidas, ou todas falham. Quando uma transação falha, o estado do objeto permanecerá inalterado.

Consistência uma transação estabelece um estado de transição válido. Isto não exclui a existência de estados intermediários inválidos durante sua execução.

Isolamento transações concorrentes não interferem entre si. Para uma transação T é como se as outras transações ocorressem ou *antes* de T, ou *depois* de T.

Uma transação é um conjunto de operações sobre o estado de um objeto (banco de dados, composição de objetos, etc.) que satisfazem as seguintes propriedades (ACID):

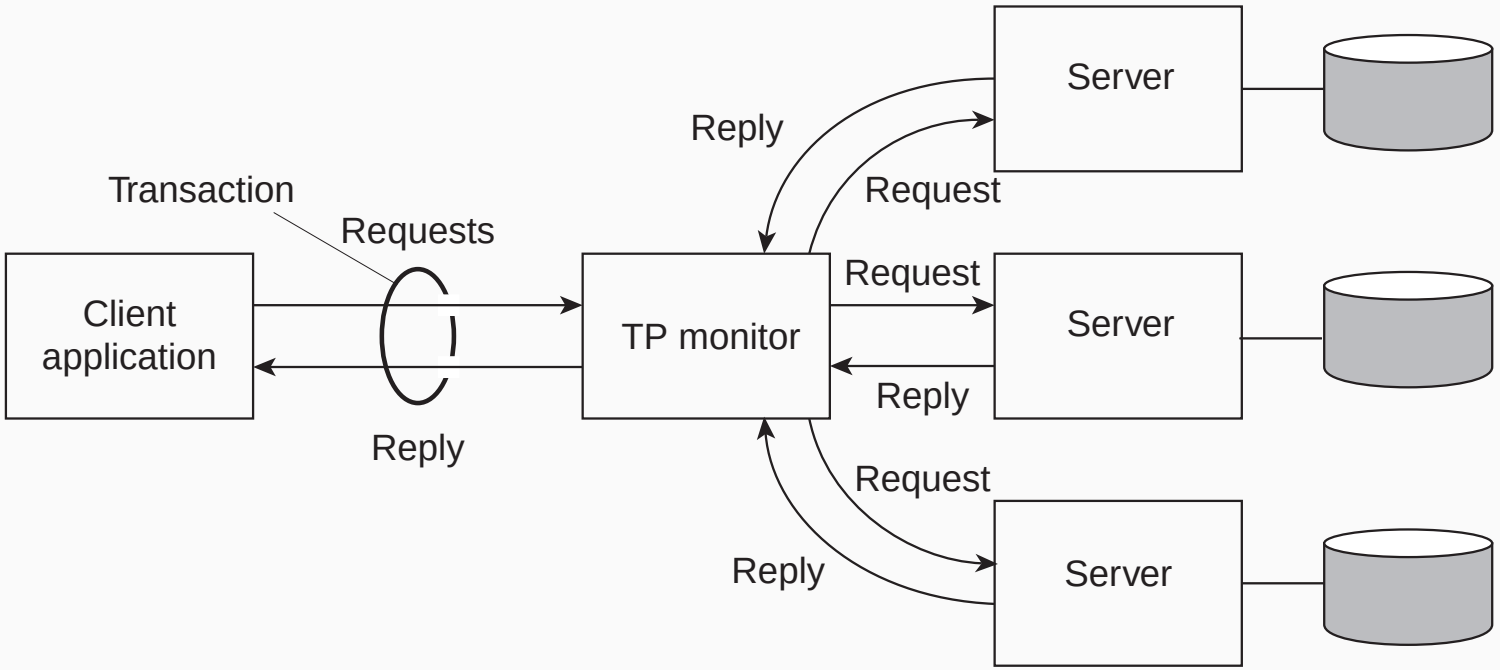
Atomicidade ou todas as operações são bem sucedidas, ou todas falham. Quando uma transação falha, o estado do objeto permanecerá inalterado.

Consistência uma transação estabelece um estado de transição válido. Isto não exclui a existência de estados intermediários inválidos durante sua execução.

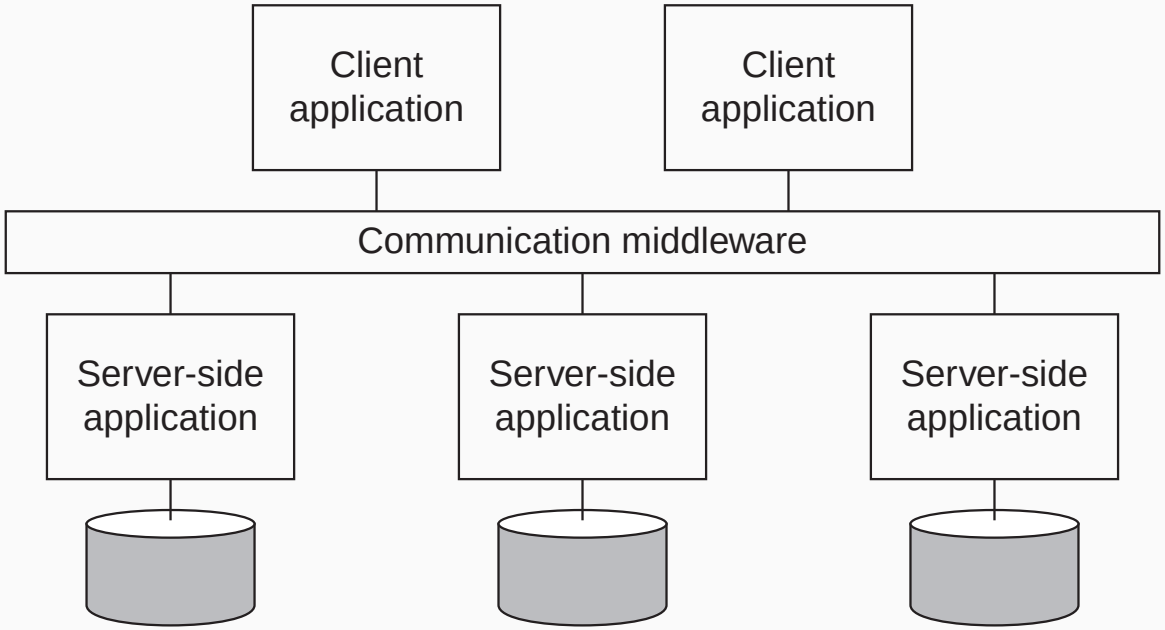
Isolamento transações concorrentes não interferem entre si. Para uma transação T é como se as outras transações ocorressem ou *antes* de T, ou *depois* de T.

Durabilidade Após o término de uma transação, seus efeitos são permanentes: mudanças de estado sobrevivem a falhas.

Observação:
Em muitos casos, o conjunto de dados envolvidos em uma transação está distribuído em vários servidores. Um TP Monitor é responsável por coordenar a execução de uma transação.



Problema
Um TP Monitor não basta, também são necessários mecanismos para a comunicação direta entre aplicações.



- Chamada de Procedimento Remoto (RPC)
- Middleware Orientado a Mensagens (MOM)

Tendência em sistemas distribuídos; nós são pequenos, móveis e normalmente embutidos em um sistema muito maior.

Alguns requisitos:

- Mudança contextual: o sistema é parte de um ambiente onde mudanças devem ser rapidamente levadas em consideração
- Composição ad hoc: cada nó pode ser usado em diferentes maneiras, por diferentes usuários. Deve ser facilmente configurável.
- Compartilhar é o padrão: nós vão e veem, fornecendo serviços e informação compartilháveis. Pede simplicidade.

Tendência em sistemas distribuídos; nós são pequenos, móveis e normalmente embutidos em um sistema muito maior.

Alguns requisitos:

- Mudança contextual: o sistema é parte de um ambiente onde mudanças devem ser rapidamente levadas em consideração
- Composição ad hoc: cada nó pode ser usado em diferentes maneiras, por diferentes usuários. Deve ser facilmente configurável.
- Compartilhar é o padrão: nós vão e veem, fornecendo serviços e informação compartilháveis. Pede simplicidade.

Nota:
Ubiquidade e transparência de distribuição formam um bom par?

Sistemas domésticos

Devem ser completamente auto-organizáveis:

- Não deve haver um administrador do sistema
- Solução mais simples: um home box centralizado?

Sistemas domésticos

Devem ser completamente auto-organizáveis:

- Não deve haver um administrador do sistema
- Solução mais simples: um home box centralizado?

Monitorando uma pessoa

Dispositivos ficam fisicamente próximos a uma pessoa:

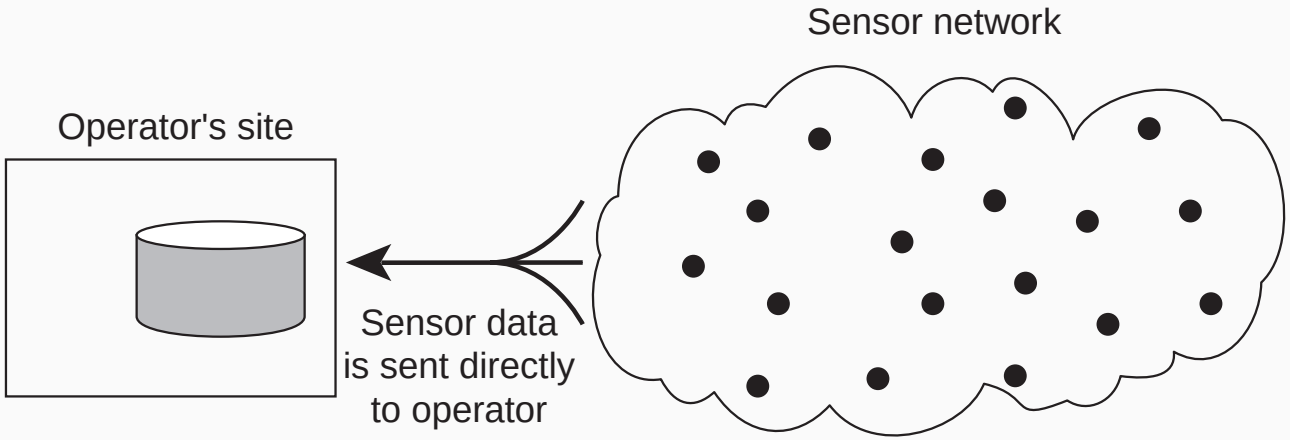
- Onde e como são armazenados os dados monitorados?
- Podemos prevenir perda de dados importantes?
- Há necessidade de gerar e propagar alertas?
- Como fazer para garantir segurança?
- Como o ambiente pode prover *feedback* online?

Características

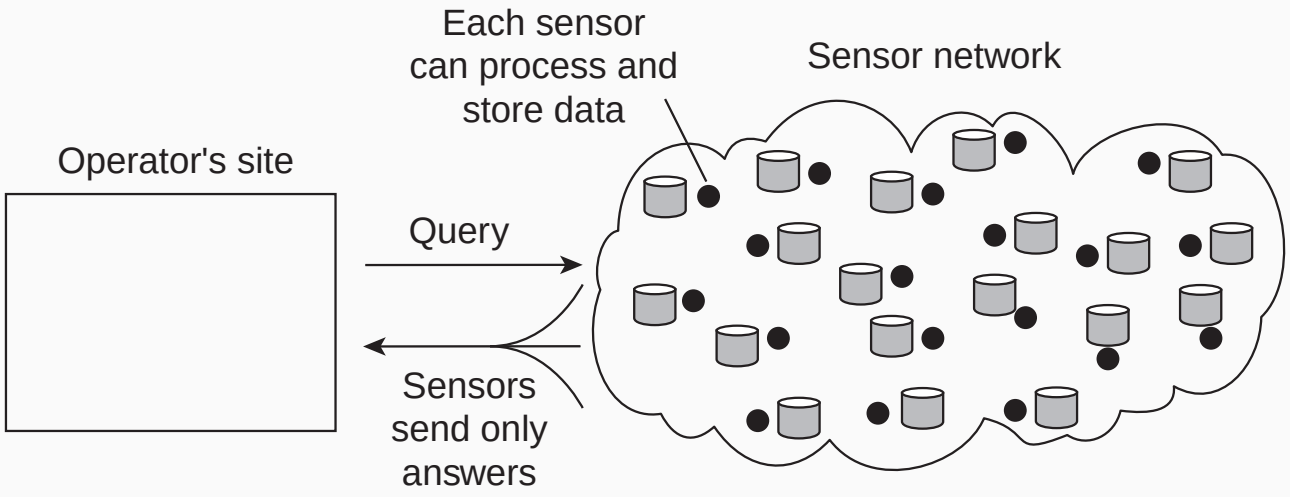
Os nós aos quais os sensores estão presos são:

- Muitos (10s–1000s)
- Simples (pouca capacidade de memória/computação/comunicação)
- Normalmente necessitam de uma bateria

REDES DE SENSORES COMO UM SISTEMA DISTRIBUÍDO



(a)



(b)

Gerenciamento de multidões

- Situação: um grande evento sem rotas fixas (exposições, festivais, etc.)
- Objetivo: guiar as pessoas de acordo com suas posições sociais:
  - direcionar pessoas com interesses similares para os mesmos locais
  - direcionar membros de um grupo para uma mesma saída no caso de uma emergência
- Objetivo: manter grupos unidos (p.ex.:, famílias)

Estimulando a mistura

- Situação: conferência com pessoas de diferentes grupos
- Objetivo: estimular pessoas de diferentes grupos a interagirem.
- Abordagem: acompanhar as interações entre os grupos:
  - Quando um aluno de SI fala com um aluno de TM: pontos de bônus para os dois alunos e para os seus respectivos grupos.
  - Pontos para o grupo são distribuídos entre os seus membros
  - Conquistas são mostradas em crachás eletrônicos (*feedback* e *intervenções sociais*)

CENÁRIOS DE APLICAÇÃO: JOGOS SOCIAIS