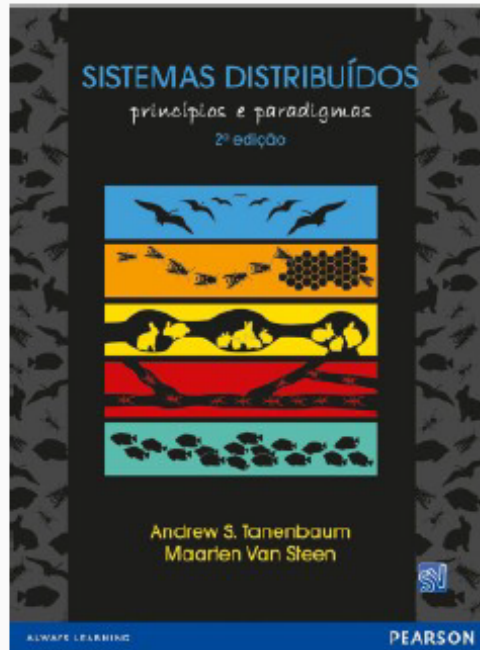

ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

INTRODUÇÃO



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.

Sistemas Distribuídos

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

Coleção de nós autônomos

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*

Organização

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

Sistema transparente

Essência

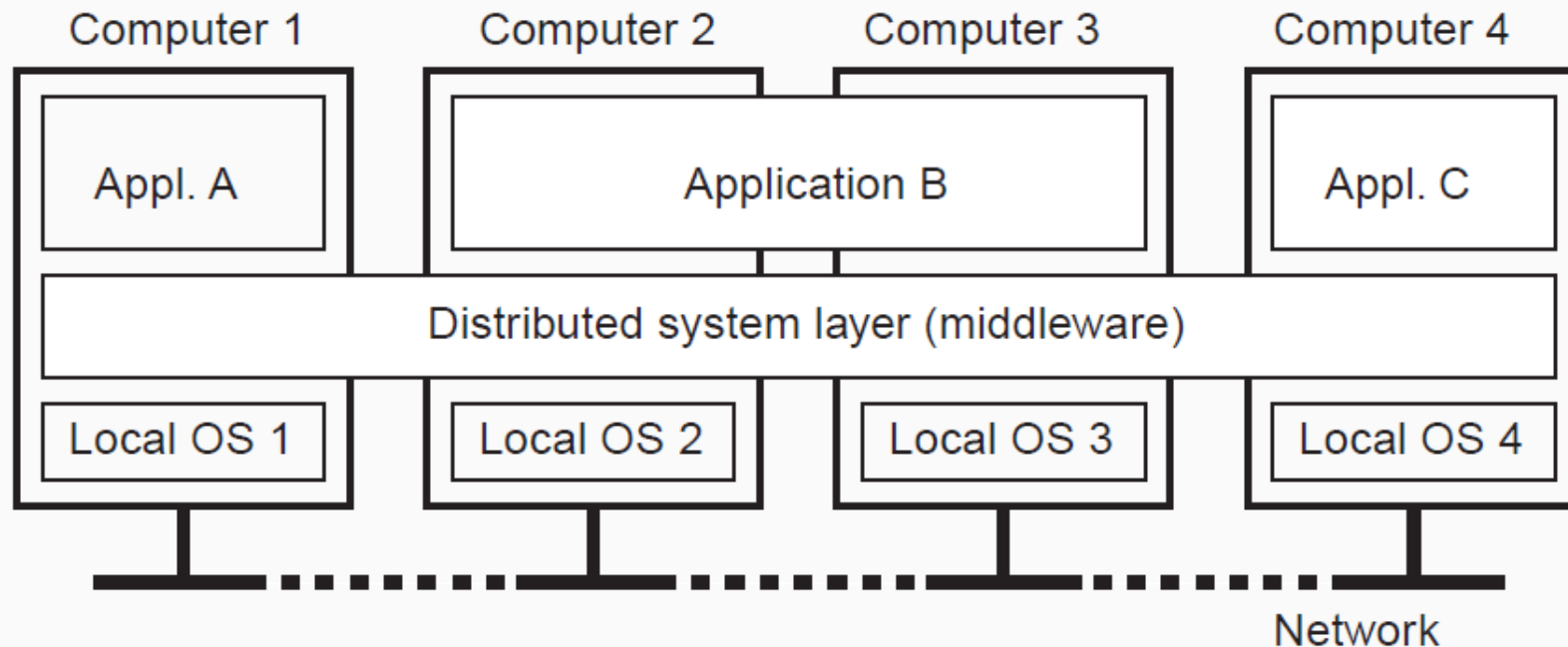
A coleção de nós opera sempre da mesma forma, não importando onde, quando ou como a interação entre um usuário e o sistema acontece

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**

Middleware



O que tem em um middleware?

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

Objetivos dos Sist. Distribuídos

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

Compartilhamento

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, à época na Sun Microsystems

Transparência

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 <i>enquanto está sendo utilizado</i>
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

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

Abertura

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**

Abertura

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

Escalabilidade

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.

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

Técnicas - Escalabilidade

Particionamento de dados e computação em muitas máquinas

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

Técnicas - Escalabilidade

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)

Técnicas - Escalabilidade

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

Problema - Escalabilidade

Aplicar técnicas para obtenção de escalabilidade é fácil, exceto por uma 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.

Armadilhas no desenvolvimento

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