Departamento de Computação - **UFSCar**

1001503
02.532-1

SISTEMAS DISTRIBUÍDOS

Prof. Dr. FREDY JOÃO VALENTE

1º Semestre 2023

25321 - SISTEMAS DISTRIBUIDOS

- CONVENCER O (A) ALUNO (A) SOBRE A IMPORTÂNCIA ATUAL DE SISTEMAS DISTRIBUÍDOS PARA O ATENDIMENTO DE REQUISITOS IMPOSTOS POR NOVAS APLICAÇÕES OU COMO SOLUÇÕES A PROBLEMAS ANTIGOS, QUE SE MOSTRAVAM INVIÁVEIS DE SEREM ALCANÇADAS COM PROCESSAMENTO CENTRALIZADO.ENSINAR OS CONCEITOS GERAIS DE SISTEMAS DE SOFTWARE PARA PROCESSAMENTO DISTRIBUÍDO E AS TÉCNICAS UTILIZADAS PARA SUAS IMPLEMENTAÇÕES EM AMBIENTES DE MULTIPROCESSADORES FRACAMENTE ACOPLADOS.ESTIMULAR O (A) ALUNO (A) A SE INTERESSAR PELA PESQUISA E DESENVOLVIMENTO DE TÉCNICAS AINDA NÃO ESTABELECIDAS PARA A SOLUÇÃO DE PROBLEMAS COMPUTACIONAIS, EM PARTICULAR EM PROBLEMAS RELACIONADOS COM A TRANSPARÊNCIA QUE SISTEMAS DISTRIBUÍDOS DEVEM APRESENTAR A USUÁRIOS E PROJETISTAS DE APLICAÇÕES E SISTEMAS.CAPACITAR O ALUNO A DESENVOLVER SOLUÇÕES COM PROCESSAMENTO DISTRIBUÍDO.
- MOTIVAÇÕES, OBJETIVOS E CARACTERIZAÇÃO DE SISTEMAS DISTRIBUÍDOS: DISTRIBUIÇÃO DOS DADOS E CONTROLE; CLASSIFICAÇÃO; DEFINIÇÃO. 2- A ARQUITETURA DE SISTEMAS DISTRIBUÍDOS: PROCESSOS PARALELOS; ESTRUTURAÇÃO MODULAR E ABSTRAÇÕES; O MODELO DE CAMADAS E INTERFACES. 3- INTERCONECÇÃO FÍSICA: TOPOLOGIA; MEIOS DE TRANSMISSÃO. 4- ASPECTOS DE PROJETO E IMPLEMENTAÇÃO: COMPARTILHAMENTO DE RECURSOS; NOMEAÇÃO E ENDEREÇAMENTO; COMUNICAÇÃO E SINCRONIZAÇÃO ENTRE PROCESSOS; PROTEÇÃO; RECUPERAÇÃODE ERROS; TOLERÂNCIA A FALHAS. 5- PROTOCOLOS E SERVIÇOS. 6- ESPECIFICAÇÃO E VALIDAÇÃO DE PROTOCOLOS.

Ementa

1	Introdução a sistemas distribuídos: objetivos, caracterização, classificação e desafios.
2	Comunicação em Sistemas Distribuídos: arquitetura em camadas, RMC, comunicação por troca de mensagens, disseminação de dados.
3	Gerenciamento de recursos: threads, processos, virtualização, escalabilidade, balanceamento de carga.
4	Coordenação, Sincronização de processos, relógio físico e lógico, exclusão mútua, eleição,
5	Tolerância a falhas, replicação e consistência
6	Modelos de Sistemas Distribuídos: baseados em objetos, serviços. Computação em nuvem.
7	Middleware e ferramentas para sistemas distribuídos, ferramentas para computação distribuída

Objetivos Específicos

Ao final da disciplina de Sistemas Distribuídos (SD) o discente deve ter atingido as seguintes competências, tanto gerais como específicas:

Aspecto **APRENDER**

- CE_Ap_1 Interagir com fontes diretas pela observação da execução de processos comunicantes em ambientes computacionais distribuídos, e análise de seu comportamento em função da arquitetura computacional distribuída, nomes de recursos distribuídos, localização, mobilidade, replicação de recursos de processamento e dados, capacidade de atendimento de transações, latência, ˜largura de banda, disponibilidade, ordenação de tarefas, propagação de mensagens, mecanismo de segurança, mecanismos de comunicação síncronos e assíncronos e paradigmas computacionais distribuídos.
- CE_Ap_2 Interagir com fontes indiretas (os diversos meios de comunicação, divulgação e difusão: abstract, relatórios técnico-científicos, relatos de pesquisa, artigos de periódicos, livros, folhetos, revistas de divulgação, jornais, arquivos, mídia eletro-eletrônica e outras, específicos da comunidade científica ou não) sobre sistemas distribuídos,
- CE_Ap_4 Realizar o duplo movimento de derivar o conhecimento das ações e as ações do conhecimento disponível, de modo que os estudantes serão capazes de abordar diferentes problemas e aptos a proporem diferentes soluções computacionais distribuídas apresentando propostas de soluções estruturadas, organizadas, coerentes e documentadas.

Aspecto **PRODUZIR**

- CE_Pro_2 Planejar procedimentos adequados para encaminhar a resolução de problemas relevantes à sistemas distribuídos proceder com análise de requisitos, conceber solução, simular testes para exequibilidade desenhar tecnicamente a solução e planejar a sua implementação
- CE_Pro_4 Implantar o planejamento realizado de uma resolução através de desenvolvimento de softwares distribuídos
- CE_Pro_5 Relatar e apresentar trabalhos realizados

Aspecto **ATUAR**

- CE_Atuar_1 Dominar conhecimentos e habilidades da área de sistemas distribuídos
- CE_Atuar_2 Dominar conhecimentos e habilidades gerais e básicas de outras áreas onde sistemas distribuídos podem ser incorporados e evoluídos
- CE_Atuar_3 Relacionar conhecimentos e habilidades de diferentes áreas para integração `sistemas distribuídos
- CE_Atuar_4 Extrapolar conhecimentos e habilidades para diferentes situações dentro de seu campo de atuação profissional.

Estratégia de Ensino

As atividades do curso serão compostas de aulas teóricas e de práticas em laboratório.

- Além disso, materiais complementares estarão disponíveis no ambiente virtual de aprendizagem da UFSCar (AVA2). Atendendo as seguintes competências específicas: (CE_Ap_1)(CE_Ap_2)(CE_Ap_4) (CE_Pro_2)
- Nas aulas teóricas, serão tratados conceitos e aspectos dos SD's, bem como estratégias, políticas e mecanismos utilizadas nas diferentes implementações SD. Atendendo as seguintes competências específicas: (CE_Ap_1)(CE_Ap_2)(CE_Ap_4)
- As aulas práticas em laboratório permitirão experiências com o uso de sistemas distribuídos, explorando suas
- diversas características, projetando, desenvolvendo e implementando soluções distribuídas para problemas reais, atendendo as seguintes competências específicas:(CE_Pro_2)(CE_Pro_4)(CE_Pro_5)(CE_Atuar_1)(CE_Atuar_2)(CE_Atuar_3)(CE_Atuar_4)

Atividades dos Alunos

- participar das aulas;
- praticar a execução de programas de SD
- projetar, planejar e implementar os programas computacionais em ambiente SD solicitados, que ilustram (e avaliam)os métodos, técnicas e tecnologias vistas nos tópico do curso, apresentar resultados das soluções em execução e em relatórios
- realizar a avaliação teórica
- Leituras críticas e apresentação de seminários

Avaliação

As avaliações serão compostas por 2 provas teóricas, atendendo as competências específicas (CE_Ap_1)(CE_Ap_2)(CE_Ap_4) pelo entrega e apresentação de 2 trabalhos práticos em SD atendendo as competências
específicas (CE_Pro_2)(CE_Pro_4)(CE_Pro_5)(CE_Atuar_1)(CE_Atuar_2)(CE_Atuar_3)(CE_Atuar_4)

A média final dos estudantes será baseada na seguinte ponderação:

a) 60% referente à média das provas - P1 e P2 - 1h40 de duração cada uma
b) 40% referente às notas dos trabalhos práticos - totalizando aproximadamente 30 horas de dedicação ao longo do semestre em atividades de projeto, questionários e seminários.

Os Trabalhos Práticos poderão ser constituídos de:

a) Implementações de programas e protótipos de sistemas distribuídos;

b) Seminários, análises e resumos críticos sobre material e assuntos específicos em sistemas distribuídos;

c) listas de exercícios;

Será aprovado o aluno que obtiver média final igual ou superior a 6,0.

Observação: Não haverá prova substitutiva.

- Serão aprovados os estudantes que obtiverem Média Final >= 6.0 e Frequência >= 75 por cento (Conforme Portaria 522/06 Art. 14 do Regimento Geral dos Cursos de Graduação da UFSCar, os estudantes que obtiverem Média Final maior ou igual a 5.0 e menor que 6.0 e Frequência mínima de 75 por cento terão direito a uma Avaliação Complementar a ser realizada no início do período letivo seguinte (determinado pela Universidade). A
- avaliação será oferecida a todos os estudantes que atingiram a nota mínima em uma única ocasião e, portanto, o não comparecimento do aluno implicará em sua reprovação. Nesse caso, o estudante que obtiver nota igual ou superior a 6 ficará com a Média Final igual a 6.
- Estará automaticamente reprovado, com nota final 0,0 (zero), o aluno que, em qualquer dos trabalhos ou provas, apresentar evidências que tenha plagiado/copiado/colado em provas e outras atividades, quer seja de colegas, de material disponível na rede, de livros, ou qualquer outra fonte.

Datas das avaliações

P1: 01/06/2023 - Tópicos 1 e 2

P2: 29/06/2023 - Tópicos 3 e 4

T1: entrega até 29/06/2023

P3: 27/07/2022 - Tópicos 5 e 6

P4: 24/08/2022 - Tópicos 7 e 8

T2: entrega até 31/08/2023

Seminários: 24 e 31/08/2023

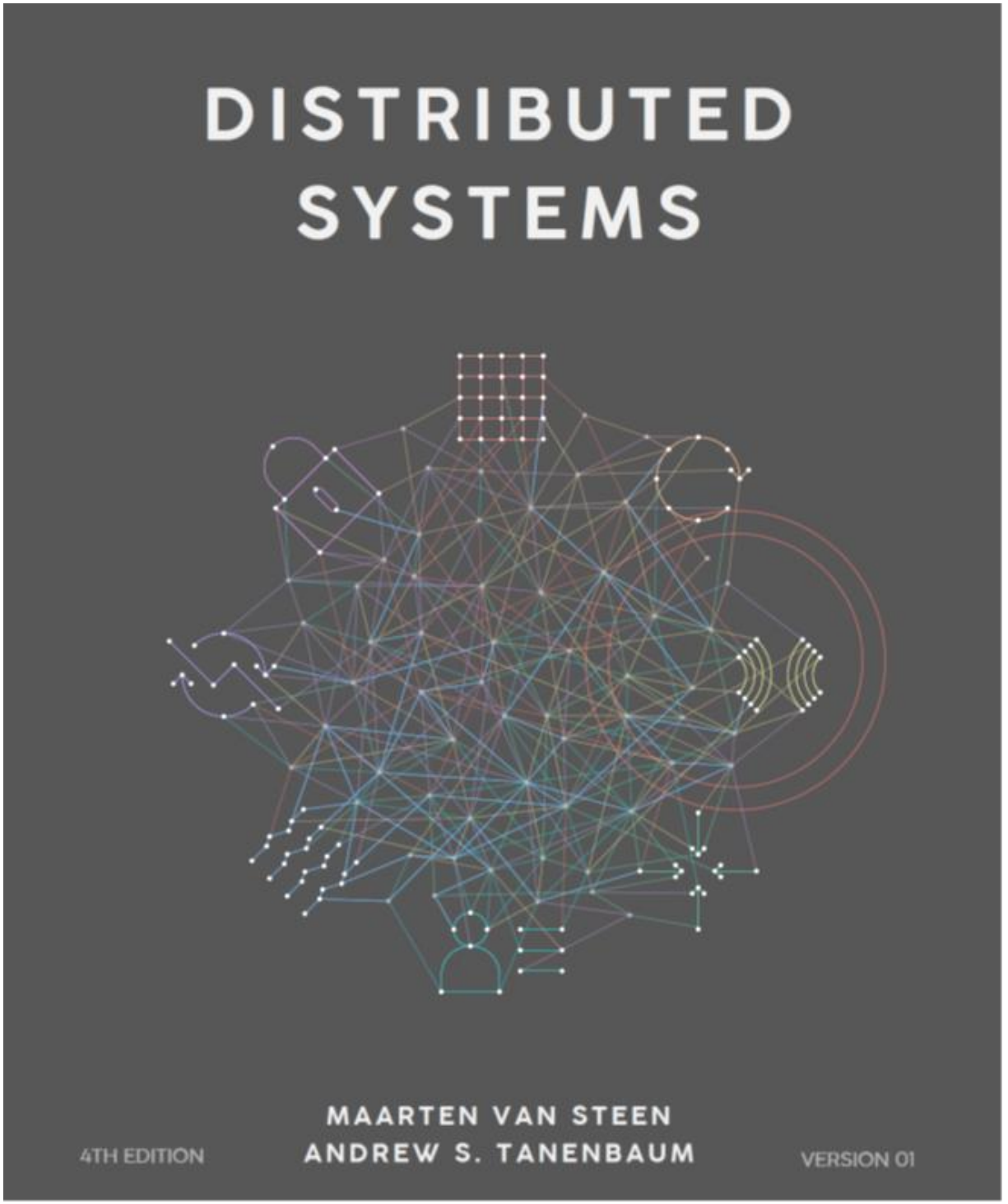
NF = 0,6*((P1+P2+P3+P4)/2) + 0,4*((T1+T2+Seminários)/3)

Bibliografia

1. Maarten van Steen, Andrew S. Tanenbaum, "Distributed Systems - 4th Edition", versão 4.01 Janeiro 2023, - disponível digitalmente - https://www.distributed-systems.net/index.php/books/ds4/
2. Tanenbaum, A. S.; Van Steen, M. "Sistemas Distribuidos: Principios E Paradigmas". Prentice Hall, 2007. Coulouris, G.; Dollimore, J. and Kindberg, T. Sistemas Distribuídos: Conceitos e Projeto. Bookman, 2007. – 1 exemplar disponível
3. Sinha, P. K. Distributed Operating systems: Concepts and Design, IEEE Computer Society Press, 1997 - 01 exemplar disponível
4. Van Steen, M. e Tanenbaum, A. *Distributed Systems 3rd edition*. Editora: Pearson 2017 - ISBN: 978-15-430573-8-6 (printed version) ISBN: 978-90-815406-2-9 (digital version) C. CACHIN, R. GUERRAOUI, L. RODRIGUES. *Introduction to Reliable and Secure Distributed Programming*, 2nd edition. Ed. Springer, 2006. ISBN 978-3-642-15259-7 e-ISBN 978-3-642-15260-3, DOI 10.1007/978-3-642-15260-3
5. Lampson, B.W.& Paul, M. & Siegert, H.J., Distributed Systems: architecture and implementation - an advanced course, Springer, Berlin, 1983.

Conhecimentos Desejados

- Necessários – Conceitos de Sistemas Operacionais e princípios, conceitos de arquiteturas de computadores
- Altamente desejado - entendimento de redes de Computadores e protocolos de rede
- Necessários: habilidades de programação – C, C++, Python


DISTRIBUTED
SYSTEMS
MAARTEN VAN STEEN
ANDREW S. TANENBAUM
4TH EDITION
VERSION 01

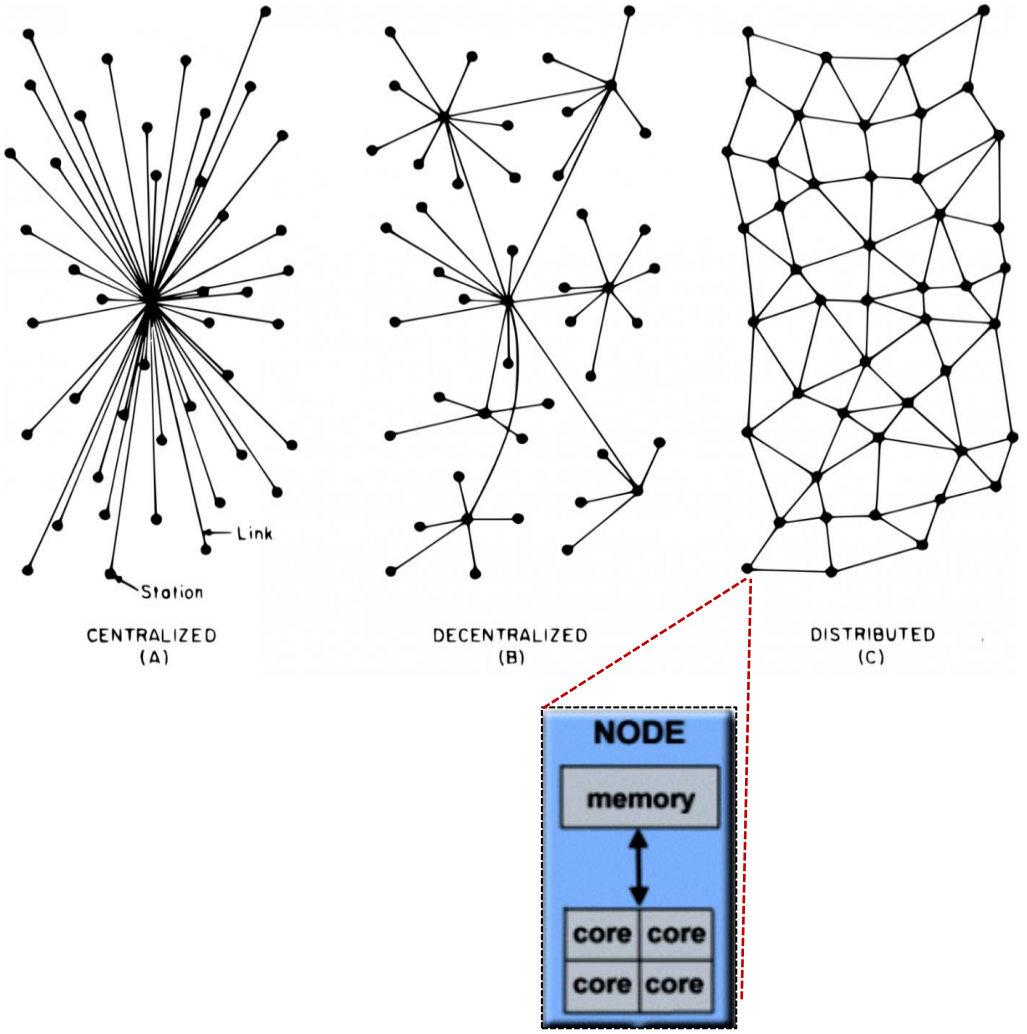
Capítulos

1. Introdução
2. Arquiteturas
3. Processos
4. Comunicação
5. Nomes
6. Coordenação
7. Consistência e Replicação
8. Tolerância a Falhas
9. Segurança

CAPÍTULO 1 INTRODUÇÃO

O QUE É UM SISTEMA DISTRIBUÍDO ?

- **Definição:**
  - Um sistema distribuído é uma coleção de elementos computacionais autônomos que aparece para seus usuários como um sistema coerente único.
- **Propriedades Características:**
  - Elementos computacionais autônomos, também conhecidos por *"nodes"* (nós), são dispositivos de hardware ou processos de software.
  - Sistema coerente único: usuários ou aplicações percebem (enxergam) um sistema único → nós (nodes) necessitam colaboração.



INTRODUÇÃO: O QUE É UM SISTEMA DISTRIBUÍDO

COLEÇÃO DE NÓS AUTÔNOMOS

- **Comportamento Independente:**
  - Cada nó é autônomo e portanto possuí sua própria noção de tempo: não existe relógio global. Leva a problemas fundamentais de sincronização e coordenação.
- **Coleção de nós:**
  - Como gerenciar associação ao grupo *(group membership)* ?
  - Como saber se você está de fato se comunicando com um membro (não) autorizado de um grupo ?

CARACTERÍSTICA 1 COLEÇÃO DE NÓS AUTÔNOMOS

ORGANIZAÇÃO

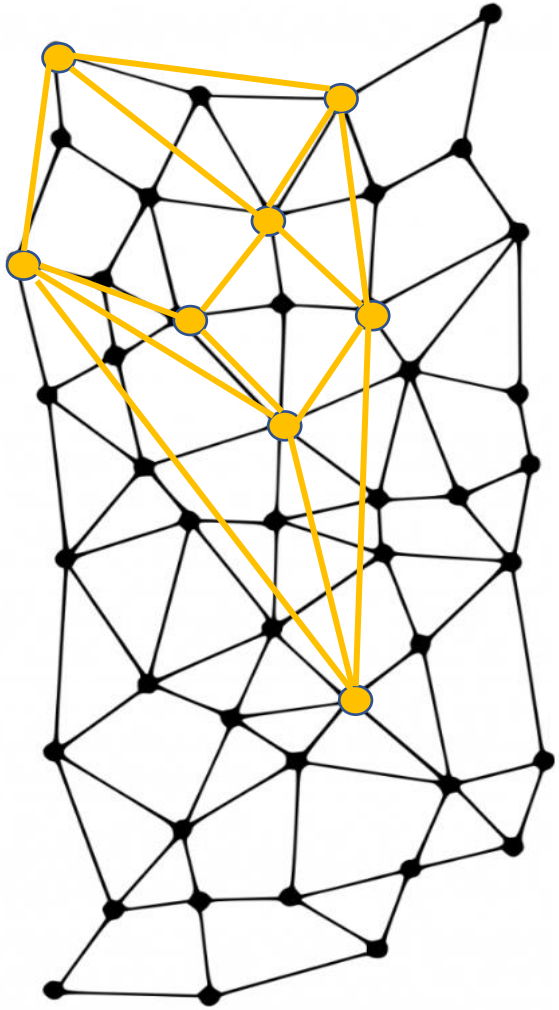
Rede Sobreposta (*Overlay Network*):

Cada nó da coleção se comunica somente com outros nós no sistema, seus vizinhos. O conjunto de vizinhos pode ser dinâmico, ou pode ainda ser conhecido somente de forma implícita (isto é, demanda *lookup*).

Tipos de redes *Overlay*:

Um exemplo bem conhecido de redes sobrepostas: sistemas *peer-to-peer*.

- **Estruturada**: cada nó tem um conjunto de vizinhos bem definidos com os quais ele (nó) pode se comunicar;
- **Não estruturada**: cada nó possuí referências para outros nós selecionados randomicamente de nós do sistema



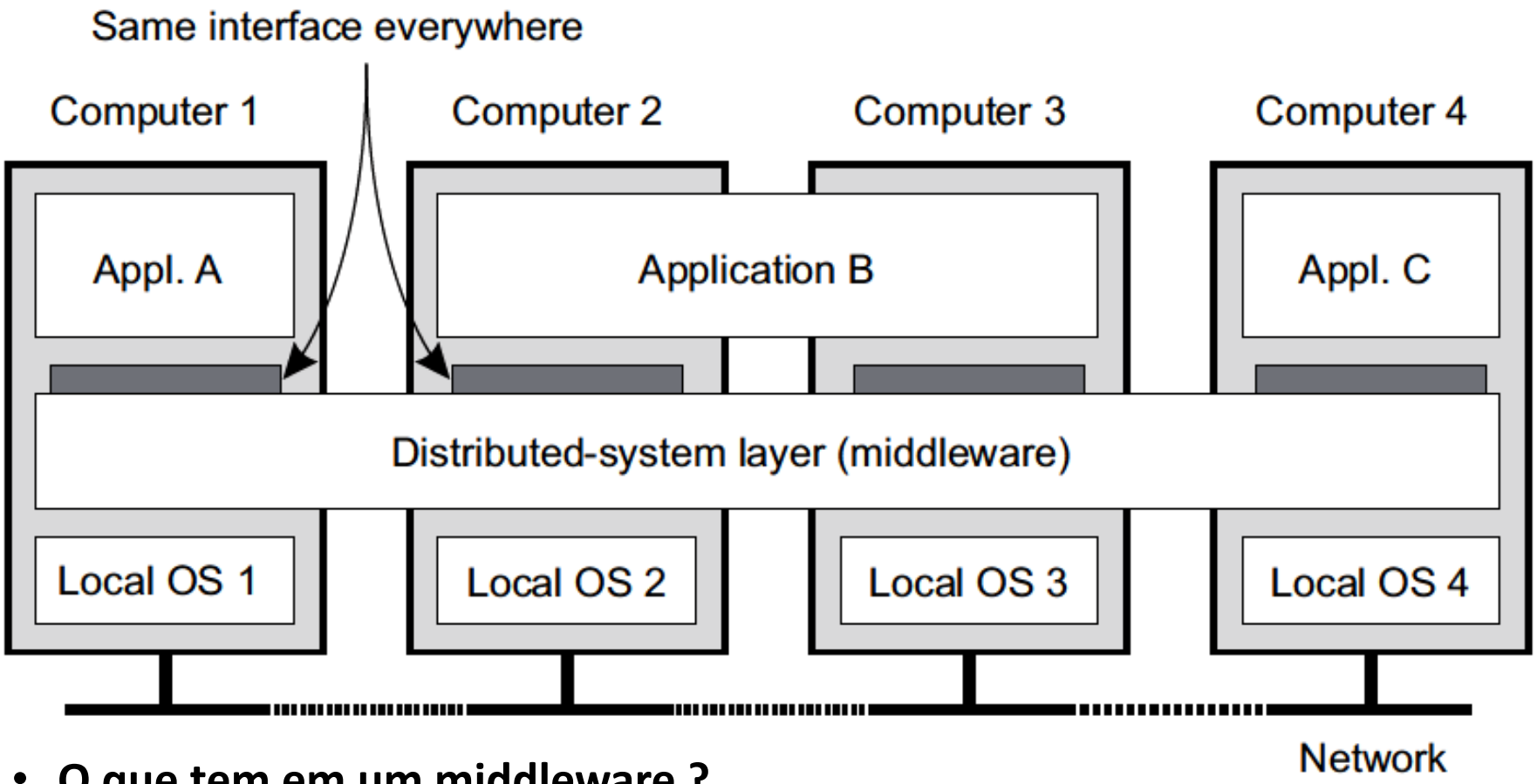
CARACTERÍSTICA 2 SISTEMA COERENTE ÚNICO

ORGANIZAÇÃO

- **Essência**
  - Uma coleção de nós como um todo opera o mesmo, independente de onde, quando e como a interação entre usuárioe sistema ocorre.
- **Exemplos**
  - Um usuário final não é capaz de dizer onde a computação está ocorrendo;
  - Onde exatamente os dados são armazenados deveria ser irrelevante para a aplicação;
  - Se os dados foram ou não replicados está completamente escondido;
- **O problema: falhas parciais**
  - É inevitável a qualquer hora somente uma parte de um sistema distribuído possa falhar. Esconder falhas parciais e recuperá-las é frequentemente difícil e em geral impossível de esconder.

MIDDLEWARE O SO DE SISTEMAS DISTRIBUÍDOS

MIDDLEWARE E SISTEMAS DISTRIBUÍDOS



- **O que tem em um middleware ?**
  - Componentes e funções de uso comum que não precisam ser implementadas nas aplicações de forma separada.

OBJETIVOS DO DESENHO (DESIGN)

O que buscamos alcançar ?

- **Suporte ao compartilhamento de recursos**
- **Transparência na distribuição**
- **Abertura (openness)**
- **Escalabilidade (scalability)**

COMPARTILHANDO RECURSOS

- **Exemplos Canônicos (de acordo com as normas)**
  - Armazenamento de arquivos compartilhados baseado em núvem
  - Sistema de *streaming* multimídia *peer-to-peer* assistido
  - Serviços de email compartilhado (pense em serviços terceirizados)
  - Compartilhamento de Web Hosting (pense em redes de distribuição de conteúdos)

- **Observação**
  - " A rede é o computador" (John Cage, então na Sun Microsystems)

TRANSPARÊNCIA NA DISTRIBUIÇÃO

TIPOS

Transparência	Descrição
Acesso	Esconde diferenças em representação de dados e como um objeto é acessado
Localidade	Esconde a localização do objeto
Relocação	Esconde que um objeto pode s mover para outra localização enquanto em uso
Migração	Esconde que um objeto pode se mover para outra localização
Replicação	Esconde que um objeto é replicado
Concorrência	Esconde que um objeto pode ser compartilhado por vários usuários independentes
Falha	Esconde falhas e recuperações de um objeto

GRAU DE TRANSPARÊNCIA

Observação

Transparência total pode ser um exagero:

- Existe latências de comunicação que não podem ser escondidas
- Esconder falhas por completo de redes e nós (teoricamente e praticamente) é impossível
  - Não dá para distinguir um computador lento de um que está falhando
  - Não dá para ter certeza que um servidor realizou uma operação antes de uma falha
- Transparência total degrada o desempenho (custo), expondo a distribuição do sistema
  - Manter replicas atualizadas de forma exata com a cópia mestre gasta tempo
  - Executar imediatamente operações de gravação no disco para tolerância a falhas

GRAU DE TRANSPARÊNCIA

Expor distribuição pode ser bom

- Fazer uso de serviços locais (achar amigos próximos)
- Ao lidar com usuários em diferentes fusos horários
- Quando fica mais fácil para um usuário entender o que está acontecendo (quando por exemplo, um servidor não responde por um longo tempo, relate-o como falho).

Conclusão

- Transparência na distribuição é um objetivo legal, mas conseguir é uma história diferente, e muitas vezes não deve ser mirado ..

ABERTURA DE SISTEMAS DISTRÍBUIDOS

Do que estamos falando ?

- Ter a capacidade de interagir com serviços e outros sistemas abertos, independente do ambiente de baixo
  - Sistemas devem estar em conformidade com interfaces bem definidas
  - Sistemas devem interoperar facilmente
  - Sistemas devem suportar portabilidade de aplicações
  - Sistemas devem ser facilmente extensíveis

POLÍTICAS X MECANISMOS

IMPLEMENTANDO ABERTURA: POLÍTICAS

- Qual nível de consistência precisamos para os dados armazenados em cache do cliente?
- Quais operações permitimos que o código baixado execute?
- Quais requisitos de QoS são ajustados em função de variação na largura de banda ?
- Qual nível de segurança de dados são necessários na comunicação ?

IMPLEMENTANDO ABERTURA: MECANISMOS

- Permitir o ajuste (dinâmico) de polítcas de cache
- Suportar diferentes níveis de confiança (trust) para códigos móveis
- Oferecer diferentes algoritmos de encriptação

SEPARAÇÃO ESTRITA

OBSERVAÇÃO

- Quanto mais rigorosa for a separação entre política e mecanismo, mais precisamos para garantir mecanismos adequados, levando potencialmente a muitos parâmetros de configurações e gerenciamento complexo.

ACHANDO EQUILÍBRIO

- Políticas de codificação rígida simplificam o gerenciamento e reduzem a complexidade ao preço de menos flexibilidade. Não há solução óbvia!

ESCALA EM SISTEMAS DISTRIBUÍDOS

OBSERVAÇÃO

- Muitos desenvolvedores de sistemas distribuídos modernos facilmente usam o adjetivo escalável sem deixar claro porque seus sistemas escalam.

PELO MENOS TRÊS COMPONENTES

- Número máximo de usuários e/ou processos (escalabilidade de tamanho)
- Máxima distância entre nós (escalabilidade geográfica)
- Número de domínios administrativos (escalabilidade administrativa)

OBSERVAÇÃO

- Muitos sistemas contabilizam somente, até um certo ponto, escalabilidade de tamanho. Frequentemente usando a solução: multiplos servidores poderosos operando independentemente em paralelo. Hoje, o desafio está na escalabilidade geográfica e administrativa.

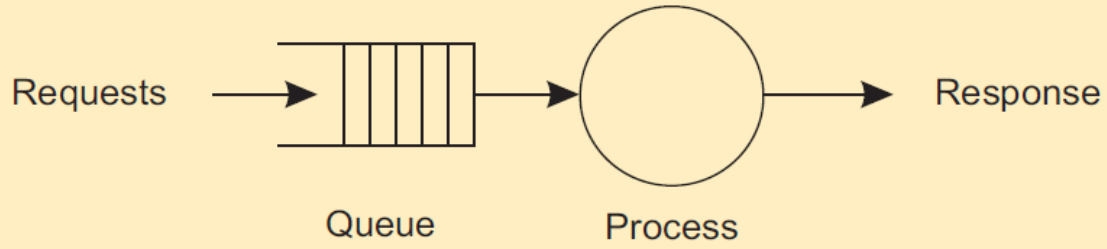
ESCALABILIDADE DE TAMANHO

CAUSAS RAIZ PARA PROBLEMAS DE ESCALABILIDADE COM SOLUÇÕES CENTRALIZADAS

- Capacidade computacional limitada por CPU´s
- Capacidade de armazenamento incluíndo taxa de transferência entre CPU´s e discos
- A rede entre usuário e o serviço centralizado

ANÁLISE FORMAL

UM SERVIÇO CENTRALIZADO PODE SER MODELADO COMO UM SISTEMA DE FILA SIMPLES



SUPOSIÇÕES E NOTAÇÕES

- A fila tem tamanho infinito → a taxa de chegada de requisições não é influenciada pelo tamanho atual da fila ou o que está sendo processado
- Taxa de chegada de requisições: λ requisições/seg
- Capacidade do processamento do serviço: μ requisições/seg

FRAÇÃO DE TEMPO PARA *k* REQUISIÇÕES NO SISTEMA

$$p_k = (1 - \frac{\lambda}{\mu})(\frac{\lambda}{\mu})^k$$

ANÁLISE FORMAL

UTILIZAÇÃO *U* DE UM SERVIÇO É A FRAÇÃO DE TEMPO QUE ELE ESTÁ OCUPADO

$$U = \sum_{k>0} p_k = 1 - p_0 = \frac{\lambda}{\mu} \Rightarrow p_k = (1-U)U^k$$

NÚMERO MÉDIO DE REQUISIÇÕES EM UM SISTEMA

$$\overline{N} = \sum_{k\geq 0} k \cdot p_k = \sum_{k\geq 0} k \cdot (1-U)U^k = (1-U)\sum_{k\geq 0} k \cdot U^k = \frac{(1-U)U}{(1-U)^2} = \frac{U}{1-U}$$

VAZÃO MÉDIA

$$X = \underbrace{U \cdot \mu}_{\text{server at work}} + \underbrace{(1-U)\cdot 0}_{\text{server idle}} = \frac{\lambda}{\mu} \cdot \mu = \lambda$$

ANÁLISE FORMAL

TEMPO DE RESPOSTA: TEMPO TOTAL PARA PROCESSAR UMA REQUISIÇÃO DEPOIS DA SUBMISSÃO

$$R = \frac{\overline{N}}{X} = \frac{S}{1-U} \Rightarrow \frac{R}{S} = \frac{1}{1-U}$$

- Onde S = $1/\mu$ sendo o tempo do serviço

OBSERVAÇÕES

- Se *U* é pequeno, a resposta para o tempo de serviço é perto de 1: uma requisição é processada imediatamente
- Se *U* cresce até 1, o sistema vai para travamento. Solução: diminúa S

PROBLEMAS COM ESCALABILIDADE GEOGRÁFICA

ESCONDER LATÊNCIAS DE COMUNICAÇÃO

- Não pode simplesmente mudar de LAN para WAN: muitos sistemas distribuídos assumem interações síncronas entre cliente e servidor: cliente envia requisição e espera por resposta: Latência pode facilmente proibir este esquema.
- Links WAN frequentemente são inerentemente não confiáveis: mudar um stream de vídeos de LAN para WAN é propenso a falhar.
- Falta de comunicação multiponto, de forma que um broadcast de busca simples não pode ser disseminado. A solução é desenvolver de forma separada, serviços de nomes e diretórios (com seu próprio problema de escalabilidade)

PROBLEMAS COM ESCALABILIDADE GEOGRÁFICA

ESSÊNCIA

- Políticas conflitantes em relação ao uso (e sobre pagamento), gerenciamento e segurança.

EXEMPLOS

- Grades (grids) computacionais: compartilha recursos caros entre diferentes domínios
- Equipamentos compartilhados: como controlar, gerenciar, e usar um rádio telescópio compartilhado construído como uma de sensores de grande escala ?

EXCEÇÃO: MUITAS REDES PEER-TO-PEER

- Sistema de compartilhamento de arquivos (baseados p. ex. no BitTorrent)
- Telefonia peer-to-peer (Skype)
- Streaming de áudio assistido por “peer” (Spotify)

Nota: usuários finais colaboram e não entidades administrativas

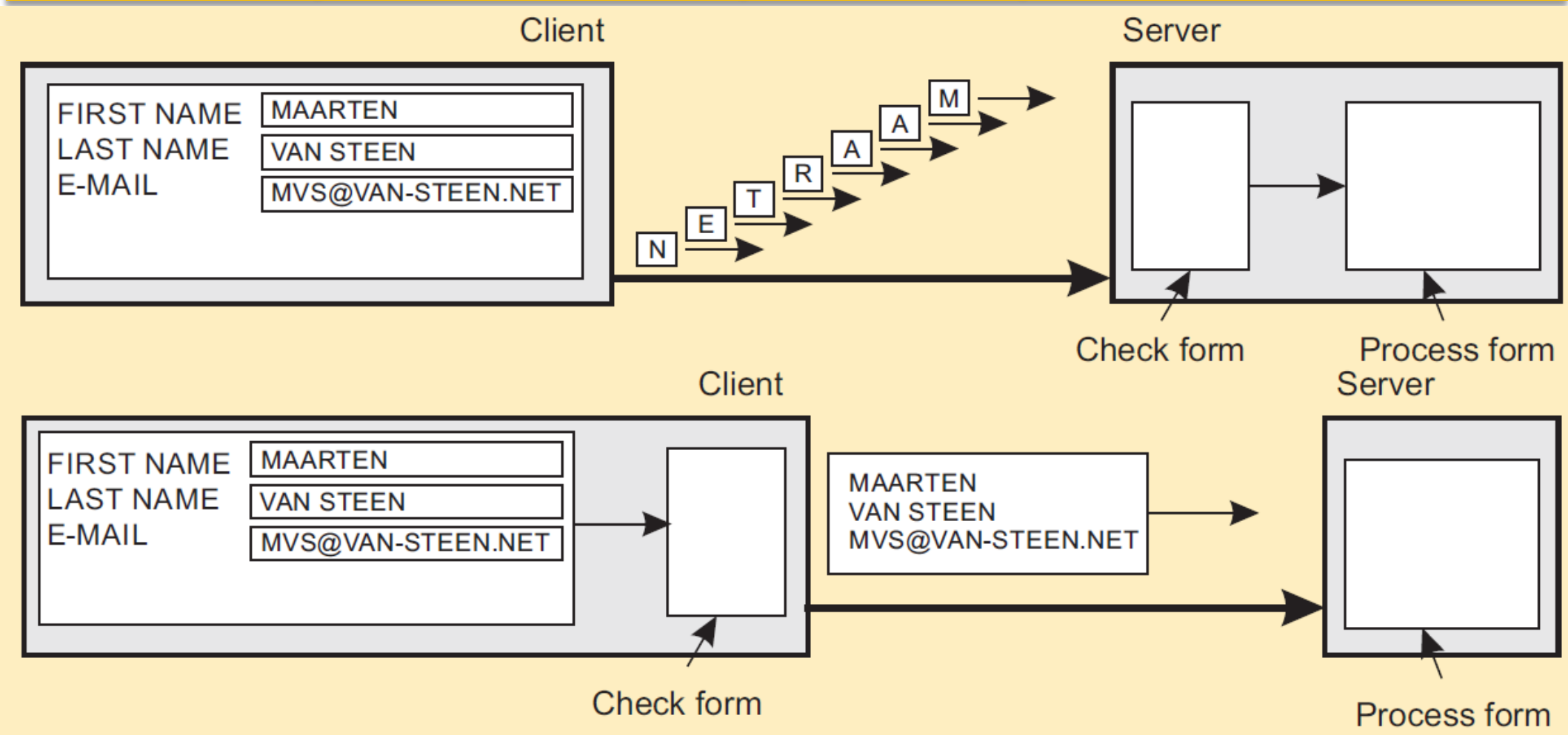
TÉCNICAS PARA ESCALABILIDADE

ESCONDER LATÊNCIAS DE COMUNICAÇÃO

- Fazer uso de comunicação assíncrona
- Ter um manipulador (handler) separado para respostas que estão chegando
- Problema: nem sempre todas as aplicações cabem neste modelo

TÉCNICAS PARA ESCALABILIDADE

FACILITAR A SOLUÇÃO ATRAVÉS DA MUDANÇA DA COMPUTAÇÃO PARA O CLIENTE



TÉCNICAS PARA ESCALABILIDADE

PARTIÇÃO DE COMPUTAÇÃO E DADOS ATRAVÉS DE MULTIPLAS MÁQUINAS

- Mover a computação para clientes (Java applets)
- Descentralização de serviços de nomes (DNS)
- Descentralização de sistemas de informação (www)

TÉCNICAS PARA ESCALABILIDADE

REPLICAÇÃO E CACHING: DISPONIBILIZA CÓPIAS DE DADOS EM DIFERENTES MÁQUINAS

- Replicação de servidores de arquivos e base de dados
- Sites WEB espelhados
- Cache WEB (nos browsers e proxies)
- Cache de arquivos (no servidor e no cliente)

ESCALABILIDADE PROBLEMA DE REPLICAÇÃO

APLICAR REPLICAÇÃO É FÁCIL, EXCETO POR UMA COISA

- A existência de múltiplas cópias (cache ou replicadas), leva a inconsistências: modificar uma cópia torna-a diferente de todas as outras,
- Manter cópias consistentes sempre de uma forma geral demanda sincronização global em cada modificação.
- Sincronização global impede soluções de larga escala

OBSERVAÇÃO

- Se pudermos tolerar inconsistências, podemos reduzir a necessidade de sincronização global, porém a tolerância de inconsistências é dependente da aplicação.

DESENVOLVIMENTO DE SISTEMAS DISTRIBUÍDOS

PITFALLS

OBSERVAÇÃO

- Muitos sistemas distribuídos são desnecessariamente complexos devido a erros que demandaram correções (*patches* – remendos). Muitas suposições falsas são feitas com frequência.

SUPOSIÇÕES FALSAS (E FREQUENTEMENTE ESCONDIDAS)

- A rede é confiável
- A rede é segura
- A rede é homogênea
- A topologia não muda
- Latência é zero
- Largura de banda é infinita
- Custo de transporte é zero
- Existe um administrador

TRÊS TIPOS DE SISTEMAS DISTRIBUÍDOS

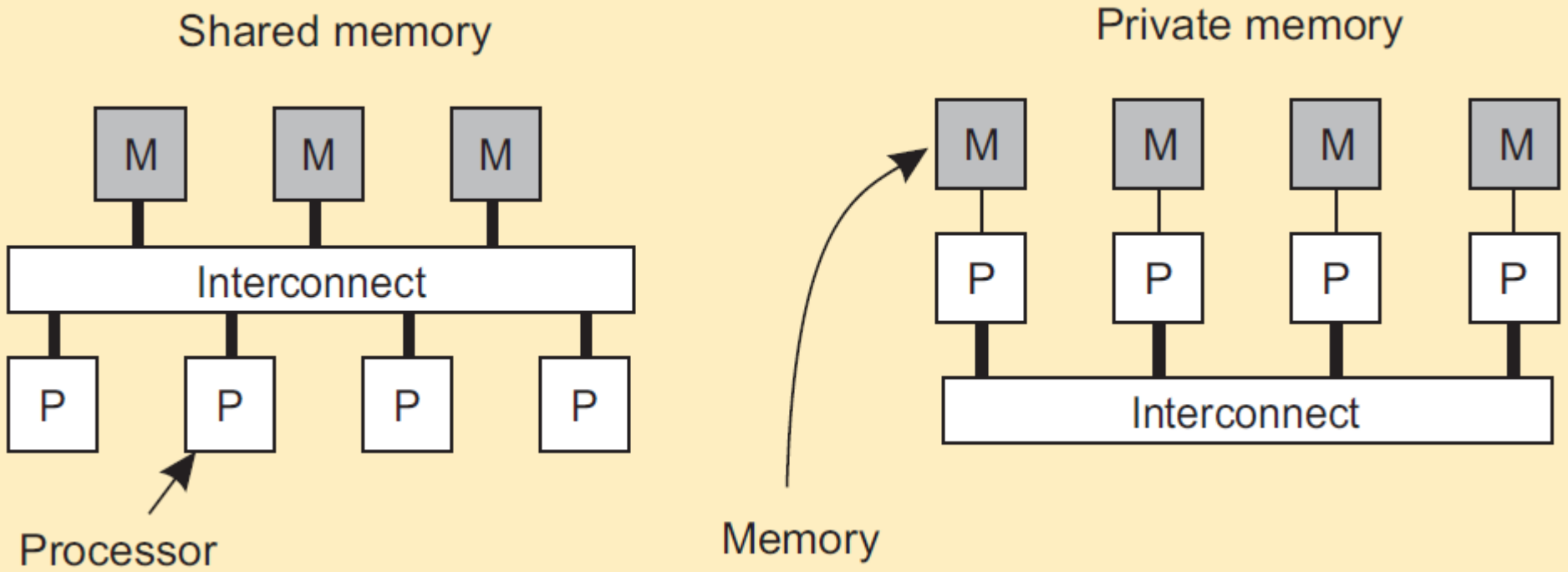
- Sistemas Computacionais Distribuídos de Alto Desempenho
- Sistemas de Informações Distribuídos
- Sistemas Distribuídos para Computação Pervasiva

COMPUTAÇÃO PARALELA

OBSERVAÇÃO

- Computação Distribuída de Alto Desempenho começou com Computação Paralela

MULTIPROCESSADOR MULTINÚCLEO X MULTICOMPUTADOR



SISTEMA DISTRIBUÍDO MEMÓRIA COMPARTILHADA

OBSERVAÇÃO

Multiprocessadores são relativamente fáceis de programar em comparação a multi-computadores, porém mostram problemas quando o número de processadores (ou núcleos) aumenta. Solução: tentar implementar um modelo de memória compartilhada no topo de um multi-computador.

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 o processador *A* endereça uma página *P* localizada no processador *B*, o SO em *A* *traps* e busca *P* de *B*, da mesma forma que iria se *P* estivesse localizado em um disco local.

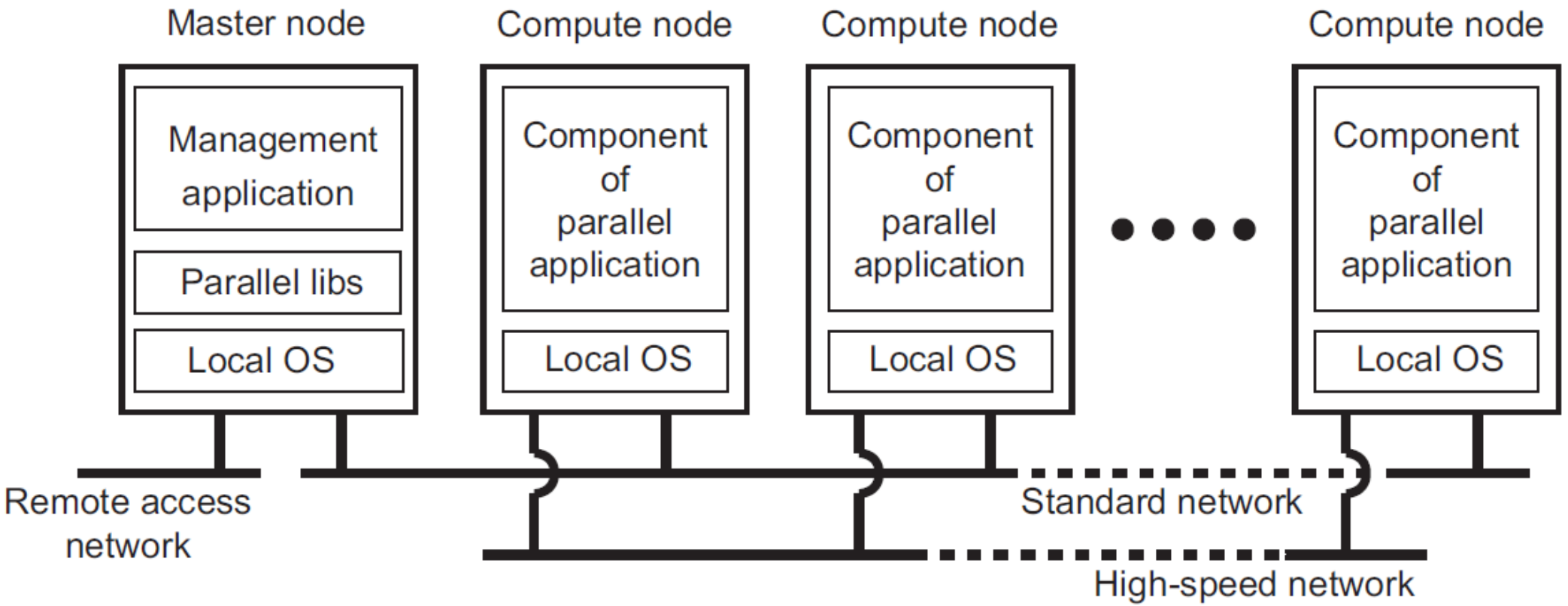
PROBLEMA

Desempenho de sistemas distribuídos com memória compartilhada nunca poderiam competir com multiprocessadores, e falhou nas expectativas de programadores. Tem sido abandonado desde então.

COMPUTAÇÃO EM CLUSTER

ESSENCIALMENTE UM GRUPO DE SISTEMAS HIGH-END CONECTADOS POR UMA LAN

- Homogêneo: mesmo SO, hardware quase idêntico
- Único nó de gerenciamento



COMPUTAÇÃO EM GRADE (GRID)

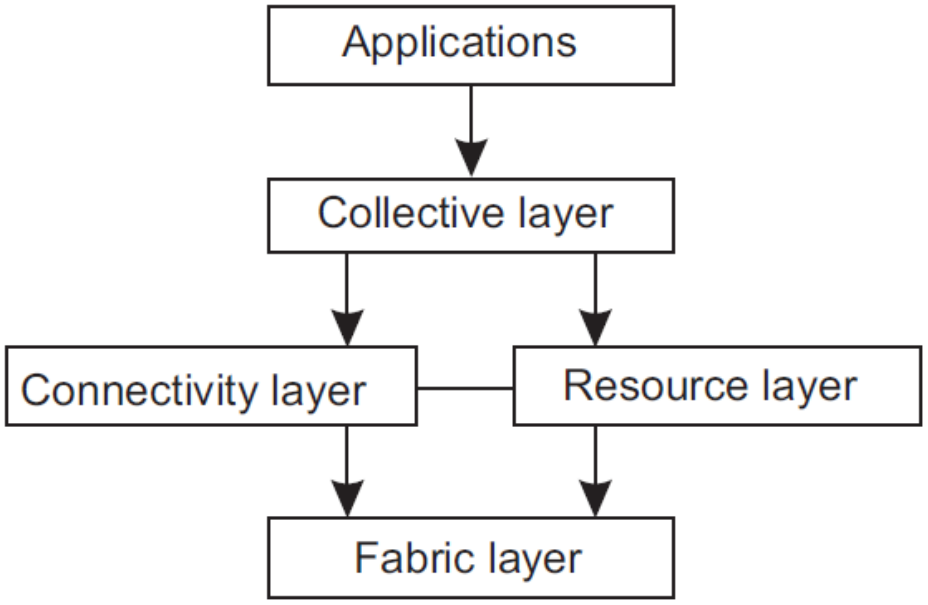
O PRÓXIMO PASSO: MUITOS NÓS DE TODOS LUGARES

- Heterogêneo
- Disperso através de diversas organizações
- Pode facilmente se espalhar por uma rede WAN

NOTA

Para permitir colaboração, grids geralmente usam organizações virtuais. Em essência, isto é um agrupamento de usuários (ou melhor: seus ID´s) que irão permitir a autorização na alocação de recursos.

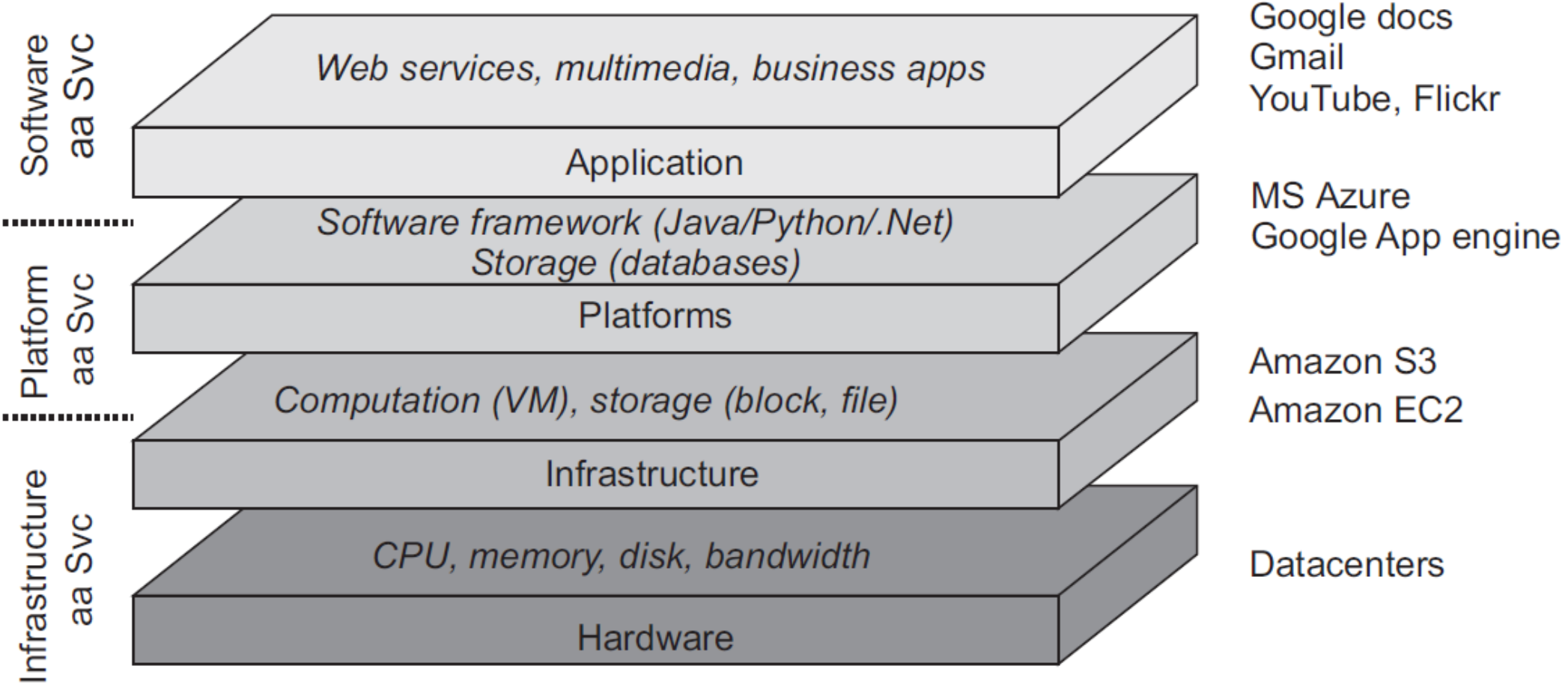
ARQUITETURA GRID COMPUTING



AS CAMADAS

- Fabrica: provê interfaces para recursos locais (para "querying" do estado e capacidades, locking etc..)
- Conectividade: protocolos de comunicação / transação, p.ex., para mover dados entre recursos. Presença de vários protocolos de autenticação.
- Recursos: Gerencia um único recurso, tal como a criação de processo ou leitura de dados.
- Coletivo: manuseia o acesso a múltiplos recursos: descoberta, escalonamento e replicação.
- Aplicação: contém as aplicações do grid em uma única organização

COMPUTAÇÃO EM NÚVEM (CLOUD)



COMPUTAÇÃO EM NÚVEM (CLOUD)

POSSUÍ 4 CAMADAS BEM DISTINTAS

- Hardware: processadores, roteadores, fontes de energia e refrigeração. Clientes nunca podem ver.
- Infraestrutura: dissemina técnicas de virtualização. Evolui ao redor da alocação e gerenciamento de dispositivos virtuais de armazenamento e servidores virtuais.
- Plataforma: provê abstrações de alto nível para armazenamento e etc. Exemplo: Sistema de armazenamento Amazon S3 oferece uma API para (criada localmente) arquivos a serem organizados e armazenados em um esquema chamado Buckets
- Aplicação: aplicações atuais tais como office suítes (processadores de texto, aplicações planilhas, aplicações de apresentação). Comparável a suíte de apps despachado junto com os SO´s

COMPUTAÇÃO EM NÚVEM É CUSTO EFICIENTE ?

OBSERVAÇÃO

Uma razão importante para o sucesso de computação em núvem é que ela permite a organização terceirizar sua infraestrutura de TI: hardware e software. Questão essencial: a terceirização é mais barata ?

ABORDAGEM

- Considere uma aplicação empresarial, modelada como uma coleção de componentes, onde cada componente *Ci* requisita um servidor *Ni.*
- Aplicação se torna um grafo direto, com o vértice representando um componente, e um arco $\langle \overrightarrow{i,j} \rangle$ presentando dados fluindo de *Ci* para *Cj*.
- Dois pesos associados por arco:
  - *Ti,j* é o número de transações por unidade de tempo que causa o fluxo de dados de *Ci* para *Cj*.
  - *Si,j* é a quantia total de dados associados com *Ti,j*

COMPUTAÇÃO EM NÚVEM É CUSTO EFICIENTE ?

PLANO DE MIGRAÇÃO

Descubra para cada componente *Ci*, quantos *ni* de seus servidores *Ni* devem migrar, de forma que os benefícios monetários reduzidos por custos adicionais para comunicação Internet, são máximos.

REQUISITOS DO PLANO DE MIGRAÇÃO

1. Regras e políticas são atendidas
2. Latências adicionais não violam limites de atrasos específicos
3. Todas as transações continuam a operar corretamente; requisições ou dados não são perdidos durante a transação.

BENEFÍCIOS COMPUTACIONAIS

ECONOMIAS (MONETÁRIAS)

- ***Bc***: benefícios da migração de um componente intensivo em computação
- ***Mc***: número total de componentes intensivos em computação migrados
- ***Bs***: benefícios da migração de componentes intensivos em armazenamento
- ***Ms***: número total de componentes intensivos em armazenamento migrados
- Obviamente, o total de benefícios é: ***Bc.Mc + Bs.Ms***

CUSTOS INTERNET

TRÁFEGO DE / PARA NÚVEM

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

- $T_{user,i}$: Transação por unidade de tempo que causa fluxo de dados para *Ci*
- $S_{user,i}$: quantidade de dados associados com $T_{user,i}$

TAXA DE TRANSAÇÕES APÓS MIGRAÇÃO

ALGUMAS ANOTAÇÕES

- *Ci,local*: conjunto de servidores de *Ci* que continuam no local
- *Ci,cloud*: conjunto de servidores de *ci* que estão na núvem
- Assuma que a distribuição de tráfego é a mesma para servidores em núvem e locais

Note que |*Ci,cloud*| = *ni*. Se *fi* = *ni/Ni*, e *si* um servidor de *Ci*

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

INTERNET CUSTOS TOTAIS

ALGUMAS ANOTAÇÕES

- *Custo local,inet*: custo internet por unidade para a parte local
- *Custo cloud,inet*: custo internet por unidade para núvem

CUSTOS E TRÁFEGO: ANTES E DEPOIS DA MIGRAÇÃO

$$Tr^*_{local,inet} = \sum_{C_{i,local}, C_{j,local}} (T^*_{i,j} S^*_{i,j} + T^*_{j,i} S^*_{j,i}) + \sum_{C_{j,local}} (T^*_{user,j} S^*_{user,j} + T^*_{j,user} S^*_{j,user})$$

$$Tr^*_{cloud,inet} = \sum_{C_{i,cloud}, C_{j,cloud}} (T^*_{i,j} S^*_{i,j} + T^*_{j,i} S^*_{j,i}) + \sum_{C_{j,cloud}} (T^*_{user,j} S^*_{user,j} + T^*_{j,user} S^*_{j,user})$$

$$costs = cost_{local,inet} (Tr^*_{local,inet} - Tr_{local,inet}) + cost_{cloud,inet} Tr^*_{cloud,inet}$$

INTEGRANDO APLICAÇÕES

SITUAÇÃO

Organizações usam muitas aplicações em rede, porém interoperabilidade é difícil de ser realizada

ABORDAGEM BÁSICA

Uma aplicação em rede que roda em um servidor disponibilizando serviços para clientes remotos: Integração: clientes combinam requisições para diferentes aplicações, despacham as requisições; coletam respostas e apresentam um resultado coerente para os usuários.

PRÓXIMO PASSO

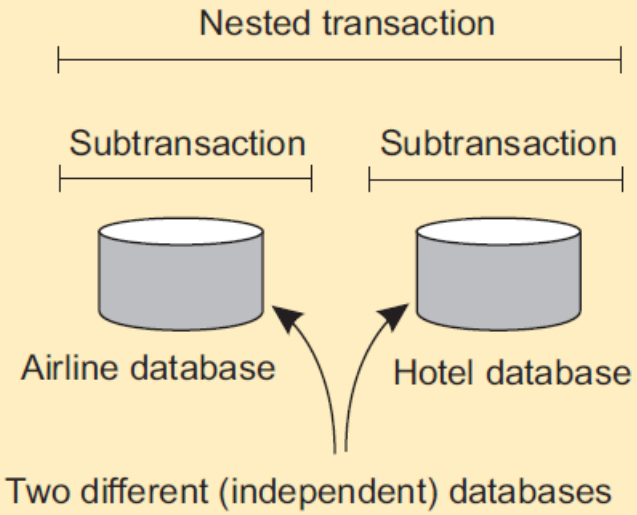
Permitir comunicação direta aplicação-para-aplicação, levando a EAI – *Enterprise Application Integration*

EXEMPLO EAI: TRANSAÇÕES (ANINHADAS)

TRANSAÇÃO

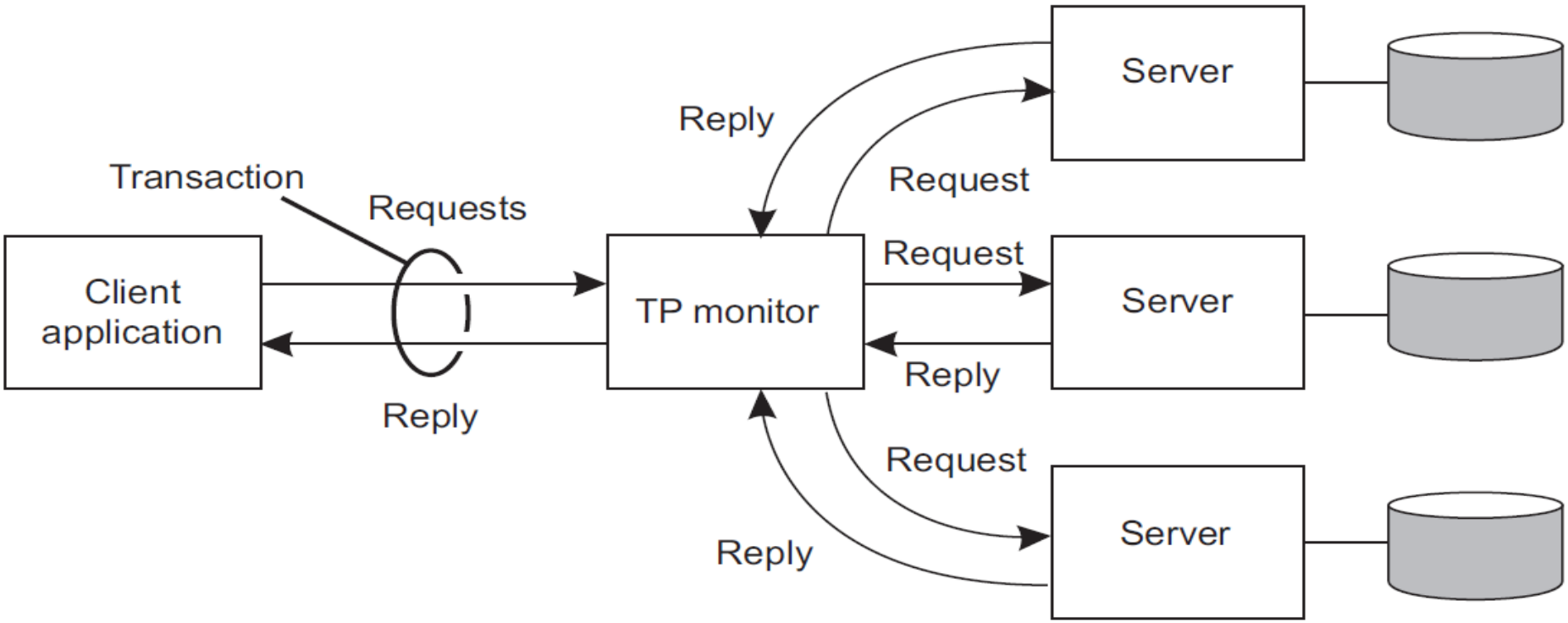
Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

QUESTÃO: TUDO-OU-NADA



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

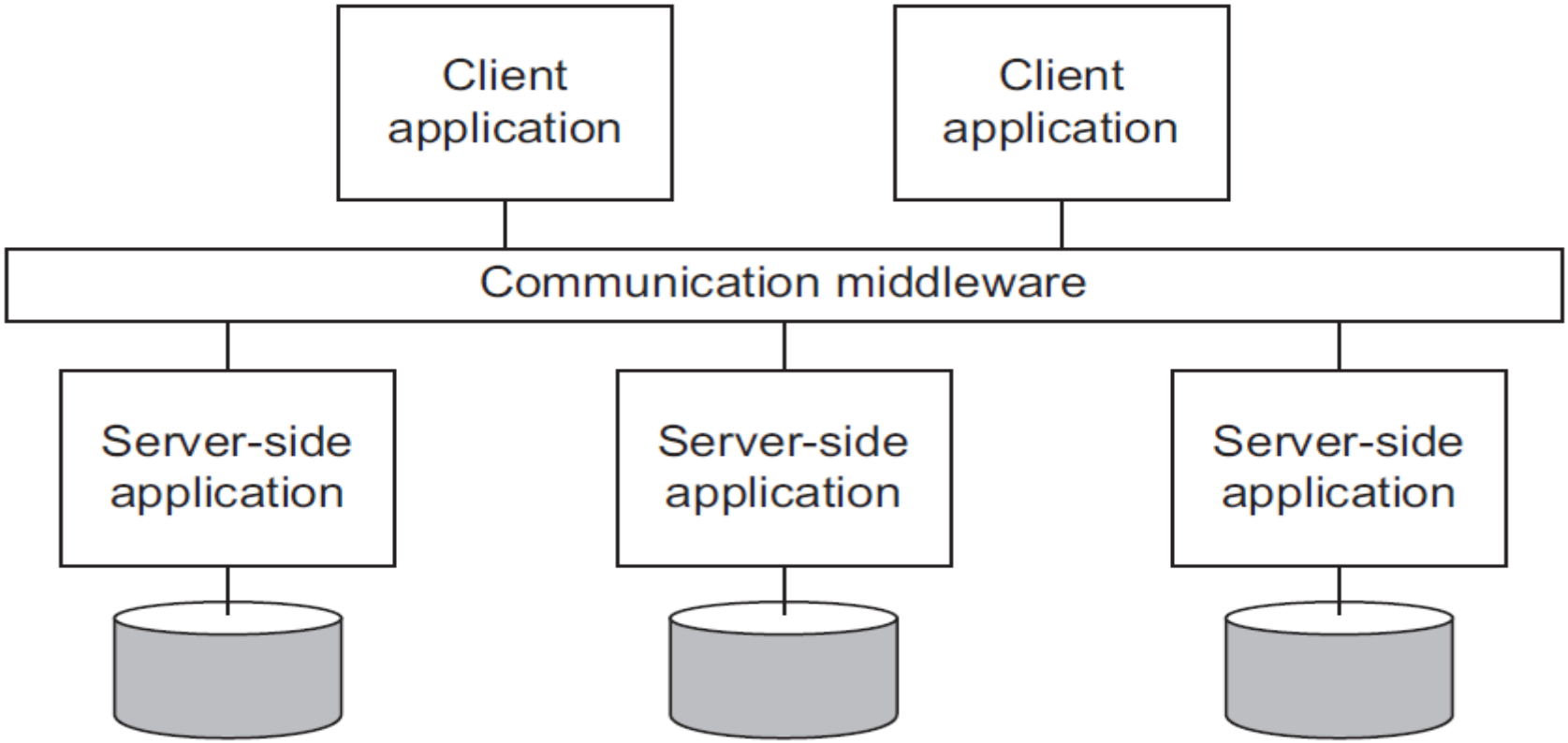
TPM: MONITOR DE PROCESSAMENTO DE TRANSAÇÕES



OBSERVAÇÃO

Em muitos casos, os dados envolvidos em uma transação é distribuído através de vários servidores. Um monitor de TP é responsável por coordenar a execução da transação

EAI E MIDDLEWARE



Middleware oferece facilidades de comunicação para integração

- RPC – Chamada Remota de Procedimento (*remote procedure call*): Requisições são enviadas através de uma chamda local a procedimento, empacotada como uma mensagem, processada, respondida através de mensagem e o resultado é retornado como um “return” de uma chamada
- MOM – *Message Oriented Middleware*: Mesnagens são enviadas a um ponto lógico de contato (**Published**), e repassadas a aplicações assinantes do tópico (**Subscribers**)

COMO INTEGRAR APLICAÇÕES

- **Transferência de arquivos**: tecnicamente simples, mas sem flexibilidade:
  - Definir formato de arquivo comum de layout
  - Definir modelo de gerenciamento de arquivo
  - Atualizar propagação e atualizar notificações
- **Base de dados compartilhada:**
  - Mais flexível, mas ainda demanda um esquema comum de dados, tem risco de “bottleneck” (gargalo)
- **RPC:**
  - eficiente quando é necessário a execução de uma série de ações
- **Mensagens:**
  - RPC demanda que o chamante e chamado estejam em estado “rodando” ao mesmo tempo
  - Mensagens permitem o “descasamento” em tempo e espaço

SISTEMAS DISTRIBUÍDOS PERVASIVOS

OBSERVAÇÃO

Sistemas distribuídos de próxima geração nos quais os nós são pequenos e moveis, e frequentemente embutidos em um sistema maior caracterizados pelo fato que o sistema naturalmente se mistura com o ambiente do usuário.

TRÊS SUBTIPOS (SOBREPOSTOS)

- **Sistemas computacionais ubiquos:** pervasivos e continuamente presentes, isto é, existe uma interação contínua entre o sistema e usuário.
- **Sistemas computacionais móveis:** pervasivos, mas com ênfase no fato que dispositivos são inerentemente móveis.
- **Redes de sensores (e atuadores):** pervasivos, com ênfase no sensoriamento existente (colaborativo) e atuação no ambiente.

SISTEMAS UBIQUOS

Elementos Principais

1. **Distribuição:** Dispositivos em rede, distribuídos, e acessíveis de forma transparente.
2. **Interação:** Interação entre usuários e dispositivos e usuários é altamente discreta.
3. **Ciência de contexto:** O sistema é ciente do contexto do usuário de forma a otimizar a interação.
4. **Autonomia:** Dispositivos operam de forma autônoma sem intervenção humana, e são assim altamente auto gerenciados.
5. **Inteligência:** O sistema como um todo pode manusear um grande gama de ações e interações dinâmicas.

COMPUTAÇÃO MÓVEL

Características Distintas

- Uma gama diversa de diferentes dispositivos móveis (smartphones, tablets, GPS´s, controle remotos, crachás ativos, dispositivos IoT, etc.
- Mobilidade implica que a localização de um dispositivo deve mudar ao longo do tempo → mudança de local dos serviços, alcance, etc. Palavra chave: descoberta (Discovery)
- Comunicação pode se tornar mais difícil: sem rota estável, e também, talvez sem garantia de conectividade → rede tolerante a disrupções

PADRÕES DE MOBILIDADE

Questão

Qual a relação entre disseminação de informação e mobilidade ? Ideia básica: um encontro permite a troca de informação (pocket-switched networks).

Estratégia de sucesso

- O mundo de Maria consiste de amigos e estranhos.
- Se Alice quer enviar uma mensagem para José: entregue a mensagem para todos seus amigos
- Amigos irão repassar a mensagem para José no primeria oportunidade de encontro

Observação

Esta estratégia funciona porque (aparentemente) existe comunidades fechadas de amigos

DETECÇÃO DE COMUNIDADE

Questão

Como detectar sua comunidade sem possuir conhecimento global ?

Construa sua lista gradualmente

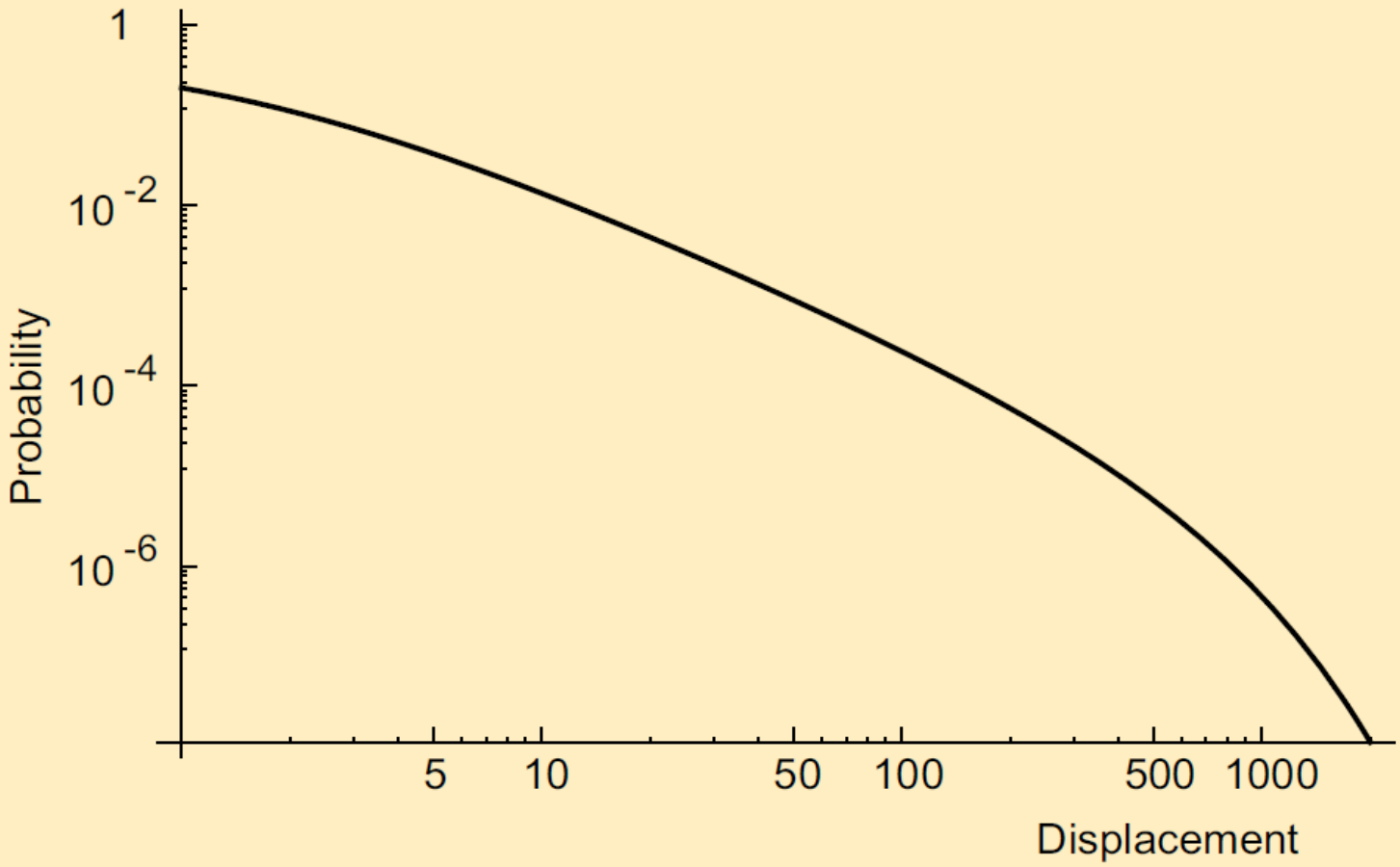
1. Nó *i* mantém um conjunto familiar *Fi* e um conjunto comunidade *Ci*, inicialmente ambos vazios.
2. Nó *i* adiciona *j* a *Ci* quando $\frac{|F_j \cap C_i|}{|F_j|} > \lambda$
3. Junte (merge) duas comunidades quando $|C_i \cap C_j| > \gamma |C_i \cup C_j|$

Experimente mostrar que $\lambda = \gamma = 0.6$ é bom.

QUANTA MOBILIDADE TÊM AS PESSOAS ?

Resultados Experimentais

Rastreando 100.000 usuários de celulares durante 6 meses leva a:



Além disto: pessoas tendem a retornar para o mesmo lugar após 24,48 ou 72 horas. Não somos muito móveis

REDES DE SENSORES

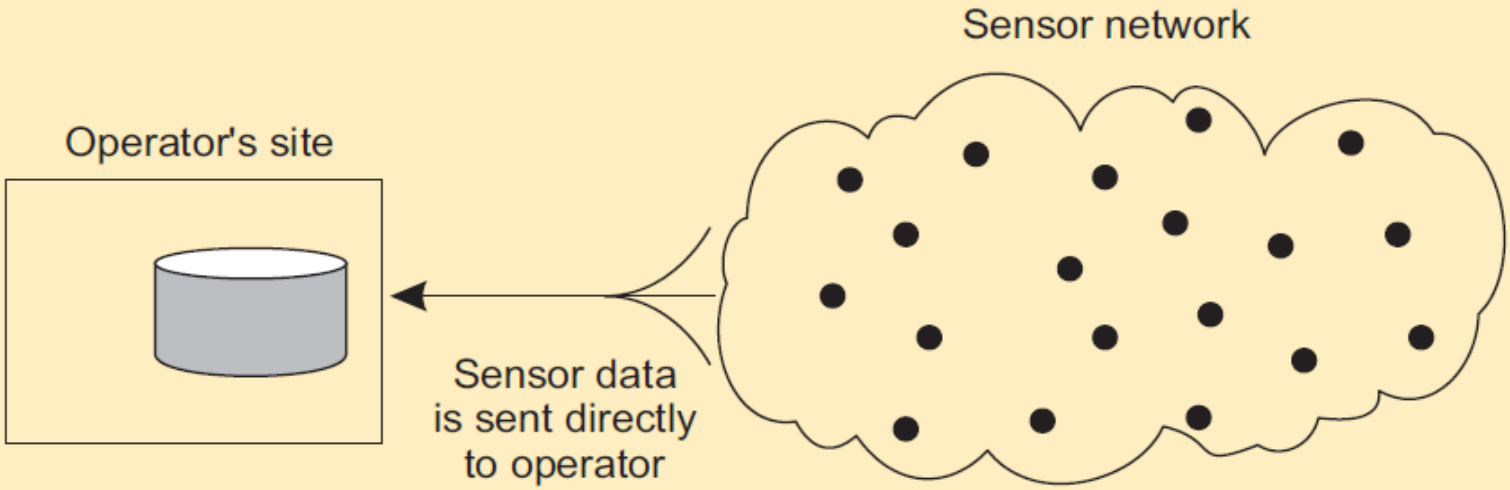
Características

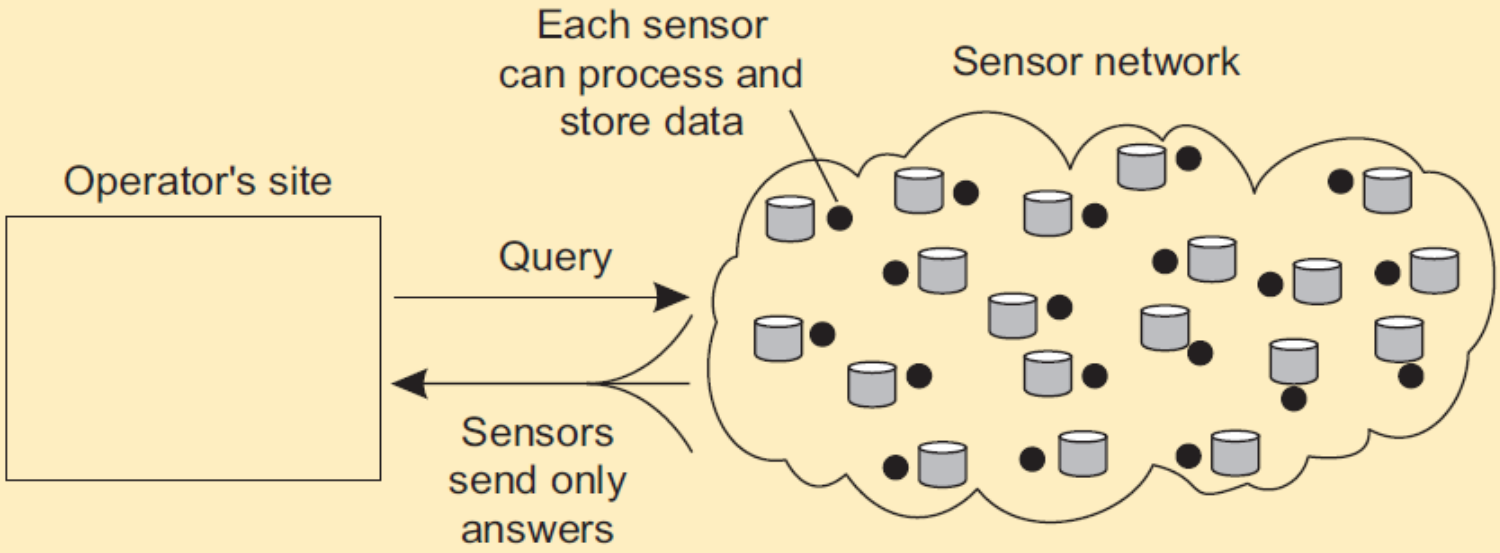
Os nós aos quais sensores estão conectados são:

- Muitos (10s a 1000s)
- Simples (baixa capacidade de memória / computação / comunicação)
- Frequentemente alimentados por bateria (ou mesmo sem bateria)

REDES DE SENSORES COMO BASE DE DADOS DISTRIBUÍDA

Dois extremos





DUTY-CYCLED NETWORKS REDES COM TRABALHO CÍCLICO

Questão

Muitas redes de sensores precisam operar com restrição de energia: introduz ciclos de trabalho (*duty-cycles*)

Definição

Um nó é ativo durante T_{active} unidades de tempo, e então é suspenso por $T_{suspended}$ unidades de tempo, para se tornar ativo de novo. *Duty cycle* τ :

$$\tau = \frac{T_{active}}{T_{active} + T_{suspended}}$$

Ciclos de trabalho típicos são de 10 a 30%, mas podem também serem menores que 1%.

MANTENDO REDES *DUTY-CYCLED* EM SINCRONISMO

Questão

Se os ciclos de trabalho são baixos, nós sensores podem não mais acordar ao mesmo tempo e tornar-se permanentemente desconectados: eles são ativos em diferentes eslotes de tempo não sobrepostos

Solução

- Cada nó *A* adota um *ID de cluster* C_A, sendo um número.
- Então um nó envia uma mensagem de *Join* durante seu período de suspensão.
- Quando *A* recebe uma mensagem de *Join* de *B* e $C_A < C_B$, então ele envia uma mensagem *Join* para seus vizinho (no cluster C_A) antes de se juntar a *B*.
- Quando $C_A > C_B$, ele envia uma mensagem *Join* para *B* durante o período ativo de *B.*

Nota

Uma vez que a mensagem de Join alcança um cluster inteiro, o *"merging"* de dois *clusters* é muito rápido. *Merging* significa: relógios reajustados