ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

Aula 01: Introdução

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

26/02/2024

Objetivos da disciplina

- Abordar os fundamentos, processos, técnicas e ferramentas para o desenvolvimento de sistemas de informação distribuídos.

Livro texto

- **Distributed Systems 4th edition** (2023), por Maarten van Steen e Andrew S. Tanenbaum
- Slides baseados no livro
- Versão em inglês do livro disponível gratuitamente no site:
  - `https://www.distributed-systems.net/index.php/books/ds4/`
- Tópicos:
  - Introdução;
  - Arquiteturas de sistemas distribuídos;
  - Processos;
  - Comunicação;
  - Nomes;
  - Coordenação;
  - Replicação;
  - Tolerância a falhas;
  - Segurança.

Programação

Data	Atividade	Conteúdo	Observações
26/2/2024	Aula 1	Introdução	Cap 1
1/3/2024	Aula 2	Introdução	Cap 1
4/3/2024	Aula 3	Arquiteturas de sistemas distribuídos	Cap 2
8/3/2024	Aula 4	Arquiteturas de sistemas distribuídos	Cap 2
11/3/2024	Aula 5	Arquiteturas de sistemas distribuídos	Cap 2
15/3/2024	Aula 6	Processos	Cap 3
18/3/2024	Aula 7	Processos	Cap 3
22/3/2024	Aula 8	Processos	Cap 3
25/3/2024	Feriado		
29/3/2024	Feriado		
1/4/2024	Aula 9	Comunicação	Cap 4
5/4/2024	Aula 10	Comunicação	Cap 4
8/4/2024	Aula 11	Comunicação	Cap 4
12/4/2024	Aula 12	Comunicação	Cap 4
15/4/2024	Aula 13	Coordenação	Cap 5
19/4/2024	Aula 14	Coordenação	Cap 5
22/4/2024	Aula 15	Coordenação	Cap 5
26/4/2024	Aula 16	Coordenação	Cap 5
29/4/2024	Prova 1	Conteúdo até aula 16	

Data	Atividade	Conteúdo	Observações
3/5/2024	Aula 17	Nomes	Cap 6
6/5/2024	Aula 18	Nomes	Cap 6
10/5/2024	Aula 19	Nomes	Cap 6
13/5/2024	Aula 20	Nomes	Cap 6
17/5/2024	Aula 21	Consistência e replicação	Cap 7
20/5/2024	Aula 22	Consistência e replicação	Cap 7
24/5/2024	Aula 23	Consistência e replicação	Cap 7
27/5/2024	Aula 24	Consistência e replicação	Cap 7
31/5/2024	Feriado		
3/6/2024	Aula 25	Tolerância a falhas	Cap 8
7/6/2024	Aula 26	Tolerância a falhas	Cap 8
10/6/2024	Aula 27	Tolerância a falhas	Cap 8
14/6/2024	Aula 28	Tolerância a falhas	Cap 8
17/6/2024	Aula 29	Segurança	Cap 9
21/6/2024	Aula 30	Segurança	Cap 9
24/6/2024	Aula 31	Segurança	Cap 9
28/6/2024	Aula 32	Segurança	Cap 9
1/7/2024	Prova 2	Todo conteúdo do curso	

O curso é PRESENCIAL

Regimento Geral da USP

SEÇÃO V — DA AVALIAÇÃO DO RENDIMENTO ESCOLAR

Artigo 84 – Será aprovado, com direito aos créditos correspondentes, o aluno que obtiver nota final igual ou superior a cinco e tenha, no mínimo, setenta por cento de frequência na disciplina.

http://www.leginf.usp.br/?resolucao=consolidada-resolucao-no-3745-de-19-de-outubro-de-1990#a84

Combinados

- Chamada feita no início da aula
- Não faça ”multitasking”

Critério de Avaliação

- Duas provas (P_1 e P_2)
- Um Exercício-Programa (EP)

Cálculo da média

$$M = 0,3P_1 + 0,5P_2 + 0,2EP$$

Informações de contato

- Prof. Dr. Renan Alves
- renanalves@usp.br
- sala A1–T10G

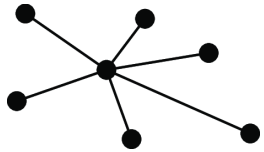
Usaremos o eDisciplinas

https://edisciplinas.usp.br/

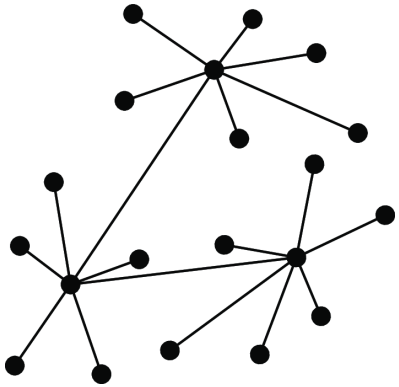
Capítulo 1

Diferença entre Distribuído e Descentralizado

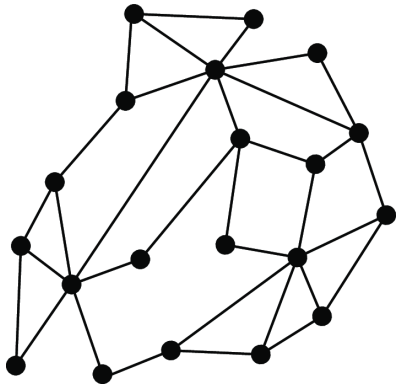
Senso comum dita que:



Centralizado



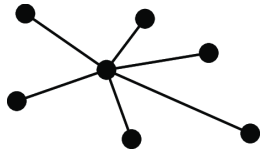
Decentralizado



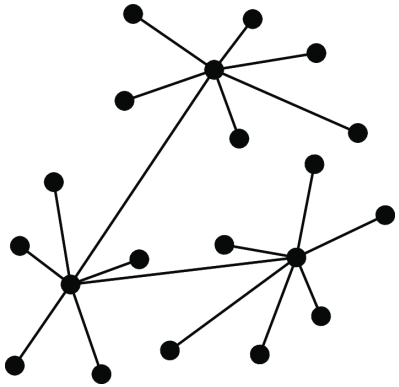
Distribuído

Diferença entre Distribuído e Descentralizado

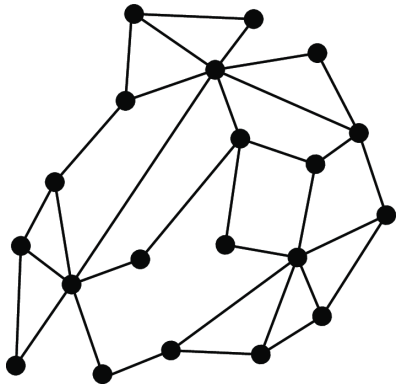
Senso comum dita que:



Centralizado



Decentralizado

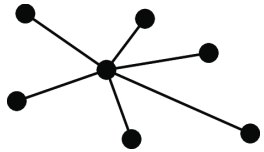


Distribuído

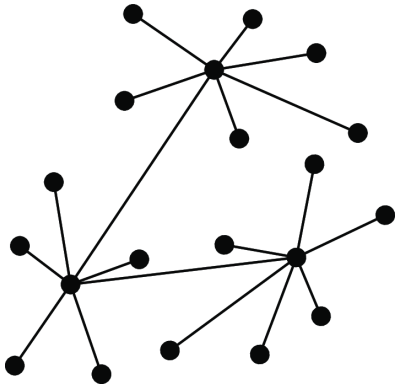
Em que ponto um sistema descentralizado se torna distribuído?

Diferença entre Distribuído e Descentralizado

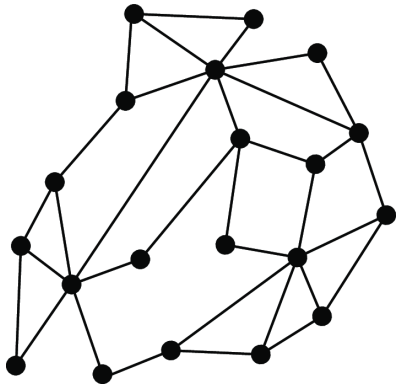
Senso comum dita que:



Centralizado



Decentralizado



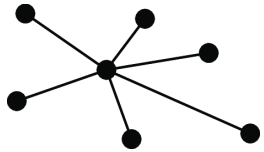
Distribuído

Em que ponto um sistema descentralizado se torna distribuído?

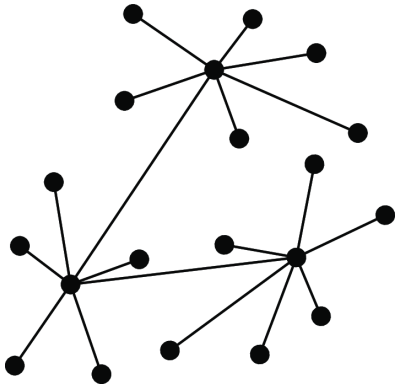
- Ao adicionar 1 conexão entre dois nós ao sistema descentralizado?

Diferença entre Distribuído e Descentralizado

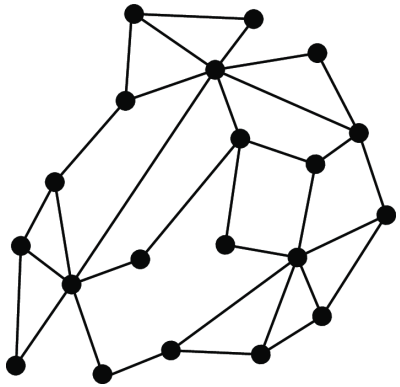
Senso comum dita que:



Centralizado



Decentralizado



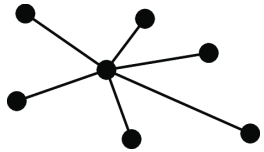
Distribuído

Em que ponto um sistema descentralizado se torna distribuído?

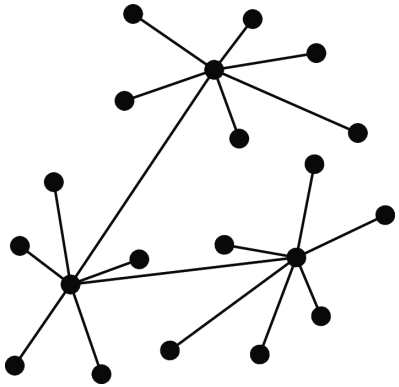
- Ao adicionar 1 conexão entre dois nós ao sistema descentralizado?
- Ao adicionar 2 conexões entre outros dois nós?

Diferença entre Distribuído e Descentralizado

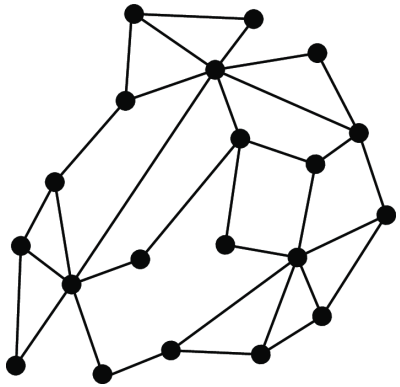
Senso comum dita que:



Centralizado



Decentralizado



Distribuído

Em que ponto um sistema descentralizado se torna distribuído?

- Ao adicionar 1 conexão entre dois nós ao sistema descentralizado?
- Ao adicionar 2 conexões entre outros dois nós?
- Generalizando: ao adicionar $k > 0$ conexões. . . ?

Abordagem de classificação alternativa

Duas visões para concretização de sistemas distribuídos

- Visão integrativa: conectar sistemas computacionais em rede pré-existentes em um sistema maior
- Visão Expansiva: um sistema computacionais em rede pré-existente é estendido com computadores adicionais

Duas definições

- Um sistema decentralizado é um sistema computacional em rede no qual processos e recursos são necessariamente espalhados por múltiplos computadores.
  - Muitas vezes esbarra em limites administrativos
  - Exemplos: aprendizado de máquina federado, blockchain, sistemas de monitoramento geograficamente dispersos
- Um sistema distribuído é um sistema computacional em rede no qual processos e recursos são suficientemente espalhados por múltiplos computadores.
  - Exemplos: servidores de e-mail, content distribution networks (CDN)

Ideias equivocadas comuns

Equívoco 1: soluções centralizadas não são escaláveis

Há uma diferença entre logicamente e fisicamente centralizada. A raiz do DNS (Domain Name System) é um exemplo:

- logicamente centralizada
- fisicamente distribuída
- decentralizada por diversas entidades

Ideias equivocadas comuns

Equívoco 1: soluções centralizadas não são escaláveis

Há uma diferença entre logicamente e fisicamente centralizada. A raiz do DNS (Domain Name System) é um exemplo:

- logicamente centralizada
- fisicamente distribuída
- decentralizada por diversas entidades

Equívoco 2: soluções centralizadas tem ponto de falha único e isso é inerentemente ruim

Ter um ponto único de falha pode ser:

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

Ideias equivocadas comuns

Equívoco 1: soluções centralizadas não são escaláveis

Há uma diferença entre logicamente e fisicamente centralizada. A raiz do DNS (Domain Name System) é um exemplo:

- logicamente centralizada
- fisicamente distribuída
- decentralizada por diversas entidades

Equívoco 2: soluções centralizadas tem ponto de falha único e isso é inerentemente ruim

Ter um ponto único de falha pode ser:

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

Em geral...

Vamos estudar sobre escalabilidade, tolerância a falhas e segurança para entender as relações custo-benefício envolvidas.

Aspectos dos sistemas distribuídos

Sistemas distribuídos são complexos, podem ser estudados sob vários aspectos

- Arquitetura: formas de organizar
- Processos: tipos de processos e como se relacionam
- Comunicação: formas de troca de dados
- Coordenação: algoritmos de coordenação independentes da aplicação
- Nomeação: como identificar os recursos
- Consistência e replicação: dados podem ser replicados para melhorar desempenho, mas é preciso atualizar de forma consistente
- Tolerância a falhas: manter sistema operante mesmo na presença de falhas parciais
- Segurança: controle de acesso e proteção aos recursos

O que queremos alcançar?

Objetivos gerais de projeto

- Compartilhamento de recursos
- Distribuição transparente
- Interfaces abertas (openness)
- Escalabilidade

Compartilhando de recursos

Exemplos clássicos

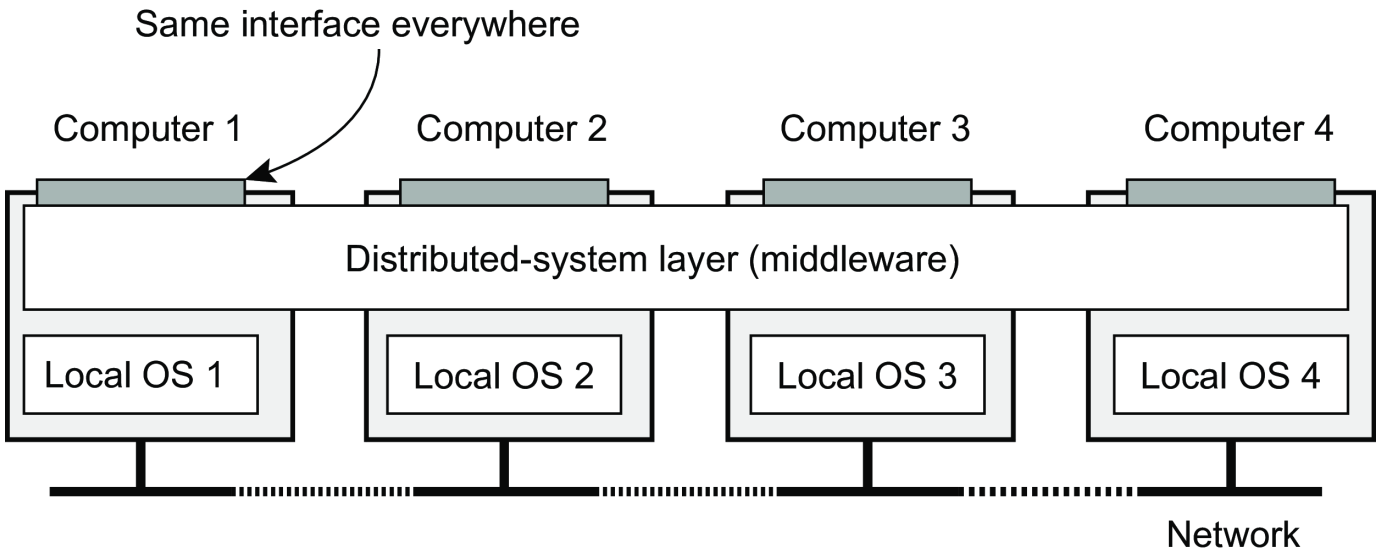
- Compartilhamento de arquivos na nuvem
- Streaming peer-to-peer
- Serviço de e-mail compartilhado (e.g. e-mail USP $\rightarrow$ gmail)
- Servidores web compartilhados

Observação

"The network is the computer"

(frase dita por John Gage, em 1984 na Sun Microsystems)

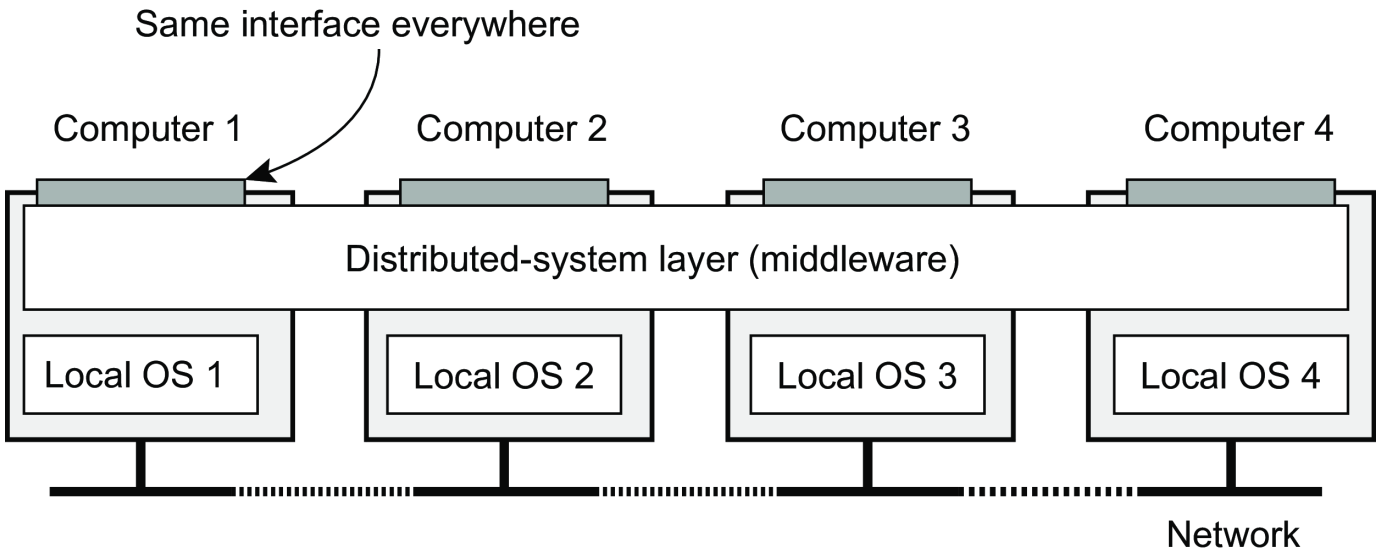
Distribuição transparente



Como definir ”transparência”?

O processo através do qual um sistema distribuído tenta mascarar o fato de que seus processos e recursos estão fisicamente distribuídos entre múltiplos computadores, possivelmente separados por longas distâncias

Distribuição transparente



Como definir ”transparência”?

O processo através do qual um sistema distribuído tenta mascarar o fato de que seus processos e recursos estão fisicamente distribuídos entre múltiplos computadores, possivelmente separados por longas distâncias

Observação

As técnicas usadas para distribuição transparente geralmente são implementadas numa camada entre as aplicações e os sistemas operacionais: o middleware

Distribuição transparente

Tipos

Transparência	Descrição
Acesso	Esconde diferenças em como os dados são representados e em como um objeto é acessado
Local	Esconde localização do objeto
Relocação	Esconde que um objeto pode ser movido para outra localização, mesmo durante seu uso
Migração	Esconde que um objeto pode ser movido para outra localização
Replicação	Esconde que um objeto é replicado
Concorrência	Esconde que um objeto pode ser compartilhado por diversos usuários de forma independente
Falha	Esconde a ocorrência de uma falha e a sua recuperação

Grau de transparência

Almejar transparência total pode ser muita coisa

Grau de transparência

Almejar transparência total pode ser muita coisa

- Há latências de comunicação que não podem ser escondidas

Grau de transparência

Almejar transparência total pode ser muita coisa

- Há latências de comunicação que não podem ser escondidas
- Esconder completamente as falhas de rede e dos nós é (teoricamente e praticamente) impossível
  - Não é possível distinguir entre uma resposta que está demorando ou a ocorrência de uma falha
  - Não é possível confirmar se uma operação foi concluída logo antes de uma falha

Grau de transparência

Almejar transparência total pode ser muita coisa

- Há latências de comunicação que não podem ser escondidas
- Esconder completamente as falhas de rede e dos nós é (teoricamente e praticamente) impossível
  - Não é possível distinguir entre uma resposta que está demorando ou a ocorrência de uma falha
  - Não é possível confirmar se uma operação foi concluída logo antes de uma falha
- Transparência total afeta o desempenho, revelando que o sistema é distribuído
  - Manter replicas exatamente atualizadas de acordo com o original leva tempo
  - Por exemplo, gravar as operações de escrita no disco imediatamente, em vez de manter apenas na memória (prevenção em caso de falha)

Grau de transparência

Expor distribuição pode ser benéfico

- Serviços baseados em localização (e.g. encontrar usuários por proximidade)
- Levar em conta fuso horários
- Facilitar interpretação do estado do sistema (e.g. estimativa de tempo para considerar que um servidor falhou)

Grau de transparência

Expor distribuição pode ser benéfico

- Serviços baseados em localização (e.g. encontrar usuários por proximidade)
- Levar em conta fuso horários
- Facilitar interpretação do estado do sistema (e.g. estimativa de tempo para considerar que um servidor falhou)

Conclusão

Distribuição transparente é um objetivo interessante, mas difícil de atingir, e, em geral, não é o objetivo principal

Interfaces abertas

Sistemas distribuídos abertos

Um sistema que possui componentes que podem ser facilmente usados por outros sistemas ou integrados a outros sistemas. O próprio sistema frequentemente será composto por outros componentes abertos.

Em outras palavras...

Ser capaz de interagir com outros sistemas de interface aberta, independentemente da infraestrutura subjacente:

- Sistemas devem seguir interfaces bem definidas
- Sistemas devem ser capaz de interoperar facilmente
- Sistemas devem suportar portabilidade de aplicações
- Sistemas devem ser fácil de estender

Políticas vs. mecanismos

Implementando openness: mecanismos

- Permitir configuração de políticas de cache
- Permitir diferentes níveis de confiança para código recebido
- Permitir configurar parâmetros de QoS por cada fluxo de dados
- Oferecer algoritmos de criptografia

Implementando openness: políticas

- Qual o nível de consistência exigido para os dados armazenados em cache no cliente?
- Quais operações podem ser realizadas por código de terceiros?
- Como cumprir os requisitos de qualidade de serviço frente a variações de banda?
- Quais são os requisitos de segurança de informação?

Sobre separação entre políticas e mecanismos

Observação

Quanto mais estrita a separação entre políticas e mecanismos, mais bem definidos devem ser os mecanismos, potencialmente gerando um grande número de parâmetros configuráveis e dificultando o gerenciamento do sistema.

Encontrando o equilíbrio

Políticas *hard-coded* simplificam o gerenciamento do sistema, reduzindo a complexidade, apesar de reduzir a flexibilidade do sistema.

Dependabilidade

Ideia geral

Um componente provê serviços para clientes. Para prover serviços, o componente pode precisar de serviços de outros componentes $\Rightarrow$ um componente depende do funcionamento de outro componente.

Mais precisamente

Um componente C depende do componente C^* se a corretude do comportamento de C's depende da corretude do comportamento de C^*'s. (Componentes são processos ou canais de comunicação.)

Dependabilidade

Requisitos de dependabilidade

Requisito	**Descrição**
Disponibilidade	Prontidão para uso
Confiabilidade	Continuidade de prestação do serviço
Segurança (Safety)	Probabilidade muito baixa de catástrofes
Manutenibilidade	Facilidade em consertar um sistema que falhou

Confiabilidade vs. Disponibilidade

Confiabilidade $R(t)$ do componente C

Probabilidade condicional de que o componente C está funcionando corretamente no intervalo de tempo $[0, t)$, assumindo que C estava operando corretamente em $T = 0$.

Métricas

- Tempo médio de falha (Mean Time To Failure) (*MTTF*)
- Tempo médio de reparo (Mean Time To Repair) (*MTTR*)
- Tempo médio entre falhas (Mean Time Between Failures) (*MTBF*): *MTTF* + *MTTR*.

Terminologia

Falha, erro e defeito

Termo	Descrição	Exemplo
Falha (failure)	Um componente não está de acordo com a especificação	Crash de programa
Erro (error)	Parte do componente que leva à falha	Bug de implementação
Defeito (fault)	Causa do erro	Programador desatento

Terminologia

Lidando com defeitos

Termo	Descrição	Exemplo
Prevenção de defeitos	Evitar ocorrência de defeitos	Não contratar programadores desatentos
Tolerância	Construir um componente para mascarar a ocorrência de defeito	Redundância de componente feito por programadores diferentes
Remoção de defeitos	Redução da prevalência ou seriedade de defeitos	Demitir programadores desatentos
Previsão de defeitos	Estimar a quantidade e consequência de defeitos	Estimar a quantidade de programadores desatentos contratados

Segurança

Observação

Um sistema distribuído que não tem segurança não provê dependabilidade

Segurança

Observação

Um sistema distribuído que não tem segurança não provê dependabilidade

Requisitos

- Confidencialidade: dados são revelados apenas para as partes autorizadas
- Integridade: dados não devem sofrer alterações indevidas (seja propositalmente por um atacante ou acidentalmente)

Segurança

Observação

Um sistema distribuído que não tem segurança não provê dependabilidade

Requisitos

- Confidencialidade: dados são revelados apenas para as partes autorizadas
- Integridade: dados não devem sofrer alterações indevidas (seja propositalmente por um atacante ou acidentalmente)

Autenticação, Autorização e Confiança

- Autenticação: verificação se identidade é verdadeira
- Autorização: verificação dos direitos de acesso atrelados à uma identidade
- Confiança: certeza que uma entidade irá se comportar de acordo com uma dada expectativa

Mecanismos de segurança

De forma simplificada

É sobre cifrar e decifrar dados com chaves de segurança.

Notação

$K(dados)$ representa que a chave K é usada para cifrar/decifrar *dados*.

Mecanismos de segurança

Criptossistema simétrico

Considerando uma chave de cifração E_K e uma chave de decifração D_K:
$\Rightarrow dado = D_K(E_K(dado))$ *se* $D_K = E_K$.
Nota: as chaves devem ser secretas.

Criptossistema assimétrico

Distinção entre uma chave pública PK e uma chave privada (secreta) SK.

- Cifração de uma mensagem de *Alice* para *Bob*:

$$dado = \underbrace{SK_{bob}(\overbrace{PK_{bob}(dado)}^{\text{Enviada por Alice}})}_{\text{Ação de Bob}}$$

- Assinatura de mensagem para *Bob* por *Alice*:

$$[dado, \underbrace{dado \stackrel{?}{=} PK_{alice}(SK_{alice}(data))}_{\text{Verificação por Bob}}] = \underbrace{[dado, SK_{alice}(dado)]}_{\text{Enviado por Alice}}$$

Mecanismos de segurança

Função de hash

Na prática, usamos funções seguras de hash: $H(dado)$ resulta em uma sequência de tamanho fixo.

- Qualquer mudança nos *dados* to $data^*$ resulta em uma saída completamente diferente $H(data^*)$.
- Dado um valor de hash h, é computacionalmente inviável encontrar um *dado* tal que $h = H(dado)$

Mecanismos de segurança

Função de hash

Na prática, usamos funções seguras de hash: $H(dado)$ resulta em uma sequência de tamanho fixo.

- Qualquer mudança nos *dados* to $data^*$ resulta em uma saída completamente diferente $H(data^*)$.
- Dado um valor de hash h, é computacionalmente inviável encontrar um *dado* tal que $h = H(dado)$

Reduzindo o custo de assinaturas digitais

Assinatura de mensagem para *Bob* por *Alice*:

$$[dado, \underbrace{H(dado) \stackrel{?}{=} PK_{alice}(sgn)}_{\text{Verificação por Bob}}] = \underbrace{[dado, H, sgn = SK_{alice}(H(dado))]}_{\text{Enviado por Alice}}$$

Escala em sistemas distribuídos

Observação

É comum desenvolvedores dizerem que seus sistema são “escaláveis” sem fazer esclarecer o porquê do sistema ser escalável

Escala em sistemas distribuídos

Observação

É comum desenvolvedores dizerem que seus sistema são “escaláveis” sem fazer esclarecer o porquê do sistema ser escalável

Ao menos três componentes

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

Escala em sistemas distribuídos

Observação

É comum desenvolvedores dizerem que seus sistema são “escaláveis” sem fazer esclarecer o porquê do sistema ser escalável

Ao menos três componentes

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

Observação

A maioria dos sistemas leva em conta apenas a escalabilidade de tamanho. A solução geralmente é composta de servidores operando independentemente em paralelo. Ainda há um desafio em escalar geográfica e administrativamente.

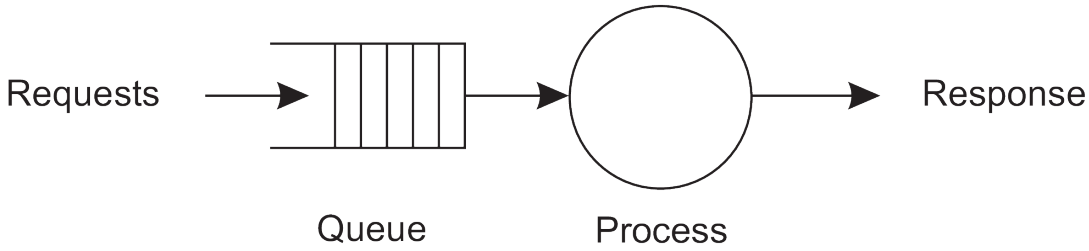
Escalabilidade de tamanho

Causas de problemas de escalabilidade em soluções centralizadas

- Capacidade computacional, limitada pelas CPUs
- Capacidade de armazenamento, incluindo a taxa de transferência entre CPUs e discos
- Capacidade da rede entre usuários e o serviço centralizado

Análise formal

Um sistema centralizado pode ser modelado como um sistema de fila simples



Premissas e notação

- Fila com capacidade infinita $\Rightarrow$ taxa de chegada de requisições não é influenciada pelo tamanho da fila ou o que está sendo processado.
- Taxa de chegada de requisições : λ
- Capacidade de processamento do servidor: μ

Porcentagem de tempo em que o sistema tem exatamente k requisições

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

Análise formal

Utilização U do serviço é a fração de tempo em que o servidor está ocupado

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

Numero médio de requisições no 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

$$X = \underbrace{U \cdot \mu}_{\text{servidor ocupado}} + \underbrace{(1-U)\cdot 0}_{\text{servidor ocioso}} = \frac{\lambda}{\mu} \cdot \mu = \lambda$$

Análise formal

Tempo de resposta: tempo total desde que a requisição é feita até ser processada

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

no qual $S = \frac{1}{\mu}$ é o tempo de serviço.

Observações

- Se U é pequeno, a razão entre tempo de resposta e tempo de serviço é próxima de 1: a requisição é processada rapidamente (pouco tempo na fila)
- A medida que U se aproxima de 100%, o sistema fica cada vez mais lento.
  Solução: diminuir S.

Problemas com escalabilidade geográfica

- Não é simples migrar a operação de LAN para WAN: muitos sistemas assumem operação síncrona entre cliente e servidor, i.e., o cliente fica parado esperando a resposta. Latência pode ser um problema.
- Conexão WAN é mais suscetível a perdas de pacotes
- Falta de comunicação do tipo um-para-muitos. Soluções usam serviçõe de nomeação e diretório (que tem seus próprias questões relacionadas a escalabilidade).

Problemas com escalabilidade administrativa

Essência

Políticas conflitantes em relação a uso, pagamento, gerenciamento e segurança

Exemplos

- Grids computacionais: compartilhamento de recursos valiosos entre diferentes domínios.
- Equipamento compartilhado: como controlar, gerenciar e usar sensores?

Exceção: diversas redes peer-to-peer

- Compartilhamento de arquivos (e.g. BitTorrent)
- Streamning assistido por pares (e.g. Spotify)

Nota: usuários colaborando, não entidades administrativas.

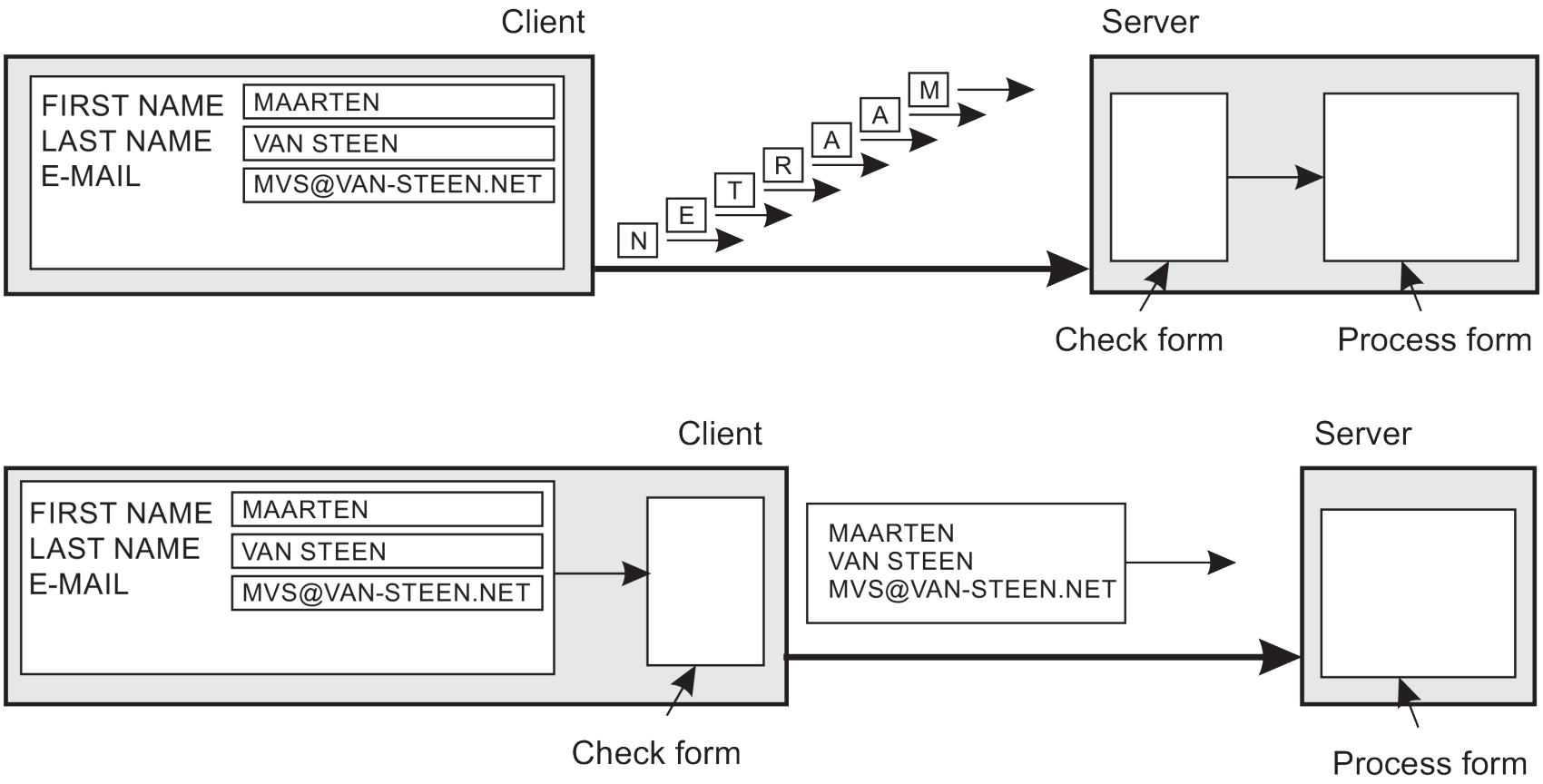
Técnicas para escalabilidade

Ocultar latências de comunicação

- Usar comunicação assíncrona
- Construir um gerenciador dedicado para tratar requisições/respostas
- Problema: não funciona para qualquer aplicação

Técnicas para escalabilidade

Facilitar solução aproximando a computação do cliente



Técnicas para escalabilidade

Particionar dados e computação entre máquinas

- Aproximar computação do cliente (e.g. Java scripts)
- Decentralized naming services (DNS)

Técnicas para escalabilidade

Replicação e caching: criar cópias de dados em máquinas diferentes

- Bases de dados com replicação/espelhamento
- Mirror de sites
- Cache de páginas web (nos navegadores e proxies)

Escalabilidade: problemas com replicação

Escalabilidade: problemas com replicação

- Múltiplas cópias resultam em inconsistências: ao modificar uma cópia, ela fica diferente das outras

Escalabilidade: problemas com replicação

- Múltiplas cópias resultam em inconsistências: ao modificar uma cópia, ela fica diferente das outras
- Manter as cópias consistentes de forma generalizada reques sincronização global após cada modificação

Escalabilidade: problemas com replicação

- Múltiplas cópias resultam em inconsistências: ao modificar uma cópia, ela fica diferente das outras
- Manter as cópias consistentes de forma generalizada reques sincronização global após cada modificação
- Sincronização global impede soluções de larga escala.

Escalabilidade: problemas com replicação

- Múltiplas cópias resultam em inconsistências: ao modificar uma cópia, ela fica diferente das outras
- Manter as cópias consistentes de forma generalizada reques sincronização global após cada modificação
- Sincronização global impede soluções de larga escala.

Observação

Se for possível tolerar um certo nível de inconsistências, pode não ser preciso realizar sincronização global (frequentemente), porém isso vai depender da aplicação