Sistemas Distribuídos

Aula 1 – Fundamentos de Sistemas Distribuídos

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

Agenda

Definição e Conceitos

Objetivos e Motivações

Características e Desafios

Modelos de Comunicação

Arquiteturas

Modelos de Falha

Exemplos Reais

Referências

Definição e Conceitos

O que é um Sistema Distribuído?

O que é um Sistema Distribuído?

▪ Definição (Tanenbaum)

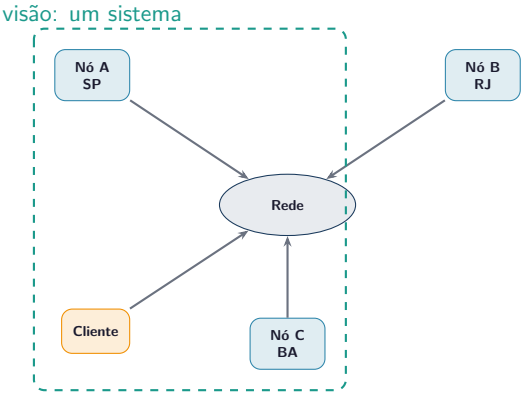
Um sistema distribuído é uma **coleção de computadores independentes** que se apresenta ao usuário como um **sistema único e coerente**.

▪ Definição (Coulouris)

Componentes localizados em **computadores em rede** que **comunicam e coordenam** suas ações apenas por **troca de mensagens**.

▪ A frase de Lamport

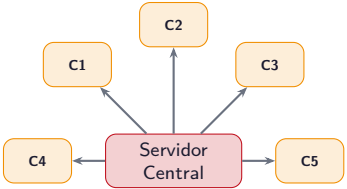
"Um sistema distribuído é aquele em que a falha de umccomputador que você nem sabia que existia pode tornarcseu próprio computador inutilizável."



Sistema Centralizado vs. Distribuído

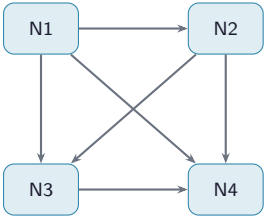
Sistema Centralizado

- Um único computador faz tudo
- Ponto único de falha
- Escalabilidade limitada
- Gerenciamento simples



Sistema Distribuído

- Vários nós cooperam
- Sem ponto único de falha (idealmente)
- Escala horizontalmente
- Gerenciamento complexo



Objetivos e Motivações

Por que usar sistemas distribuídos?

Por que usar Sistemas Distribuídos?

▪ Compartilhamento de Recursos

Permite compartilhar hardware, software e dados entre usuários e organizações. Exemplo: impressoras em rede, sistemas de arquivos distribuídos (NFS, HDFS).

▪ Escalabilidade

Adicionar mais nós para suportar mais carga (**escala horizontal**) em vez de comprar hardware mais caro (**escala vertical**). Base da computação em nuvem.

▪ Distribuição Geográfica

Dados e serviços próximos aos usuários reduzem **latência**. CDNs colocam conteúdo nos continentes onde está o público.

▪ Desempenho

Paralelismo: tarefas são divididas entre nós e executadas **simultaneamente**. Exemplo: MapReduce no Hadoop.

▪ Economia

Computadores pessoais e servidores comuns (*commodity hardware*) ligados em rede são mais baratos que supercomputadores.

▪ Tolerância a Falhas

Redundância de componentes: se um nó falha, outros assumem o trabalho. Serviços críticos (bancos, saúde) exigem **alta disponibilidade**.

Características e Desafios

O que torna os SD difíceis

▪ Transparência

O SD deve **esconder** sua natureza distribuída do usuário. Tipos de transparência:

- **Localização** — não sabe onde está o recurso
- **Migração** — recurso pode se mover
- **Replicação** — cópias são invisíveis
- **Falha** — recuperação é automática
- **Concorrência** — acesso simultâneo gerenciado

▪ Abertura (*Openness*)

Componentes seguem **interfaces padronizadas** (APIs, protocolos) que permitem **interoperabilidade** e extensibilidade. Exemplo: REST, gRPC, CORBA.

▪ Escalabilidade

Três dimensões:

- **Tamanho** — mais usuários/dados
- **Geográfica** — maior área
- **Administrativa** — mais domínios

Desafios dos Sistemas Distribuídos

▪ Ausência de relógio global

Cada nó tem seu próprio relógio. Sincronizá-los perfeitamente é **impossível** (Teorema da impossibilidade). Solução: **relógios lógicos**.

▪ Falhas parciais

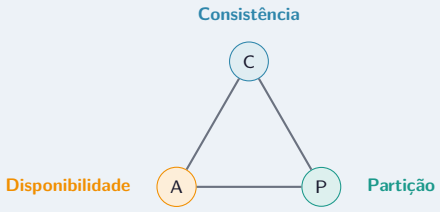
Um nó pode falhar enquanto outros continuam. O sistema deve detectar e se recuperar sem parar completamente. *"Fail-stop"* vs. *"Byzantine failures"*.

▪ Segurança

Comunicação pela rede expõe o sistema a **intercepção**, **alteração** e **ataques DoS**. Criptografia e autenticação são essenciais.

▪ Consistência vs. Disponibilidade

O **Teorema CAP** (**C**onsistency, **A**valiability, **P**artition) (Brewer, 2000): um SD só pode garantir 2 das 3 propriedades simultaneamente:



Modelos de Comunicação

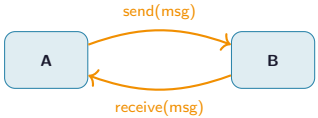
Como os nós se comunicam

Passagem de Mensagens vs. Memória Compartilhada

▪ Passagem de Mensagens

Nós trocam dados enviando e recebendo **mensagens explícitas** pela rede. É o modelo **nativo** de sistemas distribuídos.

Síncrono: emissor bloqueia até receber resposta.
Assíncrono: emissor continua; resposta chega depois.



▪ DSM — Distributed Shared Memory

Abstração que apresenta uma **memória global virtual** mesmo em nós fisicamente separados. O middleware gerencia a consistência automaticamente.

Vantagem: programação mais simples.
Desvantagem: overhead de sincronização.

▪ RPC — Remote Procedure Call

Chama uma função em outro nó **como se fosse local**. O *stub* serializa os parâmetros, envia pela rede e deserializa a resposta. Base do gRPC, CORBA, Java RMI.

Modelos de Tempo: Síncrono e Assíncrono

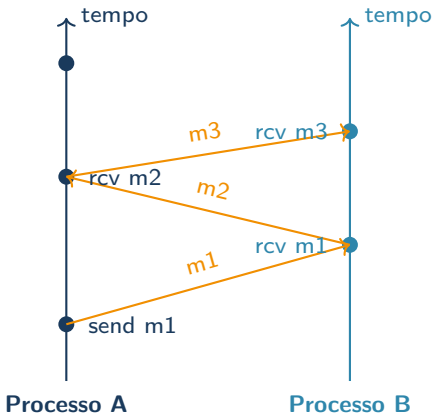
Modelo Síncrono

- Tempo de execução de cada passo, atraso de mensagem e relógio: **limitados**
- Mais simples de analisar, raramente reflete a realidade.

Modelo Assíncrono

- Execução, atraso e relógio: **sem limites**
- Modelo mais realista da Internet
- Detecção de falhas é **impossível**.
- Impossibilidade de Consenso

Diagrama espaço-tempo (mensagens)



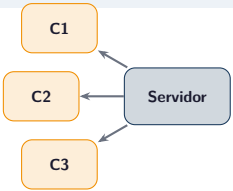
Arquiteturas de Sistemas Distribuídos

Como os componentes são organizados

▪ Cliente-Servidor

Modelo mais comum. **Servidores** oferecem serviços; **clientes** requisitam. Pode ter múltiplas camadas (*n-tier*): apresentação, lógica, dados.

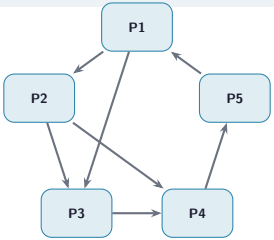
Ex: Web, e-mail, SGBD.



▪ Peer-to-Peer (P2P)

Todos os nós são **iguais**: cada um age como cliente e servidor ao mesmo tempo. Sem ponto central.

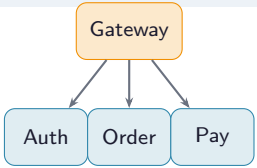
Ex: BitTorrent, Chord, Kademlia, blockchain.



▪ Microsserviços

Aplicação decomposta em **serviços pequenos e independentes**, cada um com seu banco de dados, implantados separadamente e comunicando por API.

Ex: Netflix, Amazon, Uber.

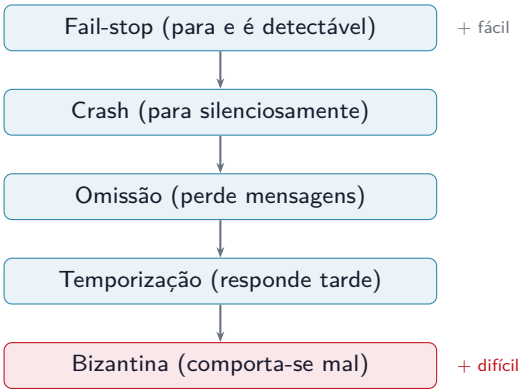


Modelos de Falha

Tipos de falha em sistemas distribuídos

Classificação de Falhas

Hierarquia de Falhas (+ simples → grave)



▪ Falha Bizantina

O nó se comporta de maneira **arbitrária e maliciosa**: envia mensagens incorretas, contraditórias ou mente deliberadamente. O nome vem do **Problema dos Generais Bizantinos** (Lamport et al., 1982).

Tolerância a falhas Bizantinas (**BFT**) requer $n \geq 3f + 1$ nós para tolerar f nós maliciosos. É a base de protocolos de consenso em **blockchain** (PBFT, Tendermint).

▪ Teorema FLP (1985)

Em um sistema assíncrono, **consenso é impossível** se ao menos um processo pode falhar por crash.

Sistemas Distribuídos na Prática

Exemplos do mundo real

SD na Prática — Exemplos

Internet e Web

- **DNS** — sistema distribuído de nomes
- **CDN** — Akamai, Cloudflare (conteúdo distribuído)
- **Motores de busca** — índice em milhares de nós

Bancos de Dados Distribuídos

- **Cassandra** — AP (sem consistência forte)
- **MongoDB** — CP (particionamento)
- **Google Spanner** — CAP com GPS/relógio atômico
- **CockroachDB** — distribuído e SQL-compatível

SD na Prática — Exemplos

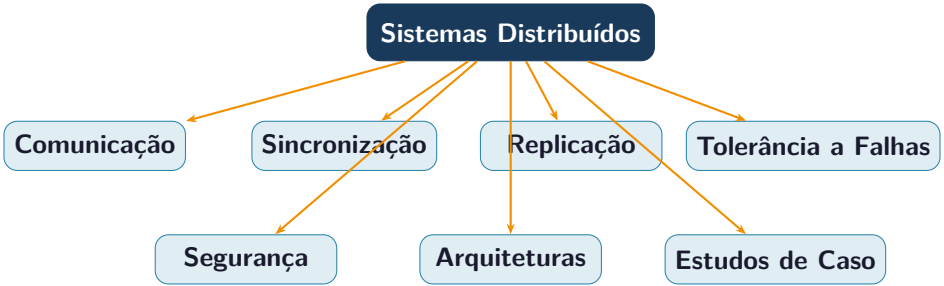
Computação em Nuvem

- **AWS, GCP, Azure** — IaaS / PaaS / SaaS
- **Kubernetes** — orquestração de contêineres
- **MapReduce / Spark** — processamento paralelo

Blockchain

- Sistema distribuído **sem autoridade central**
- Consenso via **PoW** - Proof of Work (Bitcoin) ou **PoS** - Proof of Stake (Ethereum)
- Tolerante a nós Bizantinos maliciosos

Resumo — O que estudaremos



Referências I

- TANENBAUM, A. S.; VAN STEEN, M. *Distributed Systems: Principles and Paradigms*. 3. ed. Pearson, 2017.
- COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5. ed. Addison-Wesley, 2012.
- LAMPORT, L.; SHOSTAK, R.; PEASE, M. The Byzantine Generals Problem. *ACM TOPLAS*, v. 4, n. 3, 1982.
- FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of Distributed Consensus with One Faulty Process. *JACM*, v. 32, n. 2, 1985.
- BREWER, E. Towards Robust Distributed Systems. *PODC*, 2000.