Sistemas Distribuídos

Aula 2 – Conceitos Fundamentais

Prof. Ana Carolina Sokolonski
Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana
2026

Agenda

O que é um Sistema Distribuído?

Dividir para Conquistar

Definição de Sistema Distribuído

▪ Coulouris et al.

Um SD é composto de vários computadores **autônomos**, comunicando-se via rede, **coordenando ações** e colaborando para executar processamentos, com concorrência, ausência de relógio global e falhas independentes. [1]

▪ Tanenbaum

Uma coleção de computadores independentes que se apresenta ao usuário como um **sistema único e coerente**. A distribuição deve ser **transparente** ao usuário. [2]

▪ Por que "Dividir para Conquistar"?

Demandas computacionais muito grandes exigem alto poder de processamento. A solução foi **compartilhar os recursos** de todos os computadores disponíveis na rede, resolvendo a escassez de recursos em qualquer máquina.

▪ Duas propriedades fundamentais

- **Paralelismo** — processamento simultâneo em múltiplos nós
- **Tolerância a Falhas** — continua operando mesmo com falhas parciais

Em poucas palavras

Um Sistema Distribuído é uma coleção de sistemas computacionais (hardware e software) independentes, com alguma forma de comunicação, que **colaboram** na execução de uma tarefa.

Exemplos do Dia a Dia

▪ Tarefas executadas por SDs

1. Entregue este e-mail para `fulano@uni.edu`
2. Envie o item I ao endereço E, cobrando R$350,00 da conta C
3. Autorize transferência de R$1500,00 entre contas bancárias
4. Movimente o braço robótico 3cm à direita
5. Inclua comentário com marca de tempo T na rede social

▪ Exemplos de Sistemas Distribuídos

1. **A Internet** — maior SD existente
2. Qualquer aplicação **intranet** corporativa
3. Aplicações **mobile** com servidor remoto
4. Serviços baseados em **Computação em Nuvem**
5. **Sistemas bancários** e de pagamento
6. Plataformas de **streaming** (Netflix, Spotify)
7. **Redes sociais** (Meta, X, LinkedIn)

Middleware

A cola dos Sistemas Distribuídos

O que é Middleware?

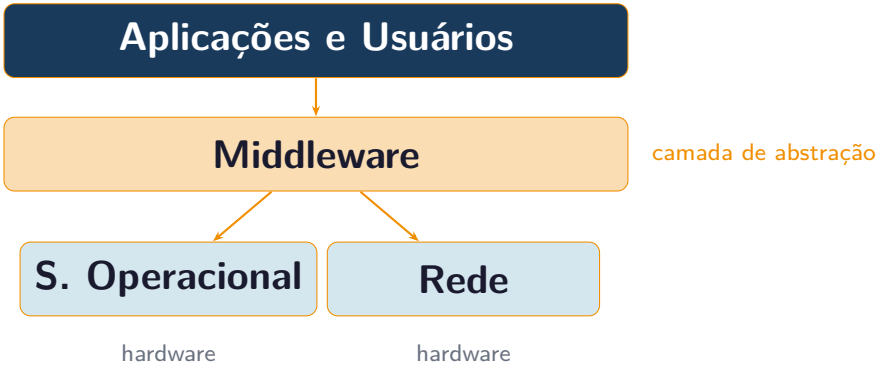
Definição

Camada de *software* situada logicamente entre duas camadas: a de **usuários e aplicações** (alto nível) e a de **Sistemas Operacionais e Comunicações** (baixo nível).
Responsável por prover a **comunicação correta entre dispositivos heterogêneos**, normalmente por meio de protocolos como chamadas remotas (RPC/RMI), mensagens (AMQP) ou serviços (REST, gRPC).

Exemplos de Middleware

- **CORBA, Java RMI** — chamadas remotas de objetos
- **RabbitMQ, Kafka** — filas de mensagens
- **gRPC, REST** — comunicação entre microsserviços
- **JDBC, ODBC** — acesso a bancos de dados

Midleware


Aplicações e Usuários
Middleware
camada de abstração
S. Operacional
Rede
hardware
hardware

Concorrência em Sistemas Distribuídos

Múltiplos processos simultâneos

▪ Paralelismo entre processos

Em SDs, vários processos executam **ao mesmo tempo** em diferentes computadores (hosts). Esse paralelismo é inato à natureza distribuída do sistema.
Além disso, cada processo individualmente pode precisar manter **várias comunicações em paralelo** com outros componentes — por isso usa-se **múltiplas threads**.

▪ Compartilhamento de Recursos

Em uma rede, execução concorrente é a norma. Usuários compartilham recursos como sites, arquivos no **Google Drive** ou impressoras, sem interferir uns nos outros.

A capacidade pode ser ampliada adicionando mais computadores — **escalabilidade horizontal**.

Desafios da Concorrência

- **Condições de corrida** — acesso simultâneo a dados
- **Deadlock** — processos se bloqueiam mutuamente
- **Starvation** — processo nunca obtém o recurso
- **Coordenação** — quem decide a ordem das ações?

Impossível ignorar

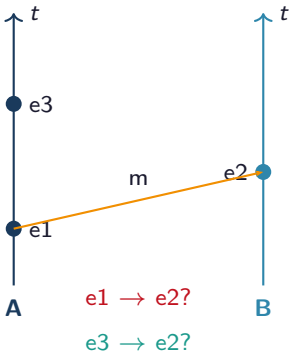
É "impossível" pensar em SDs sem pensar em **concorrência**. Componentes precisam manter várias "conversas" simultâneas.

Inexistência de Relógio Global

Um dos maiores desafios dos SDs

Soluções desenvolvidas

- **NTP** — Network Time Protocol: sincronização aproximada via rede
- **Relógios de Lamport** — ordem lógica de eventos sem relógio físico
- **Vetores de Relógio** — captura causalidade entre eventos
- **GPS/Relógio Atômico** — Google Spanner usa hardware dedicado



Tolerância a Falhas

A vida online não admite quedas

Tolerância a Falhas — Conceito

▪ Definição

Propriedade que garante a **correta operação** de um sistema mesmo na presença de falhas. Um SD tolerante a falhas mantém o funcionamento e a entrega de mensagens mesmo que parte do sistema falhe.

▪ Por que é imprescindível?

A vida *online* não admite mais que sistemas "caiam". Não podemos perder dinheiro num banco virtual ou arquivos salvos na nuvem.
Em aplicações de **missão crítica** — controle de aviões, cirurgia robótica, UTIs digitais — a falha pode custar vidas.

▪ Técnicas de tolerância

- **Replicação** — múltiplas cópias dos dados
- **Checkpoint** — salvar estado periodicamente
- **Heartbeat** — detectar nós inativos
- **Failover** — substituição automática de nós

Tipos de falha em SDs

- **Falha de nó** — computador para de funcionar
- **Falha de rede** — isola computadores sem pará-los
- **Falha de software** — bug, travamento de processo
- **Falha lenta** — nó responde, mas muito devagar
- **Falha Bizantina** — nó se comporta de forma maliciosa

A Dificuldade das Falhas em SDs

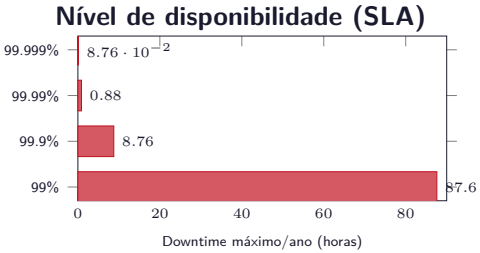
Falhas são diferentes em SDs

- Falhas de rede **isolam** computadores sem pará-los
- Programas podem não conseguir detectar se a rede falhou ou ficou muito lenta
- A falha de um computador **não é imediatamente percebida** pelos demais componentes
- Cada componente pode falhar **independentemente**

A pergunta impossível

Como diferenciar um **nó morto** de um **incrivelmente lento**? Esta questão é fundamental e está na raiz do **Teorema FLP** — consenso é impossível em sistemas assíncronos com ao menos uma falha.

A Dificuldade das Falhas em SDs



Exemplos tolerantes a falhas

Controle de Aviões e **Controle de Chão de Fábrica Automatizado** usam arquiteturas distribuídas com redundância total.

Desempenho e Confiabilidade

Por que SDs são mais rápidos e confiáveis

Desempenho em Sistemas Distribuídos

▪ Paralelismo = velocidade

A diminuição do tempo de execução é obtida pelo **paralelismo**: partes do programa são executadas em processadores diferentes **simultaneamente**.

Distribuindo a carga entre máquinas, o conjunto pode ter maior poder de processamento do que qualquer máquina individual.

▪ Confiabilidade por Redundância

SDs são potencialmente mais confiáveis pois, sendo os processadores **autônomos**, a falha de um não afeta os demais. Há redundância não só de processadores, mas de qualquer recurso necessário.
A confiabilidade aumenta ao se **replicar funções e/ou dados** nos vários processadores disponíveis.

Exemplos práticos

- **Compilação paralela** de módulos em máquinas diferentes
- **MapReduce** (Hadoop/Spark) — processa terabytes em minutos
- **Renderização** de filmes em fazendas de servidores
- **Treino de IA** distribuído em centenas de GPUs

Disponibilidade vs. Consistência

Quanto mais replicamos dados para ganhar disponibilidade, mais difícil é manter **consistência** entre as cópias. Este é o dilema central do Teorema CAP.

Transparência em Sistemas Distribuídos

O sistema deve parecer único

Os 8 Tipos de Transparência

▪ Acesso

Oculta diferenças na representação de dados e no modo como os recursos são acessados. Ex: acesso a arquivos locais e remotos.

▪ Localização

Oculta onde o recurso está fisicamente. URLs não revelam onde está o servidor.

▪ Migração

Recursos podem ser movidos sem afetar como são acessados.

▪ Relocação

Recursos podem ser relocados *enquanto são usados*. Ex: notebook trocando de redes WiFi sem perceber.

Os 8 Tipos de Transparência

▪ Replicação

Substituição automática em caso de falha. Exige transparência de localização.

▪ Concorrência

Processos compartilham recursos sem saber que o fazem nem com quem. Estado do recurso permanece consistente.

▪ Falha

O usuário não percebe que um componente falhou e se recuperou durante o uso.

▪ Persistência

Oculta se um recurso está em memória (volátil) ou em disco (persistente). Ex: objetos persistentes transparentes.

Desafios

O que torna os SDs difíceis de implementar

Principais Desafios dos Sistemas Distribuídos

1. Tolerância a Falhas

Garantir funcionamento mesmo com falhas parciais — o maior desafio.

2. Sincronização de Relógios

Não existe um relógio global; a ordem dos eventos é difícil de determinar.

3. Concorrência

Múltiplos processos acessando recursos compartilhados simultaneamente.

4. Transparência

Esconder completamente a natureza distribuída do sistema.

5. Confiabilidade e Disponibilidade

Manter o sistema funcionando 24/7, com dados íntegros.

6. Escalabilidade

Crescer para suportar mais usuários sem redesenhar o sistema.

7. Segurança

Comunicação aberta na rede expõe o sistema a ataques.

Referências I

- COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5. ed. Addison-Wesley, 2013.
- TANENBAUM, A. S.; VAN STEEN, M. *Distributed Systems: Principles and Paradigms*. 2. ed. Pearson, 2007.
- LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, v. 21, n. 7, 1978.