Introdução aos Sistemas Distribuídos

Definição: Sistemas distribuídos são um modelo de arquitetura computacional em que múltiplos computadores independentes, conectados por uma rede, trabalham juntos para apresentar aos usuários e aplicações um sistema único e coeso. Cada nó (máquina ou processo) do sistema é autônomo, mas colabora para atingir objetivos comuns, compartilhando recursos e coordenando ações.

Tecnicamente, um sistema distribuído é caracterizado por:

- **Multiplicidade de nós**: Existem vários componentes que podem estar fisicamente dispersos geograficamente.
- **Comunicação por mensagem**: A interação entre os nós se dá via troca de mensagens (não há memória compartilhada direta).
- **Falhas parciais**: Uma parte do sistema pode falhar sem necessariamente derrubar o sistema inteiro — tolerância a falhas é crucial.
- **Transparência**: Idealmente, a distribuição deve ser invisível para o usuário (transparência de localização, replicação, acesso, concorrência e falha).

Exemplos clássicos de sistemas distribuídos:

- Serviços de nuvem (AWS, Azure, GCP)
- Sistemas de arquivos distribuídos (Google File System, HDFS)
- Bancos de dados distribuídos (Cassandra, CockroachDB)
- Redes blockchain
- Aplicações peer-to-peer (BitTorrent)

Desafios principais:

- Consistência de dados (CAP Theorem)
- Sincronização de relógios e eventos (problemas de ordenação)

- Tolerância a falhas e recuperação
- Escalabilidade e balanceamento de carga
- Segurança e autenticação distribuída

Principal Objetivo

O **principal objetivo** de um sistema distribuído é **fornecer ao usuário uma visão unificada e consistente de um conjunto de recursos e serviços que, na prática, estão fisicamente dispersos**, garantindo ao mesmo tempo **eficiência, escalabilidade, tolerância a falhas e transparência**.

Se formos detalhar isso em termos mais técnicos:

- **Unificar** o acesso a recursos espalhados (processamento, armazenamento, rede).
- **Maximizar disponibilidade e confiabilidade** (se uma máquina falha, o sistema continua funcionando).
- **Permitir escalabilidade horizontal** (crescimento adicionando mais nós ao invés de máquinas maiores).
- **Otimizar o desempenho** (distribuindo a carga e executando processos mais próximos de onde os dados estão).
- **Abstrair a complexidade** da comunicação e da coordenação entre as partes.

Comunicação por Passagem de Mensagem

Vamos fortalecer alguns conceitos da Comunicação por Passagem de Mensagem:

Remetente (Sender ou Emissor) - É o processo ou componente que gera e envia a mensagem.

- Ele é o iniciador da comunicação.
- Prepara a mensagem (payload + metadados) e envia pela rede.

- Pode ser um servidor, um cliente, ou um serviço intermediário.

Exemplo: Um app mobile enviando uma requisição para um servidor de autenticação.

Receptor (Receiver) - É o processo ou componente que recebe a mensagem.

- Ele escuta ativamente ou fica pronto para receber a mensagem.
- Quando a mensagem chega, ele a interpreta e toma uma ação (como responder ou processar os dados).

Exemplo: Um servidor que recebe a requisição de login e valida as credenciais.

Mensagem - É a **unidade de comunicação** entre remetente e receptor.

- Contém dados (ex: comando, informação, solicitação) e metadados.
- Pode ser pequena (ex: "ping") ou grande (ex: envio de um vídeo).

Componentes de uma mensagem típica:

- Identificador único
- Origem e destino
- Tipo da mensagem (ex: solicitação, resposta, erro)
- Dados úteis (payload)

Canal de Comunicação (Communication Channel) - É o **meio** através do qual a mensagem viaja.

- Pode ser uma conexão de rede TCP, uma fila de mensagens (como RabbitMQ), um socket, HTTP, etc.
- Garante, ou não, características como: ordem, entrega garantida, segurança.

Protocolo de Comunicação - É o **conjunto de regras** que define:

- Como as mensagens são formatadas
- Como são enviadas e recebidas
- Como erros são tratados
- Como a conexão é estabelecida ou terminada

Exemplos de protocolos:

- TCP, UDP (baixo nível)
- HTTP, WebSocket, gRPC (alto nível)

Protocolo de Comunicação (IPC) em Sistemas Distribuídos: Um protocolo de comunicação é um conjunto de regras, formatos e convenções que define como duas ou mais entidades (processos, sistemas ou dispositivos) se comunicam entre si.

Tipos de IPC em Sistemas Distribuídos:

HTTP/HTTPS

- Comunicação padrão na web e APIs REST.
- Baseado em texto (human-readable).
- Requisição-resposta (síncrono).
- Suporte a segurança via TLS (HTTPS).

gRPC

- Framework de chamadas remotas (RPC) da Google.
- Usa HTTP/2 como transporte (binário, multiplexado).

- Suporta comunicação síncrona e streaming.
- Usa Protocol Buffers para serialização eficiente.

AMQP

- Advanced Message Queuing Protocol.
- Focado em mensagens assíncronas confiáveis.
- Usado por sistemas de fila como RabbitMQ.

MQTT

- Protocolo leve para comunicação Pub/Sub.
- Usado em IoT (Internet of Things).
- Comunicação eficiente em redes instáveis.

WebSocket

- Comunicação bidirecional em tempo real sobre TCP.
- Abre uma conexão persistente (diferente de HTTP tradicional).

TCP vs UDP

- TCP: Entrega confiável, ordenada (ideal para dados críticos).
- UDP: Entrega rápida, sem garantias (ideal para áudio/vídeo em tempo real).

Tipos de Comunicação por Destinatário - modos de endereçamento ou modos de entrega de mensagem em sistemas distribuídos.

Em sistemas distribuídos (e redes em geral), as mensagens podem ser enviadas **para diferentes quantidades de receptores**, dependendo da estratégia de comunicação.

Unicast

Definição:

- Comunicação **um para um** (1:1).
- Um remetente envia a mensagem para **um único receptor específico**.

Características:

- Privado e direto.
- Mais controle sobre a entrega.
- Mais custo de rede se precisar falar com muitos (pois replica a mensagem).

Exemplo prático:

- Você acessando um site — seu navegador se conecta diretamente ao servidor específico.

[Emissor] ---> [Receptor]

Multicast

Definição:

- Comunicação **um para muitos** (1:N), mas **não para todos**.

- Um remetente envia uma mensagem para **um grupo específico** de receptores interessados.

Características:

- Mais eficiente do que vários unicasts separados.
- Só membros do grupo recebem.
- Exige suporte de rede/configuração para funcionar corretamente.

Exemplo prático:

- Transmissão de vídeo ao vivo para um grupo de usuários inscritos.
- Atualizações em jogos multiplayer.

[Emissor] ---> [Grupo de Receptores]

Broadcast

Definição:

- Comunicação **um para todos** (1:Todos).
- A mensagem é enviada para **todos os nós** da rede, sem exceção.

Características:

- Muito custo de rede (todos processam).
- Nem sempre desejável (gasto e ruído).

Exemplo prático:

- DHCP (quando uma máquina entra na rede e pede IP, ela envia broadcast).
- Mensagem de "quero me registrar" em redes locais.

[Emissor] ---> [Todos na rede]

Anycast (menos falado, mas importante)

Definição:

- Comunicação **um para um**, **mas** escolhendo o receptor **mais próximo** ou **mais disponível** entre vários possíveis.

Características:

- Alta performance e redundância.
- Balanceamento de carga natural.

Exemplo prático:

- Servidores DNS públicos (como o 8.8.8.8 da Google): você consulta o mesmo IP, mas é atendido pelo servidor geograficamente mais próximo.

[Emissor] ---> [Receptor mais próximo de um grupo]

Métodos de comunicação - Existem dois grandes métodos de comunicação entre processos distribuídos:

Comunicação Síncrona (Tipicamente mais confiável)

Definição:

- No envio síncrono, o remetente envia a mensagem e espera uma resposta ou uma confirmação antes de continuar seu processamento.

- O receptor também está disponível e pronto para responder imediatamente.

Características:

- Comunicação bloqueante: o emissor *pausa* até receber uma resposta.
- Facilidade de programação (modelo mais próximo de chamada de função local).
- Melhor controle de ordem, consistência e garantia de entrega.
- Tende a ser mais lento em redes instáveis (porque depende da resposta).

Exemplos de tecnologias:

- **RPC (Remote Procedure Call)**
- **gRPC (Google RPC)**
- **Chamadas HTTP/REST tradicionais**

Analogia prática: Você liga para alguém no telefone e só desliga depois que a pessoa atende e vocês terminam a conversa.

Comunicação Assíncrona (Tipicamente não tão confiável)

Definição:

- No envio assíncrono, o remetente envia a mensagem e continua seu trabalho sem esperar uma resposta imediata.
- O receptor pode processar a mensagem depois, quando estiver disponível.

Características:

- Comunicação não bloqueante: o emissor segue seu fluxo normal após enviar.

- Mais escalável: permite lidar com grandes volumes de mensagens sem travar processos.
- Sujeito a problemas de:
    - Perda de mensagens
    - Entrega duplicada
    - Entrega fora de ordem
- Normalmente exige estratégias de reentrega, confirmação manual ou consistência eventual.

Exemplos de tecnologias:

- **Message Queues (RabbitMQ, Kafka, SQS)**
- **Event-driven systems (Sistemas orientados a eventos)**

Analogia prática: Você envia uma carta pelo correio — você não espera resposta imediata, nem sabe se ela vai chegar, nem quando.

Modelos de Organização de Sistemas Distribuídos

São **formas de estruturar os componentes** (clientes, servidores, nós) e **definir quem fala com quem** e **quem faz o quê**.

Existem **três modelos principais**:

Modelo Cliente-Servidor (Client-Server) - Dois papéis claros: Cliente: Solicita serviços ou dados e Servidor: Fornece serviços ou dados.

Características:

- Centralização dos recursos.
- Fácil de controlar e gerenciar.
- Escalabilidade limitada (o servidor pode virar gargalo).

- Muito usado na Web (ex.: navegar num site).

Exemplo prático:

- Aplicação Web: Navegador (cliente) faz requisição HTTP para servidor web (servidor).

[Cliente] ---> [Servidor]
<---

Modelo Peer-to-Peer (P2P) - Todos os nós (peers) podem atuar tanto como clientes quanto como servidores ao mesmo tempo.

Características:

- Descentralizado (sem servidor central).
- Alta resiliência: se um nó falhar, outros continuam operando.
- Difícil de controlar e gerenciar (dinamismo alto).
- Alta escalabilidade.

Exemplo prático:

- BitTorrent: computadores compartilham pedaços de arquivos diretamente uns com os outros.
- Blockchain: nós validam e propagam transações entre si.

[Peer] <--> [Peer] <--> [Peer]

Modelo Multicamadas (Multitier ou N-Tier) - Divisão do sistema em várias camadas, cada uma com sua responsabilidade.

Características:

- Separação de preocupações (Frontend, Backend, Banco de Dados, etc.).
- Melhora a manutenção e escalabilidade.
- Base para arquiteturas modernas (ex.: Microservices).

Exemplo prático:

- **Aplicação 3 camadas:**
    1. Cliente (UI)
    2. Servidor de Aplicação (lógica de negócios)
    3. Banco de Dados (persistência)

[Cliente] ---> [Servidor de Aplicação] ---> [Banco de Dados]