# Sistemas Distribuídos

Resumo

**Rafael Rodrigues**

LEIC
Instituto Superior Técnico
2022/2023

# Contents


**1 Introdução** **3**

**2 Remote Procedure Call (RPC)** **4**
- 2.1 Arquitetura cliente-servidor . . . 4
- 2.2 Interfaces de comunicação . . . 4
  - 2.2.1 UDP . . . 4
  - 2.2.2 TCP . . . 4
- 2.3 Serialização . . . 4
- 2.4 Semânticas de execução . . . 5
- 2.5 Serviços de Nomes . . . 5
- 2.6 Web Services . . . 6
  - 2.6.1 SOAP . . . 6
  - 2.6.2 REST . . . 6
- 2.7 gRPC . . . 7
  - 2.7.1 Protocol Buffers (protobuf) . . . 7
  - 2.7.2 Stubs . . . 7
  - 2.7.3 Run-time . . . 7
  - 2.7.4 Gestor de nomes . . . 7

**3 Sincronização de Relógios** **8**
- 3.1 Algoritmo de Cristian . . . 8
- 3.2 Algoritmo de Berkeley . . . 8
- 3.3 Network Time Protocol (NTP) . . . 8

**4 Relógios Lógicos** **9**
- 4.1 Lamport . . . 9
- 4.2 Vector clocks . . . 9

**5 Registos** **9**
- 5.1 Modêlos de Coerência . . . 9
- 5.2 Algoritmo ABD . . . 10

**6 Espaços de tuplos** **11**
- 6.1 Implementação do Put . . . 11
- 6.2 Implementação do Read . . . 11
- 6.3 Implementação do Take . . . 11

**7 Estados Globais** **12**

**8 Eleição de líder** **12**
- 8.1 Algoritmo baseado em anel . . . 12
- 8.2 Algoritmo "bully" . . . 12

**9 Exclusão Mútua** **12**
- 9.1 Algoritmo de Ricart-Agrawala . . . 12
- 9.2 Algoritmo de Maekawa . . . 12

**10 Difusão em ordem total** **13**

**11 Replicação** **14**
11.1 Coerência forte . . . . . 14
11.1.1 Primary-backup (replicação passiva) . . . . . 14
11.1.2 Replicação de máquina de estados (replicação ativa) . . . . . 14
11.1.3 Primary-backup vs RME . . . . . 14
11.2 Checkpoint recovery . . . . . 14
11.3 Coerência fraca . . . . . 15
11.3.1 Gossip . . . . . 15
11.3.2 Bayou . . . . . 15

**12 Consenso** **16**
12.1 Floodset consensus . . . . . 16

**13 Transações distribuídas** **17**
13.1 2-phase commit (2PC) . . . . . 17

**14 Segurança** **18**

# 1 Introdução

**Sistema Distribuído** - Sistema de componentes software ou hardware localizados em computadores ligados em rede, que comunicam e coordenam as suas ações através de troca de mensagens.

# 2 Remote Procedure Call (RPC)

Estrutura a programação distribuída com base na chamada (pelo clientes) de procedimentos que se executam remotamente (no servidor).
O cliente envia um pedido e o servidor envia uma resposta.

## 2.1 Arquitetura cliente-servidor

- Servidores mantêm recursos e server pedidos de operações sobre esses recursos.
- Servidores podem ser clientes de outros servidores.
- Simples e permite distribuir sistemas centralizados muito diretamente.
- Limitado pela capacidade do servidor e pela rede que o liga aos clientes.

## 2.2 Interfaces de comunicação

### 2.2.1 UDP

- **Socket sem ligação**:
  - Normalmente serve mais de 2 interlocutores (1 para n).
  - Normalmente não fiável (perdas, duplicação, reordenação).
- **Socket Datagram**: canal sem ligação, bidirecional, não fiável, interface tipo mensagem.
- Maior eficiência.
- Menor overhead.

### 2.2.2 TCP

- **Socket com ligação**:
  - Normalmente serve 2 interlocutores.
  - Normalmente fiável, bidirecional e garante sequencialidade.
- **Socket Stream**: canal com ligação, bidirecional, fiável, interface tipo sequência de octetos.
- Oferece um canal fiável sobre uma rede não fiável.
- Para cada invocação remota precisamos de 2 pares de mensagens adicionais.
- Gestão de fluxo é redundante para as invocações simples do nosso sistema.
- ACKs desnecessários.
- É usado caso as mensagens sejam maiores que um datagrama UDP.

## 2.3 Serialização

## 2.4 Semânticas de execução

**Função idempotente** - Operação que, se executada repetidamente, produz o mesmo estado no servidor e resultado devolvido ao cliente do que se só executada 1 vez

- **Talvez (maybe)** - Protocolo não pretende tolerar nenhuma falta pelo que nada faz para recuperar de uma situação de erro.
  - O stub cliente não recebe uma resposta num prazo limite
  - O timeout expira e a chamada retorna com erro
  - Neste caso o cliente não sabe se o pedido foi executado ou não
- **Pelo-menos-uma-vez (at-least-once)** - Para serviços com funções idempotentes.
  - O stub cliente não recebe uma resposta num prazo limite
  - O timeout expira e o stub cliente repete o pedido até obter uma resposta
  - Se receber uma resposta o cliente tem a garantia que o pedido foi executado pelo menos uma vez
  - Para evitar que o cliente fique permanentemente bloqueado em caso de falha do servidor existe um segundo timeout mais amplo
- **No-máximo-uma-vez (at-most-once)** - Protocolo de controlo tem de: (1) identificar os pedidos para detetar repetições no servidor; (2) manter estado no servidor acerca dos pedidos em curso ou que já foram atendidos.
  - O stub cliente não recebe uma resposta num prazo limite
  - O timeout expira e o stub cliente repete o pedido
  - O servidor não executa pedidos repetidos
  - Se receber uma resposta, o cliente tem a garantia que o pedido foi executado no máximo uma vez
- **Exatamente-uma-vez (exactly-once)** - Servidor e cliente com funcionamento transacional
  - O stub cliente não recebe uma resposta num prazo limite
  - O timeout expira e o stub cliente repete o pedido
  - O servidor não executa pedidos repetidos
  - Se o servidor falhar existe a garantia de fazer rollback ao estado do servidor de modo a não ficar com pedidos "a meio"

## 2.5 Serviços de Nomes

## 2.6 Web Services

### 2.6.1 SOAP

Desvantagens:

- Obriga ao uso de XML
- Desempenho e escalabilidade limitados
  - Representações em XML são longas, marshalling/unmarshalling é pesado
  - Não permite caching de dados do lado do cliente
- Relativamente difícil de implementar

### 2.6.2 REST

Vantagens:

- Permite uso de alternativas ao XML, como o JSON (mais eficiente).
- Clientes que pretendam agir sobre um recurso, recebem cópia por inteiro dos dados
  - Permite caching no cliente (melhor escalabilidade)

## 2.7 gRPC

É um mecanismo de RPC moderno, com estrutura semelhante aos RPC tradicionais, mas desenvolvido com foco em serviços cloud.
Usado em comunicação para sistemas distribuídos com requisitos de baixa latência e elevada escalabilidade.
É um protocolo **eficiente**, **preciso** e **independente da linguagem** de programação.

Características do sistema gRPC:

- O gRPC usa uma IDL (protobuf) para definir os tipos de dados e as operações.
- Disponibiliza ferramenta de geração de código a partir do IDL
  - Trata da conversão de dados
  - Gestão da invocação remota
- Permite ter chamadas remotas síncronas e assíncronas
  - **Chamadas síncronas** - esperam pela resposta
  - **Chamadas assíncronas** - a resposta é verificada mais tarde

### 2.7.1 Protocol Buffers (protobuf)

- Os protocol buffers (protobuf) são uma IDL para descrever mensagens e operações
  - Permite estender/modificar os esquemas, mantendo compatibilidade com versões anteriores
  - O protobuf é um formato canónico
  - A apresentação de dados tem uma estrutura explícita
- O compilador protoc converte a IDL em código para uma grande variedade de linguagens de programação
- As camadas mais baixas do gRPC não dependem da IDL
  - Em teoria é possível usar alternativas ao protobuf

Formato **mais eficiente** que JSON e XML.
Permite uma boa integração com várias linguagens

### 2.7.2 Stubs

Objeto usado pelo cliente, que expõe a interface do serviço. A invocação de um método no stub vai serializar e traduzir os dados do pedido. Segue-se depois a invocação remota. É no stub que as restrições de interface e dos tipos de dados são verificadas.

### 2.7.3 Run-time

### 2.7.4 Gestor de nomes

# 3 Sincronização de Relógios

- **Externa** - relógios dos processos são sincronizados através de uma **referência externa**
- **Interna** - relógios dos processos de um sistema **sincronizam-se entre si**

## 3.1 Algoritmo de Cristian

Os relógios dos clientes são sincronizados pelo relógio de um **servidor de tempo** (sincronização externa).

1. Servidor $S$ lê o valor dos outros relógios.
   $T_{S_i} = T_{env_i} + T_{rec_i}/2$
   $delta_i = T_S - T_i$
   $erro_i = \pm\ RTT_i/2$
2. Indica a todos os participantes para ajustarem o seu relógio (incluindo o seu).
   $ajuste_i = \overline{T} + delta_i$

Diferença máxima = Soma dos dois maiores valores de erro
Precisão = $\pm\,(RTT/2 - min)$

## 3.2 Algoritmo de Berkeley

1. É escolhido um líder através de um processo de eleição.
2. O líder pergunta os tempos aos seus servidores.
3. O líder calcula o tempo de cada máquina tendo em atenção o RTT.
4. O líder calcula a média dos tempos, ignorando os outliers.
5. Envia o valor (positivo ou negativo) que cada máquina deve ajustar.

## 3.3 Network Time Protocol (NTP)

# 4 Relógios Lógicos

Relacão happens-before/aconteceu-antes ($\rightarrow$)

1. se $a$ e $b$ são eventos do mesmo processo, se $a$ ocorre antes de $b$, então $a \rightarrow b$
2. se $a$ indica um evento envio de mensagem, e $b$ o evento da receção dessa mensagem, então $a \rightarrow b$

**Transitividade**: se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$
**Eventos concorrentes**: se nem $a \rightarrow b$, nem $b \rightarrow a$, então $a \parallel b$

## 4.1 Lamport

1. se $a \rightarrow b$, então $C(a) < C(b)$
   - se os eventos ocorrerem no mesmo processo, e $a$ ocorre *antes* de $b$, então $C(a) < C(b)$
   - se $a$ for o evento envio de mensagem e $b$ a sua receção, então $C(a) < C(b)$
2. o valor de $C(e)$ *nunca decresce*
   - As correções ao relógio devem ser feitas sempre por incrementos

## 4.2 Vector clocks

1. se $C(a) < C(b)$, então $a \rightarrow b$
   - $V_a < V_b$ se pelo menos um elemento de $V_a$ for menor e nenhum for maior que $V_b$

# 5 Registos

## 5.1 Modêlos de Coerência

- Safe
  - Apenas um processo pode escrever e vários podem ler (1 escritor - múltiplos leitores)
  - Se uma leitura não for concorrente com uma escrita, lê o último valor escrito.
  - Se uma leitura for concorrente com uma escrita, pode retornar um valor arbitrário
- Regular
  - Apenas um processo pode escrever e vários podem ler (1 escritor - múltiplos leitores)
  - Se uma leitura não for concorrente com uma escrita, lê o último valor escrito.
  - Se uma leitura for concorrente com uma escrita, retorna o valor anterior ou o valor que está a ser escrito
- Atomic
  - O resultado da execução é equivalente ao resultado de uma execução em que todas as escritas e leituras ocorrem instantaneamente num ponto entre o início e o fim da operação.

## 5.2 Algoritmo ABD

# 6 Espaços de tuplos

Operações num espaço de tuplos:

- Put: coloca um tuplo no espaço
- Read: lê um tuplo do espaço
- Take: lê e remove um tuplo do espaço

O Read e Take são **operações bloqueantes** (se não existir o tuplo) e aceitam "wildcards".

O Take, sendo bloqueante, permite sincronizar processos.

A exclusão mútua pode ser concretizada através de um tuplo $< token >$

- um processo remove (Take) do espaço antes de aceder ao recurso
- e volta a colocar no espaço depois de aceder ao recurso

Abstração muito elegante pois permite partilha de memória e sincronização através de uma interface uniforme.

Num espaço de tuplos, podem existir várias cópias do mesmo tuplo.

O equivalente a uma escrita é feito através de um Take seguido de um Put.

O Take permite fazer operações que noutros tipos de memória partilhada mais "convencionais" requerem uma instrução do tipo compare-and-swap.

A operação Take requer ordem total.

## 6.1 Implementação do Put

1. A réplica onde o pedido é feito faz multicast do Put para todas as outras réplicas e responde ao cliente
2. Ao receber o pedido, as réplicas inserem o tuplo e enviam ACK
3. O passo 1 é repetido até a réplica original receber o ACK de todas as outras réplicas

## 6.2 Implementação do Read

1. A réplica onde o pedido é feito faz multicast do Read para todas as outras réplicas
2. Ao receber o pedido, as réplicas retornam o tuplo que faz "match"
3. Ao receber a primeira resposta, a réplica retorna ao cliente, ignorando as respostas seguintes
4. O passo 1 é repetido até se receber uma resposta

## 6.3 Implementação do Take

# 7 Estados Globais

Dado um evento $e_2$ pertencente ao corte, se $e_1 \rightarrow e_2$ então, para o corte ser coerente, $e_1$ tem de pertencer ao corte. Se $e_1 \rightarrow e_2$ e apenas $e_1$ pertencer ao corte, este continua a ser coerente.

# 8 Eleição de líder

## 8.1 Algoritmo baseado em anel

## 8.2 Algoritmo "bully"

No melhor cenário, o algoritmo de "Bully" consegue terminar a eleição trocando menos mensagens que o de anel.

# 9 Exclusão Mútua

## 9.1 Algoritmo de Ricart-Agrawala

## 9.2 Algoritmo de Maekawa

# 10 Difusão em ordem total

# 11 Replicação

## 11.1 Coerência forte

### 11.1.1 Primary-backup (replicação passiva)

1. FE envia pedido ao primário (usando no-máximo-uma-vez)
2. Primário ordena os pedidos por ordem de chegada
3. Primário executa pedido e guarda resposta
4. Primário envia aos secundários: (novo estado, resposta, id.pedido) e espera por ack de todos os secundários
5. Primário responde ao front-end, que retorna ao cliente

### 11.1.2 Replicação de máquina de estados (replicação ativa)

1. FE envia pedido (juntamente com um relógio lógico de Lamport) a **todas as réplicas usando difusão em ordem total** (mas sem esperar pelas réplicas em falha)
2. O algoritmo de difusão entrega o pedido a todas (quando chegar a ver deste pedido na ordem total)
3. Cada réplica executa o pedido
4. Cada réplica responde ao cliente, que usa a primeira resposta

Os pedidos necessitam de ser entregues por ordem total.

### 11.1.3 Primary-backup vs RME

- Primário-secundário
  - Suporta operações não deterministas (o líder decide o resultado)
  - Se o líder produzir um valor errado, este valor é propagado para as réplicas
- Replicação máquina de estados
  - Se uma réplica produzir um valor errado não afeta as outras réplicas
  - A operações necessitam de ser deterministas

## 11.2 Checkpoint recovery

Técnica que para tolerar faltas guarda o estado da aplicação periodicamente, de forma a poder relançar a mesma sem ter de começar do início.

## 11.3 Coerência fraca

### 11.3.1 Gossip

Pedidos de leitura (query):

- Réplica verifica se $pedido.prev \leq valueTS$

Pedidos de escrita (update):

- responde e incrementa $pedido.prev_{[i]}$

### 11.3.2 Bayou

# 12 Consenso

- Conjunto de N processos
- Cada processo propõe um valor (input)
- Todos os processos decidem o mesmo valor (output)
  - O valor decidido deve ser um dos valores propostos
  - Pode ser qualquer um dos valores propostos

## 12.1 Floodset consensus

# 13 Transações distribuídas

Uma transação distribuída implica executar múltiplas suboperações em nós diferentes.

- Caso tudo corra bem, todas as suboperações são confirmadas **(commit)**.
- Caso algo corra mal, todas as suboperações são anuladas **(abort)**.

Propriedades ACID:

- Atomic - Atomicidade
- Consistent - Coerência
- Isolated - Isolamento
- Durable - Durabilidade

## 13.1 2-phase commit (2PC)

1. O coordenador envia uma mensagem denominada "prepare" a todos os participantes.
2. Se o participante pode confirmar a transação envia "ok" ao coordenador.
3. Se o participante não puder confirmar a transação envia "not-ok" ao coordenador.
4. Se o coordenador receber "ok" de todos os participantes decide:
    - confirmar a transacção
    - Acrescenta ao seu "log" uma entrada indicando que a transação foi confirmada
    - e avisa os outros (que registam no seu log)
5. Se o coordenador receber "not-ok" de pelo menos um participante decide:
    - abortar a transacção
    - Acrescenta ao seu "log" uma entrada indicando que a transação foi abortada
    - e avisa os outros (que registam no seu log)

Este algoritmo é bloqueante se o coordenador falha

# 14 Segurança