INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Algoritmos de replicação e consistência

Por que replicar dados?

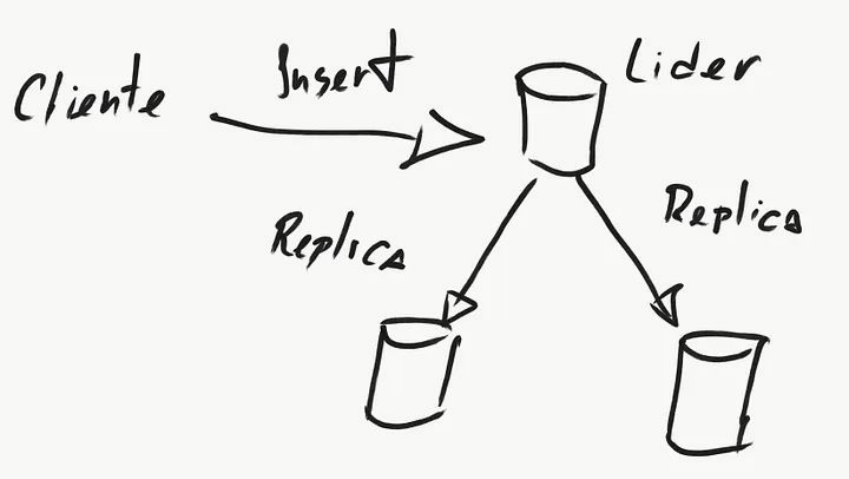
- A replicação de dados é uma técnica importante em sistemas distribuídos para
  - Aprimorar a confiabilidade
    - Possibilidade de continuar trabalhando após uma queda de uma réplica por meio da comutação para uma das outras réplicas.
    - Melhor proteção contra dados corrompidos.
  - Melhorar o desempenho
    - Ampliação de um sistema em quantidade e área geográfica
- Para garantir o sucesso nessa operação, precisamos manter réplicas consistentes, assegurando que quando uma cópia for atualizada, todas as outras também deverão ser.

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Estratégias mais comuns de replicação

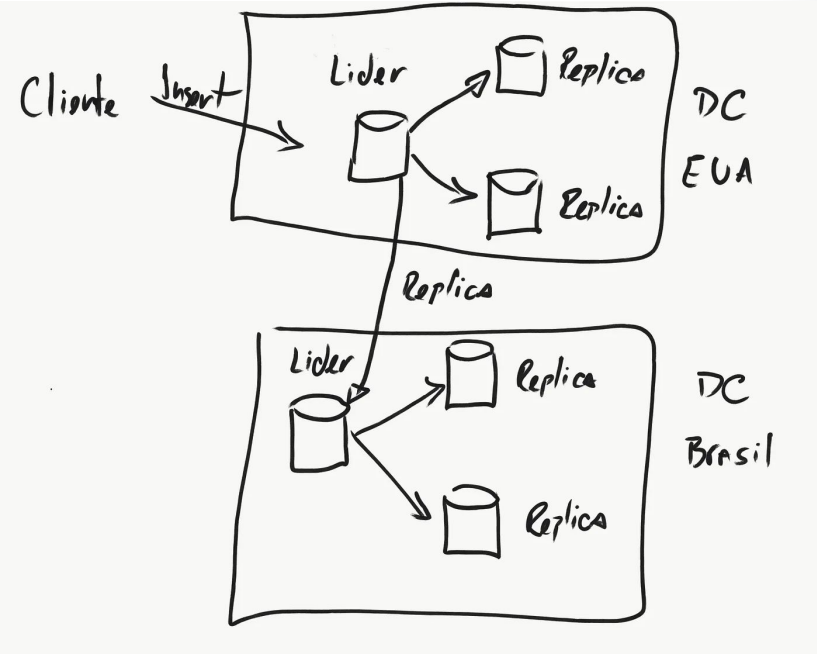
Replicação de líder único

- Estratégia mais comum para se garantir disponibilidade em caso de falha.
- O cluster sempre elege um líder, usando algum que será o único responsável por aceitar requisições de escrita. As requisições de leitura são aceitas por qualquer nó, líder ou réplica.
- O nó líder commita a escrita e envia uma resposta ao cliente. Em seguida, de maneira assíncrona, é enviado aos nós réplicas a mesma operação para que o mesmo dado seja escrito nos outros nós e manter todas as réplicas consistentes.
- A escrita nas réplicas também pode ser feita de maneira síncrona, com o cliente recebendo a resposta da requisição somente depois da escrita ter sido replicada em todos os nós. Porém, isso proporciona uma performance ruim que pode causar enorme gargalo.



Replicação com múltiplos líderes

- Estratégia mais comum em ambientes multi-datacenter, caso exista versões da aplicação rodando em mais de um datacenter diferente, para acesso mais rápido em diferentes regiões.
- É possível ter um cluster em múltiplas regiões e ter um líder por região.
- O líder que commitou o dado replica a operação de escrita com os líderes das outras regiões, que por sua vez, enviam os dados para seus respectivos nós réplicas.



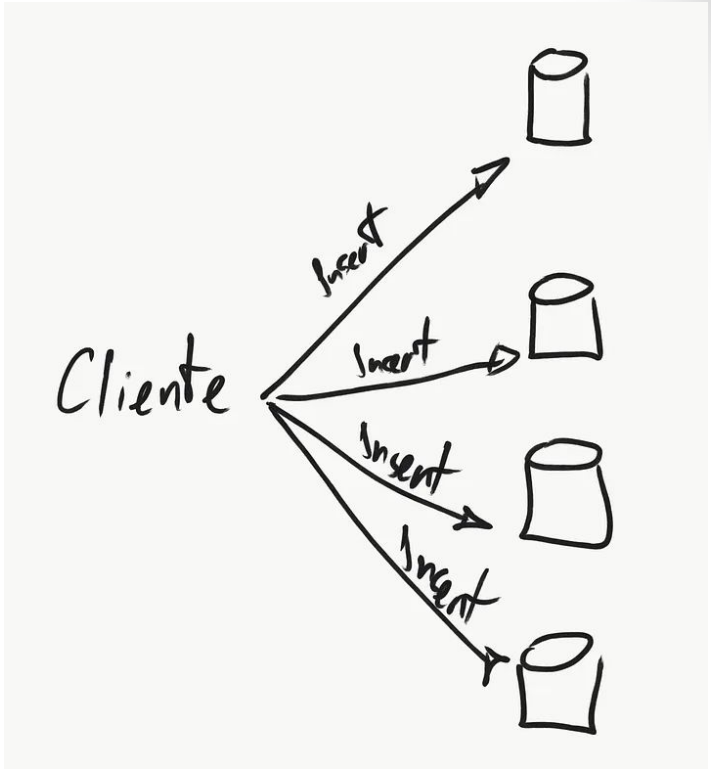
Replicação sem líder

- O cliente envia a requisição para vários nós, como não há um líder coordenando as operações, é necessário utilizar uma técnica chamada Quorum, que consiste basicamente em ter confirmação de uma maioria dos nós no cluster.

- Uma operação em um cluster com N nós, para ser considerada bem sucedida, deve ser confirmada por no mínimo K nós. O número K é o que chamamos de Quorum. Esse número K pode ser diferente para operações de leitura e escrita, ou ser igual para ambas operações, desde que esse número satisfaça a seguinte condição:

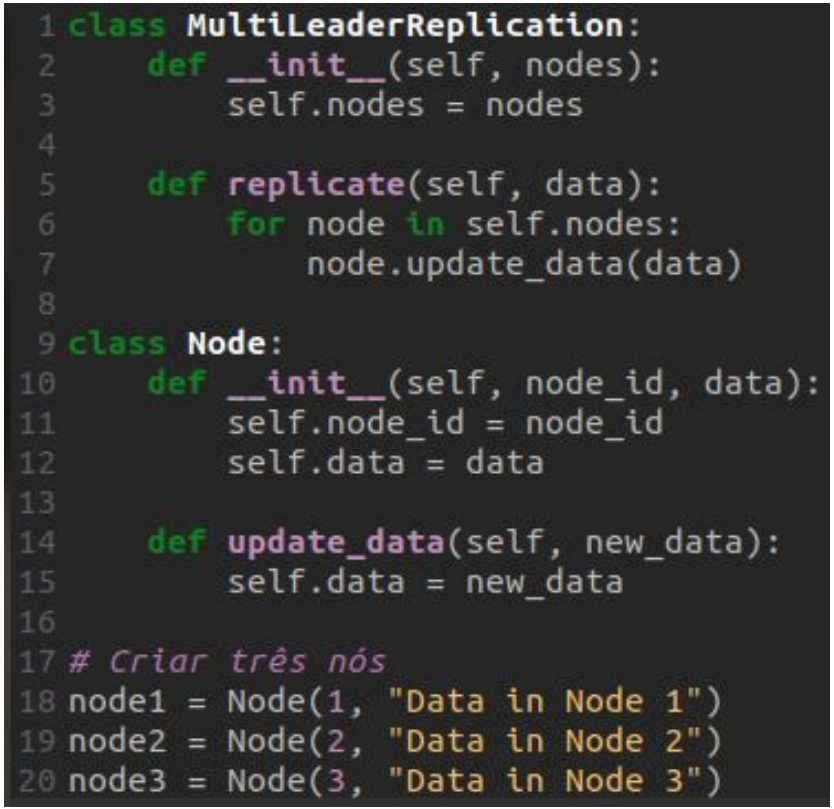
  K_leitura + K_escrita > N

- Você pode definir o valor de K como sendo (N + 1) / 2, arredondado para cima. Então caso haja 4 nós no cluster, as operações teriam que ser confirmadas por um Quorum de ao menos 3 nós.



Exemplo:

Replicação com múltiplos líderes



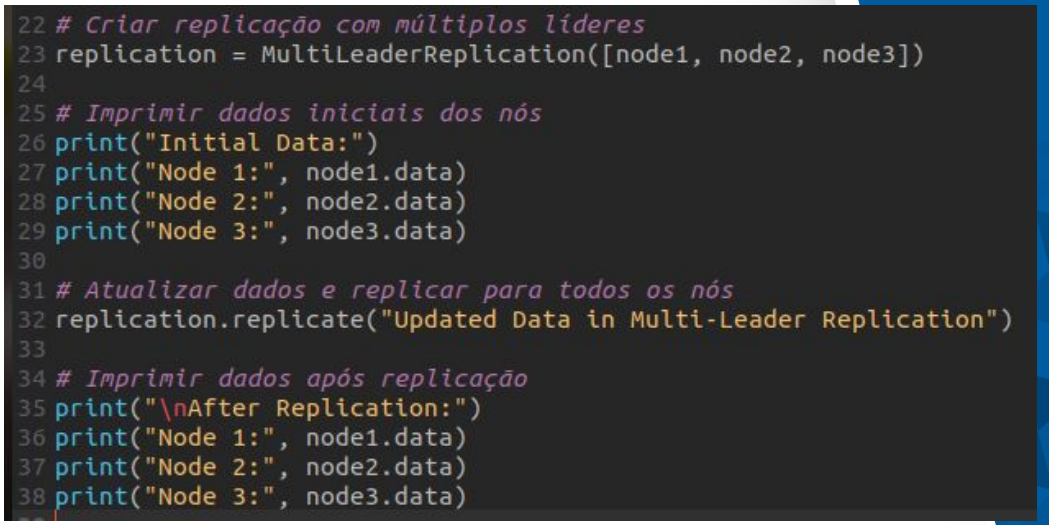
```python
class MultiLeaderReplication:
    def __init__(self, nodes):
        self.nodes = nodes

    def replicate(self, data):
        for node in self.nodes:
            node.update_data(data)

class Node:
    def __init__(self, node_id, data):
        self.node_id = node_id
        self.data = data

    def update_data(self, new_data):
        self.data = new_data

# Criar três nós
node1 = Node(1, "Data in Node 1")
node2 = Node(2, "Data in Node 2")
node3 = Node(3, "Data in Node 3")
```



```python
# Criar replicação com múltiplos líderes
replication = MultiLeaderReplication([node1, node2, node3])

# Imprimir dados iniciais dos nós
print("Initial Data:")
print("Node 1:", node1.data)
print("Node 2:", node2.data)
print("Node 3:", node3.data)

# Atualizar dados e replicar para todos os nós
replication.replicate("Updated Data in Multi-Leader Replication")

# Imprimir dados após replicação
print("\nAfter Replication:")
print("Node 1:", node1.data)
print("Node 2:", node2.data)
print("Node 3:", node3.data)
```

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Estratégias de consistência

Modelo de consistência centrados em dados

É um contrato entre um data store (armazém de dados) distribuído e os processos, no qual o data store define precisamente o resultado de operações concorrentes de leitura e escrita.

Exemplo: Algoritmo de Paxos

- Consistência contínua
  - Busca manter um certo nível de consistência de dados mesmo quando ocorrem atualizações concorrentes em diferentes partes do sistema. Em vez de garantir uma consistência rigorosa o tempo todo, a consistência contínua permite uma flexibilidade controlada, onde os sistemas podem operar com diferentes níveis de consistência com base nos requisitos e no contexto da aplicação.
  - Réplicas podem diferir em relação a seus valores numéricos
  - Réplicas podem diferir em relação à desatualização relativa
  - Pode haver diferenças no número e na ordem das operações de atualizações realizadas

Modelo de consistência centrados em dados

- Consistência sequencial
  - O resultado de qualquer execução é o mesmo, como se as operações de todos os processos fossem executadas na mesma ordem sequencial e as operações de cada processo aparecessem na mesma ordem especificada pelo programa.
  - Esse modelo enfatiza a preservação da ordem sequencial das operações conforme são executadas, independentemente de como elas ocorrem fisicamente nos nós do sistema.
  - Principais pontos:
    - Ordem das operações
    - Garantia de consistência (As operações acontecem na ordem real em que foram realizadas)
    - Independe de latências ou atrasos na rede
    - Restrição de acesso (Pode causar bloqueios temporários para garantir a ordem sequencial)
    - Impacto na escalabilidade e desempenho

Modelo de consistência centrados em dados

- Consistência causal
  - Concentra-se na preservação da ordem causal as operações.
    - Garante que se uma operação A causou uma operação B, então a operação B não será considerada como ocorrendo antes da operação A.
  - Esse modelo não exige uma ordem global estrita para as operações, mas preserva a relação de causalidade entre elas.
  - Principais pontos
    - Preservação da ordem causal
    - Relações de causa e efeito
    - Flexibilidade e desempenho (Operações independentes podem ocorrer em paralelo)
    - Garantia de ordem parcial em relação a operações dependentes

Modelos de consistência centrados no cliente

Tem como objetivo mostrar que manter a consistência em todo o sistema seja desnecessário se nos concentramos no que os clientes precisam, ao invés daquilo que deve ser mantido pelos servidores.

- Leituras monotônicas
    - As leituras não retrocedem no tempo; ou seja, se um cliente leu um determinado valor, todas as leituras subsequentes desse cliente verão valores iguais ou mais recentes.
- Escritas Monotônicas
    - As operações de escrita de um cliente são aplicadas em uma ordem consistente, assegurando que a ordem das escritas do cliente seja preservada em todos os nós.
- Read-Your-Writes (Leia suas escritas)
    - Garante que um cliente sempre veja suas próprias operações de escrita ao realizar leituras subsequentes, proporcionando uma visão consistente de suas ações.
- Write-Follows-Reads (Escrita segue leituras)
    - As operações de escrita de um cliente ocorrem após todas as operações de leitura prévias do mesmo cliente, mantendo a sequência de leitura e escrita.

Referências Bibliográficas

[1] TANENBAUM, Andrew S.; VAN STEEN, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2. ed. São Paulo: Pearson Addison Wesley, 2007.

[2] FERREIRA, Breno. Dados distribuídos: Replicação de dados. Disponível em: https://brenocferreira.medium.com/dados-distribu%C3%ADdos-replica%C3%A7%C3%A3o-de-dados-3d45b0914e71. 07 de julho de 2020.