UNIVERSIDADE DE SÃO PAULO - USP
Instituto De Ciências Matemáticas e de Computação - ICMC

---

Profa. Elaine Parros Machado de Sousa

# Trabalho 2 de Arquitetura de SGBDs

## SCC0243 - Arquitetura de Sistemas Gerenciadores de Base de Dados

Lívia Lelis 12543822
Lourenço de Salles Roselino 11796805
Samuel Figueiredo Veronez 12542626

São Carlos - SP
06/2025

# Sumário

# 1 Introdução

O crescimento das aplicações digitais e o aumento do volume de dados exigiram novas soluções para armazenamento eficiente e escalável. Os bancos de dados NoSQL surgiram para atender a essas necessidades, oferecendo flexibilidade, alta disponibilidade e desempenho em ambientes distribuídos.

Nesse contexto, para o segundo projeto da disciplina SCC0243 - Arquitetura de Sistemas Gerenciadores de Base de Dados, decidimos comparar a eficiência de diferentes configurações do Apache Cassandra executando um benchmark padronizado.

O objetivo do projeto é implementar e avaliar as diferenças de performance e overheads de diferentes configurações de uma base de dados NoSQL Apache Cassandra. Com esse objetivo, utilizaremos o benchmark Yahoo! Cloud Serving Benchmarking (YCSB).

Todos os códigos e instruções para a execução do projeto estão disponíveis no repositório do projeto, disponível em <https://github.com/VH-Evil-Inc/asgbd>.

# 2 Fundamentação Teórica

## 2.1 Banco de Dados Distribuído

Um banco de dados distribuído é um banco de dados cujos dados estão armazenados em diferentes locais, mas que se intercomunicam de forma a parecerem um único banco de dados para o usuário.

A distribuição dos dados entre várias máquinas possibilita melhorar a escalabilidade, a disponibilidade e a tolerância a falhas do sistema. Entretanto, essa distribuição implica em custos adicionais de comunicação e sincronização entre os nós do sistema.

No contexto da distribuição dos dados em bancos de dados distribuídos, tanto as tuplas quanto os atributos que as compõem podem ser fragmentados entre os nós do sistema, e cópias desses dados podem ser replicadas em mais de um nó.

### 2.1.1 Replicação

No que diz respeito à redundância dos dados entre os nós do sistema, a replicação consiste na cópia de um fragmento de dados em mais de um nó. Essa cópia pode ser feita de forma parcial ou completa, e uma maior redundância dos dados implica em maior disponibilidade e tolerância a falhas do sistema, mas aumenta os custos de armazenamento e sincronização.

### 2.1.2 Alocação de Dados

A alocação de dados em um banco de dados distribuído é o processo pelo qual se decide onde os dados serão armazenados, levando em consideração a configuração dos nós do sistema e as métricas de desempenho desejadas. O processo de alocação de dados pode ser feito de forma centralizada ou descentralizada.

### 2.1.3 Transações

Transações em bases de dados distribuídas envolvem múltiplos nós ou locais onde os dados estão armazenados, mas precisam ser executadas de forma coordenada para garantir as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID). Mesmo que uma transação envolva atualizações em diferentes servidores, ela deve ser concluída integralmente em todos eles ou ser completamente desfeita em caso de falha, mantendo o sistema em um estado consistente. Para garantir essas propriedades, sistemas distribuídos utilizam protocolos especiais, como o protocolo de commit em duas fases (Two-Phase Commit – 2PC), que coordenam a aprovação e execução das operações entre todos os nós participantes. Esses mecanismos são essenciais para evitar inconsistências e garantir a confiabilidade das transações./

## 2.2 Apache Cassandra

Para o desenvolvimento do projeto, foi utilizado o Apache Cassandra, um banco de dados NoSQL distribuído.

### 2.2.1 Estrutura de Dados e Particionamento

Os dados no Cassandra são organizados em keyspaces (equivalentes a bancos de dados), que agrupam tabelas relacionadas. Cada tabela é composta por linhas e colunas, sendo que cada linha é identificada por uma chave primária composta por um partition key e, opcionalmente, colunas de ordenação (clustering columns). O partition key é fundamental para a distribuição dos dados: ele é transformado em um token por meio de um particionador, e esse token determina em qual nó do cluster a linha será armazenada.

A distribuição dos dados é feita de forma que cada nó do cluster seja responsável por um intervalo do espaço de tokens. Quando novos nós são adicionados, o espaço de tokens é redistribuído automaticamente, promovendo o balanceamento de carga e a escalabilidade linear do sistema.

### 2.2.2 Replicação e Tolerância a Falhas

Cassandra implementa replicação configurável por keyspace, permitindo definir o fator de replicação (quantidade de cópias de cada dado) e a estratégia de replicação mais adequada ao ambiente. O mecanismo de replicação, junto à arquitetura descentralizada, garante alta disponibilidade e recuperação automática de falhas, com mecanismos como hinted handoff para garantir consistência e reparo automático de dados entre nós.

### 2.2.3 Consistência e Transações

Um dos grandes diferenciais do Cassandra é a consistência ajustável (tunable consistency), sendo que o usuário pode definir, para cada operação, quantos nós precisam confirmar uma leitura ou escrita para que ela seja considerada bem-sucedida.

Isso permite ajustar entre priorizar consistência forte ou disponibilidade, conforme a necessidade da aplicação. Por padrão, o que diz respeito ao teorema CAP, o Cassandra opera por padrão como um sistema AP (alta disponibilidade e tolerância a partições), mas

pode ser configurado para comportar-se como CP(consistência e tolerância a partições) em cenários específicos.

Cassandra não implementa transações distribuídas entre múltiplas partições, mas oferece suporte a transações leves (lightweight transactions) com isolamento serial em nível de partição.

#### 2.2.4 Escalabilidade

Cassandra é reconhecido pela adição ou remoção de nós poder ser feita sem downtime, e o throughput de leitura e escrita cresce linearmente com o número de nós. Isso o torna ideal para aplicações que exigem alta disponibilidade, tolerância a falhas e capacidade de lidar com grandes volumes de dados distribuídos globalmente.

## 3 Sistema Proposto

O sistema proposto foi desenvolvido para comparar o desempenho de diferentes arquiteturas e configurações de uma base de dados do Apache Cassandra.

O benchmark utilizado será o Yahoo! Cloud Serving Benchmarking (YCSB), desenvolvido pelo Yahoo!, e reconhecido como um padrão do mercado para avaliar desempenho e escalabilidade de sistemas de banco de dados NoSQL.

Para garantir fácil reprodutibilidade e configuração, ambos os ambientes serão configurados utilizando o Docker Compose, sendo executado em um ambiente de nuvem no Digital Ocean mediado pelo Terraform.

### 3.1 Apache Cassandra

O Apache Cassandra é um banco de dados NoSQL distribuído, projetado para lidar com grandes volumes de dados em ambientes distribuídos. Decidimos por comparar o desempenho do Cassandra em diferentes configurações, avaliando como a distribuição de dados, replicação e configuração de nós afetam o desempenho e a escalabilidade do sistema.

As configurações a serem testadas incluem:

- Configuração padrão do Cassandra com um único nó;
- Configuração com 3 nós, sem replicação dos dados;
- Configuração com 3 nós, com replicação dos dados;

### 3.2 Yahoo! Cloud Serving Benchmarking (YCSB)

O YCSB é um benchmark amplamente utilizado para avaliar o desempenho de sistemas de banco de dados NoSQL. Ele permite medir a latência e a taxa de transferência de operações de leitura e escrita em diferentes configurações de banco de dados. Para este projeto, utilizaremos o YCSB para executar uma série de testes em cada configuração do Cassandra, com foco em medir o throughput e a latência das operações.

O YCSB utiliza um modelo de dados simples baseado em chave-valor. O formato padrão do banco de dados é uma tabela chamada geralmente de usertable, que possui:

- Uma chave primária YCSB_KEY;
- Um conjunto de colunas de dados, FIELD0, FIELD1, ..., FIELD9 (por padrão, são 10 campos), cada uma armazenando por padrão valores do tipo string.

O YCSB executa um conjunto padrão de operações que simulam o comportamento de aplicações reais sobre bancos NoSQL ou relacionais. As operações padrão são:

- Insert: Insere um novo registro.
- Read: Lê um registro com base na chave, podendo ler todos os campos ou apenas um campo específico.
- Update: Atualiza um ou mais campos de um registro existente.
- Delete: Remove um registro com base na chave.
- Scan: Faz uma varredura sequencial de múltiplos registros a partir de uma chave inicial, retornando um número configurável de registros.

# 4 Implementação

O tanto o sistema como a execução do benchmark foram implementado utilizando o Docker Compose e o Terraform.

Os scripts de inicialização dos ambientes e execução dos benchmarks foram automatizados, facilitando a replicação dos experimentos e a coleta dos resultados. Os comandos para reproduzir o sistema e executar o benchmark estão disponíveis no readme do repositório.

# 5 Experimentos e Resultados

Para comparar o desempenho das arquiteturas propostas, realizamos o benchmark YCSB com as configurações do Cassandra descritas na Seção de Sistema Proposto. Os resultados estão descritos a seguir, com foco em throughput, latência e overheads de cada configuração.

## 5.1 Resultados do Benchmark YCSB

Tabela 1 – Tempos de benchmark no YCSB

| Configuração | Tempo de Carregamento (ms) | Tempo de Execução (ms) |
|---|---|---|
| Nó único | 386 275 | 625 923 |
| 3 nós sem replicação | 224 126 | 225 164 |
| 3 nós com replicação | 477 892 | 552 229 |

Tabela 2 – Resultados do benchmark YCSB - Único nó

| Operação | Operações | Latência Média (µs) | Latência Mín (µs) | Latência Máx (µs) | 95% (µs) | 99% (µs) |
|---|---|---|---|---|---|---|
| INSERT | 10 000 000 | 1 222,82 | 197 | 148 607 | 1 961 | 5 299 |
| READ | 5 000 753 | 2 296,55 | 220 | 135 295 | 4 583 | 11 775 |
| UPDATE | 4 999 247 | 1 681,11 | 159 | 149 759 | 2 923 | 7 011 |

Tabela 3 – Resultados agregados do benchmark YCSB - 3 nós sem replicação

| Operação | Operações | Latência Média (µs) | Latência Mín (µs) | Latência Máx (µs) | 95% (µs) | 99% (µs) |
|---|---|---|---|---|---|---|
| INSERT | 10 000 000 | 703,01 | 159 | 139 135 | 1 084 | 2 669 |
| READ | 4 998 961 | 759,82 | 210 | 97 151 | 1 205 | 2 307 |
| UPDATE | 5 001 039 | 656,09 | 160 | 95 935 | 1 087 | 1 964 |

Tabela 4 – Resultados agregados do benchmark YCSB - 3 nós com replicação

| Operação | Operações | Latência Média (µs) | Latência Mín (µs) | Latência Máx (µs) | 95% (µs) | 99% (µs) |
|---|---|---|---|---|---|---|
| INSERT | 10 000 000 | 1 514,36 | 186 | 149 119 | 3 843 | 9 103 |
| READ | 5 000 119 | 2 334,83 | 268 | 125 119 | 5 563 | 11 191 |
| UPDATE | 4 999 881 | 1 170,27 | 182 | 114 431 | 3 099 | 5 983 |

## 5.2 Análise Comparativa dos Resultados

A análise dos resultados obtidos evidencia diferenças significativas no desempenho das configurações avaliadas. O cluster com três nós sem replicação apresentou as menores latências médias para todas as operações (INSERT, READ e UPDATE), demonstrando maior eficiência na distribuição da carga e no processamento das requisições.

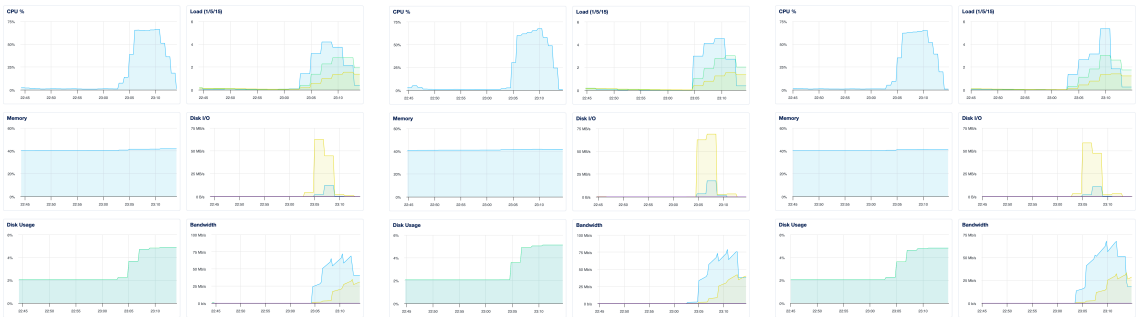
A configuração com replicação no cluster de três nós resultou em um aumento considerável nas médias das latências, especialmente nas operações de escrita e leitura. Esse aumento pode ser atribuído ao overhead inerente ao mecanismo de replicação, que demanda sincronização adicional entre os nós do cluster.

Comparando-se o desempenho do nó único com as demais configurações, observa-se que apresentou métricas semelhantes em relação à configuração de cluster com replicação, entretanto não dispõe dos mesmos níveis de disponibilidade e tolerância a falhas.

### 5.2.1 Análise dos Recursos Utilizados no Cluster

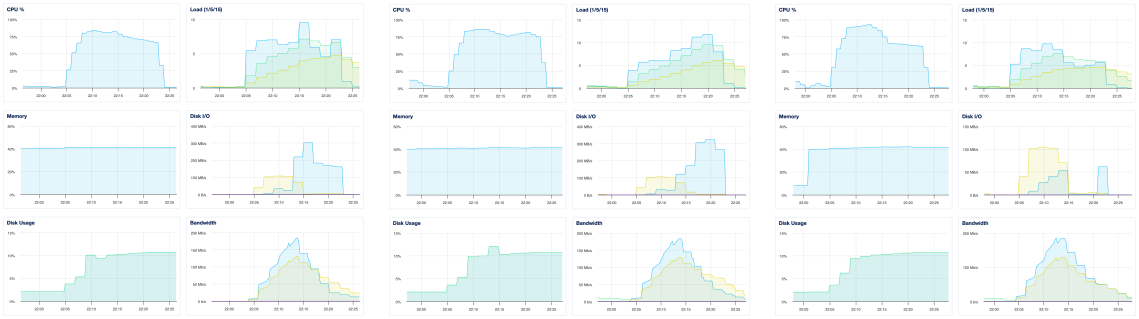
Além das métricas de latência e throughput, decidimos comparar o uso de recursos do cluster durante a execução do benchmark YCSB. Abaixo estão os gráficos de uso de recursos dos nós que compõem cada cluster:

Sem Replicação



Nó 1 Nó 2 Nó 3

Com Replicação



Nó 1 Nó 2 Nó 3

Comparando o gráfico do uso de recursos do cluster sem replicação com o do cluster com replicação, observamos que para as arquiteturas utilizadas, com excessão do uso de memória semelhante, todas métricas apresentam um aumento significativo no uso de recursos quando a replicação é ativada. As métricas de leitura e escrita de disco e de uso de rede são as que mais se destacam, sendo que os clusters com replicação superaram em mais do dobro do uso desses recursos pelos cluster sem replicação.

De modo geral, obsevarmos que o overhead causado pela replicação é significativo, tanto em tempo como uso de recursos, mas é um custo necessário em aplicações que exijam maior disponibilidade e tolerância a falhas. Para ambientes em que a latência é o principal critério de desempenho, a configuração de cluster sem replicação se mostra mais adequada.

## 6 Conclusão

Os experimentos realizados mostraram que a configuração de cluster com três nós sem replicação apresentou o melhor desempenho em termos de latência para todas as operações avaliadas (inserção, leitura e atualização), evidenciando a eficiência do balanceamento de carga em ambientes distribuídos. A ativação da replicação, embora tenha aumentado significativamente as latências médias e no uso de recursos das máquinas, é fundamental para aplicações que demandam alta disponibilidade e tolerância a falhas, características essenciais em muitos cenários de produção. O nó único, por sua vez, apresentou desempenho inferior em relação às demais configurações, reforçando as vantagens da distribuição de dados e do paralelismo proporcionados pelo Cassandra em ambientes distribuídos.

## 7 Trabalhos Futuros

Como trabalhos futuros, sugere-se a realização de experimentos com outras configurações de replicação e o teste de outros sistemas gerenciadores de base de dados distribuídos.

## 8 Referências


- Apache Cassandra. Disponível em: <https://cassandra.apache.org/>. Acesso em: 20 jun. 2025.

- YCSB - Benchant. Disponível em: <https://benchant.com/blog/ycsb>. Acesso em: 20 jun. 2025.
- Terraform - Documentation. Disponível em: <https://developer.hashicorp.com/terraform>. Acesso em: 20 jun. 2025.