como o  armazena

TRILHÕES

de mensagens

O DISCORD

O Discord é uma plataforma de comunicação largamente utilizada nos dias de hoje. Foi lançado em maio de 2015 e rapidamente conquistou o gosto do público. Em 2016, a plataforma, extraordinariamente, já registrava uma base ativa mensal de 10 milhões de usuários.

O DISCORD USA AS MELHORES TECNOLOGIAS DO MERCADO, CERTO?

ERRADO!

ERRADO!

2015

ITERAÇÃO RÁPIDA

POUCA DOR DE CABEÇA

MIGRAÇÃO RÁPIDA NO FUTURO

2015-2016

ESTRUTURA FLEXÍVEL → DESENVOLVIMENTO RÁPIDO

SEM SHARDING + REPLICA SET UNICO → SIMPLICIDADE

2015-2016

ESTRUTURA FLEXÍVEL → DESENVOLVIMENTO RÁPIDO

SEM SHARDING + REPLICA SET UNICO → SIMPLICIDADE

→ MIGRAÇÃO FÁCIL NO FUTURO

FUNCIONOU!

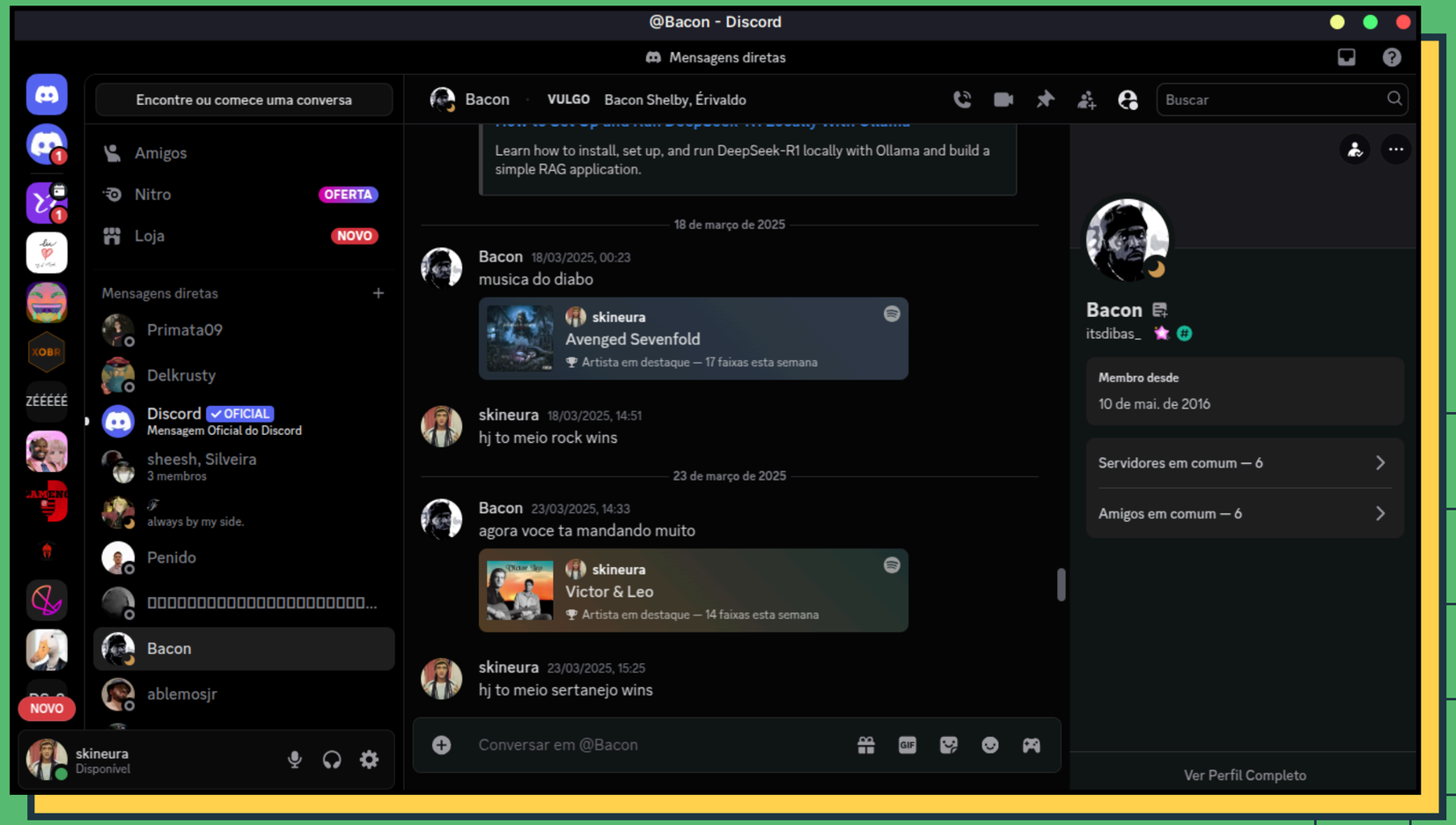
FUNCIONOU!

PARA O QUE ELES PRECISAVAM

ATÉ 100 MILHÕES DE MENSAGENS CONSEGUIRAM UM SISTEMA BEM ROBUSTO SEM ESCALA HORIZONTAL E UM ÚNICO ÍNDICE

COMO AS MENSAGENS ERAM ARMAZENADAS?

- Uma conversa pertence à um canal e elas possuem uma ordem de tempo nas mensagens enviadas.
- índice composto nos atributos channel_id e created_at



O COMEÇO DO FIM

AS MENSAGENS ARMAZENADAS PASSARAM DE 100 MILHÕES

DADOS NÃO CABEM NA MEMÓRIA

→

+ SEEKS NO DISCO

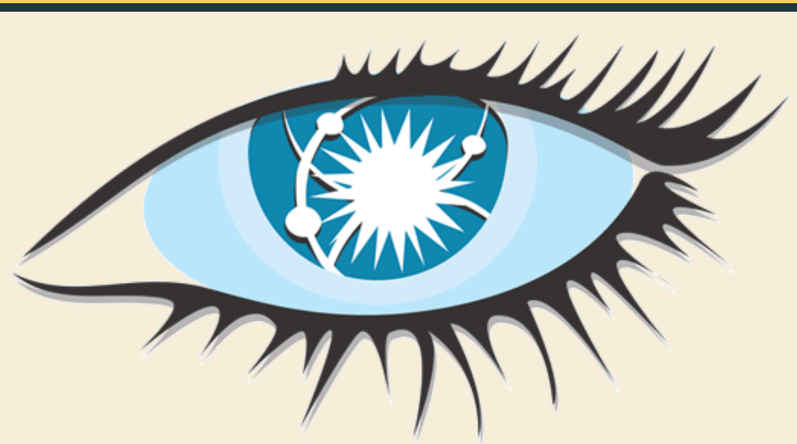
→

+ LATÊNCIA

MIGRAÇÃO PARA O CASSANDRA (2016/2017)

REQUISITOS DEFINIDOS

- ESCALABILIDADE LINEAR
- PERFORMANCE PREVISIVEL
- RECUPERAÇÃO AUTOMÁTICA
- NÃO É UMA BLOB-STORE
- BAIXA MANUTENÇÃO
- OPEN-SOURCE
- TECNOLOGIA CONSOLIDADA



CASSANDRA

KKV-STORE

KEY KEY VALUE

1° K Chave de partição: determina o nó e localização fisíca do dado no servidor

2° K Chave de agrupamento: identificador único da tupla e atributo de ordenamento

KKV-STORE

KEY KEY VALUE

1° K

CANAL A

CANAL B

CANAL C

msg 16/07/2024 - 16:37

msg 16/07/2024 - 16:39

msg 16/07/2024 - 16:40

2° K

NO DISCORD

```
CREATE TABLE messages (
  channel_id bigint,
  message_id bigint,
  author_id bigint,
  content text,
  PRIMARY KEY (channel_id, message_id)
) WITH CLUSTERING ORDER BY (message_id DESC);
```

1° K

2° K

PROBLEMA COM ESSA MODELAGEM

PARTIÇÕES GRANDES DEMAIS

SOLUÇÃO: BUCKETING

```
CREATE TABLE messages (
    channel_id bigint,
    bucket int,
    message_id bigint,
    author_id bigint,
    content text,
    PRIMARY KEY ((channel_id, bucket), message_id)
) WITH CLUSTERING ORDER BY (message_id DESC);
```

1° K

2° K

SUBINDO EM PROD

Modificaram o código para fazer os reads/writes tanto do Mongo quanto do Cassandra

author_id == null ? Como?

CONSISTÊNCIA EVENTUAL

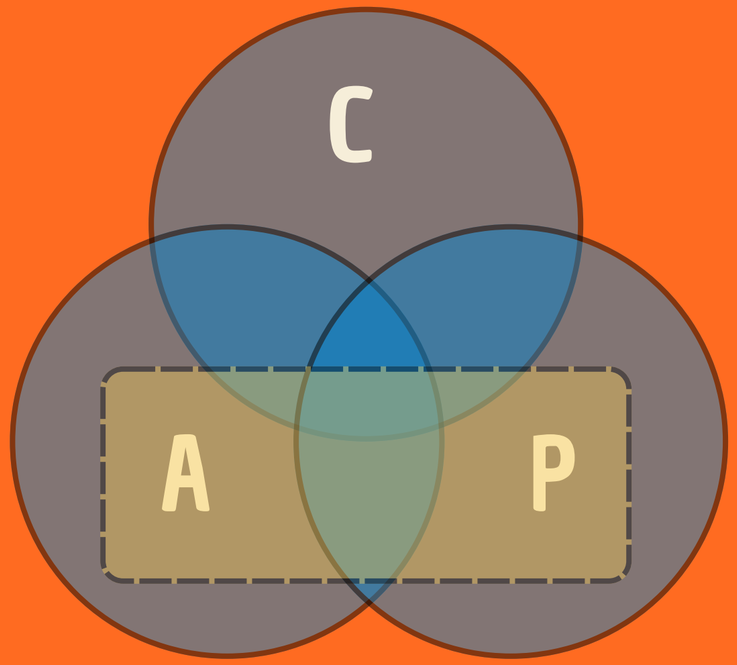
ALTA DISPONIBILIDADE

RESISTÊNCIA A PARTIÇÃO

CASSANDRA É AP

UPSERT

LAST-WRITE-WINS P/ COLUNA




1
STAN
HE

SOLUÇÃO: MENSAGEM COM AUTHOR_ID == NULL → CORROMPIDA

WRITES INEFICIENTES

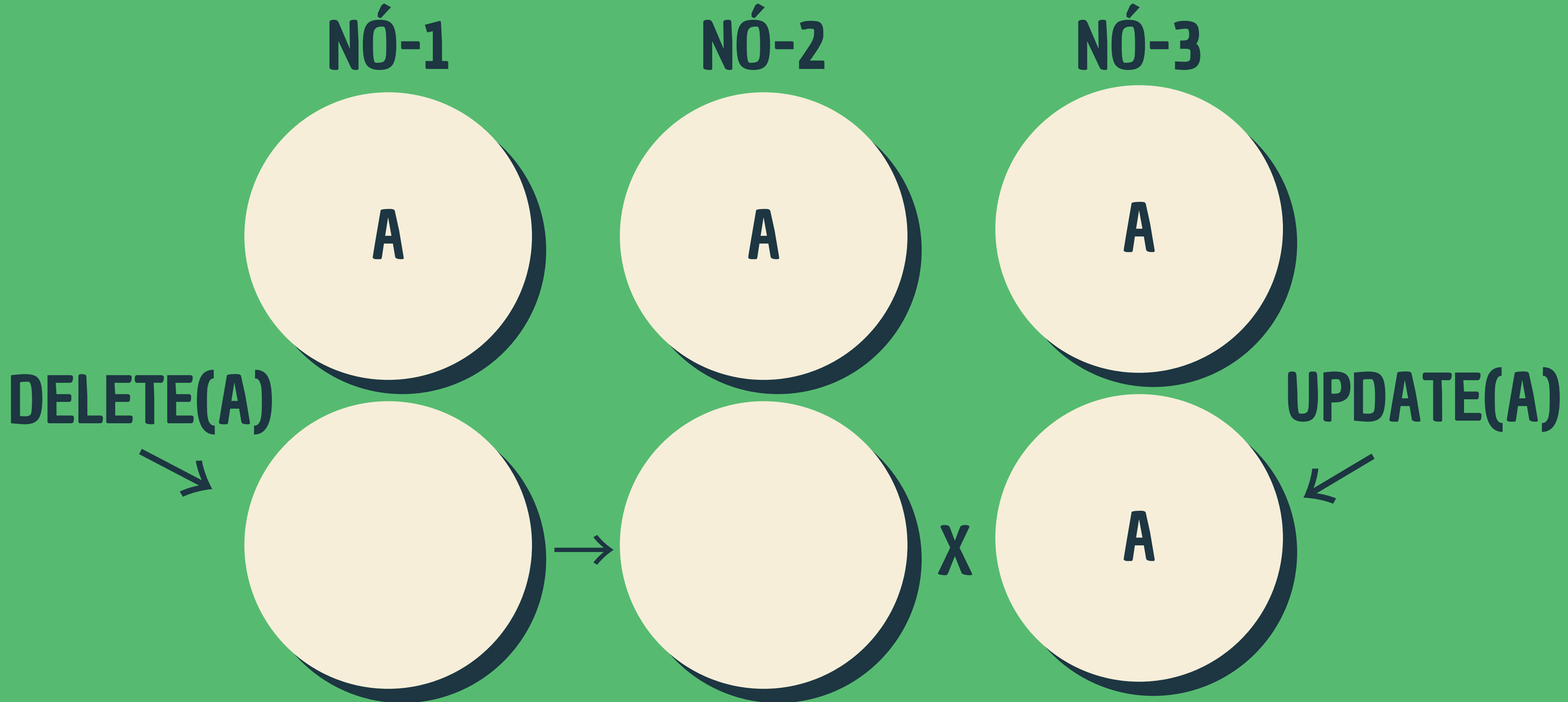
- No Cassandra, null → tombstone
- De 16 campos da mensagem, estavam escrevendo 12 campos nulls.

TOMBSTONES:

DADOS DELETADOS SÃO MARCADOS COMO TOMBSTONES AFIM DE PRESERVAR A CONSISTÊNCIA EVENTUAL.

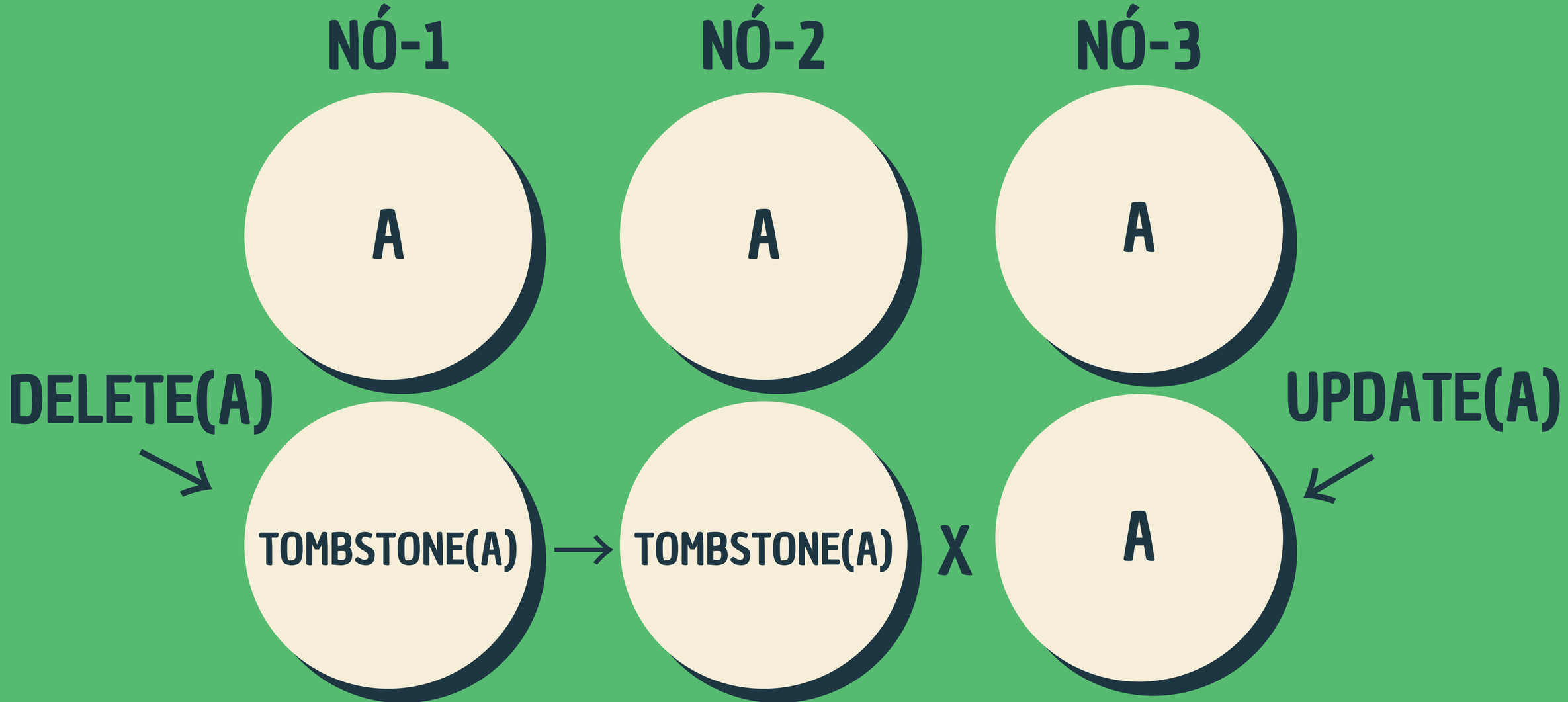
TOMBSTONES:

DADOS DELETADOS SÃO MARCADOS COMO TOMBSTONES AFIM DE PRESERVAR A CONSISTÊNCIA EVENTUAL.



TOMBSTONES:

DADOS DELETADOS SÃO MARCADOS COMO TOMBSTONES AFIM DE PRESERVAR A CONSISTÊNCIA EVENTUAL.

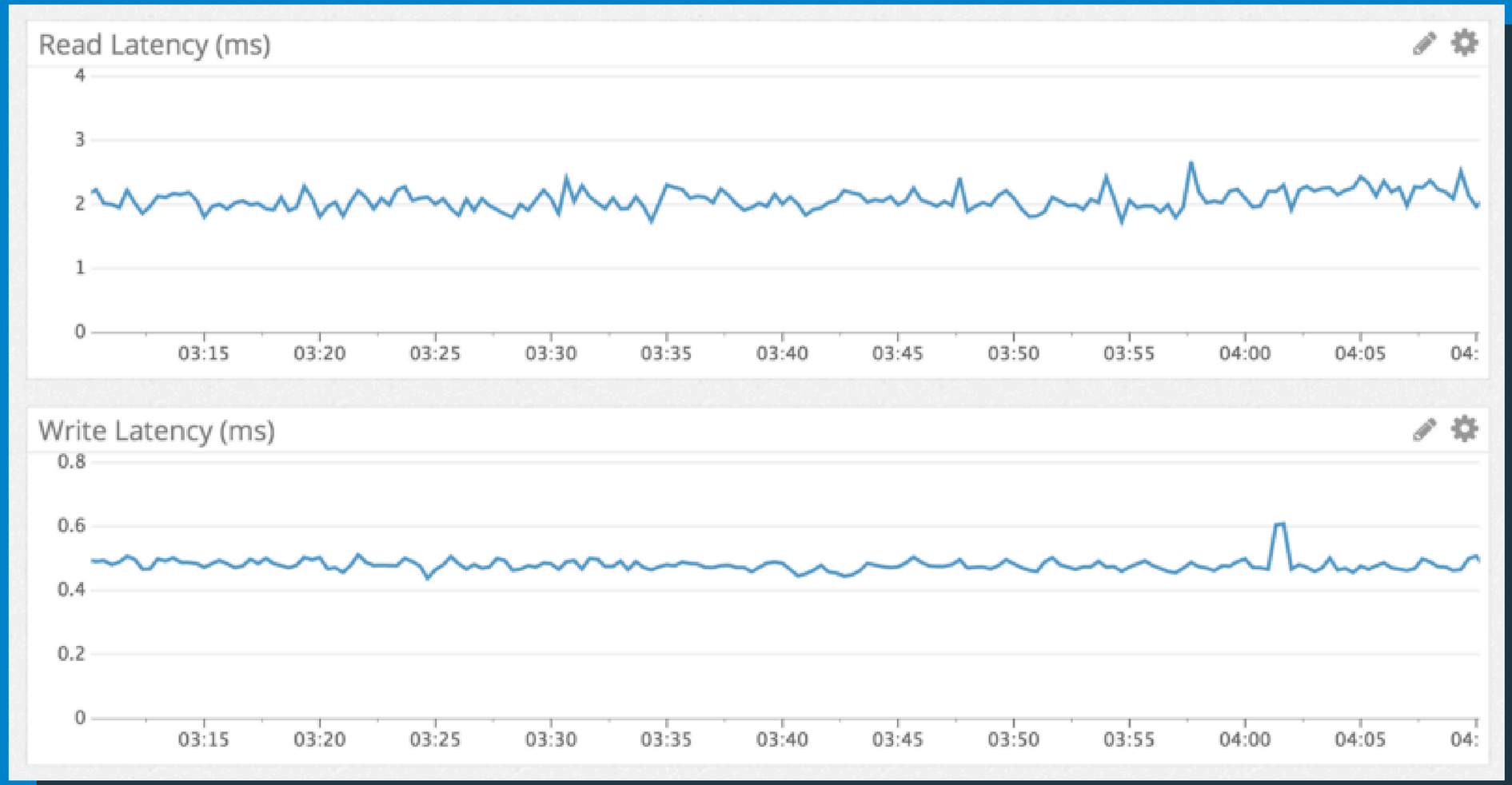


DISCORD X TOMBSTONE:

NO CASO ANTERIOR, O DELETE/UPDATE ACONTECE EM NÓS DIFERENTES.

NO CASO DO DISCORD, ACONTECEM NO MESMO NÓ, RESULTANDO EM UMA TUPLA ZUMBI: UM DADO QUE RESSUCITOU DE FORMA INCONSISTENTE, ERA PARA SER UM TOMBSTONE, MAS FOI RESSUCITADO POR CONTA DO UPSERT.

RESULTADOS


Read Latency (ms)
4
3
2
1
0
03:15
03:20
03:25
03:30
03:35
03:40
03:45
03:50
03:55
04:00
04:05
Write Latency (ms)
0.8
0.6
0.4
0.2
0
03:15
03:20
03:25
03:30
03:35
03:40
03:45
03:50
03:55
04:00
04:05

A GRANDE SURPRESA

EM UM BELO DIA, O CASSANDRA PAROU DE FUNCIONAR DIREITO.

- Pausas de até 10 segundos para Garbage Collection
- Servidor público do subreddit “Puzzles & Dragons” teve todas suas milhões mensagens, exceto uma, apagadas via API do Discord
- Resultado: milhões de tombstones que o Cassandra precisava checar

SOLUÇÕES

- Redução do tempo de vida dos tombstones
- Otimização da query

MIGRAÇÃO PARA O SCYLLADB (2023)

AO ATINGIR A MARCA DE TRILHÕES DE MENSAGENS, O CASSANDRA PASSOU A APRESENTAR LIMITAÇÕES

LIMITAÇÕES

LATÊNCIA IMPREVISÍVEL

PAUSAS DE GC

HOT PARTITIONS

MANUTENÇÃO CUSTOSA

COMPACTAÇÃO INEFICIENTE

SCYLLADB

- Compatível com Cassandra
- Escrito em C++
- Arquitetura shard-per-core

ARQUITETURA SHARD-PER-CORE

Cada núcleo físico do processador (core) é responsável por um shard (partição de dados) independente dentro do mesmo nó, processando requisições de forma isolada e paralela.

- Remove locks globais
- Processo mais previsível e eficiente
- Maior aproveitamento de hardware

MOTIVAÇÕES

- **Eliminação de pausas de GC:** não depende da JVM
- **Isolamento de carga:** graças à shard-per-core
- **Desempenho superior:** testes e benchmarks
- **Facilidade de manutenção:** compactação, reparo e monitoramento

BENCHMARKS

Métrica	ScyllaDB (shard-per-core)	Cassandra (tradicional)
Throughput máximo (ops/s)	até 300k	até 80k
Latência p99 de leitura	até 4.5 ms	até 265 ms
Latência p99 de escrita	até 7.9ms	até 494 ms
Tempo para adicionar nó	~37 minutos	~107 minutos
Tempo para substituir nó	~54 minutos	~209 minutos
Tempo de compaction	~36 minutos (32x mais rápido)	~22 a 38 horas

Fontes: ScyllaDB vs Cassandra Performance Comparison, ScyllaDB Benchmarks, ScyllaDB vs Apache Cassandra

RUST DATA SERVICES

Problema

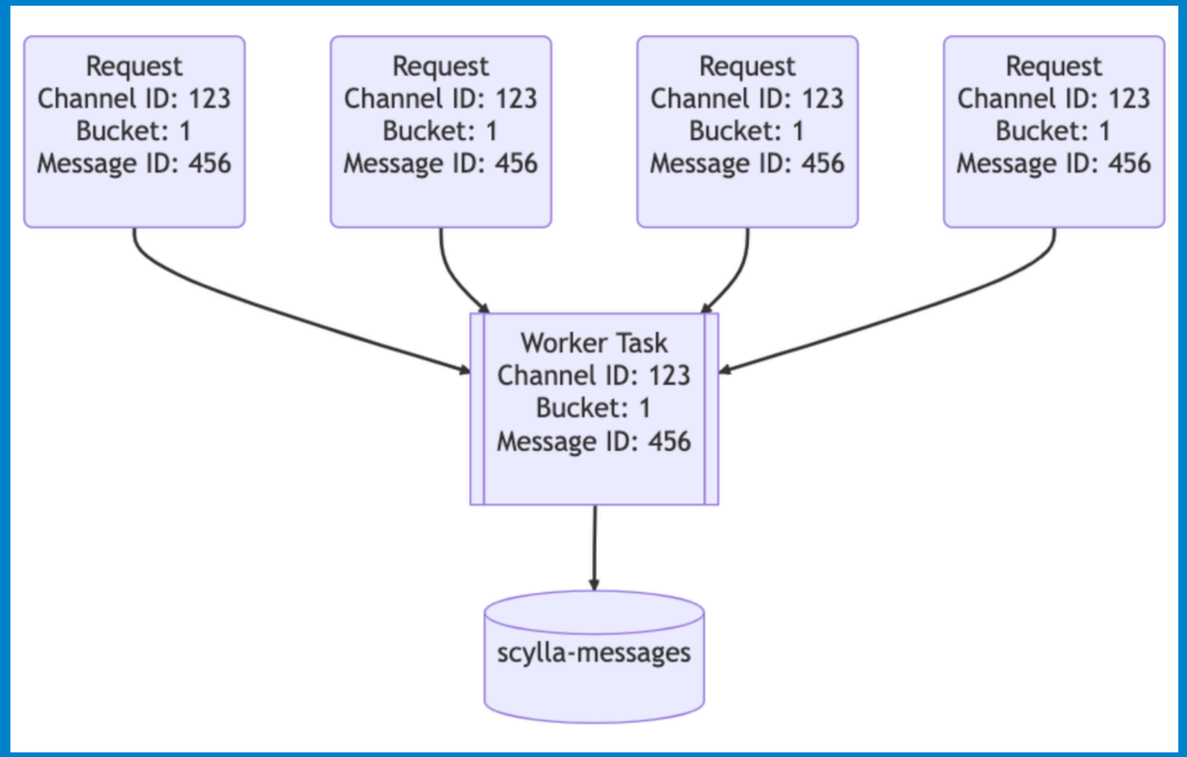
- Picos de tráfego em determinados canais
- Menções @eveyone ou grandes anúncios/eventos

Solução Proposta

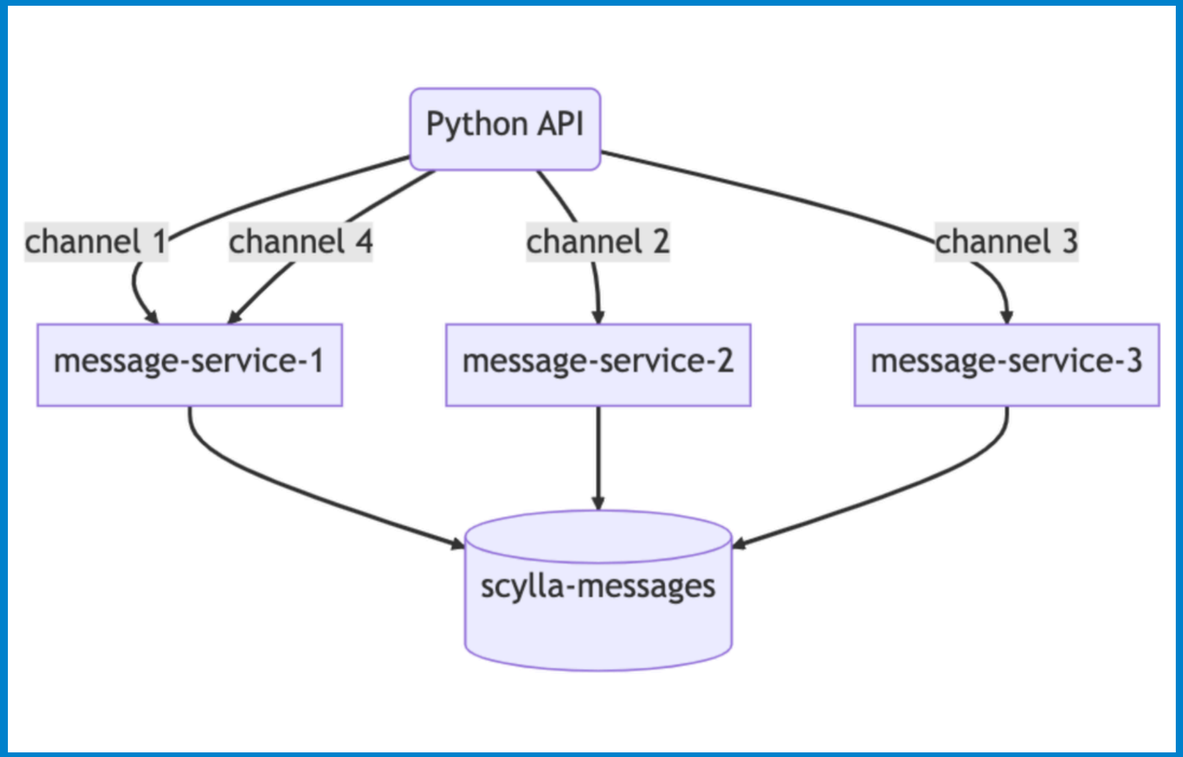
- Criação de intermediário entre API e BD
- Coalescência de requisições
- Roteamento consistente
- Concorrência segura

RUST DATA SERVICES

Coalescência de Requisições



Roteamento Consistente



A MIGRAÇÃO

- **Leitura reversa:** impeditivo, essencial para a aplicação
- **Provisionamento do novo cluster:** SSDs com RAID
- **Desenvolvimento de ferramenta de migração**
- **Gravação duplicada:** novas mensagens salvas no Scylla e Cassandra
- **Validação automática:** comparações entre leituras nos dois bancos
- **Resolução de tombstones:** travamento nos últimos 0,0001%

MODELAGEM DE DADOS

NÃO MUDOU!

- Chave primária (channel_id, bucket)
- Ordenação por message_id DESC
- Otimizações nativas ScyllaDB como a distribuição otimizada de shards

RESULTADOS

- **Redução de nós:** 177 para 72
- **Aumento de capaidade dos nós:** 9TB vs 4TB
- **Latência reduzida:** p99 de 40-125ms para 15ms em leitura. Estabilizou em 5ms na escrita.
- **Redução de custo operacional:** menos nós garantem menos manutenções manuais e maior previsibilidade
- **Resiliência comprovada:** suportou picos globais sem degradação significativa de performance