Banco de Dados Cassandra

Bruno Rafael dos Santos, Fernando H.

Fundação Universidade do Estado de Santa Catarina

26 de novembro de 2024

Sumário

Problema

Problema

- O problema deste trabalho envolve uma empresa que deseja monitorar seu parque computacional e linha de produção. Esse monitoramento ocorre em tempo real por meio de sensores que coletam e transmitem os dados, com o objetivo de identificar rapidamente a ocorrência de incêndios.
- Ao detectar um incêndio, é crucial que o sistema afetado seja imediatamente interrompido para evitar maiores danos. Além disso, a empresa precisa enviar alarmes para os bombeiros internos.
- O sistema requer disponibilidade de 99,99%.

Proposta

Proposta

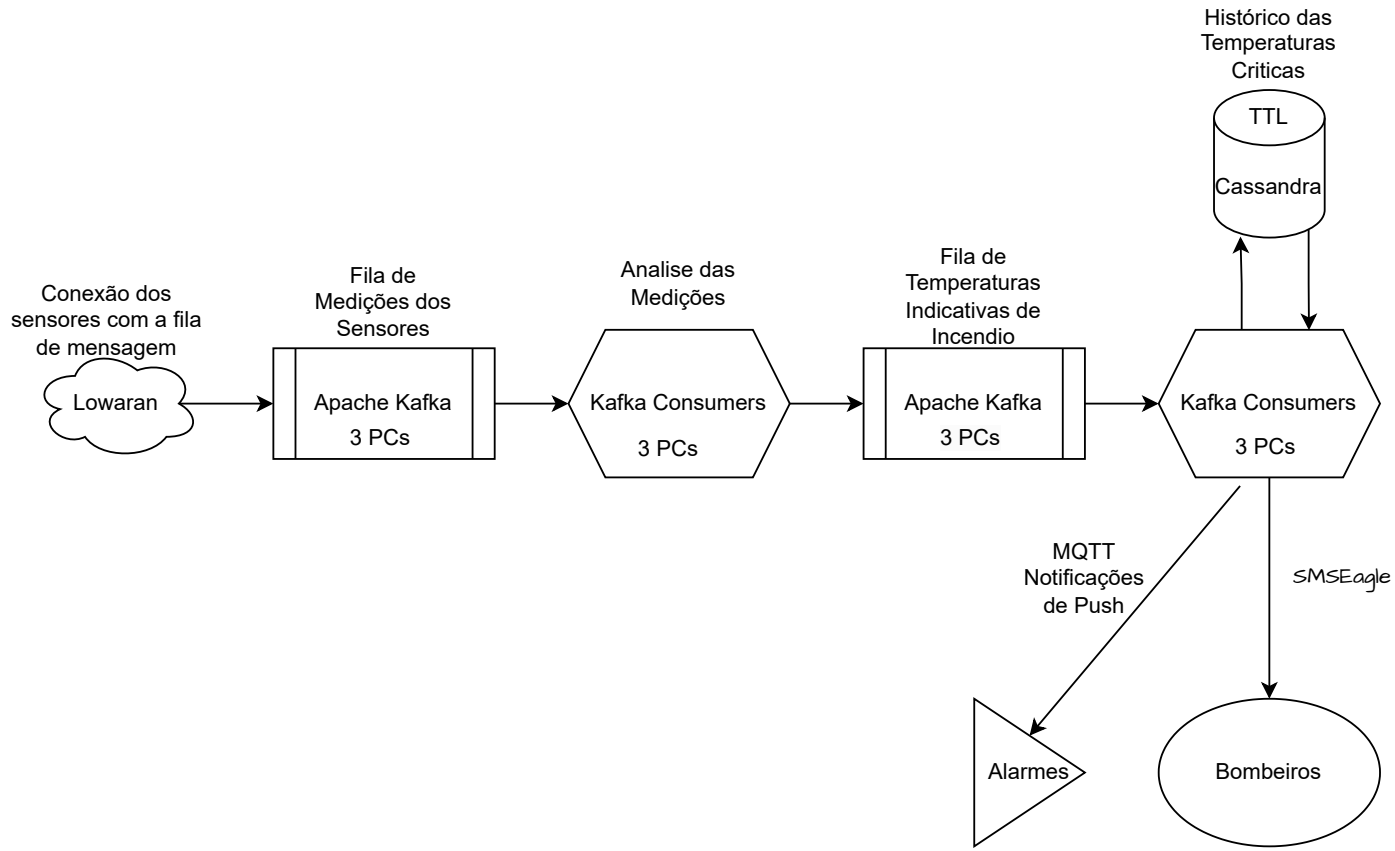

Figura: Proposta, elaborado pelos autores.

Proposta

- Cada PC com 8 núcleos e 16 threads custa em torno de R$2.500,00.
- Assumindo que cada PC consegue processar uma leitura a cada 1ms por thread, suporta 16 mil sensores, considerando que cada sensor realiza uma leitura por segundo.
- Para garantir a disponibilidade, para cada etapa da solução, serão atribuídos três PCs.
- O custo inicial do projeto, consederando as 15 máquinas necessárias será de R$30.000,00.

Proposta

- O projeto inicial suportará até 16 mil sensores.
- Para manter a disponibilidade em 99,99% será necessario adicionar uma máquina redundante a cada duas novas, portanto a cada 32 mil novos sensores, será necessário adicionar 3 máquinas na primeira fila de mensagens e 3 máquinas no grupo responsável por analisar as medições. Isso totaliza um custo de R$15.000,00 para cada 32 mil novos sensores.
- O custo médio de cada novo sensor será de R$0,46.

Introdução ao Banco de Dados Cassandra

Introdução ao Banco de Dados Cassandra

- Conforme o teorema CAP nenhum banco de dados pode atender completamente os requisitos de: consistência (*consistency*), disponibilidade (*availability*) e tolerância a partições (*partition tolerance*) [1].
- Cassandra é um banco de dados com foco na disponibilidade e tolerância a partições que é considerado "eventualmente consistente"[1] .
- De acordo com [1], o modelo de dados do Cassandra pode ser classificado como *"partitioned row store"* com dados armazenados em tabelas *hash* multidimensionais e esparsas.

Introdução ao Banco de Dados Cassandra

- Nesse contexto, ser esparsa (*sparse*) indica que toda linha pode ter uma ou mais colunas, porém as linhas de uma mesma tabela não precisam ter todas as colunas da tabela [1].
- Em "partitioned row store", "partitioned" indica que cada linhas está associada à exatamente uma *partition key* [1].
- Também segundo [1], o banco de dados Cassandra é muitas vezes erroneamente mencionado como "wide column store".

Introdução ao Banco de Dados Cassandra

- A organização das tabelas em Cassandra é feita por meio de chaves primárias conhecidas como *composite keys* [1].
- **Composite Key:** é uma chave primária composta por uma *partition key* e um conjunto (opcional) de *clustering columns* [1].
- **Partition Key:** é o primeiro campo de uma *composite key* (chave primária) e é responsável por determinar em que nó serão armazenadas as linhas [1]; este campo pode conter mais de uma coluna (basta definir tal como um tupla de colunas, o que será mostrado adiante).

Introdução ao Banco de Dados Cassandra

- **Partição:** é o grupo representado por uma *partition key*.
- **Clustering Columns:** servem para ordenar os dados em uma partição [1].
- **Static Columns:** armazenam valores que não fazem parte da *primary key* e contém um valor único por *partition key* [1].

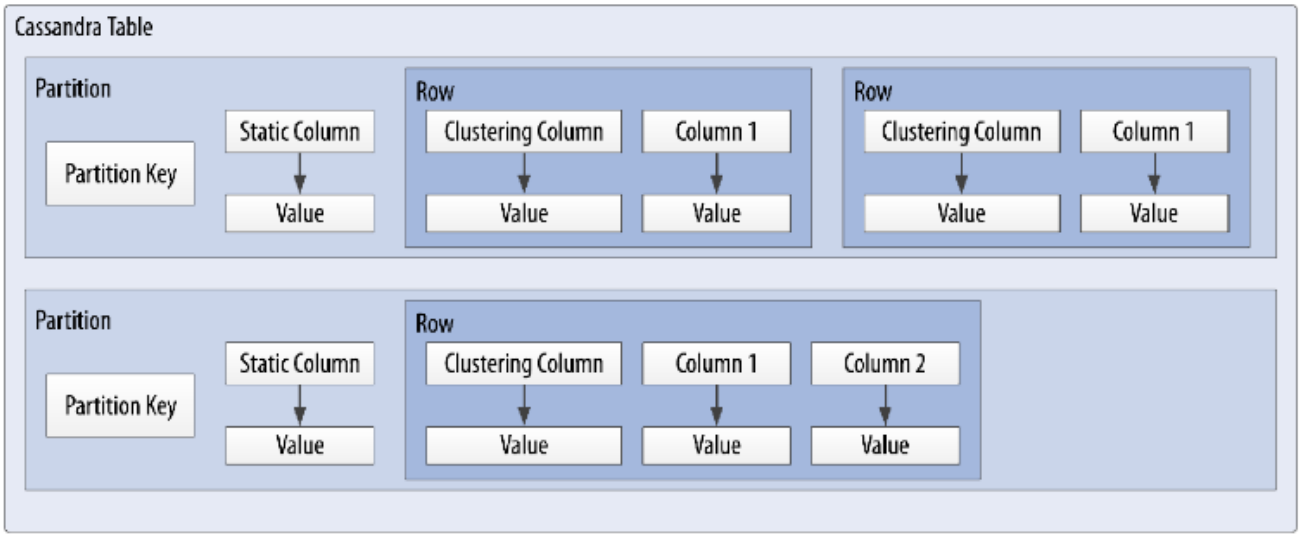

Figura: "Tabela com Partições em Cassandra" [1, p.59, tradução nossa], retirada de [1].

Introdução ao Banco de Dados Cassandra

- **Keyspaces:** de maneira semelhante aos bancos de dados relacionais contém tabelas (*tables*) e atributos que definem como o keyspace irá se comportar (um exemplo disso são os atributos relacionados a replicação) [1].
- **Cluster:** é um anel de nós, onde cada nó é um instância de Cassandra (normalmente em máquinas diferentes) [1].

Introdução ao Banco de Dados Cassandra

- **Timestamps:** nas tabelas em Cassandra há uma coluna implícita que é utilizada para manter os dados atualizados em caso de alterações conflitantes (*last write wins approach*) [1].
  - Essa coluna pode ser visualizada por meio da função `writetime()` que recebe como argumento uma coluna, no entanto está coluna não pode pertencer a chave primária da tabela (não é possível visualizar *timestamps* de colunas da chave primária) [1].

Introdução ao Banco de Dados Cassandra

- Para gerenciar o armazenamento e as buscas o banco de dados Cassandra fornece os seguintes atributos para configuração da topologia dos clusters [1]:
  - **Racks:** contém um grupo de nós estão supostamente próximos.
  - **Datacenter:** contém um grupo de *racks* supostamente distantes.

  Por padrão o Cassandra utiliza uma configuração em que todos os clusters estão dentro do mesmo *rack* e portanto mesmo *datacenter*, sendo os nomes destes *rack1* e *datacenter1* respectivamente.

Introdução ao Banco de Dados Cassandra

- **Tokens:** a organização de como os dados serão armazenados em um anel utiliza *tokens*, isto é, valores numéricos, que são associadas a cada um dos nós no anel.
  - Um nó associado a um *token* é responsável por todos os valores de *token* maiores que os do *token* associado ao do nó anterior.
  - Um dado é associado a um nó de acordo com o resultado de uma função *hash* que tem como argumento a *partition key* do respectivo dado.
  - Para replicações o dado é sempre guardado no nó associado ao *token* calculado e o restante destes nós é definido pela polítca de replicação configurada [1].

Cassandra em Docker

Cassandra utilizando Docker Composer

- Será explicado aqui como criar um cluster de 3 nós por meio de 3 contêiners tendo como base o conteúdo exposto em [2].
- Para fazer download de uma imagem de Cassandra em docker basta usar o seguinte comando:

```
$ docker pull cassandra
```

- Para criar e rodar um contêiner com a imagem de Cassandra é usado o seguinte comando:

```
$ docker run --name cassandra-1 -d cassandra:latest
```

Cassandra em Docker

- Para se obter as informações sobre os nós do cluster ao qual pertence o nó rodando na máquina `cassandra-1` pode-se utilizar o comando `nodetool` no contêiner:

```
$ docker exec -i -t cassandra-1 bash -c 'nodetool status'
```

- Para adicionar o segundo nó ao cluster é necessário criar um segundo contêiner e conectá-lo ao contêiner `cassandra-1`, o que pode ser feito por:

```
$ docker run --name cassandra-2 -d --link
cassandra-1:cassandra cassandra:latest
```

Cassandra em Docker

- Por fim adicionando o terceiro nó ao cluster:

```
$ docker run --name cassandra-3 -d --link
cassandra-1:cassandra cassandra:latest
```

- Agora, pode-se por exemplo, acessar em quaisquer um dos contêineres a linha de comando *cqlsh* do banco de dados Cassandra:

```
$ docker exec -it cassandra-1 bash -c 'cqlsh'
```

Cassandra em Docker

- **Fator de replicação:** este valor indica o número de nós de um cluster que receberão cópias de um dado quando este for inserido.
- Como tem-se 3 nós no cluster pode-se criar um *keyspace* com um fator de replicação 3, para que os dados sejam replicados em 3 nós:

```
CREATE KEYSPACE sensores
WITH REPLICATION = {
    'class' : 'SimpleStrategy',
    'replication_factor' : 3
};
```

onde a estratégia de replicação *SimpleStrategy* não considera a existência de *racks* e coloca as replicas em sequência nos nós no sentido horário [2].

Cassandra em Docker

- Considerando o problema deste trabalho pode-se fazer uma inserção com tempo de vida para os dados (usando TTL):

```
INSERT INTO temperaturas(id_sensor,
momento_sensor , temperatura_graus) VALUES
('04', now(), 25.0) USING TTL 10;
```

- A tabela para armazenamento das temperaturas dos sensores foi criada por meio do seguinte comando:

```
CREATE TABLE temperaturas (
id_sensor text,
momento_sensor timeuuid,
temperatura_graus float,
PRIMARY KEY (id_sensor, momento_sensor));
```

Cassandra em Docker

- Foram testados, utilizando a biblioteca *cassandra-driver* em *Python*, os tempos (em segundos) de inserções (comando `INSERT`) e buscas (comando `SELECT`) com tamanhos de dados variados, isto é, de tempo para realização da inserção de quantidades diferentes de linhas e de busca nestas respectivas quantidades.
- Os resultados obtidos foram os seguintes:

nº linhas	tempo para inserção	tempo para busca
16	0.0128286	0.0227280
2^8	1.2099252	0.0018699
2^{12}	2.0133822	0.0014019
2^{16}	25.0366664	0.0014555
2^{20}	416.7740293	0.0300076

Cassandra em Docker

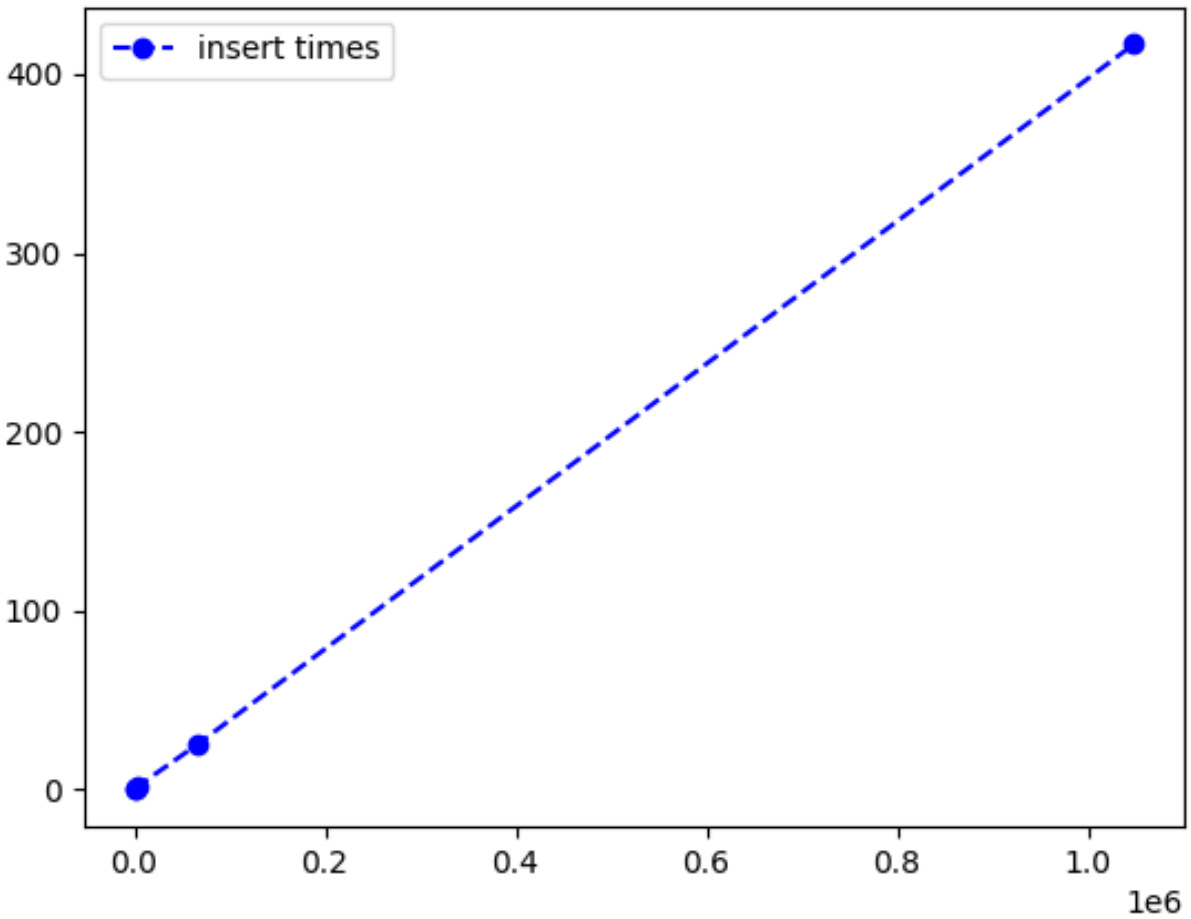

Figura: Gráfico de tempo e número de inserções.

Cassandra em Docker

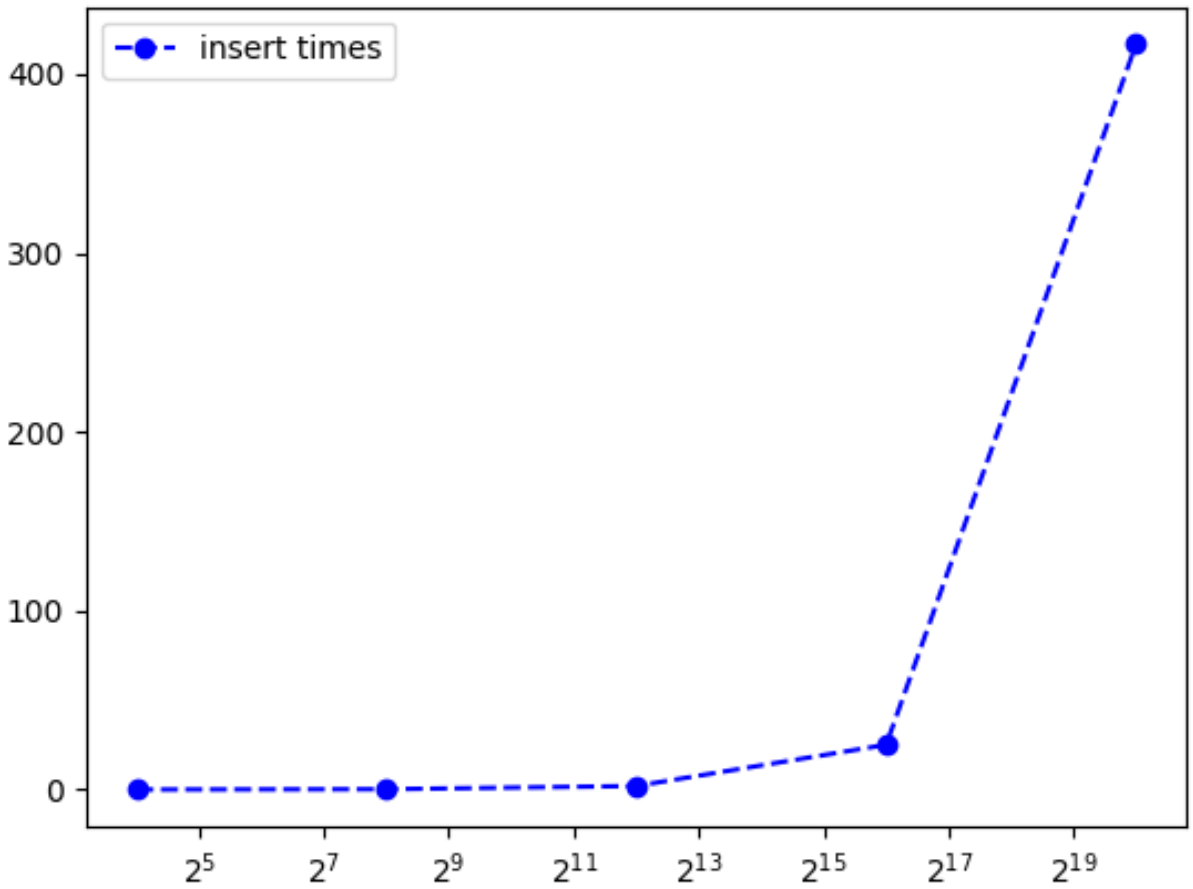

Figura: Gráfico de tempo e número de inserções em escala logarítmica.

Cassandra em Docker

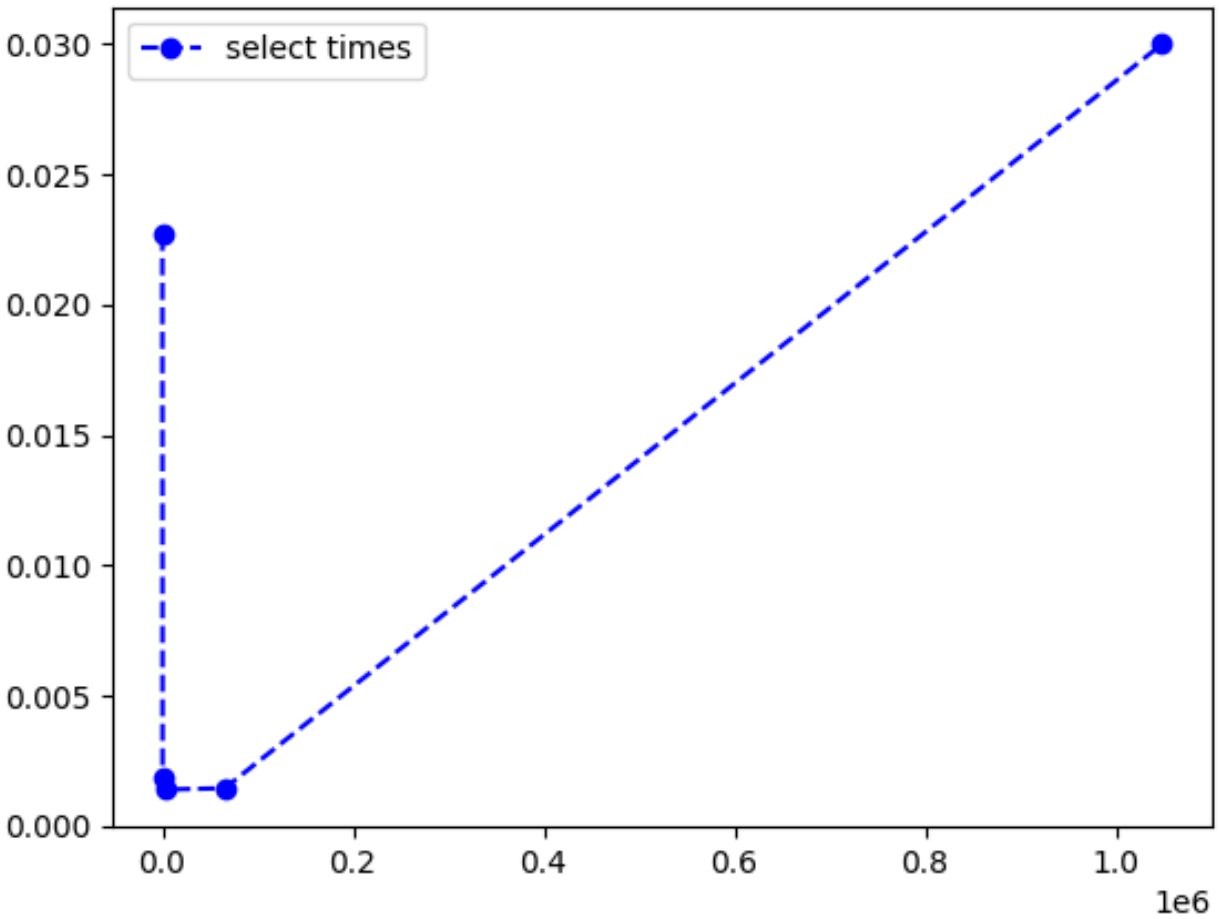

Figura: Gráfico de tempo de busca e número de linhas.

Cassandra em Docker

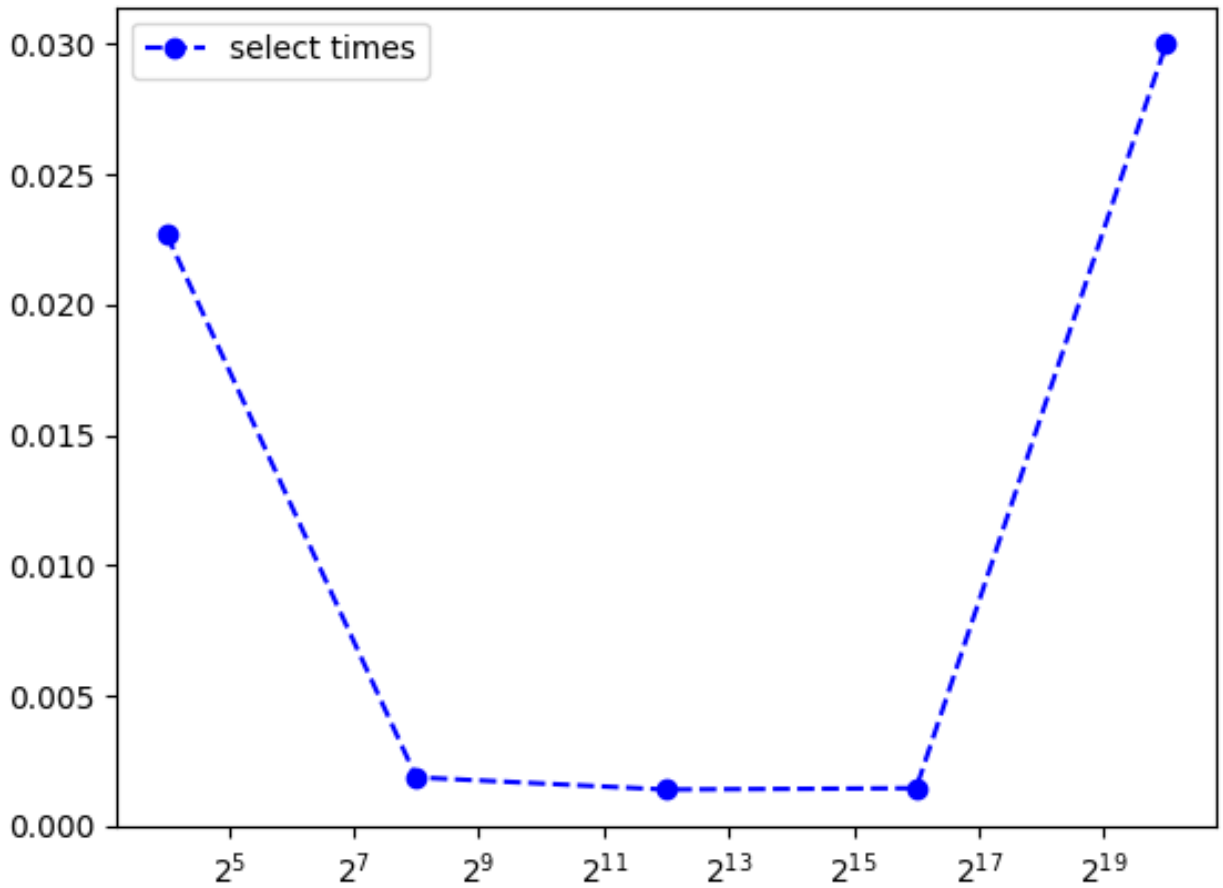

Figura: Gráfico de tempo de busca e número de linhas em escala logarítmica.

Cassandra em Docker

- Quanto as dificuldades encontradas na realização desse trabalho vale destacar o seguinte:
  - Para uso da biblioteca *cassandra-driver* foi necessário utilizar a versão 3.11 do *python*, pois a biblioteca é feita para as versões 3.8 até 3.12.
  - Existem maneiras mais interessantes e provavelmente mais didáticas para o uso de Cassandra em Docker (usando `docker composer` como apresentado em [3] por exemplo), porém não caberia no intervalo de tempo da apresentação trazer uma destas formas de uso no lugar da que foi aqui apresentada.

Bibliografia

Bibliografia I

[1] Jeff Carpenter and Eben Hewitt. *Cassandra: The Definitive Guide*. Cassandra. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 3 edition, 2020.

[2] Stelios Sotiriadis and Dimitris Kargatzis. Lab8: Introduction to apache cassandra, fevereiro 2020.

[3] Nate Toscano. Establishing a multi-node cassandra cluster using docker-compose, abril 2023.