Um Estudo de Bancos de Dados Distribuídos em um Dataset Criminal Utilizando Apache Hive

Ítalo L. F. Portinho[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, s/nº – 24.210-346 – Niterói – RJ – Brazil

italons@gmail.com, italoleite@id.uff.br

Abstract. *The PolRoute is a dataset with information on criminal incidents on the streets of São Paulo, aimed at serving as a knowledge base to propose efficient police patrolling routes. Given the large cardinality of the dataset, this work proposes a distribution project using horizontal fragmentation for the database derived from the dataset, utilizing the Apache Hive middleware to provide parallelism in query processing. Based on 7 proposed queries, we compared the performance of processing in an unfragmented schema versus a fragmented schema, concluding that performance increases significantly with fragmentation, although depending on the table size, the gain may not be worthwhile.*

Resumo. *O PolRoute é um dataset com informações sobre ocorrências criminais nas ruas de São Paulo com o propósito de servir como base de conhecimento para propor rotas de patrulhamento policial eficientes. Dada a grande cardinalidade do conjunto de dados este trabalho propõe um projeto de distribuição utilizando fragmentação horizontal para o banco de dados originado do dataset, utilizando o middleware Apache Hive para prover paralelismo no processamento das consultas. Partindo de 7 consultas propostas, comparamos o desempenho do processamento em um schema sem fragmentação com um schema fragmentado chegando à conclusão que o desempenho aumenta consideravelmente com a fragmentação, porém dependendo do tamanho da tabela, pode não valer a pena o ganho.*

1. Introdução

É um fato bem conhecido que a criminalidade é uma questão aberta, porém importante, na maioria dos centros urbanos ao redor do mundo. Especialmente no Brasil, criar soluções para reduzir os índices de criminalidade é uma prioridade máxima. Para reduzir esses índices, muitas cidades estão adotando técnicas de policiamento preditivo. As técnicas de policiamento preditivo são altamente baseadas na extração de conhecimento valioso de um conjunto massivo de dados que contém informações sobre tempos, locais e tipos de crimes passados. O conhecimento extraído é então utilizado para fornecer insights aos departamentos de polícia, definindo onde a polícia deve manter sua presença. Esses conjuntos de dados também podem ser usados para uma tarefa crítica de policiamento preditivo: definir onde as patrulhas policiais devem atuar. Tais patrulhas são comumente definidas para cobrir uma série de pontos quentes de crime (áreas que apresentam altos níveis de criminalidade) e têm algumas restrições a serem consideradas (número de policiais disponíveis, viaturas, etc.). Portanto, definir a rota de cada veículo policial é um problema complexo de otimização, uma vez que, na maioria dos casos, existem muitos pontos quentes e os recursos disponíveis são escassos, ou seja, a quantidade de veículos e policiais disponíveis é muito menor do que o necessário. Infelizmente, dados de qualidade sobre taxas de criminalidade não são fáceis de obter. Com o objetivo de enfrentar esse problema, foi proposto o conjunto de dados PolRoute-DS, um conjunto de dados projetado para fomentar o desenvolvimento e a avaliação de abordagens de roteamento policial em grandes centros urbanos. O PolRoute-DS combina a estrutura espacial da cidade de interesse (no contexto deste artigo, a cidade de São Paulo), representada como um grafo conectado e direcionado de segmentos de ruas, com dados criminais obtidos de fontes públicas.

O *dataset* é composto de 6 arquivos csv com informações sobre o tipo de ocorrência criminal, em qual bairro/distrito, segmento de rua com seus vértices, e em qual período de tempo separado por ano, mês, dia da semana, dia e período do dia. Os arquivos são: *time.csv, crime.csv, segment.csv, vertice.csv, district.csv e neighborhood.csv*. Abaixo especificamos um *schema* com as relações para guiar o resto do trabalho:

district(id, name, geometry)

neighborhood(id, name, geometry)

time(id int, period, day, month, year, weekday)

vertice(id, label, district_id, neighborhood_id, zone_id)

district_id referencia district(id)

neighborhood_id referencia neighborhood(id)

segment(id, geometry, oneway, length, final_vertice_id, start_vertice_id)

final_vertice_id referencia vertice(id)

start_vertice_id referencia vertice(id)

crime(id, total_feminicide, total_homicide, total_felony_murder, total_bodily_harm, total_theft_cellphone, total_armed_robbery_cellphone, total_theft_auto, total_armed_robbery_auto, segment_id, time_id)

Partindo de 7 perguntas sobre o *dataset*, esse trabalho propõe nas próximas seções traduzir elas para a linguagem SQL, e dado o grande volume de dados do conjunto, criar um projeto de distribuição para o banco de dados, e executar as consultas em um *cluster* operando um *middleware* (Apache Hive) que proporcione processamento distribuído dessas consultas. Mais do que obter o resultado das perguntas propostas, vamos comparar o tempo médio de processamento das consultas sem o projeto de distribuição aplicado às tabelas com o projeto de distribuição aplicado às tabelas. As perguntas propostas são:

1. Qual o total de crimes por tipo e por segmento das ruas do distrito de IGUATEMI durante o ano de 2016?
2. Qual o total de crimes por tipo e por segmento das ruas do distrito de IGUATEMI entre 2006 e 2016?
3. Qual o total de ocorrências de Roubo de Celular e roubo de carro no bairro de SANTA EFIGÊNIA em 2015?
4. Qual o total de crimes por tipo em vias de mão única da cidade durante o ano de 2012?
5. Qual o total de roubos de carro e celular em todos os segmentos durante o ano de 2017?
6. Quais os ID's de segmentos que possuíam o maior índice criminal (soma de ocorrências de todos os tipos de crimes) , durante o mês de Novembro de 2010?
7. Quais os ID's dos segmentos que possuíam o maior índice criminal (soma de ocorrências de todos os tipos de crimes) durante os finais de semana do ano de 2018?

2. O Projeto de Distribuição

Primeiramente vamos descrever as perguntas propostas na seção anterior na linguagem SQL, para que possamos analisar melhor as dependências entre as tabelas e quais atributos serão usados em cada consulta. Ficam então as perguntas traduzidas em 7 consultas SQL descritas abaixo:

QUERY 1: Qual o total de crimes por tipo e por segmento das ruas do distrito de IGUATEMI durante o ano de 2016?

```
select
    crime.segment_id,
    sum(crime.total_feminicide),
    sum(crime.total_homicide),
    sum(crime.total_felony_murder),
    sum(crime.total_bodily_harm),
    sum(crime.total_theft_cellphone),
```

```
        sum(crime.total_armed_robbery_cellphone),
        sum(crime.total_theft_auto),
        sum(crime.total_armed_robbery_auto)
    from    vertice, district, segment, crime, time
    where     crime.segment_id = segment.id
        and  (segment.start_vertice_id  =  vertice.id  OR
segment.final_vertice_id = vertice.id)
        and vertice.district_id = district.id
        and district.name = 'IGUATEMI'
        and crime.time_id = time.id
        and time.year = 2016
        group by crime.segment_id;
```

QUERY 2: Qual o total de crimes por tipo e por segmento das ruas do distrito de IGUATEMI entre 2006 e 2016?

```
    select
        crime.segment_id,
        sum(crime.total_feminicide),
        sum(crime.total_homicide),
        sum(crime.total_felony_murder),
        sum(crime.total_bodily_harm),
        sum(crime.total_theft_cellphone),
        sum(crime.total_armed_robbery_cellphone),
        sum(crime.total_theft_auto),
        sum(crime.total_armed_robbery_auto)
    from
        vertice, district, segment, crime, time


    where
        crime.segment_id = segment.id
        and  (segment.start_vertice_id  =  vertice.id  OR
segment.final_vertice_id = vertice.id)
        and vertice.district_id = district.id
        and district.name = 'IGUATEMI'
        and crime.time_id = time.id
        and time.year between 2006 and 2016
```

```
    group by crime.segment_id;
```

QUERY 3: Qual o total de ocorrências de Roubo de Celular e roubo de carro no bairro de SANTA EFIGÊNIA em 2015?

```
select
    neighborhood.name,
    time.year,
    sum(crime.total_theft_cellphone),
    sum(crime.total_armed_robbery_cellphone),
    sum(crime.total_theft_auto),
    sum(crime.total_armed_robbery_auto)
from
    vertice, neighborhood, segment, crime, time


where
    crime.segment_id = segment.id
     and (segment.start_vertice_id = vertice.id OR
segment.final_vertice_id = vertice.id)
    and vertice.neighborhood_id = neighborhood.id
    --and neighborhood.name LIKE 'Santa Efig%'
    and neighborhood.id = 6
    and crime.time_id = time.id
    and time.year = 2015
group by neighborhood.name, time.year;
```

QUERY 4: Qual o total de crimes por tipo em vias de mão única da cidade durante o ano de 2012?

```
select
    time.year as ano,
    segment.oneway as mao_dupla,
    sum(crime.total_feminicide) as feminicidio,
    sum(crime.total_homicide) as homicidio,
    sum(crime.total_felony_murder) as assassinato,
    sum(crime.total_bodily_harm) as lesao_corporal,
    sum(crime.total_theft_cellphone) as furto_celular,
          sum(crime.total_armed_robbery_cellphone) as
roubo_celular,
```

```
    sum(crime.total_theft_auto) as furto_carro,
    sum(crime.total_armed_robbery_auto) as roubo_carro
from
    segment, crime, time

where
    crime.segment_id = segment.id
    and segment.oneway = 'yes'
    and crime.time_id = time.id
    and time.year = 2012
group by time.year, segment.oneway;
```

QUERY 5: Qual o total de roubos de carro e celular em todos os segmentos durante o ano de 2017?

```
select
    time.year as ano,
    sum(crime.total_theft_cellphone) as furto_cel,
        sum(crime.total_armed_robbery_cellphone) as
roubo_cel,
    sum(crime.total_theft_auto) as furto_carro,
    sum(crime.total_armed_robbery_auto) as roubo_carro
from
    crime, time

where crime.time_id = time.id
    and time.year = 2017
group by time.year;
```

QUERY 6: Quais os IDs de segmentos que possuíam o maior índice criminal (soma de ocorrências de todos os tipos de crimes), durante o mês de Novembro de 2010?

```
select
    segment.id,
    time.month,
    time.year,
    sum(crime.total_feminicide) +
```

```
        sum(crime.total_homicide) +
        sum(crime.total_felony_murder) +
        sum(crime.total_bodily_harm) +
        sum(crime.total_theft_cellphone) +
        sum(crime.total_armed_robbery_cellphone) +
        sum(crime.total_theft_auto) +
        sum(crime.total_armed_robbery_auto) as soma_total
    from
        segment, crime, time


    where
        crime.segment_id = segment.id
        and crime.time_id = time.id
        and time.year = 2010
        and time.month = 11
    group by segment.id, time.month, time.year order by
soma_total DESC;
```

QUERY 7: Quais os IDs dos segmentos que possuíam o maior índice criminal (soma de ocorrências de todos os tipos de crimes) durante os finais de semana do ano de 2018?

```
    select
        segment.id,
        time.year,
        sum(crime.total_feminicide) +
        sum(crime.total_homicide) +
        sum(crime.total_felony_murder) +
        sum(crime.total_bodily_harm) +
        sum(crime.total_theft_cellphone) +
        sum(crime.total_armed_robbery_cellphone) +
        sum(crime.total_theft_auto) +
        sum(crime.total_armed_robbery_auto) as soma_total
    from
        segment, crime, time
```

```
    where
        crime.segment_id = segment.id
        and crime.time_id = time.id
        and time.year = 2018
        and time.weekday IN ('saturday', 'monday')
    group by segment.id, time.year order by soma_total
DESC LIMIT 100;
```

Uma vez de posse das consultas mais frequentes no banco de dados, vamos expressar as relações entre as tabelas com um diagrama dono-membro(figura 1) com o propósito de determinar quais delas receberão uma fragmentação horizontal primária(FHP) e quais terão fragmentação horizontal derivada(FHD) a partir destas. Analisando o diagrama, determina-se que as tabelas *crime*, *district* e *neighborhood* receberão fragmentação horizontal primária. A tabela vértice receberá fragmentação horizontal derivada da tabela *district*, pois possui duas junções com essa tabela e apenas um com *neighborhood*[Özsu, M. Tamer, 2011]. Pelo mesmo motivo a tabela *crime* receberá fragmentação horizontal derivada da tabela *time*, pois possuí mais junções com essa tabela do que com a tabela *segment*.

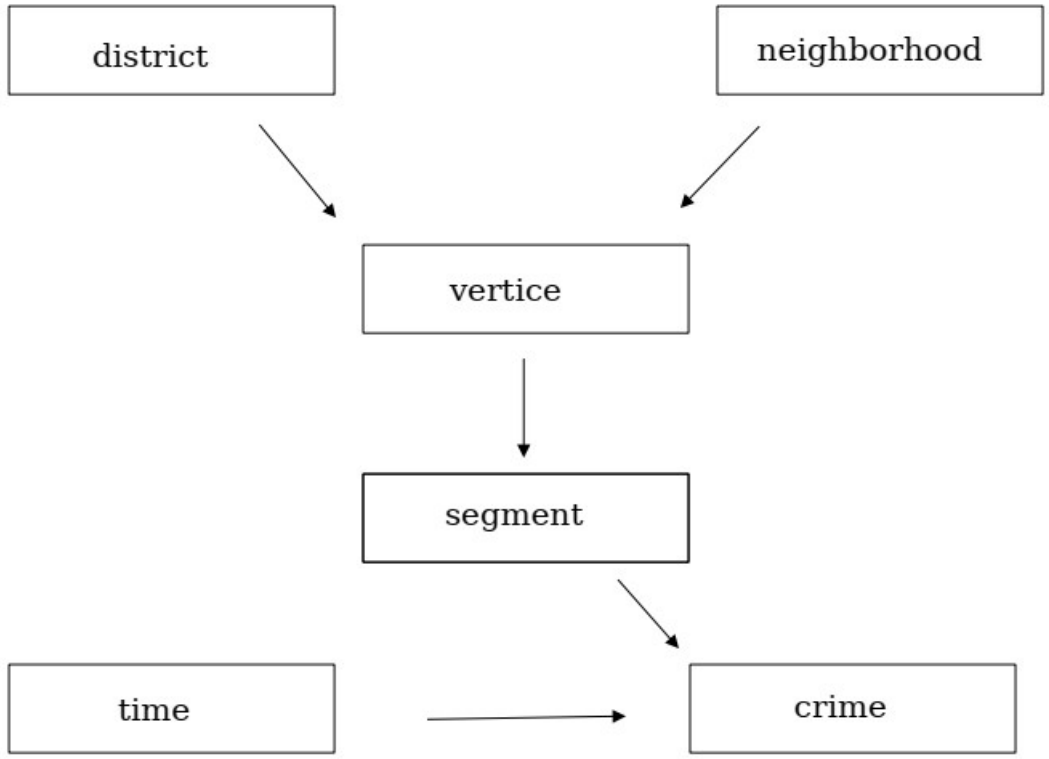

Figura 1: Diagrama dono-membro.

Das consultas, conseguimos obter os predicados simples de cada tabela com FHP(tabela 1) e as tabelas *district* e *neighborhood* terão dois fragmentos, consequência de seus dois mintermos(pois cada uma tem somente um predicado simples e o número de mintermos é $2^{nº\ predicados\ simples}$). Portanto as tabelas com fragmentação derivada de *district*, *vertice* e *segment*, também terão dois fragmentos. No entanto, a tabela time usando essa abordagem teria 2^{11} mintermos, sendo totalmente impraticável seguirmos com essa estratégia. Utilizaremos, ao invés disso, o conceito de *partitioning* no Apache Hive, que criará fragmentos da tabela *time* de acordo com um campo informado por nós durante a criação dela. Apesar do Hive permitir *partitioning* por vários campos, no caso da tabela *time* ao particionar por mais de um campo estávamos obtendo um erro por exceder o número de partições possível. Portanto optamos por particionar somente pelo campo *year*, obtendo 99 partições.

Por conveniência, também vamos usar o mecanismo de *partitioning* nas tabelas *district* e *neighborhood*. Para os fragmentos de *district* e *neighborhood*, também vamos omitir o campo com a geometria, pois ele não será usado nesse trabalho. Vale observar também que para todas as tabelas fragmentadas, quer usem *partitioning* ou não, vamos adicionar o sufixo "frag" ao seu nome, para melhor orientação. Assim o DDL das tabelas com fragmentação horizontal primária vai ficar como a seguir:

```
CREATE TABLE IF NOT EXISTS time_frag(
    id int,
    period string,
    day int,
    month int,
    weekday string
)
    partitioned by (year int);

CREATE TABLE IF NOT EXISTS neighborhood_frag(
    id int
)
    partitioned by (name string);

CREATE TABLE IF NOT EXISTS district_frag(
    id int
)
    partitioned by (name string);
```

O DDL das tabelas com fragmentação horizontal derivada(*vertice*, *segment* e *crime*) ficará igual ao das tabelas originais. Para a nomenclatura, além do sufixo "frag",

adicionaremos o número do fragmento. Portanto vertice tera os fragmentos vertice_frag_1, e vertice_frag_2. A tabela *segment* terá os fragmentos segment_frag_1 e segment_frag_2. No caso da tabela time, como ao particionar obtivemos um total de 99 partições, ficaria inviável para o escopo deste trabalho criar tantos fragmentos derivados. Optamos então por criar fragmentos somente com os anos que vamos presentes em cada consulta, e usar esse ano após o sufixo "frag" para nomear a tabela. Como exemplo o fragmento da tabela time para o ano 2015 ficará: time_frag_2015.

Tabela 1. Predicados simples

Time	District	neighborhood
Year = 2016	Name = 'IGUATEMI'	Name = 'SANTA EFIGÊNIA'
Year >= 2006		
Year <= 2016		
Year = 2015		
Year = 2012		
Year = 2017		
Year = 2010		
month = 11		
Weekday = 'saturday'		
Weekday = 'sunday'		

Para dar a carga dos dados nas tabelas particionadas utilizaremos o comando INSERT FROM SELECT. A diferença entre as tabelas com FHP e FHD será que para as primeiras será um select simples pegando dados da tabela original, e para as derivadas será um select com uma junção, porém pegando somente os dados da tabela membro, emulando um *semi-join* da álgebra relacional. Todos os comandos estarão disponíveis no repositório do trabalho no github, que ficará público após a data de entrega: https://github.com/italoportinho/trabalho-bdd-2024-1. Abaixo segue os inserts na tabela district_frag, no fragmento segment_frag_1, e no fragmento crime_frag_2016:

```
INSERT OVERWRITE TABLE district_frag
partition(name)
SELECT
    id,
    name
    FROM district;


INSERT OVERWRITE TABLE segment_frag_1
SELECT
    segment.*
    FROM segment, vertice_frag_1
    WHERE segment.start_vertice_id = vertice_frag_1.id
```

```
OR segment.final_vertice_id = vertice_frag_1.id;
```

```
INSERT OVERWRITE TABLE crime_frag_2016
SELECT
    crime.*
    FROM crime, time_frag
    WHERE crime.time_id = time_frag.id
    and time_frag.year = 2016;
```

3. O Cluster

O Apache Hadoop é um conjunto de utilitários de software de código aberto que facilita o uso de uma rede de muitos computadores para resolver problemas envolvendo grandes quantidades de dados e computação. Ele fornece um framework de software para o armazenamento distribuído e processamento de dados volumosos usando o modelo de programação MapReduce. O núcleo do Hadoop consiste em uma parte de armazenamento, conhecida como *Hadoop Distributed File System* (HDFS), e uma parte de processamento que é o modelo de programação *MapReduce*. Hadoop divide arquivos em blocos grandes e os distribui entre nós em um *cluster*. Em seguida, ele transfere código empacotado para os nós para processar os dados em paralelo. Este método aproveita a localidade dos dados, onde os nós manipulam os dados aos quais têm acesso. Isso permite que o conjunto de dados seja processado de maneira mais rápida e eficiente do que em uma arquitetura de supercomputador convencional que depende de um sistema de arquivo paralelo, onde a computação e os dados são distribuídos via rede de alta velocidade. Para o propósito deste trabalho vamos detalhar um pouco melhor duas partes do Hadoop: o *NameNode* e o *DataNode*

O *NameNode* está no centro do *cluster* Hadoop. Ele mantém a árvore de diretórios de todos os arquivos no sistema de arquivos e rastreia onde os dados do arquivo são realmente mantidos no *cluster*. O *DataNode*, armazena os dados reais do arquivo. Os arquivos são replicados em vários *DataNodes* em um cluster para maior confiabilidade. Especificamente, se pensarmos em termos de Hive, os dados armazenados em suas tabelas são espalhados pelos *DataNodes* dentro do *cluster* e é responsabilidade do *NameNode* rastrear esses blocos de dados. Neste trabalho, para simplificar, vamos utilizar somente um *DataNode*.

O Apache Hive é um projeto de software de *datawarehouse* construído sobre o Apache Hadoop para fornecer consulta e análise de dados em escala massiva. Hive oferece uma interface semelhante ao SQL para consultar dados armazenados em vários bancos de dados e sistemas de arquivos que se integram ao Hadoop. Consultas SQL tradicionais precisam ser implementadas na API Java do MapReduce para executar aplicativos e consultas SQL em dados distribuídos no entanto Hive fornece a abstração SQL necessária para integrar consultas semelhantes ao SQL (HiveQL) no Java subjacente sem a necessidade de implementar consultas na API Java de baixo nível. Hive facilita a integração de linguagens de consulta baseadas em SQL com o Hadoop, que é comumente usado em aplicações de *data warehousing*.

Da arquitetura do Hive vamos especificar melhor o conceito de *MetaStore*. O

Hive *MetaStore* armazena metadados para cada uma das tabelas, como esquema e localização no sistema de arquivos do Hadoop. Também inclui os metadados da partição que ajudam o driver a rastrear o progresso de vários conjuntos de dados distribuídos pelo cluster.

Vamos utilizar *containers* docker para implementar nosso cluster. Ele será constituído de 5 containers:

- namenode: operando uma imagem docker do hadoop;
- datanode: operando uma imagem docker do hadoop. Depende do namenode;
- hive-server: operando uma imagem do Hive. Depende do metastore;
- metastore: operando uma imagem do Hive. Depende do metastore-postgresql;
- metastore-postgresql: operando uma imagem do postgresql. Depende do datanode;

O *container* metastore-postgresql é utilizado para persistir os dados gerados pelo container metastore. Todo o código para subir o cluster a partir dos containers docker, bem como todas as consutas e o DDL das tabelas fragmentadas e não fragmentadas está disponível no repositório https://github.com/italoportinho/trabalho-bdd-2024-1, atualmente privado mas que será tornado público após a data de entrega do trabalho.

4. O Experimento

O *hardware* utilizado foi um *notebook* com processador Intel® Core i3-3110M 2.40GHz com 4 núcleos e 8 GB de memória RAM, rodando o sistema operacional Ubuntu 22.04.03 LTS. Foi utilizado o software *docker*(26.1.4) para criar os *containers* do cluster, que rodam imagens com Apache Hadoop(2.7.4) e Apache Hive(2.3.2) como visto nas seções anteriores.

Dentro de uma pasta foram criados os arquivos hadoop-hive.env, com as configurações para o namenode, datanode e hive-server, e docker-compose.yml, com as configurações necessárias para subir o 5 *containers* do nosso *cluster*. Nesse mesmo diretório também foi criada a pasta polroute, aonde deve ser descompactado o dataset, e foram criados dois arquivos HQL. Um contém o DDL das tabelas sem fragmentação(database_ddl_nofrag.hql), o outro contém o DDL das tabelas fragmentadas e os comandos de carga de dados para as tabelas com FHD(database_ddl_frag_partition.hql). Com essa estrutura de diretórios e arquivos configurada, podemos rodar no terminal o comando abaixo para subir o *cluster*:

sudo docker compose up

Com o *cluster* rodando, podemos abrir um outro terminal e entrar na linha de comando do *container* hive-server:

sudo docker exec -it hive-server /bin/bash

Agora podemos navegar ao diretório polroute, e executar os arquivo HQL com o DDL das tabelas. No exemplo abaixo vamos carregar as tabelas não fragmentadas:

cd ..

cd polroute

hive -f database_ddl_nofrag.hql

Com as tabelas criadas podemos dar carga nelas a partir dos dados de cada um dos arquivos csv presentes no diretório polroute:

hadoop fs -put crime.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/crime

hadoop fs -put time.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/time

hadoop fs -put district.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/district

hadoop fs -put neighborhood.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/neighborhood

hadoop fs -put vertice.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/vertice

hadoop fs -put segment.csv hdfs://namenode:8020/user/hive/warehouse/trabalho_bdd_nofrag.db/segment

Não devemos nos surpreender com o fato dessa carga ser extremamente rápida(mais ou menos 1 segundo) mesmo para os maiores arquivos. O que o hadoop faz é apenas copiar o arquivo informado para a sua localização dentro do HDFS. O Apache Hive também não faz nenhuma validação dos dados no momento da escrita, somente na leitura (*schema on-read*).

Com as tabelas criadas e preenchidas, podemos acessar a linha de comando do Hive e começar a rodar nossas queries.

hive

use trabalho_bdd_nofrag

Todo o procedimento de criação do *database* e da carga de dados é análogo para as tabelas fragmentadas. Cada *query*, fragmentada e não fragmentada, foi executada 10 vezes, com tempos muito semelhantes entre cada execução, e a médias dos tempos de execução está sumarizada na tabela 2. A query 7 foi forçada a retornar somente 100 linhas, pois estávamos interessados apenas nos maiores valores. Podemos observar também um print da execução da query 1(figura 3), ainda não fragmentada, e vemos que o processamento está distribuído entre os 4 núcleos do computador, em comparação com o estado dos núcleos antes da execução da query(figura 2).

Nas análises da *queries* 6 e 7 existe uma observação importante a ser feita. O valor entre parênteses na coluna do tempo de processamento médio das *queries* fragmentadas indica o tempo de processamento usando a tabela *segment* original, não fragmentada, ao invés da união de seus fragmentos, e podemos constatar que esse tempo é bem menor do que o registrado para *query* com a tabela fragmentada. Isso indica que a fragmentação pode não valer a pena nesse caso, seja pelo pequeno tamanho da tabela, ou quantidade de tuplas envolvidas no *join*, ou o fato da tabela *crime*, com seus mais de

7 milhões de tuplas, ser o grande gargalo desse banco de dados.

Tabela 2. Tempo médio de execução das consultas.

Consulta	T- não frag(seg)	T - frag(seg)	Linhas	Ganho
Query 1	221,39	38,18	926	82,75%
Query 2	227,53	44,26	1554	80,55%
Query 3	60,04	51,99	1	13,04%
Query 4	89,83	22,88	1	74,55%
Query 5	22,42	14,23	1	36,53%
Query 6	110,42	28,46 (14,03)	187	74,26%
Query 7	104,65	42,11 (27,22)	100*	59,76%

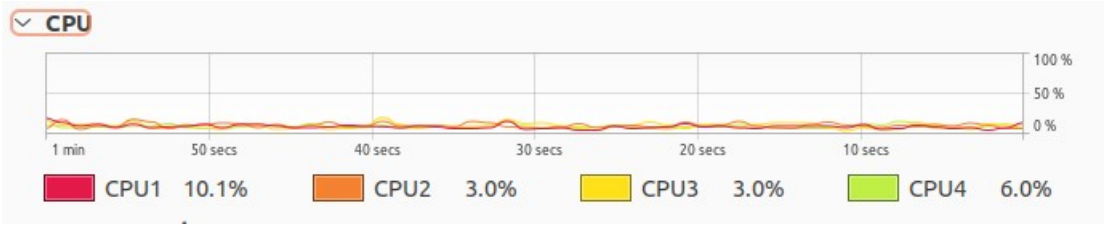

Figura 1: Carga dos núcleos da CPU sem nenhuma consulta rodando

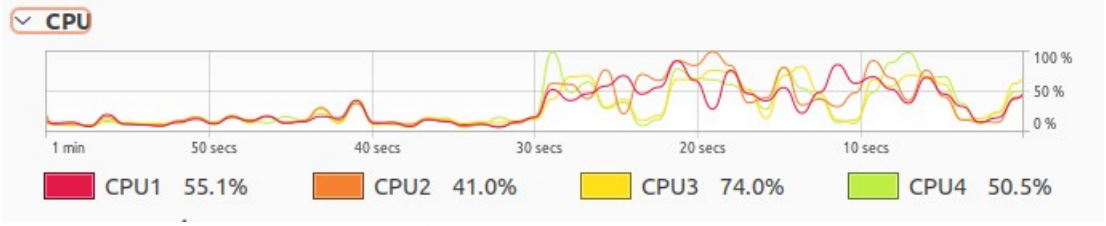

Figura 2: Carga dos núcleos da CPU com a query 1 fragmentada.

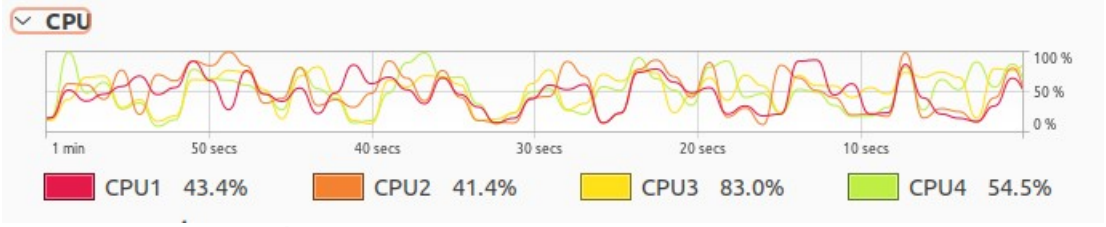

Figura 3: Carga das núcleos da CPU com a query 1 fragmentada, passado mais tempo.

Referências

[1] Cunha Sá, B., Muller, G., Banni, M., Santos, W., Lage, M., Rosseti, I., Frota, Y. and de Oliveira, D. 2022. PolRoute-DS: a Crime Dataset for Optimization-based

Police Patrol Routing. Journal of Information and Data Management. 13, 1 (Aug. 2022). DOI:https://doi.org/10.5753/jidm.2022.2355.

[2] Özsu, M. Tamer, Valduriez, P. (2011) “Principles of Distributed Database Systems - Third Edition”, p.81-98. Springer