CENTRO FEDERAL DE EDUCACAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Análise de desempenho de bancos de dados NoSQL em consultas de trajetória

Arthur Vinicius N de S Santa Rita

Iuri Bloch Valladares

Prof. Orientador:

Eduardo Soares Ogasawara, D.Sc.

Rio de Janeiro,
Julho de 2017

CENTRO FEDERAL DE EDUCACAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Análise de desempenho de bancos de dados NoSQL em consultas de trajetória

Arthur Vinicius N de S Santa Rita
Iuri Bloch Valladares

Projeto final apresentado em cumprimento às normas do Departamento de Educação Superior do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Prof. Orientador:
Eduardo Soares Ogasawara, D.Sc.

Rio de Janeiro,
Julho de 2017

Santa Rita, Arthur V. N. S.

Bloch, Iuri Valladares.

Análise de desempenho de bancos de dados NoSQL em problemas de trajetória / Arthur Vinicius Nascimento de Souza Santa Rita, Iuri Bloch Valladares - 2016.

x, 33 f; enc.

Conclusão de Curso, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2016.

Bibliografia: f, 30–33

RESUMO

Atualmente dados são gerados cada vez mais em maior quantidade, com esse crescimento vem a dificuldade de analisar todos esses dados, gerando o conceito de *Big Data*. Os bancos de dados relacionais nem sempre possuem um bom desempenho na análise de grandes quantidades de dados, por isso estão sendo usados os bancos de dados não relacionais (*NoSQL*) para analisar grandes volumes de dados. Nesse artigo o objetivo será testar o desempenho entre um banco de dados orientado a linhas e um banco de dados orientado a colunas. Serão utilizadas na experimentação consultas típicas de trajetória e os dados utilizados serão informações geradas por GPS de objetos em movimento.

Palavras-chave: Bigdata; NoSQL; PostGIS; MonetDB; PostgreSQL; trajetória

ABSTRACT

Currently data is generated increasingly greater amount, with this growth comes the difficulty of analyzing all this data, generating the concept of Big Data. Relational databases do not always have a good performance in the analysis of large amounts of data, so are being used non-relational databases (NoSQL) to analyze large volumes of data. In this article the goal will be to test performance between a row-oriented database and a column-oriented database. Typical trajectory queries will be used in the experiment and the data used will be GPS generated information of moving objects.

Keywords: Bigdata; NoSQL; PostGIS; MonetDB; PostgreSQL; Trajectory

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

Capítulo 1

Introdução

O crescimento na utilização de sensores para detecção de localização (GPS,RFID,etc), impulsionado recentemente pela popularização dos smartphones e aplicações que utilizam esse serviço, tem produzido um grande volume de dados. Aliado ao maior acesso à rede de computadores, é possível transmitir essas informações de forma mais rápida e eficiente. A coleta de dados oriundos desses dispositivos também deve ser observado. Devido ao grande volume de dados que esses sensores e dispositivos podem gerar, se inicia um problema que se enquadra no contexto do Big Data.

O acesso a dados gerados por esses dispositivos, possibilita que aplicações sejam desenvolvidas para auxiliar tomadas de decisões por indivíduos e organizações, simplificando situações do dia-a-dia [Ubaldi, 2013].

O portal de dados abertos da Prefeitura do Rio de Janeiro [Rio, 2016] disponibiliza um conjunto diverso de dados, relacionados com equipamentos e politicas da administração pública. Esses dados fornecem informações sobre mobilidade urbana, saúde, educação, arrecadações e despesas, entre outros. Essas publicações vêm em concordância com algumas recomendações do W3C para dados governamentais abertos [Bennett and Harvey, 2009]. Onde é necessário que sejam publicados dados em diferentes formatos.

Os dados oriundos dos deslocamentos de ônibus pela cidade, particularmente, contêm informações de georreferenciamento. A partir, deles podemos formular análises sobre problemas de trajetória. Os dados provenientes dos GPS, instalados em cada veículo, são responsáveis pelo maior volume de dados. Devido ao grande volume e complexidade dos dados gerados, é necessário realizar processo de ETL para aplicar transformações e limpeza de ruídos

As consultas de trajetória possuem aplicações em diversas áreas, não sendo limitados somente à analise de tráfego, como naturalmente poderiam ser associadas. Na Biologia, há o interesse em analisar o comportamento de animais, observando seu padrão de migração, por exemplo. Em aplicações militares, observar a movimentação de tropas inimigas. Identificar grupos que realizam movimentos em conjunto numa rede a fim de prever ataques. Em todas essas situações podemos utilizar consultas de trajetória, a partir de dados de localização, para

responder à essas questões.

O conceito Big Data aborda algumas características, como por exemplo grande volume de dados, complexidade dos dados, e uma taxa elevada de crescimento na quantidade deles [Wu et al., 2014]. A diversificação de dados, como os semi-estruturados ou não estruturados, comumente encontrados na Web, também não é atendida pelo modelo tradicional [Lóscio et al., 2011]. É necessário repensar formas para gerenciar uma grande quantidade de dados, com essas características, de maneira eficiente. Com o objetivo de atender algumas dessas demandas, surgem que são criados para receber um grande volume de dados, conhecidos como NoSQL.

Frequentemente atender à problemática do Big Data, é associado diretamente com bancos de dados NoSQL sendo, as vezes, confundidos como sinônimos, ou algo que é exclusivo à eles. É verdade que os bancos de dados que estão nessa categoria resolvem muitos problemas dessa área, porém é uma classificação muito abrangente. E, implementações do modelo relacional não podem ser descartadas como parte da solução. É compreendido como NoSQL os modelos que não se baseiam no relacional, dessa forma estão incluídos os modelos orientados à documento, orientados à matriz, grafos, hadoop, entre outros. Apesar de serem completamente diferentes um dos outros, estão englobados pela mesma terminologia.

Um único modelo não é suficiente para resolver todos os desafios relacionados ao gerenciamento de dados (armazenamento e análise de dados), sendo necessário utilizar outros em conjunto [Stonebraker et al., 2005]. Entender como cada um funciona, seus pontos fortes e seus pontos fracos para diferentes tipos de consultas e dados, é essencial para avançarmos na solução dos problemas de trajetória.

Dessa forma iremos desenvolver uma análise comparativa entre os modelos, a fim de documentar situações em que cada um pode agregar valor ou apontar aquelas nas quais, apresentem baixo desempenho. Levando em consideração a diferença entre eles, poderão existir limitações, tanto para popular os dados, quanto à realizações de determinadas consultas. Dessa forma, uma preocupação será em relação a quais estratégias iremos adotar, a fim de tornar a comparação equilibrada e justa.

Neste trabalho serão comparados dois modelos de banco de dados. O primeiro, o modelo relacional (orientado a linhas) armazena os dados em tuplas de uma tabela. O segundo modelo armazena os dados em colunas (orientado a colunas). Com exceção do primeiro modelo, o segundo modelo pertence a categoria dos modelos NoSQL. Utilizaremos os mesmos dados espaço temporais para serem carregados nos dois e, além disso, serão submetidos, também,

à consultas de trajetória semelhantes com a mesma complexidade. Com isso, será possível comparar de forma equilibrada o desempenho entre eles, uma vez que as condições e variantes são equivalentes.

O restante desse trabalho está dividido da seguinte forma: Capítulo II Dados de Trajetórias, onde é apresentado o conceito de trajetória e tipos de consultas que podem ser aplicadas à dados geo-espacias. Sendo divididas em consultas que se baseiam nas trajetórias, e consultas que se baseiam em ponto de interesse. No Capítulo III abordamos as principais características dos modelos de bancos de dados que iremos considerar para análise de desempenho. Serão explicitados as características da arquitetura, para cada modelo, que adicionam uma vantagem à eles. Assim como os pontos que possam representar uma desvantagem tanto do modelo relacional quanto ao modelo de colunas.

No capítulo IV, iremos abordar toda metodologia utilizada para realização deste trabalho, como apresentação da base de dados, definições das consultas de trajetória, modelagem de dados, como os dados foram processados e as métricas que serão utilizadas em cada consulta. O capítulo V será apresentada a avaliação experimental que será feita, apresentando também os sistemas utilizados e as consultas utilizadas.

No capítulo VI é apresentada a conclusão a partir dos resultados encontrados através da fase experimental, mostrando gráficos para indicar o desempenho que cada sistema teve.

Capítulo 2

Trajetória

2.1 Trajetória

Trajetória é o nome dado ao percurso realizado por um determinado objeto em movimento, com base em um sistema de coordenadas predefinido. Os pontos que compõem essa trajetória são formados pela coordenada e o tempo em que a observação foi feita. O intervalo de tempo é delimitado pelo instante em que o objeto inicia um dado percurso e o instante em que o objeto termina o mesmo percurso. A trajetória é composta por um inicio, fim, movimento e pode conter paradas [Spaccapietra et al., 2008].

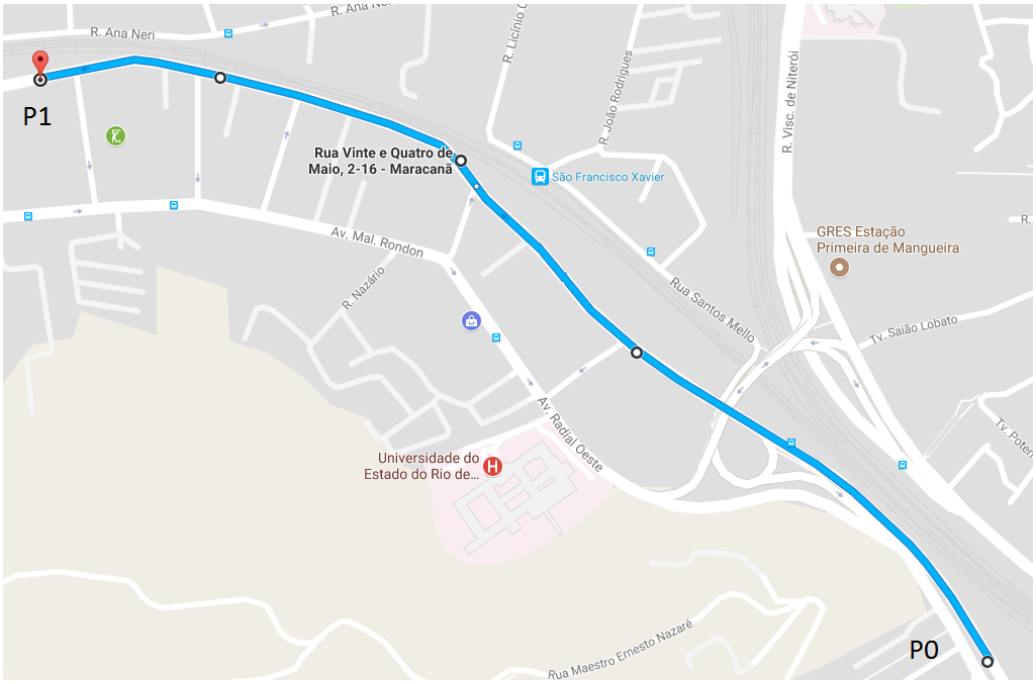

Figura 1: Trajetória

Tabela 1: Modelo de Trajetória

Ponto	Hora-Observaçao	Latitude	Longitude	Sequência
P0	2014-06-12 15:27:21	-22.9081820	-43.2385410	1
P1	2014-06-12 15:30:15	-22.9045850	-43.2429700	2

O início e fim estão estritamente ligados a como será analisada uma determinada série temporal. Usando, como exemplo a entrega de produtos de uma empresa, o entregador é o objeto em movimento e podemos analisar sua trajetória. O instante em que o entregador sai do estoque é caracterizado como o início da trajetória, enquanto o retorno à origem, caracteriza o fim.

Durante o percurso podem existir várias paradas, quando o entregador não está executando nenhum movimento. Cada entrega representa uma parada para este exemplo. Dessa forma uma trajetória pode ser composta por um conjunto de subtrajetórias. A trajetória, para a análise que desejamos fazer nesse caso, é o caminho total percorrido [Spaccapietra et al., 2008].

2.2 Modelo de trajetória

A partir do que foi apresentado na seção 2.1 como uma trajetória, o banco de dados que será utilizado nesse trabalho apresenta uma característica de segmento. Todo registro possui a latitude e longitude da observação e um número representando a sequência da observação. Dessa forma é possível traçar a trajetória a partir da sequência de registros do mesmo objeto em movimento, como mostra a tabela 2. É possível também, obter subtrajetórias de cada objeto em movimento observando os registros entre o registro que representa o início da trajetória e o registro que representa o fim da trajetória (na tabela 2 são representados pelos registro de sequência 1 e 5 respectivamente) desse objeto em movimento.

Tabela 2: Modelo de Trajetória

Ponto	Id	Latitude	Longitude	Sequência
P0	B71161	-22.9081820	-43.2385410	1
P1	B71161	-22.9045850	-43.2429700	2
P2	B71161	-22.9023550	-43.2451970	3
P3	B71161	-22.9013880	-43.2482260	4
P4	B71161	-22.9014110	-43.2505040	5

Para ilustrar a explicação anterior pode se usar como exemplo a figura 2. De forma que cada ponto, por exemplo o P0, seria um registro de um objeto em movimento específico, como mostra a tabela 2. Nesse exemplo, o objeto em movimento possuiria 5 registros. P0 seria o primeiro registro (primeira observação sobre o objeto em movimento) e o P4 seria o último registro (última observação sobre o objeto em movimento). Essa trajetória possui diversas subtrajetórias algumas seriam por exemplo: P1-P2-P3, P2-P3.

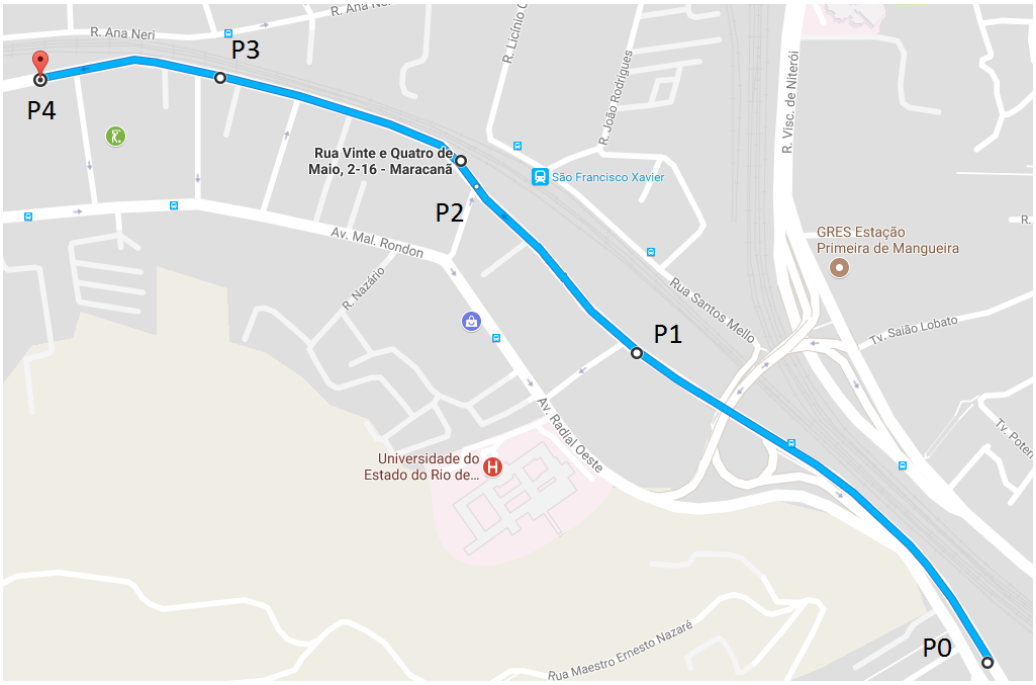

Figura 2: Modelo de trajetória

2.3 Consultas de trajetória

A partir da trajetória dos objetos em movimento, podemos formular consultas que busquem identificar diversos padrões ou informações relevantes para algumas aplicações. Essas consultas auxiliam, por exemplo, na análise de tráfego, comportamento de indivíduos ou animais (objetos em movimento). Se por exemplo, um motorista estiver interessado em saber quais são os postos de gasolina em sua rota de navegação, verificar se outros veículos estarão passando pela mesma estrada, ocasionando trânsito, ou o menor caminho até o seu destino dentre diversas opções.

Existem duas classes de consultas típicas de trajetória que atendem essas necessidades. A relação entre objetos em movimento e suas trajetórias (*T-Query*) e a relação entre objeto em movimento e um Ponto de interesse (*P-Query*) [Zheng and Zhou, 2011]. Abaixo, tais consultas e suas definições serão apresentadas.

2.3.1 P-Query

P-Query é uma classificação de consultas de trajetória baseada em pontos de interesse (POIs). Nesse tipo de consulta os POIs podem ser parâmetros para obter o retorno da consulta ou o próprio retorno dessa consulta. Os POIs podem ser por exemplo, postos de gasolina, praças, pontos de ônibus ou até mesmo pontos específicos de alguma trajetória.

Uma abordagem por exemplo, tem como objetivo encontrar um ou mais POIs que vão satisfazer uma relação espaço-temporal com uma ou mais trajetórias ou com objetos em movimento. Um clássico tipo dessa P-Query, é o de recuperar o objeto em movimento mais próximo (*nearest neighbor* [Roussopoulos et al., 1995]) de todo POI em uma trajetória [Cao et al., 2010]. Essa consulta tem como retorno os POIs, sendo a trajetória o parâmetro. Usando como exemplo um motorista que percorre uma estrada (trajetória definida), essa consulta poderia retornar os restaurantes, nessa trajetória, que estão abertos no horário do almoço.

Outra abordagem de P-Query visa encontrar uma ou mais trajetórias ou objetos em movimento dado um ou mais pontos de interesse. Temos como exemplo, a consulta baseada em um ponto único [Goldberg et al., 1992], que tem como objetivo encontrar a trajetória mais próxima de um POI. Além da consulta baseada em um ponto único, temos a consulta de trajetória de múltiplos pontos [Getoor and Sahami, 1999]. Nessa consulta, dado um pequeno conjunto de pontos, o objetivo é encontrar a trajetória que melhor percorra esses pontos geograficamente.

2.3.2 T-Query

T-Query é uma classificação de consulta de trajetória que tem como foco as trajetórias. Uma maneira de trabalhar com esse consulta é tratar a trajetória como uma sequência de pontos. É uma consulta que tem como retorno uma trajetória ou conclusões tiradas a partir dessa trajetória, como por exemplo quais ou quantos objetos em movimento passam por essa trajetória em um determinado período de tempo.

Uma abordagem da T-Query identifica trajetórias com características similares, ou seja, possuem objetos em movimento com as mesmas características ou possuem tráfego intenso em momentos iguais durante o dia. Ela é capaz de identificar subtrajetórias para auxiliar com tráfego intenso e encontrar rotas alternativas que ajudem a amenizar o tráfego [Kumar et al., 2005].

Capítulo 3

Modelos de Bancos de Dados

Um banco de dados representa aspectos do mundo real, sendo uma coleção de dados coerentes, com algum significado específico, permitindo a usuários ou aplicações armazenar e recuperar informações. Para a utilizar o banco de dados, é frequentemente utilizado um sistema de gerenciamento de banco de dados (SGBD), que simplificam o processo de definição, construção, manipulação e compartilhamento dos dados entre usuários e aplicações [Elmasri, 2008]. O SGBD disponibiliza uma interface para o usuário atualizar, incluir, deletar e consultar os dados armazenados. O modelo mais comum utilizado é o relacional, que utiliza a linguagem SQL (*Structured Query Language*), mas também existem outros modelos, denominados NoSQL que utilizam linguagens específicas para definição de dados e consultas.

3.1 Banco de Dados Relacional

Um banco de dados relacional possui uma coleção de tabelas, aonde cada uma possui um nome único. As tabelas são formalmente denominadas relações e representam entidades do mundo real. Cada tupla, ou linha, um conjunto de dados relacionados. A tabela representa um conjunto desses relacionamentos. Existe uma ligação muito próxima entre o conceito de tabela e o conceito matemático de relacionamento, de onde o modelo de banco de dados relacional tem seu nome [Silberschatz et al., 1997].

Tabela 3: Armazenamento por linha

Id	Código	Linha	Sequência
3459816	C20261	601	478
3460233	C21008	308	331
3461093	C21009	315	335
3462641	C21012	502	810
3480847	C21052	314	62

A definição do nome da tabela e seus atributos, com os respectivos tipos, formam o esquema, criando restrições para garantir integridade. A Tabela 3 representa a tabela *Ônibus* que possui

três atributos. Cada linha corresponde a um ônibus da cidade do Rio de Janeiro, e *Código*, *Linha*, *Sequência* são atributos dessa tabela.

Os SGBD de banco de dados relacionais implementam as propriedades ACID que é um acrônimo Atomicidade, Consistência, Isolamento, Durabilidade. Essas propriedades garantem a confiabilidade do banco de dados. A seguir serão explicados de forma individual cada propriedade.

A-atomicidade, na transação ou se faz tudo, ou não se faz nada. Pensando que em uma transação podemos ter mais de uma operação, então, em uma transação realizamos a inclusão de um cliente novo, a geração de uma nota fiscal e a baixa no estoque do produto vendido, ao final desta transação, devemos confirmar a transação por inteiro e gravar todas estas operações, se esta transação não se confirmar ao final, nenhuma destas operações pode ser gravada no banco de dados, garantindo assim a atomicidade da transação [Elmasri, 2008].

C-consistência, Tem por objetivo garantir que o banco de dados antes da transação esteja consistente e, que após a transação o banco permaneça consistente, sem problemas de integridade. Por exemplo, se um cliente em um banco possui 100 reais em sua conta e quer fazer uma retirada de 110 reais dessa mesma conta, essa transação não poderia ser concluída pois a consistência do banco de dados não estaria garantida deixando a conta com um saldo negativo [Elmasri, 2008].

I-isolamento, tem como objetivo garantir que nenhuma transação seja interferida por outra até que ela seja completada. No entanto existem transações que podem ocorrer de forma simultânea sob os mesmos dados, como por exemplo consultas. A seguir serão aplicadas as duas situações em exemplos práticos: Duas transações são iniciadas, as duas estão ligadas diretamente ao mesmo registro no banco de dados, a primeira atualizando, a segunda consultando, o isolamento nos garantirá que a transação de consulta somente será executada após a transação de atualização ser completada. No momento de consultar, podemos imaginar um sistema de vendas, que o mesmo produto pode ser consultado várias vezes ao mesmo tempo, visando saber o valor deste produto [Elmasri, 2008].

D-durabilidade, esta propriedade garante que a informação gravada no banco de dados dure de forma imutável até que alguma outra transação de atualização, ou exclusão afete-a. Este conceito garante que os dados não sejam corrompidos, ou seja, desapareçam ou se modifiquem sem motivo aparente [Elmasri, 2008].

3.2 Banco de Dados Não-Relacional (NoSQL)

O termo NoSQL é definido como um sistema que apresenta uma interface de consulta que não é apenas SQL [Brown and Wilson]. Essa é uma classificação bastante abrangente, onde modelos com características distintas são enquadrados. Esses modelos visam suprir necessidades que o relacional não consegue atender. Foram desenvolvidos para resolver problemas de Big Data como o armazenamento de grandes volumes de dados de forma eficiente, requerimentos de acesso, alta escalabilidade e disponibilidade com menos custo operacional e de gestão [Han et al., 2011].

Geralmente o NoSQL é utilizado quando se procura consistência, disponibilidade e que tenha tolerância a falhas. De acordo com o modelo CAP (Consistência, Disponibilidade e Tolerância à Partição), onde a Consistência é a característica de que o sistema fique consistente após uma operação, Disponibilidade é a característica de que o sistema seja assegurado que permaneça ativo durante um período de tempo determinado e Tolerância à Partição é a característica de que o sistema continue operando mesmo após uma falha na rede. Porém em modelos de banco de dados, apenas dois desses três itens podem ser corretamente satisfeitos[Gilbert and Lynch, 2002]

NoSQL não é apenas mais um modelo de banco de dados, mas um termo que define uma classe de modelos[Hecht and Jablonski, 2011], dentro dessa classe de modelos podemos citar algumas classes que seriam Armazenamento Chave-Valor (*Key-Value Store Databases*), Orientado a Coluna (*Column-Oriented Databases*), Orientado a Documento (*Document Store Databases*),Orientado a gráfico (*Graph Databases*).

Neste trabalho, iremos utilizar o modelo orientado a coluna, pois este modelo vem sendo aplicado na substituição da metodologia de armazenamento em linhas, devido a ambos tratarem do mesmo tipo de dados, trabalharem com a linguagem SQL e pela sua superioridade em desempenho com grandes quantidades de dados para certas aplicações.

3.3 Modelo de Banco de Dados Orientado a Coluna

Nesse modelo de dados, os atributos de uma relação são particionados em colunas. Cada coluna é uma representação binária contendo um *id* e o valor do atributo. O *id* é utilizado no mapeamento para um único valor e como identificador para a junção entre as colunas. A relação é composta pelo conjunto das relações binárias. Dessa forma é feito a reconstrução das tuplas,

com as junções das colunas, materializando a relação [Idreos, 2010].

Tabela 4: Armazenamento por coluna

Id	Código
3459816	C20261
3460233	C21008
3461093	C21009
3462641	C21012
3480847	C21052

Id	Linha
3459816	601
3460233	308
3461093	315
3462641	502
3480847	314

Id	Sequência
3459816	478
3460233	331
3461093	335
3462641	810
3480847	62

Os SGBD de bancos de dados relacionais implementam as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), enquanto os SGBD de modelos de dados NoSQL não garantem as propriedades ACID, mas garantem propriedades BASE (Basicamente Disponível, Estado Leve, Eventualmente Consistente) [Nayak et al., 2013].

O BASE é diametralmente oposta ao ACID, onde ACID é pessimista e força a consistência no final de cada operação e o BASE é otimista e aceita que a consistência do banco de dados estará em um estado de fluxo. Apesar disso parecer impossível de lidar, na realidade é bastante gerenciável e leva a níveis de escalabilidade que não podem ser obtidos com ACID [Pritchett, 2008]. Ou seja, enquanto no modelo BASE a disponibilidade tolera a falha parciais e a persistência não é efetivada em tempo real, no modelo ACID a cada operação realizada, ocorre a persistência após.

Tabela 5: ACID x BASE

ACID	BASE
Consistência forte	Fraca consistência
Isolamento	Foco em disponibilidade
Conservador (pessimista)	Agressivo (otimista)
Evolução difícil (esquema, por exemplo)	Evolução mais fácil

Uma aplicação funciona basicamente todo o tempo (Basicamente Disponível), não tem de ser consistente a todo tempo (Estado Leve) e o sistema torna-se consistente no momento devido (Eventualmente Consistente).

3.4 Comparação entres os modelos de bancos de dados

Nesse tópico serão comparados os modelos de bancos de dados orientado a linhas e o modelo de banco de dados orientado a colunas. A decisão de comparar os dois modelos se tornou

justificável uma vez que os dois modelos de banco de dados apresentam características iguais [MonetDB, 2016; Postgres, 2016]. A seguir serão apresentadas algumas vantagens e desvantagens do modelo de banco de dados orientado a colunas em relação ao modelo de banco de dados orientado a linhas.

Uma das vantagens seria o melhor uso da banda larga, em um modelo de banco de dados orientando a colunas somente os atributos que são acessados pela consulta são alocados em disco. Já no modelo de banco de dados orientado a linhas os atributos atrelados ao atributo consultado também são carregados em disco [Khoshafian et al., 1987].

A compressão de dados no modelo orientado a colunas também funciona de forma mais eficiente, uma vez que ao armazenar dados do mesmo domínio juntos o modelo orientado a colunas melhora o conceito de localidade e também a taxa de compressão de dados é aumentada. Dessa forma, ocorre uma diminuição no uso de banda larga uma vez que é mais fácil transferir dados comprimidos [Abadi et al., 2006].

Melhor utilização de memória cache, uma linha de cache tende a ser maior que a os atributos da tupla, logo uma linha de cache pode conter atributos irrelevantes no modelo de banco de dados orientado a linhas. Dessa forma é desperdiçado espaço na cache e reduz a frequência de *hits* [Ailamaki et al., 2001].

A seguir serão apresentadas desvantagens do modelo de banco de dados orientados a colunas em relação ao bando de dados relacional. O disco lê cada bloco enquanto mais de uma coluna é lida em paralelo, aumentando o tempo da busca em disco. Porém, se o *pre-fetch* do disco for do tamanho ideal, esse custo pode ser controlado [Abadi et al., 2007].

Outra desvantagem é o alto tempo de inserção de dados. Uma vez que é preciso inserir muitos dados em locais diferentes do disco o desempenho não é satisfatório. Porque é preciso atualizar o disco para cada tupla inserida, esse custo é detectado com uma inserção em massa é aplicada [Abadi et al., 2007].

Todas as vantagens e desvantagens apresentadas devem ser levadas em consideração no momento de escolher um dos modelos para se trabalhar. Uma vez apresentadas essas características do modelo de banco de dados orientado a colunas, elas aparentam ser mais capazes de trabalhar com grandes quantidades de dados, portanto será feita a comparação de desempenho com o modelo de banco da dados orientado a linhas, para mostrar se o armazenamento em colunas é de fato mais eficiente.

Capítulo 4

Metodologia

A análise do desempenho possui como objetivo principal saber se o modelo colunar abordado possui alguma vantagem em relação ao modelo relacional que é o modelo tradicional e mais usado atualmente. Desta forma para desenvolver essa pesquisa, é preciso buscar uma base de dados sólida e que abordasse um tema ligado a trajetórias, tema apresentada no Capítulo II. A partir desses dados serão executadas consultas típicas de trajetória nos modelos de banco de dados apresentados no Capítulo III. A metodologia para análise do desempenho dos modelos de bancos de dados é composta por quatro etapas conforme apresentado na Figura 3. A Base de Dados aborda a origem e como está estruturado os dados neste trabalho, já em Processamento dos Dados é explicado todo o processo realizado desde da coleta dos dados até a importação nos bancos de dados. Em Definição das consultas de trajetória abordamos os conceitos e o retorno de cada consulta utilizada e Métricas para avaliação mostra as práticas utilizadas para avaliação de cada consulta. As próximas seções, aborda em detalhes as atividades executadas em cada um destes estágios.

Figura 3: Etapas utilizadas na metodologia

4.1 Base de dados

Os dados para a modelagem do teste de desempenho foram obtidos por meio de bases de dados públicas providas por instituições governamentais e possuem informações relacionadas aos ônibus da cidade do Rio de Janeiro. A cidade do Rio de Janeiro possui uma frota de ônibus com mais de 8.000 ônibus urbanos. Além dos ônibus, a cidade do Rio de Janeiro possui também o metrô, barcas e o VLT (Veículo Leve sob Trilho) que dividem a responsabilidade de realizar

o transportar a população diariamente. Cada ônibus possui um aparelho de GPS, onde coleta as informações diárias de cada ônibus, ou seja, coleta a latitude e longitude por onde o ônibus passou, velocidade, data e outras informações.

A informação do GPS de cada ônibus é disponibilizada pelo Portal de Dados Abertos da Prefeitura do Rio de Janeiro (Data.rio), esse portal é uma ferramenta fornecida pelo governo municipal para que pessoas possam utilizar os dados que são gerados pela cidade. Os dados abertos, no Portal de Dados Abertos da Prefeitura do Rio de Janeiro, são organizados de tal maneira que permita sua reutilização em aplicativos digitais desenvolvidos pela sociedade. Isso proporciona ao cidadão um melhor entendimento do governo municipal, no acesso aos serviços públicos e na participação no planejamento, desenvolvimento das políticas públicas e melhor conhecimento da cidade.

O Portal de Dados Abertos da Prefeitura do Rio de Janeiro disponibiliza uma serie de dados referentes a transporte, esporte, entretenimento e outros. Alguns desses dados são atualizados diariamente e disponibilizados no portal. Os dados disponibilizados pelo portal englobam os anos de 2012 até hoje. Através desse portal, recolhemos os dados referentes aos ônibus que circulavam durante o período da Copa do Mundo de 2014, evento que aconteceu na cidade entre o mês de junho e julho.

Na Figura 4, podemos ver o modelo de dados que foi utilizado. A entidade ônibus representa cada veiculo pertencente a uma linha. A entidade ponto representa os atributos referentes a localização do ônibus. E por fim, temos a entidade data que representa a hora e data que o ônibus passou pelo ponto observado.

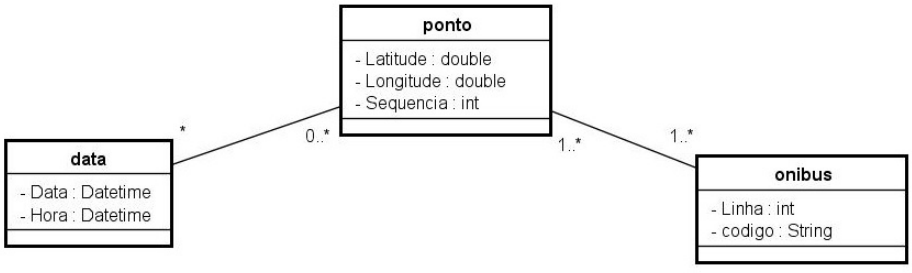

Figura 4: Modelo de dados

4.2 Processamento dos Dados

Na base de dados escolhida, os registros dos ônibus possuem coordenadas de localização, como latitude e longitude, data e hora, linha, código e sequência de cada observação gerada pelo

GPS de cada ônibus. As etapas realizadas para tornar esses dados disponíveis para execução das consultas, consistem em: Limpeza dos dados, gerar o CSV e a carga em cada modelo de banco de dados. Na Figura 5, podemos ver com mais detalhe todo o processo executado, desde da coleta dos dados no Data. Rio até a importação dos dados nos bancos de dados. Abaixo, será detalhado como cada processo será realizado.

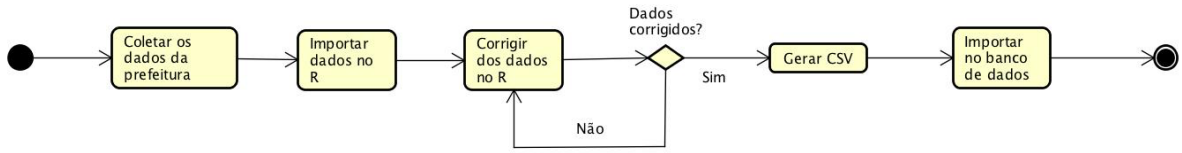

Figura 5: Processo utilizado para carga dos dados

Os dados foram disponibilizados e coletados do portal Data.rio, sendo disponibilizado no formato JSON. A geração desses dados é feita em tempo real, minuto a minuto, e ficam disponível por um curto período de tempo, não sendo possível recuperar os dados para uma carga histórica.

Após a coleta dos dados referente ao período escolhido, foi realizada a importação dos dados no R para realização das correções. Após importar os dados no R, foi observado que alguns dados estavam incompletos e outros estavam repetidos.

Na etapa corrigir dados no R, foi feita a limpeza dos dados, detectando os *outliers* e removendo essas tuplas. Além disso, alguns registros, como por exemplo velocidade, estavam em branco, ou seja, com valores faltantes, também foram removidos da base.

Com a limpeza dos dados realizada, através do R foi gerado o dataset no formato CSV para realizar a importação em cada modelo de banco de dados.

4.3 Definição das Consultas de trajetória

Os dados que serão objetos de análise desse trabalho são referentes as informações geradas pelo GPS instalado nos ônibus da cidade do Rio de Janeiro [Rio, 2016]. Através desses dados recolhidos, serão utilizadas as localizações dos ônibus. A partir dessas informações, serão realizadas consultas que estejam enquadradas em cada uma das classificações descritas no Capítulo II. Dessa forma serão formuladas duas consultas para serem executadas em cada modelo de banco de dados.

Tabela 6: Consultas de trajetória

Consulta	Descrição	Classificação
Q1	Obter os ônibus que passaram por um ponto determinado.	P-Query
Q2	Obter os ônibus que passam por N pontos determinados formando uma trajetória.	T-Query

A Tabela 6 mostra a descrição e classificação de cada consulta que será executada em banco de dados pertencente a cada modelo de dados abordado. Ao todo serão executadas duas consultas, onde dentro de cada consulta poderá existir variações de valores. Abaixo será descrito cada consulta abordada na Tabela 6.

Na consulta Q1, serão retornados o código e o número das linhas dos ônibus que passam por um ponto de interesse determinado. Nesta consulta haverá uma variação utilizando três tipos de ponto de interesse diferentes. Os pontos são: Praça General Osório, Maracanã e a Praia de Copacabana (Posto 5).

Para execução da consulta foram definidos intervalos entre a latitude e longitude de cada ponto. Esta consulta é classificada como uma P-Query, visto que estamos utilizando um ponto de interesse para ter como retorno as linhas dos ônibus que passam por esse ponto de interesse.

Na consulta Q2, serão retornados o código e o número das linhas dos ônibus que passam pela trajetória determinada. Esta consulta possui uma variação utilizando N pontos sequenciais formando uma trajetória. Cada trajetória formada será denominada como uma variação dentro da consulta, essas variações serão denominadas como TM, onde M é equivalente ao número de variações. Todas as trajetórias serão criadas utilizando pontos localizados na cidade do Rio de Janeiro.

Para execução da consulta foram definidos intervalos entre a latitude e longitude de cada ponto. Esta consulta é classificada como uma T-Query, pois está sendo verificado quais e quantos ônibus percorrem a mesma trajetória.

4.4 Métricas para avaliação

Cada consulta será executada quatro vezes em intervalo de tempos distintos, ou seja, não serão executadas consecutivamente. Esse método de execução, evita que o cache seja armazenado e comprometa no resultado do tempo da consulta.

Após contabilizados o tempo de cada execução, será feito um cálculo médio para saber o

tempo que cada modelo de dados levou para obter o retorno esperado. A medição do tempo será feita em segundos e ambos os modelos receberão os mesmos valores como parâmetros. Após contabilizar os tempos de cada execução será feita uma análise no plano de execução da consulta para entender o motivo do tempo da execução da consulta.

As métricas explicadas acima serão utilizadas em cada variação da consulta Q1 e em cada variação da consulta Q2. As consultas serão executadas na linguagem SQL, visto que, ambos os modelos de bancos possuem compatibilidade, com isso não haverá vantagem para nenhum deles.

E para apresentar os resultados das variações das consultas, serão gerados gráficos em barra para ter uma melhor visualização da comparação do tempo que cada variação de cada consulta levou para ser executada por completo em cada modelo de banco de dados. E uma tabela com o tempo geral de cada consulta.

Capítulo 5

Avaliação Experimental

Nossa análise foi realizada com o objetivo principal saber se o modelo colunar abordado possui alguma vantagem em relação ao modelo relacional. Ou seja, saber se realmente o modelo colunar é mais rápido que o modelo relacional, e também saber qual a diferença em segundos que cada um leva para executar as consultas abordados no capítulo 4.

Para desenvolver essa análise, a avaliação experimental foi dividida em sistemas avaliados, dataset e avaliação das consultas. Em sistemas avaliados, é abordado sobre o gerenciador de cada modelo de banco de dados. Em dataset, é possível entender sobre os dados utilizados, quantas tuplas foram importadas e como a tabela foi criada com os atributos em cada gerenciador de banco de dados. Por fim em avaliação das consultas, é abordado o tempo de execução de cada gerenciador e o respectivo plano de execução gerado.

5.1 Sistemas Avaliados

5.1.1 PostgreSQL

PostgreSQL é um sistema gerenciador de banco de dados relacional, onde uma das suas principais características é ser um sistema de código livre (open-source). Há bastante tempo em atividade, além de ser um sistema gratuito, ganhou bastante reputação por não só apenas salvar os dados seguramente, mas também pela confiabilidade e integridade dos dados.

O PostgreSQL evoluiu a partir do projeto Ingres na Universidade da Califórnia, em Berkeley. Quando Michael Stonebraker iniciou um projeto para abordar os problemas com os sistemas de banco de dados contemporâneos que se tornaram cada vez mais claros no início dos anos 80. Esse projeto tinha como objetivo agregar o menor número de recursos necessários para suportar as aplicações da época. Alguns desses recursos era a capacidade de definir tipos e descrever relacionamentos entre as tabelas do banco [Momjian, 2001].

Após publicação de uma série de artigos descrevendo o sistema, teve sua primeira versão publicada. E a cada ano, era lançado cada versão com melhorias em relação a versão anterior

com adição de recursos. Em 1994, estudantes de graduação de Berkeley, Andrew Yu e Jolly Chen, substituíram o intérprete de linguagem de consulta por um para a linguagem de consulta SQL. Após esse ano, o PostgreSQL ganhou uma licença liberal permitindo que fosse um programa modificável [Drake and Worsley, 2002].

Totalmente compatível com as propriedades ACID, além de incluir a maioria dos tipos de dados como inteiro, numérico, carácter e outros, também possui suporte para *materialized views, triggers,functions e stored procedures* [Drake and Worsley, 2002].

Com uma grande comunidade ativa e bastante utilizado em meios acadêmicos e comerciais, o PostgreSQL é considerado um dos os bancos de dados de código aberto mais avançado do mundo, pois fornece uma grande variedade de recursos que normalmente só se encontram no banco de dados comercial. Podemos citar como um dos seus principais recursos, um amplo suporte à linguagem SQL, suporte a Controle de Concorrência em Múltiplas Versões (MVCC), replicação assíncrona, transações aninhadas, otimizador de consultas sofisticados. É altamente escalável tanto na grande quantidade de dados que pode gerenciar quanto na quantidade de usuários simultâneos que pode acomodar. Além disso, possui é compatível algumas linguagens de programação, como JAVA, C/C++ e Phyton [Douglas and Douglas, 2003].

5.1.2 PostGIS

O PostGIS é uma extensão de banco de dados espacial para o popular gerenciador de banco de dados relacional PostgreSQL. Permitindo que o PostgreSQL seja usado em sistemas de informações geográficas (GIS) e aplicações de mapeamento na web. Como o PostgreSQL, O PostGIS também é um projeto de código aberto, sendo desenvolvido e mantido pela empresa Refractions Research.

Após a tentativa de colocar dados espaciais no PostgreSQL, apesar de possuir nativamente tipos geométricos, eram muito limitados para dados e análises do SIG (ou GIS) (Sistemas de Informação Geográfica). Através desse cenário e com enfrentando problemas para armazenar dados espaciais, a empresa Refractions Research resolveu desenvolver uma extensão para o PostgreSQL, visto que o PostgreSQL já era utilizado pela empresa e existia a capacidade de adicionar extensões por ser código livre (*open source*) [Ramsey et al., 2005].

Em 2001, após sua primeira implementação, obteve um resultado muito superior ao que era esperado, sendo cerca de dez vezes mais rápido que o uso do subsistema genérico BLOB (objeto binário). Após esse resultado, foram adicionadas melhorias permitindo adicionar um

índice espacial, através do suporte para ligações R-Tree ao subsistema de índices espaciais GiST [Ramsey et al., 2005].

O PostGIS adiciona tipos extras (geometria, geografia) ao PostgreSQL. Também adiciona funções, operadores e aprimoramentos de índice que se aplicam a esses tipos espaciais. Essas funções adicionais, operadores, vinculações de índice e tipos, aumentam o poder do PostgreSQL, tornando-se um sistema de gerenciamento de banco de dados espacial rápido, funcional e robusto [Obe and Hsu, 2015].

5.1.3 MonetDB

MonetDB é um sistema pioneiro de banco de dados orientado a coluna, onde suas principais características é ser um sistema NoSQL de código livre (*open-source*). Foi desenvolvido para ser usado em banco de dados em larga escala, ou seja, com grande quantidade de dados armazenada. Diferente dos tradicionais bancos orientado a linha, como PostgreSQL, o armazenamento em coluna promove uma solução moderna e escalável sem precisar de um grande investimento em servidores (hardware) [MonetDB, 2016].

MonetDB é compatível com sistemas operacionais Linux e Windows XP [MonetDB, 2016]. E também, é compatível com o padrão SQL 2003, ou seja, consegue interpretar as consultas escritas na linguagem SQL, tendo suporte a *joins, views, triggers e stored procedures*. Além disso, possui integração com a linguagem R, essa integração é feita a partir de um módulo instalado [MonetDB, 2016]. É também compatível com as propriedades ACID e suporta uma grande quantidade de interfaces de programação (JDBC, ODBC, PHP, Python, RoR, C/C++, Perl) [Idreos et al., 2012].

É composto por dois níveis o *front-end e back-end*, onde o *front-end* possui módulos para suporte a idiomas de consulta, como o SQL, e o *back-end* que atua como uma máquina virtual de álgebra BAT (Associação binaria de tabelas), onde internamente o MonetDB armazena colunas usando arquivos mapeados em memoria [Boncz et al., 2008]. Na Figura 6, apresenta os dois níveis ilustrados.

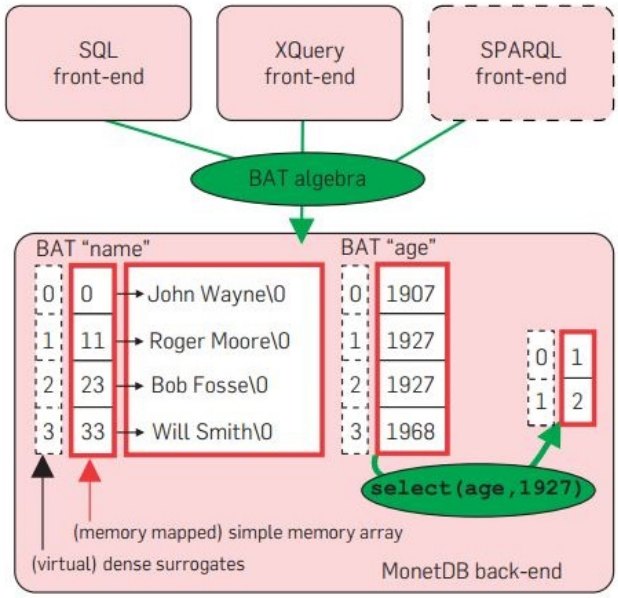

Figura 6: Arquitetura interna MonetDB, retirada de [Boncz et al., 2008]

5.2 Dataset

O dataset possui as trajetórias dos ônibus percorridas durante o evento da Copa do Mundo de 2014 realizado no estado do Rio de Janeiro. Os dados representam um período de trinta dias de observações, foram coletados no intervalo que iniciou na data 12 de junho de 2014 e terminou na data 12 de julho de 2014.

O dataset possui um total de 99.397.279 tuplas, onde a cada tupla possui as informações de latitude, longitude, data, linha e código referente ao ônibus. O arquivo com extensão .CSV possui um tamanho de 5.420.059 kilobytes (KB), ou 5,420059 gigabytes (GB), que será lido e inserido em cada banco de dados.

Os dados foram inseridos nos bancos de dados em uma única tabela chamada de trajetória. Na Tabela 7, podemos ver como ficou definido os tipos de dados para cada coluna em cada banco de dados.

Tabela 7: Colunas da tabela trajetoria

PostgreSQL / MonetDB	PostGIS
data (timestamp) codigo (varchar(255)) linha (varchar(255)) sequencia (decimal) latitude (decimal) longitude (decimal)	data (timestamp) codigo (varchar(255)) linha (varchar(255)) sequencia (decimal) geo (GEOMETRY(POINT))

Como podemos observar na Tabela 7, as colunas latitude e longitude foram transformadas em uma única coluna geo para utilização das funções geométricas encontradas no PostGIS.

5.3 Avaliação das Consultas

5.3.1 Consulta Q1

Essa consulta é definida como uma P-Query, pois estamos definindo um ponto de interesse. Para realizar a consulta, definimos uma estrutura onde passamos as coordenadas (latitude e longitude) de um ponto de interesse e obtemos como retorno as linhas e códigos dos ônibus que passaram por esse ponto. Esse ponto de interesse foi definido através de intervalos de valores da latitude e longitude. Nas Figuras 7 e 8, podemos visualizar a estrutura da consulta utilizada nos bancos.

```
a = LOAD ‘data’ USING BinStorage AS (user);
b = GROUP a BY user;
c = FOREACH b GENERATE COUNT(a) AS cnt;
d = ORDER c BY cnt;
```

select t.codigo, t.linha from trajetoria t

Figura 7: Consulta utilizada no PostgreSQL e MonetDB

```
SELECT t.codigo,t.linha
FROM trajetoria t
WHERE
ST_Contains( ST_MakePolygon( ST_GeometryFromText('LINESTRING(COORDENADAS DO PONTO DE OBSERVACAO)') )  ,
geo );
```

Figura 8: Consulta utilizada no PostGIS

Utilizando as consultas acima, definimos alguns pontos que utilizamos neste trabalho. Na Tabela 8 podemos ver os pontos e como foi definido o intervalo de cada ponto de interesse.

Tabela 8: Pontos de interesse

Descrição	Menor (Latitude, Longitude)	Maior (Latitude, Longitude)
Praça General Osório	(-22.98575, -43.19859)	(-22.98574, -43.19709)
Estádio Maracanã	(-22.91395, -43.23115)	(-22.91380, -43.23090)
Praia de Copacabana(Posto 5)	(-22.98119, -43.18954)	(-22.98011, -43.18934)

Nas Figuras 9, 10 e 11, podemos ver o plano de execução de cada gerenciador de banco de dados ao executar a consulta.

```
------------------------------------------------------------------------------------------------------------
--------------------------
Seq Scan on trajetoria t  (cost=0.00..2881634.76 rows=24 width=10)
  Filter: ((latitude > '-22.98575'::numeric) AND (latitude < '-22.98574'::numeric) AND (longitude >= '-43.19859'::numeric) AND (longitude
<= '-43.19709'::numeric))
```

Figura 9: PostgreSQL - Plano de execução

```
-----------------------------------
 HashAggregate  (cost=27116602.73..27116602.74 rows=1 width=3)
   Group Key: linha
   ->  Seq Scan on trajetoria t  (cost=0.00..27116519.90 rows=33132 width=3)
         Filter: (('010300000001000000050000003108AC1C5AFC36C005A8A9656B9945C0EA5BE67459FC36C0302FC03E3A9945C0DCF4673F52FC36C0E1D1C6116B9945C0F8A57EDE54FC36C00C59DDEA399945C03108AC1C5AFC36C005A8A9656B9945C0
'::geometry ~ geo) AND _st_contains('010300000001000000050000003108AC1C5AFC36C005A8A9656B9945C0EA5BE67459FC36C0302FC03E3A9945C0DCF4673F52FC36C0E1D1C6116B9945C0F8A57EDE54FC36C00C59DDEA399945C03108AC1C5AFC36C
005A8A9656B9945C0'::geometry, geo))
```

Figura 10: PostGIS - Plano de execução

```
+----------------------------------------------------------------------------------------------------------+
| rel                                                                                                      |
+==========================================================================================================+
| project (                                                                                                |
| | select (                                                                                               |
| | | table(sys.trajetoria) [ trajetoria.codigo as t.codigo, trajetoria.linha as t.linha, trajetoria.latitude as t.latitude, trajetoria. |
: longitude as t.longitude ] COUNT                                                                          :
| | ) [ decimal(18,7)[sys.sql_neg(decimal(7,5) "2298575")] < t.latitude < decimal(18,7)[sys.sql_neg(decimal(7,5) "2298574")], decimal(18 |
: ,7)[sys.sql_neg(decimal(7,5) "4319859")] <= t.longitude <= decimal(18,7)[sys.sql_neg(decimal(7,5) "4319709")] ]  :
| ) [ t.codigo, t.linha ]                                                                                  |
+----------------------------------------------------------------------------------------------------------+
```

Figura 11: MonetDB - Plano de execução

Ao analisar os planos de execução, temos um grande custo na projeção da consulta, ou seja, no intervalo das latitude e longitude, e no atributo geo que é referente ao PostGIS. No resultado final, esse grande custo é diretamente influenciável no tempo de execução da consulta.

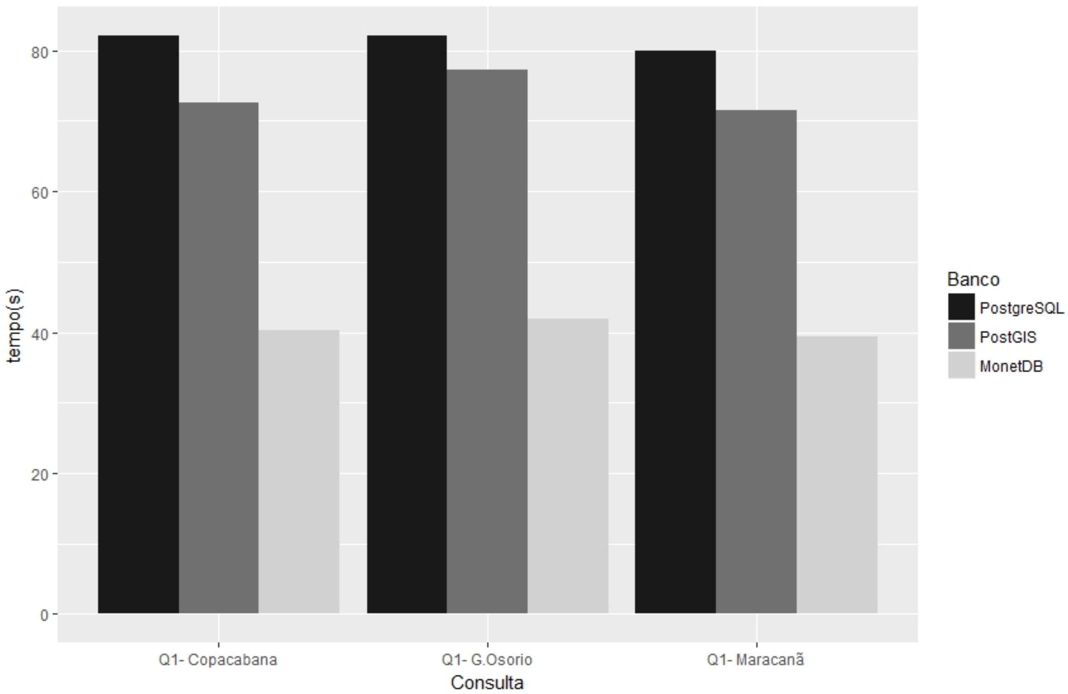

Figura 12: Tempo de execução dos bancos

Ao analisar o gráfico da Figura 12, pode concluir que o MonetDB leva grande vantagem em relação aos demais bancos, quando estamos analisando o ponto de interesse, já o PostGIS possui uma pequena vantagem em relação ao PostgreSQL.

5.3.2 Consulta Q2

Essa consulta é definida como uma T-Query. Para realizarmos essa consulta definimos uma estrutura onde passamos as coordenadas (latitude e longitude) de pontos sequenciais. O retorno da consulta são as linhas e códigos dos ônibus que passam por esses pontos, ou seja, que realizaram a mesma trajetória na mesma data. Nas Figuras 13 e 14, podemos visualizar a estrutura da consulta utilizada.

```
SELECT t.linha,t.codigo,t.sequencia,t2.linha,t2.codigo,t2.sequencia
FROM trajetoria t, trajetoria t2
WHERE t.codigo=tt.codigo AND t.data  = tt.data
AND tt.sequencia <= t.sequencia+10 AND tt.sequencia > t.sequencia
AND t.latitude >= menor latitude AND t.latitude <= maior latitude
AND t.longitude >= menor longitude AND t.longitude <= maior longitude
AND t2.latitude >= menor latitude AND t2.latitude <= maior latitude
AND t2.longitude >= menor longitude AND t2.longitude <= maior longitude ;
```

Figura 13: Consulta utilizada no PostgreSQL e MonetDB

```
SELECT t.linha,t.codigo,t.sequencia,tt.linha,tt.codigo,tt.sequencia
FROM trajetoria t,trajetoria tt
WHERE
ST_Contains( ST_MakePolygon( ST_GeometryFromText('LINESTRING(COORDENADAS DO PONTO DE OBSERVACAO)') )  ,
t.geo )
AND t.codigo=tt.codigo AND t.data  = tt.data
AND tt.sequencia  <= t.sequencia+10 AND tt.sequencia > t.sequencia
AND ST_Contains( ST_MakePolygon( ST_GeometryFromText('LINESTRING(COORDENADAS DO PONTO DE OBSERVACAO)') )  ,
tt.geo );
```

Figura 14: Consulta utilizada no PostGIS

Nesta consulta utilizamos três variações chamadas de T1, T2 e T3, onde cada variação possui trajetórias com dois, três e quatro pontos de observação. Abaixo podemos ver cada variação com seus respectivos pontos de observação.

Na tabela 9, podemos ver na variação T1, uma trajetória composta por dois pontos, com seus pontos definidos com suas respectivas coordenadas e sendo ilustrados na Figura 15.

Tabela 9: Pontos da Variação T1

Ponto	Menor (Latitude, Longitude)	Maior (Latitude, Longitude)
P0	(-22.97899,-43.22430)	(-22.97888,-43.22418)
P1	(-22.96445,-43.21475)	(-22.96430,-43.21460)

Figura 15: Trajetória T1

Na variação T2, temos uma trajetória composta por três pontos. Na tabela 10 temos os valores de latitude e longitude dos pontos, já na Figura 16 temos a ilustração de toda a trajetória.

Tabela 10: Pontos da Variação T2

Ponto	Menor (Latitude, Longitude)	Maior (Latitude, Longitude)
P0	(-22.91019,-43.20525)	(-22.90995,-43.20504)
P1	(-22.90456,-43.18877)	(-22.90454,-43.18843)
P2	(-22.90184,-43.17891)	(-22.90167,-43.17890)

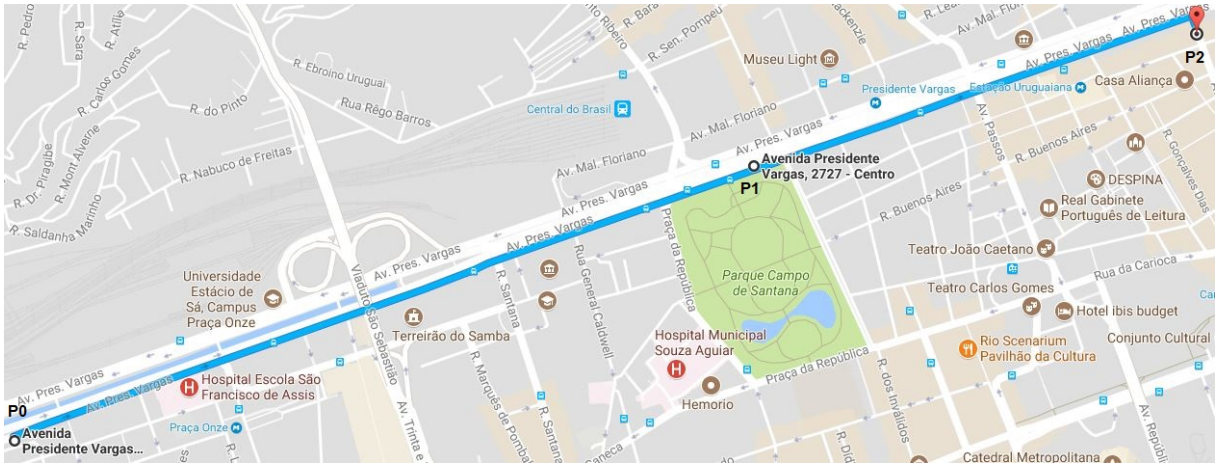

Figura 16: Trajetória T2

Na variação T3, temos uma trajetória composta por quatro pontos. Na tabela 11, temos uma visão dos valores de latitude e longitude dos pontos, já na Figura 17 temos a ilustração de toda a trajetória.

Tabela 11: Variação T3

Ponto	Menor (Latitude, Longitude)	Maior (Latitude, Longitude)
P0	(-22.92577, -43.25834)	(-22.92575, -43.25832)
P1	(-22.91871, -43.24066)	(-22.91870, -43.24065)
P2	(-22.91718, -43.22502)	(-22.91717, -43.22501)
P3	(-22.91284, -43.21599)	(-22.91283, -43.21597)

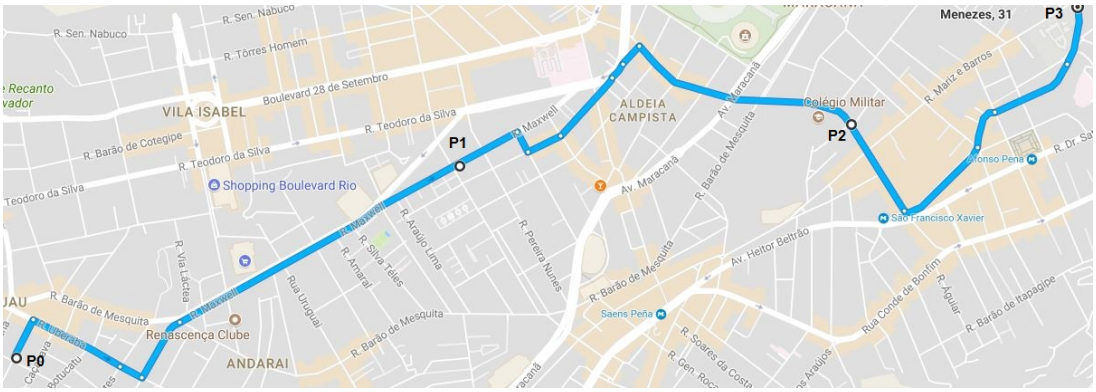

Figura 17: Trajetória T3

A variação T4, temos uma trajetória composta por cinco pontos. Na tabela 12, temos uma visão dos valores de latitude e longitude dos pontos, já na Figura 18 temos a ilustração de toda a trajetória.

Tabela 12: Variação T4

Ponto	Menor (Latitude, Longitude)	Maior (Latitude, Longitude)
P0	(-22.96892, -43.18053)	(-22.96884, -43.18030)
P1	(-22.96623, -43.17576)	(-22.96613, -43.17536)
P2	(-22.96563, -43.17448)	(-22.96558, -43.17416)
P3	(-22.96358, -43.17418)	(-22.96328, -43.17414)
P4	(-22.95411, -43.17712)	(-22.95402, -43.17694)

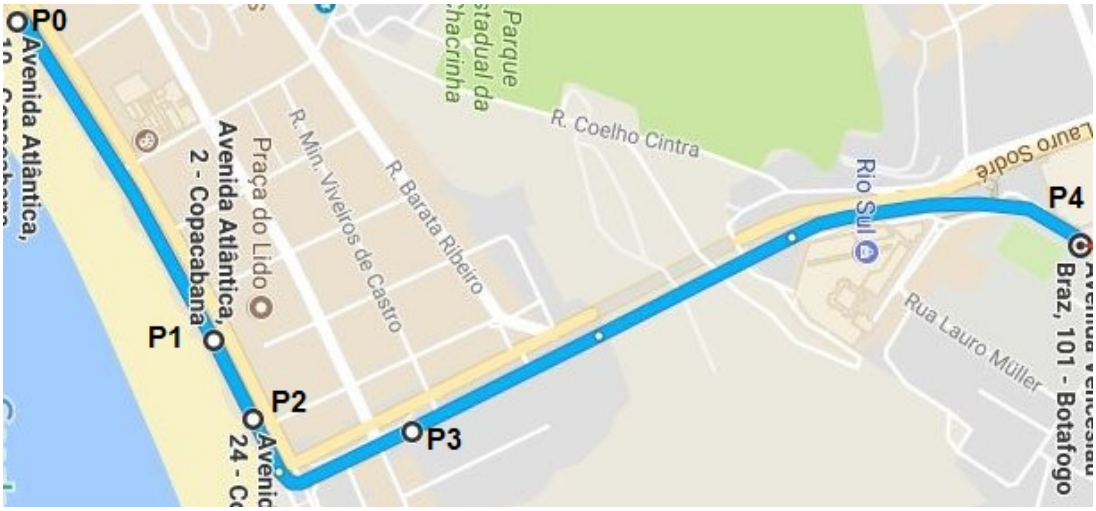

Figura 18: Trajetória T4

Ao executar as consultas, foram analisados os planos de execução de cada gerenciador de banco de dados. Nas figuras 19, 20 e 21, podemos ver o plano de execução da consulta Q2 utilizando os parâmetros da trajetória T1.

```
-----------------------------------------------
 Nested Loop  (cost=0.00..5763279.82 rows=1 width=30)
   Join Filter: ((tt.sequencia > t.sequencia) AND ((t.codigo)::text = (tt.codigo)::text) AND (tt.sequencia <= (t.sequencia + '10'::numeric
)) AND ((t.data)::date = (tt.data)::date))
   ->  Seq Scan on trajetoria tt  (cost=0.00..2881634.76 rows=19 width=23)
         Filter: ((latitude >= '-22.96445'::numeric) AND (latitude <= '-22.96430'::numeric) AND (longitude >= '-43.21475'::numeric) AND (l
ongitude <= '-43.21460'::numeric))
   ->  Materialize  (cost=0.00..2881634.85 rows=18 width=23)
         ->  Seq Scan on trajetoria t  (cost=0.00..2881634.76 rows=18 width=23)
               Filter: ((latitude >= '-22.97899'::numeric) AND (latitude <= '-22.97888'::numeric) AND (longitude >= '-43.22430'::numeric)
AND (longitude <= '-43.22418'::numeric))
```

Figura 19: PostgreSQL - Plano de execução

```
----------------------------------------
 Merge Join  (cost=54238014.88..54238857.99 rows=94 width=30)
   Merge Cond: (((t.codigo)::text = (tt.codigo)::text) AND (((t.data)::date) = ((tt.data)::date)))
   Join Filter: ((tt.sequencia > t.sequencia) AND (tt.sequencia <= (t.sequencia + '10'::numeric)))
   ->  Sort  (cost=27119007.44..27119090.27 rows=33132 width=23)
         Sort Key: t.codigo, ((t.data)::date)
         ->  Seq Scan on trajetoria t  (cost=0.00..27116519.90 rows=33132 width=23)
               Filter: (('0103000000010000000500000006D1CB1169FFA36C0143FC6DCB59C45C0A661F88898FA36C0465F419AB19C45C0C3F5285C8FFA36C0143FC6DCB59C45C0DFA63FFB91FA36C00D1AFA27B89C45C06D1CB1169FFA36C0143FC6DCB59C45C0'::geometry ~ geo) AND _st_contains('0103000000010000000500000006D1CB1169FFA36C0143FC6DCB59C45C0A661F88898FA36C0465F419AB19C45C0C3F5285C8FFA36C0143FC6DCB59C45C0DFA63FFB91FA36C00D1AFA27B89C45C06D1CB1169FFA36C0143FC6DCB59C45C0'::geometry, geo))
   ->  Sort  (cost=27119007.44..27119090.27 rows=33132 width=23)
         Sort Key: tt.codigo, ((tt.data)::date)
         ->  Seq Scan on trajetoria tt  (cost=0.00..27116519.90 rows=33132 width=23)
               Filter: (('0103000000010000000500000090A0F831E6F636C0736891ED7C9B45C0668863SDDCF636C05DDC4603789B45C09EEA909BE1F636C06C43C5387F9B45C013622EA9DAF636C0F8E28BF6789B45C090A0F831E6F636C0736891ED7C9B45C0'::geometry ~ geo) AND _st_contains('0103000000010000000500000090A0F831E6F636C0736891ED7C9B45C0668863SDDCF636C05DDC4603789B45C09EEA909BE1F636C06C43C5387F9B45C013622EA9DAF636C0F8E28BF6789B45C090A0F831E6F636C0736891ED7C9B45C0'::geometry, geo))
```

Figura 20: PostGIS - Plano de execução

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| rel                                                                                                                                                                                           |
+===============================================================================================================================================================================================+
| project (                                                                                                                                                                                     |
| | join (                                                                                                                                                                                      |
| | | project (                                                                                                                                                                                 |
| | | | select (                                                                                                                                                                                |
| | | | | table(sys.trajetoria) [ trajetoria.data as t.data, trajetoria.codigo as t.codigo, trajetoria.linha as t.linha, trajetoria.lati |
: tude as t.latitude, trajetoria.longitude as t.longitude, trajetoria.sequencia as t.sequencia ] COUNT                                                                                         :
| | | | ) [ decimal(18,7)[sys.sql_neg(decimal(7,5) "2297899")] <= t.latitude <= decimal(18,7)[sys.sql_neg(decimal(7,5) "2297888")], deci |
: mal(18,7)[sys.sql_neg(decimal(7,5) "4322430")] <= t.longitude <= decimal(18,7)[sys.sql_neg(decimal(7,5) "4322418")] ]                                                                        :
| | | ) [ t.data, t.codigo, t.linha, t.latitude, t.longitude, t.sequencia, date[t.data] as L15.L15, t.sequencia as L20.L20, sys.sql_add( |
: t.sequencia, decimal(18,7)[tinyint "10"]) as L21.L21 ],                                                                                                                                     :
| | | project (                                                                                                                                                                                 |
| | | | select (                                                                                                                                                                                |
| | | | | table(sys.trajetoria) [ trajetoria.data as tt.data, trajetoria.codigo as tt.codigo, trajetoria.linha as tt.linha, trajetoria.l |
: atitude as tt.latitude, trajetoria.longitude as tt.longitude, trajetoria.sequencia as tt.sequencia ] COUNT                                                                                 :
| | | | ) [ decimal(18,7)[sys.sql_neg(decimal(7,5) "2296445")] <= tt.latitude <= decimal(18,7)[sys.sql_neg(decimal(7,5) "2296430")], dec |
: imal(18,7)[sys.sql_neg(decimal(7,5) "4321475")] <= tt.longitude <= decimal(18,7)[sys.sql_neg(decimal(7,5) "4321460")] ]                                                                     :
| | ) [ tt.data, tt.codigo, tt.linha, tt.latitude, tt.longitude, tt.sequencia, date[tt.data] as L16.L16, tt.sequencia as L17.L17 ]                                                               |
| | ) [ t.codigo = tt.codigo, L15.L15 = L16.L16, L20.L20 < L17.L17 <= L21.L21 ]                                                                                                                 |
| ) [ t.linha, t.codigo, t.sequencia, tt.linha, tt.codigo, tt.sequencia ]                                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Figura 21: MonetDB - Plano de execução

Ao analisar os planos de execução, temos um grande custo na projeção da consulta, ou seja, no intervalo das latitude e longitude e no atributo geo, no caso do PostGIS. E com isso, influenciando no resultado final do tempo de execução da consulta.

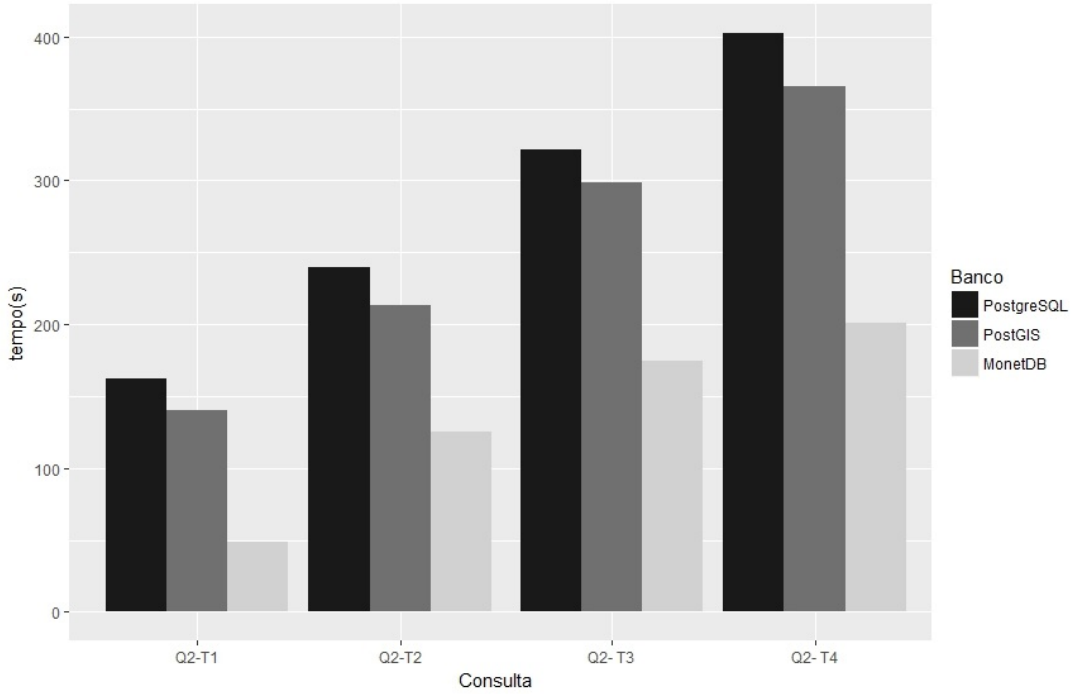

Figura 22: Tempo de execução dos bancos

Ao analisar o gráfico da figura 22, pode concluir que o MonetDB leva grande vantagem em relação aos demais bancos em todas as variações de trajetória, enquanto o PostGIS possui vantagem em relação ao PostgreSQL que aumenta quando o número de pontos na trajetória aumenta.

Capítulo 6

Conclusão

Analisar o desempenho de bancos NoSQL e comparar com banco de dados relacionais, utilizando consultas baseadas em trajetória mostrou ser pouco explorado. Este trabalho mostra que utilizando conceitos de trajetória é possível criar consultas que podem ser utilizadas para comparar o desempenho entre os bancos de dados.

Para ser possível a realização dessa pesquisa, coletamos os dados do *dataset* proveniente do Portal de Dados Abertos da Prefeitura do Rio de Janeiro (data.rio). Uma grande quantidade de dados foi utilizada, de junho de 2014 ao mês de julho de 2014, período referente a Copa do Mundo 2014. Realizamos algumas atividades de pré-processamento, como limpeza dos dados gerados, para melhorar a qualidade dos resultados.

Nesse trabalho, é abordado o conceito de cada modelo de banco de dados, tanto o relacional (orientado a linha) quanto o colunar (orientado a coluna). Na secção III.4 são feitas comparações entre os dois modelos, mostrando as vantagens e desvantagens de cada modelo. Essa comparação nos mostrou uma vantagem do modelo colunar sobre o modelo relacional, justificando uma análise comparativa entre os modelos usando dados de trajetória.

Na avaliação experimental, são apresentadas características sobre cada gerenciador de banco de dados utilizado, mostrando também a forma que os atributos foram criados em cada um deles. Foram usados um total de 99.397.279 tuplas para o teste de desempenho dos bancos de dados, uma quantidade grande o suficiente para os bancos de dados se esforçarem para encontrar o resultado de cada consulta e assim poder mostrar se realmente existe uma diferença de desempenho relevante entre cada modelo de banco de dados.

Foram executadas duas consultas, sendo elas Q1 (P-Query) e Q2 (T-Query), tendo dentro de cada consulta três variações. Na Q1, a diferença entre cada variação são as variáveis das consultas que possuem valores de diferentes em cada uma, ou seja, ponto de observação diferente. Na Q2, a cada variação mudam as quantidades de pontos de observação por trajetória e mudam também as trajetórias.

Após a execução da consulta Q1, os resultados mostraram uma grande vantagem do MonetDB em relação aos outros bancos, onde em alguns casos o tempo de execução foi a metade

em relação aos outros. Na consulta Q2, o MonetDB obteve novamente o melhor desempenho em relação aos demais bancos.

Concluímos que a vantagem do MonetDB em relação aos demais bancos está relacionada pelo fato de ser um banco NoSQL, preparado para receber grande volume de dados e devido ao seu modelo colunar, onde mostrou grande vantagem para realizar a projeção da consulta, retornando os dados em menor tempo de execução.

Referências Bibliográficas

Abadi, D., Madden, S., and Ferreira, M. (2006). Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM.

Abadi, D. J. et al. (2007). Column stores for wide and sparse data. In *CIDR*, pages 292–297.

Ailamaki, A., DeWitt, D. J., Hill, M. D., and Skounakis, M. (2001). Weaving relations for cache performance. In *VLDB*, volume 1, pages 169–180.

Bennett, D. and Harvey, A. (2009). Publishing open government data. *World Wide Web Consortium*.

Boncz, P. A., Kersten, M. L., and Manegold, S. (2008). Breaking the memory wall in monetdb. *Communications of the ACM*, 51(12):77–85.

Brown, A. and Wilson, G. The architecture of open source applications: Elegance, evolution, and a few fearless hacks, volume i. aosabook. org, march 2012. *Online at http://aosabook. org*.

Cao, X., Cong, G., and Jensen, C. S. (2010). Mining significant semantic locations from gps data. *Proceedings of the VLDB Endowment*, 3(1-2):1009–1020.

Douglas, K. and Douglas, S. (2003). *PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases*. SAMS publishing.

Drake, J. D. and Worsley, J. C. (2002). *Practical PostgreSQL*. "O'Reilly Media, Inc.".

Elmasri, R. (2008). *Fundamentals of database systems*. Pearson Education India.

Getoor, L. and Sahami, M. (1999). Using probabilistic relational models for collaborative filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*. Citeseer.

Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59.

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE.

Hecht, R. and Jablonski, S. (2011). Nosql evaluation: A use case oriented survey. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341.

Idreos, S. (2010). Database cracking: Towards auto-tuning database kernels. *CWI and University of Amsterdam*.

Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, S., Kersten, M., et al. (2012). Monetdb: Two decades of research in column-oriented database architectures. *A Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering*, 35(1):40–45.

Khoshafian, S., Copeland, G., Jagodits, T., Boral, H., and Valduriez, P. (1987). A query processing strategy for the decomposed storage model. In *Data Engineering, 1987 IEEE Third International Conference on*, pages 636–643. IEEE.

Kumar, P., Singh, V., and Reddy, D. (2005). Advanced traveler information system for hyderabad city. *Intelligent Transportation Systems, IEEE Transactions on*, 6(1):26–37.

Lóscio, B. F., OLIVEIRA, H. R. d., and PONTES, J. C. d. S. (2011). Nosql no desenvolvimento de aplicações web colaborativas. *VIII Simpósio Brasileiro de Sistemas Colaborativos, Brasil*.

Momjian, B. (2001). *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York.

MonetDB (2016). Portal do MonetDb. https://www.monetdb.org/.

Nayak, A., Poriya, A., and Poojary, D. (2013). Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19.

Obe, R. O. and Hsu, L. S. (2015). *PostGIS in action*. Manning Publications Co.

Postgres (2016). Portal do PostgreSql. https://www.postgresql.org/.

Pritchett, D. (2008). Base: An acid alternative. *Queue*, 6(3):48–55.

Ramsey, P. et al. (2005). Postgis manual. *Refractions Research Inc*.

Rio, P. (2016). Portal de dados abertos da prefeitura do Rio de Janeiro. http://data.rio/.

Roussopoulos, N., Kelley, S., and Vincent, F. (1995). Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79. ACM.

Silberschatz, A., Korth, H. F., Sudarshan, S., et al. (1997). *Database system concepts*, volume 4. McGraw-Hill New York.

Spaccapietra, S., Parent, C., Damiani, M. L., de Macedo, J. A., Porto, F., and Vangenot, C. (2008). A conceptual view on trajectories. *Data & knowledge engineering*, 65(1):126–146.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., et al. (2005). C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment.

Ubaldi, B. (2013). Open government data.

Wu, X., Zhu, X., Wu, G. Q., and Ding, W. (2014). Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107.

Zheng, Y. and Zhou, X. (2011). *Computing with spatial trajectories*. Springer Science & Business Media.