MARCOS VINICIUS PONTAROLO

PARALELIZAÇÃO DE CONSULTAS SOBRE DADOS DE MOBILIDADE

(*versão pré-defesa, compilada em 20 de janeiro de 2024*)

Tese apresentada como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação no Programa de Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carmem Satie Hara.

CURITIBA PR

2024

RESUMO

No planejamento de cidades inteligentes, melhorar a mobilidade urbana é um dos objetivos principais a se alcançar. Diversos aspectos são levados em consideração para alcançar este objetivo, como determinar a quantidade de faixas, a direção da via, os cruzamentos, a sincronização dos sinaleiros, além de outros fatores que podem ser determinantes para o sucesso de um bom planejamento. Uma das maneiras de coletar os dados necessários que irão viabilizar os estudos para determinar as mudanças necessárias é o monitoramento dos eventos de trânsito que ocorrem nessas regiões, tais como os engarrafamentos, irregularidades e alertas. Para que essas fontes de informações se tornem *insights* importantes para as tomadas de decisões, é necessário estabelecer relações espaço-temporais entre os diferentes tipos de eventos, além de incluir outros tipos de objetos geométricos fixos, como escolas, parques, bairros e ruas para enriquecer as consultas e obter informações relevantes. Um ponto importante acerca de consultas espaço-temporais é que elas normalmente possuem um alto custo computacional para serem realizadas, sendo importante a otimização da forma como as consultas são processadas para que consigamos obter um desempenho satisfatório. Com o intuito de otimizar as consultas espaço-temporais relacionadas a eventos de trânsito, nós criamos um método de indexação para dados de mobilidade (MIDM), uma estrutura de índice e um modelo de armazenamento para os eventos que podem ser representados como pontos ou um conjunto de pontos. O MIDM é baseado na técnica de tesselação, que divide a área de interesse em um grid de tamanho fixo, particionando os registros por células, denominada também como célula geográfica(CG). Cada CG é associado a um conjunto de blocos que contém os registros de eventos. Nesta forma linear de consulta, há também um índice *bitmap* para filtrar as células geográficas que contém os registros necessários para a consulta realizada. Foram realizados experimentos comparando o MIDM com uma estrutura criada no *PostgreSQL*, que utiliza a extensão *PostGIS* e adota um índice *R-tree* para a indexação dos dados. Criando uma configuração apropriada do tamanho dos CGs, o MIDM obteve uma redução do tempo de processamento das consultas, sendo assim mais eficiente que a solução no *PostgreSQL*. O MIDM também foi idealizado para que fosse facilmente adaptado para receber dados em fluxo e ser extensível para o processamento paralelo. Em busca de elevar ainda mais os resultados do processamento do método, foi pensando em uma estrutura para prosseguir com a paralelização do MIDM, pensando também na facilidade da sua implementação. Para isso, a primeira abordagem foi pensada utilizando a solução com o ecossistema do *Hadoop* juntamente com o processamento do *Spark*, migrando assim o sistema de arquivos do MIDM e modificando um pouco da sua estrutura, como a eliminação do índice bitmap para filtrar os CGs. A implementação paralela não se mostrou efetiva após a realização dos experimentos em comparação com o MIDM sequencial e precisa ser melhor investigada, realizando mais experimentos.

Palavras-chave: Geoprocessamento. Mobilidade. Trânsito.

ABSTRACT

In smart city planning, improving urban mobility is one of the main objectives to achieve. Several aspects are taken into consideration to reach this goal. Determining the number of lanes, the road direction, intersections, traffic signal synchronization, among other factors, can be crucial for the success of a good plan. One efficient way to gather the necessary variables that will enable studies to determine the necessary changes is by monitoring traffic alerts occurring in these regions, such as jams, irregularities, and other types of isolated alerts. To transform these sources of information into important insights for decision-making, it is necessary to establish spatio-temporal relationships among different types of events, as well as include other fixed geometric objects, such as schools, parks, neighborhoods, and streets, to enrich queries and obtain relevant information. An important aspect of spatio-temporal queries is that they often have a high computational cost to be performed, making it important to optimize how queries are processed to achieve satisfactory performance while reducing computational cost. With the aim of optimizing spatio-temporal queries related to traffic events, we have developed a method for indexing traffic events(MIDET), an index structure, and a storage model for events that can be represented as points or a set of points. The MIDET is based on the tessellation technique, which divides the area of interest into a grid of fixed size by partitioning records into cells, also known as geographic cells (GCs). Each GC is associated with a set of blocks containing the actual records. In this linear query form, there is also a bitmap index to filter the geographic cells containing the records needed for the query performed. Experiments were conducted comparing the MIDET with a structure created in PostgreSQL, which uses the PostGIS extension and adopts an R-tree index for data indexing. By creating an appropriate configuration of GC sizes, the MIDET achieved a reduction in query processing time, thus being more efficient than the solution in PostgreSQL. The MIDET was also designed to be easily adaptable to receive streaming data and be extensible for parallel processing. In pursuit of further enhancing the method's processing results, a structure for proceeding with the parallelization of MIDET was designed, also considering the ease of its implementation. For this purpose, the initial approach was designed using the Hadoop ecosystem together with Spark processing, thus migrating the MIDET file system and making some modifications to its structure, such as eliminating the bitmap index to filter the GCs. The adopted parallel solution did not prove effective after conducting experiments compared to the sequential MIDM and needs further investigation through additional experiments.

Keywords: Geoprocessing. Mobility. Traffic.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os eventos de trânsito tem se tornado cada vez mais comum em nosso cotidiano desde a popularização dos automóveis e o aumento dos centros urbanos. Engarrafamentos, irregularidades, alertas, ruas fechadas ou em manutenção e outros diversos alertas são exemplos de eventos de trânsito que pertencem a esse ambiente e são reportados em tempo real por aplicativos como o *Waze* ou portais como o Portal do Departamento de Transporte de *Maryland* (Maryland, 2022).

Esses diversos dados espaço-temporais são uma importante fonte de informação para realizar estudos sobre o planejando urbano e podem servir para detectar padrões entre os eventos de forma que o produto dos estudos sobre eventos subsidiem ações que buscam melhorar a mobilidade como um todo, como a sincronização de sinaleiros, o planejamento das vias, manutenções preventivas. Os diversos estudos e análise sobre os dados normalmente envolvem estabelecer relações espaciais entre os dados, além de incluir dados geográficos relevantes para o estudo, como os dados de ruas, bairros, praças e outros objetos fixos. Ou seja, um estudo requer a combinação de mais de um elemento espacial que pode envolver uma relação temporal entre os dados. Um exemplo de consulta simples que podemos estar interessados seria determinar os locais de ocorrência de engarrafamentos e alertas no mesmo período de tempo e região, ou até verificar qual rua tem a maior incidência de engarrafamentos dentro de um determinado bairro em determinada época do ano, ou até determinar a sazonalidade das ocorrências, como no final de ano ou feriados prolongados que normalmente aumentam o volume de carros nas rodovias. Realizar essa junção espaço-temporal estabelecendo relações entre os dados de forma a gerar estudos sólidos geralmente tem um alto custo computacional de processamento associado.

Há diversos trabalhos anteriores focados na questão de aprimorar o desempenho de consultas com relacionamentos espaciais, tal como é referenciado nos trabalhos recentes de (Chaudhry et al., 2020) e (Mahmood et al., 2019). Um dos métodos clássicos e consequentemente mais utilizado para melhorar o desempenho de consultas é a indexação. A indexação espacial, envolve duas etapas: a etapa de filtro e a etapa de refinamento. A etapa de filtro reduz o conjunto de busca para os objetos espaciais que realmente são candidatos aos resultados da consulta, normalmente esse filtro é realizado sobre a aproximação da geometria do objeto, como o retângulo envolvente mínimo ou *minimum bounding rectangle* (MBR). Na etapa de refinamento, o conjunto de busca com os objetos espaciais restantes é avaliado para verificar se eles atendem aos requisitos da consulta espacial. Uma indexação espacial pode ser representada por dois tipos: indexação orientada ao espaço ou orientado por dados. A Árvore R (Guttman, 1984), é um exemplo de índice de indexação que cria um particionamento orientado por dados, no qual o atributo espacial dos objetos determina em como eles serão armazenados e clusterizados. Já na indexação orientada a espaço, há um particionamento da região de interesse utilizando a técnica

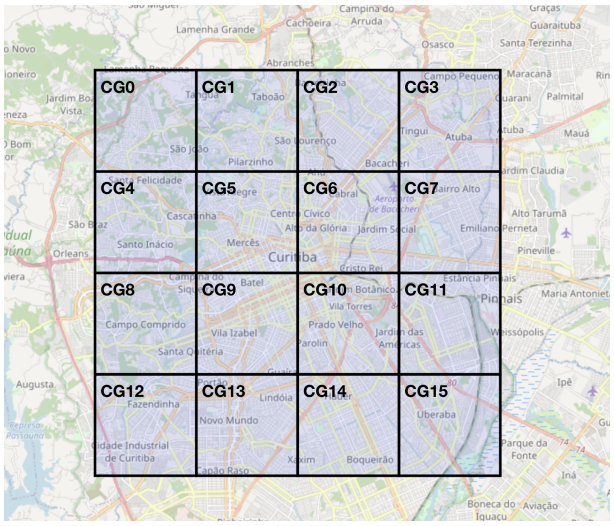


Figura 1.1: Exemplo de tesselação com Células Geográficas

de tesselação, na qual a área é dividida em células geográficas (CGs) de tamanho fixo. A figura 1.1 ilustra um exemplo de tesselação. Nessa técnica, cada CG possui um conjunto de dados associados a ele, composto por objetos, cuja geometria sobrepõe a do CG. Há diversos trabalhos sobre estrutura de dados e sistemas que utilizaram o particionamento espacial como índice, tal como o *Grid File* (Nievergelt et al., 1984), *Filter trees* (Sevcik e Koudas, 1996), PBSM (Patel e DeWitt, 1996), *Paradise* (Patel et al., 1997) e o SATO (Vo et al., 2014).

Focando em analisar os dados de trânsito para coletar informações relevantes no desenvolvimento de cidades inteligentes, o nosso interesse é realizar consultas históricas analíticas com dados espaço-temporais envolvendo os eventos de trânsito. Em sua grande maioria, os dados de trânsito podem ser representados como pontos espaço-temporais, tal como um pequeno acidente em uma via pública, ou coleções de pontos, como no caso de um congestionamento que pode alcançar quilômetros de distância. O aplicativo *Waze* representa todos os dados espaciais apenas com os dois tipos exemplificados. Como foco do trabalho, estamos interessados somente em dados espaciais sobre eventos de trânsito que podem ser representados como pontos ou uma coleção de pontos.

1.1 MIDM

Nos trabalhos realizados, foi proposto um **M**étodo de **I**ndexação para **D**ados de **M**obilidade (MIDM), que é baseado na indexação orientada ao espaço, utilizando a tesselação, com uma grade fixa para o particionamento.

Quando utilizamos a técnica de indexação orientado ao espaço, há alguns problemas típicos que podem aparecer, como: objetos que estão na borda da célula ou até que cruzam várias células, causando uma distribuição não uniforme desses dados, como no caso de uma rodovia congestionada. Há também o problema de má distribuição dos dados ao longos dos CGs, como em uma região central na qual há uma maior concetração de eventos de trânsito,

principalmente em horários de pico. O MIDM trata o primeiro caso da seguinte maneira: os conjuntos de pontos que atravessam mais de um CG são separados em subconjuntos distribuídos para cada CG, enquanto os dados do evento como o identificador e o *timestamp* são mantidos. Cada CG é associado a um conjunto de blocos de tamanho fixo (que correspondem as paginas de disco) contendo os registros. Um arquivo é criado para cada intervalo de tempo (configurado pelo usuário) e associado a um índice *bitmap*. O vetor de *bitmap* contém uma entrada para cada CG com o valor configurado inicialmente em 0 e sendo modificado para 1 caso exista pelo menos um evento daquele tipo no CG e no intervalo de tempo definido. O índice *bitmap* é utilizado pelo MIDM na etapa de filtro, utilizando operações sobre os *bitmaps* para recuperar apenas os arquivos necessários do disco. Os índices *bitmap* são eficazes neste contexto, pois foram consideradas apenas operações de inserção de dados. Os índices *bitmap* possuem uma limitação bem conhecida na literatura, não sendo eficaz para a utilização em conjunto de dados com operações de atualização frequentes.

1.2 MIDM PARALELO

O MIDM foi pensado para ser facilmente estendido para processar consultas com dados chegando em tempo real(*stream*) e também na implementação do processamento paralelo das consultas históricas. Neste trabalho, o foco foi implementar o processamento paralelo como uma forma de melhoria do método, utilizando formas simples de implementação para medir a eficácia do paralelismo.

O processador do método foi programado em C. Porém, para que seja realizada a implementação de uma paralelização no código C, é necessário recorrer a bibliotecas como o OpenMP e OpenMPI, que exigem uma maior maturidade de programação paralela e considerando todos os cenários possíveis que podem ser enfrentados no MIDM, porém, a execução do processador juntamente com bibliotecas nativas da linguagem pode ter um desempenho superior a qualquer outro *framework* utilizado na abordagem. Considerando uma primeira versão da paralelização, com o intuito de medir a eficácia do processamento, foi optado por uma solução de simples implementação e completa para que pudéssemos realizar os experimentos necessários para medir a eficiencia do processamento paralelo.

A solução encontrada para avançar com a paralelização foi o *Spark*, juntamente com a API *PySpark*, que integra um ambiente nativo *Java* com *Kernel* Python. O *Spark* é um *framework Open-source* e amplamente utilizado para processamento de dados em ambientes analíticos, principalmente em processos que envolvem ETL (*extract*, *transform* e *load*). Possui uma vasta biblioteca de funções para trabalhar com as transformações necessárias dos dados, além de funções específicas para a análise de dados.

1.3 ORGANIZAÇÃO DA MONOGRAFIA

No capítulo 2 mostraremos os trabalhos relacionados ao tema e como eles influenciaram na estrutura do MIDM. No capítulo 3 serão mostrados detalhes do **M**étodo de **I**ndexação para **D**ados de **M**obilidade (MIDM), fruto de um trabalho anterior, contando com os experimentos realizados e a evolução do método ao longo dos estudos, além de introduzir os conceitos inicias do método e os problemas que ele resolve nas consultas espaço-temporais de eventos de trânsito. Seguindo para o capítulo 4, apresentamos a versão paralela do MIDM, contando com os detalhes de como ela foi implementada, além das dificuldades envolvidas no processo de paralelização, juntamente com os experimentos realizados em comparação com a versão sequencial do método. Mostramos também a arquitetura proposta para a paralelização, explicando as tecnologias utilizadas como o HDFS e Spark, as estruturas de dados criadas e as bibliotecas necessárias para o bom desempenho das consultas. O capítulo 5, conclui o trabalho, resumindo as contribuições realizadas com o MIDM, bem como as publicações e os prêmios que foram conquistados.

2 TRABALHOS RELACIONADOS

De acordo com (Aji et al., 2015), há seis métodos mais utilizados na literatura para o particionamento espacial: grade fixa, divisão binária, *strip*, *boundary optimized strip*, curva de Hilbert e *sort-tile-recursive*. Uma análise é realizada sobre a qualidade da partições geradas nos dois conjunto de dados, juntamente com o desempenho de junção em um *cluster* utilizando o *MapReduce*, e o custo do particionamento dos dados. Por conta da facilidade do particionamento utilizando a grade fixa, o custo para a geração de novas partições é muito menor que as outras formas de particionamento. Por conta disso, a técnica de grade fixa pode ser uma boa estratégia para o armazenamento de dados espaço-temporais em fluxo, o que motivou a adoção da grade fixa para o MIDM. Muitas das técnicas utilizadas para o particionamento espacial não foram propostas para uma estratégia de armazenamento de dados, mas apenas para a etapa de filtro das junções espaciais. O particionamento com grade fixa também foi utilizado para a etapa de filtro de uma junção utilizando dados em disco por PBSM (Patel e DeWitt, 1996), e também foi explorado para a execução em memória primária na etapa de refinamento para cada CG (Tsitsigkos et al., 2019), além de uma técnica para a execução paralela (Vo et al., 2014; Eldawy e Mokbel, 2015; Yu et al., 2015). O MIDM é similar ao PBSM com as etapas para a execução das junções espaciais, porém não adota o mesmo modelo de armazenamento.

A estrutura de arquivos do PBSM é semelhante ao *grid file* (Nievergelt et al., 1984), um dos primeiros estudos dedicados ao armazenamento de dados utilizando o particionamento em grade. O *grid file* é um dos diversos métodos de acesso a dados representados como ponto detalhados em uma pesquisa sobre métodos de acesso multidimensionais (Gaede e Günther, 1998). Esse método propõe uma relação *many-to-one* (muitos para um) entre as células da grade e as páginas de disco. O espaço é dividido em células e, quando a capacidade da página é excedida, ocorre uma divisão de página. Adicionalmente, a cada divisão, a grade inteira dobra o número de células. Entretanto, as células que não causaram a divisão mantêm suas referências à mesma página de disco. Já o MIDM é similar ao *grid file*, mas estabelece uma relação de *one-to-many* (um para muitos) entre as células da grade e as páginas de disco. O MIDM mantém uma partição fixa da grade e, ao exceder a capacidade da página, aloca-se uma página adicional (denominada bloco) para a mesma partição. Alguns estudos, ao otimizarem os arquivos de grade, propuseram estruturas baseadas em árvores para associar células da grade às páginas de disco. Enquanto o *GE-tree* (Shin et al., 2019) adota uma *quad-tree*, o MSI (Al-Badarneh et al., 2013) opta por uma *R-tree*. Em contraste a esses trabalhos, o MIDM utiliza um vetor de *bitmap* para representar a presença de dados em cada célula, visando lidar com a distribuição desigual dos dados, e armazena a relação entre células, arquivos e páginas em um banco de dados relacional (*PostgreSQL*).

Semelhante ao MIDM, há estudos que utilizam uma indexação baseada em *bitmaps*, associando a uma estrutura de árvore (Neto et al., 2013; Siqueira et al., 2012) ou vinculada à particionamento orientado a espaço, como o BPSJ (Shohdy et al., 2015). O BPSJ propõe um algoritmo paralelo de junção espacial que explora o espaço para encontrar pontos adequados para dividir o espaço, buscando gerar partições balanceadas, seguido de uma junção baseada no *bitmap* em memória. No entanto, os vetores de *bitmap* têm o tamanho do conjunto de dados. Já os vetores de *bitmap* do MIDM são significativamente menores, correspondendo ao tamanho da grade, além da utilização dos *bitmaps* comprimidos. Outro estudo, (Antoine et al., 2011), adota um índice de *bitmap*. No entanto, diferentemente do MIDM, considera uma partição hierárquica do espaço para evitar que objetos ultrapassem as fronteiras das células, associando os objetos à partição de menor nível que os contenha completamente. Cada partição e nível é associado a um arquivo e índice de *bitmap*. Por outro lado, o MIDM associa um arquivo a cada intervalo de tempo e considera uma estrutura de grade fixa de único nível.

3 O MIDM

Este capítulo introduz o MIDM, um **M**étodo de **I**ndexação para **D**ados de **M**obilidade. Este método assume a existência de diversos tipos de eventos de trânsito, como os engarrafamentos, alertas e irregularidades. O MIDM é baseado em dois princípios básicos: a definição de intervalos de tempo de tamanhos iguais e, para cada intervalo, a aplicação de um particionamento orientado a espaço, que no caso é a técnica de tesselação, dividindo a área de interesse em células geográficas. A motivação para a escolha dos métodos aplicados é otimizar consultas espaço-temporais que contém junções entre os objetos. Ou seja, consultas que combinam mais de um *dataset* baseadas em seus atributos espaciais e temporais.

Para a configuração do MIDM, se faz necessário estabelecer o tamanho dos intervalos de tempo que serão utilizados para a indexação temporal, além de estabelecer o tamanho do *grid* que será formado na área de interesse. Esses dados são armazenados em uma tabela de metadados que será utilizada posteriormente na execução das consultas.

Cada célula geográfica pode ser associada a diversos blocos que contém os registros dos eventos de trânsito. Os eventos que ocorrem na mesma CG são armazenados em conjunto. Os registros contém um atributo espacial, que pode ser denotado por um ponto ou um conjunto de pontos. Para os eventos de trânsito que são definidos por um conjunto de pontos (como os engarrafamentos e irregularidades), quando ocorre um caso do evento atravessar mais de uma CG, o MIDM automaticamente divide a coleção de pontos em subconjuntos, distribuindo os conjuntos para as CGs pertencentes, juntamente com a replicação dos outros campos do evento. Mais especificamente, ao trabalhar com um objeto que contém 4 pontos, o MIDM verifica se os 4 pontos estão localizados em mais de um CG. Em caso afirmativo, então é criado um subconjunto dos 4 pontos determinando quantos pontos estão localizados em cada CG, podendo ter uma distribuição de 2 pontos para o *CG1* e outros 2 pontos para o *CG2*, dando origem a dois registros, um registro será armazenado no bloco do *CG1* e o outro registro será armazenado no bloco pertencente ao *CG2*. Ambos os registros terão os mesmos dados, exceto o atributo espacial, que será recriado de acordo com a geometria formada pelos pontos de cada subconjunto.

O MIDM adota um modelo misto de armazenamento baseado em arquivos e um banco de dados relacional que contém as informações necessárias para otimizar a velocidade de acesso aos arquivos, além de armazenar os metadados necessários para as consultas.

3.1 ORGANIZAÇÃO DO MIDM

No MIDM, um arquivo é criado para cada intervalo de tempo associado a um tipo de evento. Cada arquivo é composto por um conjunto de blocos de tamanho fixo, que contém diversos registros. Os blocos são utilizados como unidade de transferência entre o disco e a memória primária e possuem eventos de trânsito que ocorreram na mesma CG. Quando o número

Figura 3.1: Fluxo dos dados no MIDM

387 387 2 1506543240 1506557436
18 1506543240 R. Anita Garibaldi 0101000020E61000006EE00ED4296D48C0D6E6FF5547523AC0
8 1506557436 R. Aubé 0101000020E61000001538D906EE6A48C011514CDE004F3AC0

388 388 3 1506516407 1506550851
11 1506516407 R. Xanxerê 0101000020E61000006B6116DA396B48C04776A565A4463AC0
17 1506549772 R. Marcos Welmuth 0101000020E6100000B64DF1B8A86C48C06EA12B11A8463AC0
16 1506550851 R. Da. Francisca 0101000020E6100000A92EE065866B48C03B35971B0C493AC0

...

Figura 3.2: Estrutura do arquivo de dados

de registros excede o tamanho do bloco definido, um novo bloco é criado. Ao final do intervalo de tempo definido pelo usuário, o espaço livre restante de cada bloco não será preenchido. Em resumo, cada intervalo de tempo e tipo de evento possuem um arquivo correspondente, e nesse arquivo contém um conjunto de blocos para cada CG. A Figura 3.1 mostra um exemplo do fluxo que o dado percorre até ser armazenado dentro dos arquivos do MIDM, sendo carregado diretamente de uma base de dados brutos, passando pelo processo de tesselação e divisão em células geográficas, sendo armazenado nos blocos correspondentes. Por fim, demonstra a organização de cada CG com seus blocos, enfatizando que cada bloco possui eventos que ocorreram apenas naquela CG.

Os registros de eventos de trânsito possuem tamanho variado, dessa forma, cada bloco possui um cabeçalho (*header*) com informações importantes para que o processador consiga acessar os seus registros. A Figura 3.2 ilustra a organização e a estrutura do *header*. Ele é composto por: identificador do bloco, indetificador da CG correspondente, quantidade de registros do bloco, o menor *timestampo* dos registros e o maior *timestamp* encontrado dentro do bloco.

3.2 INDEXAÇÃO

Para otimizar a forma de resgatar e carregar em memória primária os blocos que contém os eventos da mesma CG, foi proposto uma estrutura inspirada em um índice invertido, mapeando

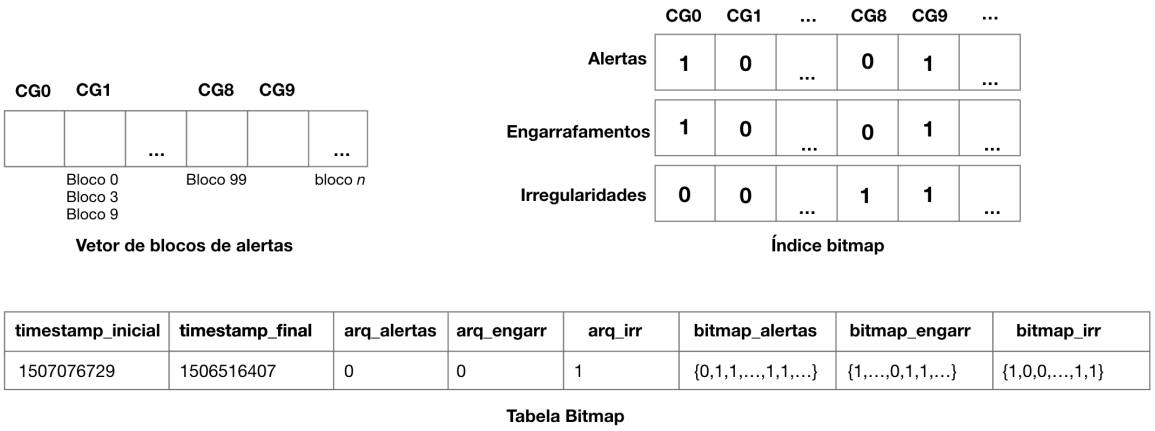


Figura 3.3: Indexação utilizando *Bitmaps*

as CGs para os blocos, como ilustrado no vetor de blocos de alertas da figura 3.3. Essa estrutura auxilia na execução de consultas que buscam eventos que ocorreram em uma dada área de interesse. Porém, ainda não é considerada a dimensão temporal. O MIDM Propõe um índice *Bitmap* para otimizar consultas espaciais. Cada intervalo de tempo e cada tipo de evento possui um vetor de *bitmap* associado a ele, que é indexado pela CG. Os índices do vetor de *bitmap* variam de 0 (CG 0) até o número máximo de CG (por exemplo, CG 999), sendo inicializados com o valor 0 e recebendo o valor 1 se naquele arquivo e intervalo de tempo há registros do evento que ocorreu naquela CG. A Figura 3.3 mostra 3 índices, cada um sendo associado a um arquivo. Observando o arquivo de eventos de engarrafamentos, pode-se visualizar que o arquivo contém registros que ocorreram na CG 0 e 9, porém não há registros que ocorreram nas CGs 1 e 8.

Ao armazenar esses *bitmaps*, é possível realizar operações padrões como *and* e *or* entre diversos tipos de eventos e computar rapidamente os resultados, criando assim um filtro para resgatar somente as CGs que contém os candidatos da consulta para a etapa de refinamento.

Todos os índices *Bitmap* associados aos arquivos são armazenados em um banco de dados relacional, foi utilizado o SGBD *PostgreSQL* em uma tabela denominada de *Bitmap*. Cada linha desta tabela corresponde a um intervalo de tempo. Os atributos desta tabela são: os *Timestamps* de inicio e final do período de tempo, a quantidade de arquivos e os índices *Bitmaps* para cada tipo de evento associado ao intervalo de tempo da linha.

Uma tabela de vetor de blocos também é criada para cada tipo de evento. Essa tabela associa cada CG com os blocos que compõem os arquivos, e corresponde ao índice invertido, também ilustrada na Figura 3.3. A tabela contém 3 atributos: um contador de arquivo, o identificador da CG e o endereço do bloco dentro do arquivo.

3.3 CARREGAMENTO DOS DADOS

O MIDM possui um processo de geração das bases de dados necessárias para a execução das consultas. Ao inicializar, os seguintes parâmetros serão requisitados: a área de interesse, o

tamanho máximo do bloco, o tamanho do *grid*, ou seja, o tamanho da área de tesselação, que é dada pelo número de linhas e colunas, automaticamente o algoritmo realiza o cálculo do tamanho de cada célula geográfica, e por último é necessário especificar o tamanho do intervalo de tempo, definido por um *timestamp*.

Para cada novo tipo de evento, é necessário alterar a estrutura de dados utilizada. A Figura 3.3 demonstra que há um atributo para cada tipo de evento, armazenando os dados necessários para as operações de busca do bloco e as operações de *bitmap*. Dado um tipo de evento já estruturado nas bases de dados, no início de cada intervalo de tempo, uma página com o tamanho de bloco definido anteriormente é alocado na memória primária para cada CG. Um vetor de *bits* indexando cada CG é criado na estrutura, com todos as entradas configuradas em 0 inicialmente. Um novo arquivo também é criado. O método assume que os eventos são reportados em ordem cronológica. Ao registrar um novo evento, o MIDM determina em qual CG o objeto está localizado, e se necessário, realiza a divisão dos subconjuntos de pontos pelo atributo espacial, assim como detalhado anteriormente. A cada nova inserção, é verificado se o bloco já atingiu o tamanho máximo definido. Em caso afirmativo, o bloco é transferido para o arquivo em disco e uma nova linha é criada na tabela de vetor de bloco do evento em questão, referenciando o arquivo utilizado, a CG e o endereço do bloco. Após a escrita em disco, uma nova página é alocada na memória para armazenar novos eventos. Caso haja espaço remanescente em uma página da memória, um novo registro é adicionado a ela normalmente. Ao registrar um novo evento na célula geográfica, o vetor de *bits* é configurado como 1 na CG correspondente.

Após a leitura de todo o intervalo de tempo, todas as páginas são escritas em disco, o arquivo atual é fechado e uma nova linha na tabela de *bitmap* é criada, com a informação dos arquivos e os *bitmaps* criados durante o carregamento.

3.4 PROCESSAMENTO DAS CONSULTAS

Todo o fluxo do processamento das consultas do MIDM é ilustrado na Figura 3.4. A consulta se inicia ao fornecer a opção pretendida, como uma junção entre alertas e engarrafamentos que ocorreram em uma rua específica, ou até uma geometria para realizar a junção espacial com os dados das tabelas. Após inicializar a consulta, os arquivos e os vetores de *bitmaps* são recuperados da tabela de *bitmaps*, considerando o intervalo de tempo pesquisado. Na próxima etapa, são realizadas as operações *bit* a *bit*, selecionando as CGs com valor 1 em ambos os vetores envolvidos na operação lógica. Com essa operação, evitamos uma busca completa nos arquivos de eventos, retornando apenas os endereços dos blocos dentro dos arquivos que contém os registros das CGs específicas que foram selecionadas nas consultas. Após identificar os arquivos e os endereços que serão utilizados, os blocos são recuperados dos arquivos físicos e lidos diretamente para a memória principal, utilizando as informações dos *headers* para acessar os registros de cada bloco. Por fim, são realizadas operações de junções com os registros na memória principal utilizando os parâmetros da consulta requisitada.

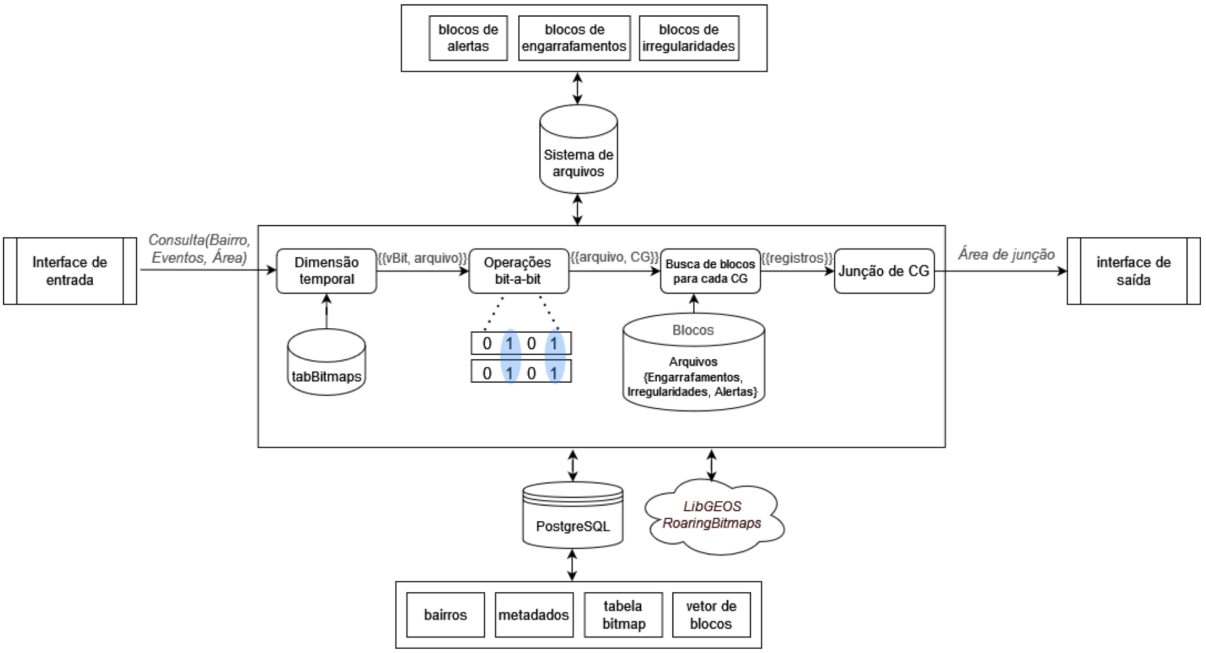


Figura 3.4: Arquitetura de processamento do MIDM

No MIDM, é assumido que todas as operações de junções e os filtros possam ser executados na memória principal, por isso é importante escolher um tamanho adequado para cada bloco na criação das bases de dados. É importante que o número de blocos em cada CG não ultrapasse a capacidade da memória. Caso contrário, podem ser necessários novos acessos a disco para realizar um *swapping* de dados, prejudicando o desempenho da consulta.

3.5 DETALHAMENTO DA IMPLEMENTAÇÃO

Para que o método tenha desempenho satisfatório, a implementação do módulo principal do MIDM foi desenvolvida na linguagem C, compilada utilizando o *GCC* 11.0.0. Para o suporte com os dados brutos recebidos dos sistemas externos, além do armazenamento dos metadados necessários, foi utilizado um sistema de banco de dados relacional, mais especificamente o *PostgreSQL*, que possui suporte a extensões que realizam operações com objetos espaciais, como o *PostGIS* utilizado ao longo do trabalho. Para a conexão entre a versão compilada em C e o banco de dados *PostgreSQL*, foi utilizada a biblioteca C *libpq*. Para a criação, armazenamento e o processamento dos *Bitmaps*, foi utilizado o *Roaring Bitmaps* (Lemire et al., 2018). O *Roaring* foi escolhido com base no artigo (Wang et al., 2017), que realizou diversos experimentos e estudos de caso, concluindo que o *Roaring* é uma das melhores alternativas *Open-source* em comparação com outros métodos de compressão de *bitmaps*. O *Roaring* possui uma biblioteca em C para a utilização, bem como uma extensão do *PostgreSQL* denominada *pg_roaringbitmap*.

3.6 EXPERIMENTOS REALIZADOS

Para realizar os experimentos do MIDM, foi utilizado um conjunto de dados coletados pelo aplicativo de trânsito *Waze*, que foi obtido através de uma parceria com o projeto "Cidade Inteligente de Joinville". A base do *Waze* contém 13 *Gigabytes (GB)* de dados, sendo referente ao período de setembro de 2017 a setembro de 2018, contendo três tipos de eventos: alertas, irregularidades e engarrafamentos. Algumas transformações e modificações foram realizadas dos dados brutos provenientes da base do aplicativo para que se alguns atributos fossem melhor explorados, eliminando também atributos pouco relevantes para os estudos. Para a comparação, uma base relacional com os mesmos atributos utilizados pelos registros do MIDM foi criada no banco de dados relacional, sendo indexada e clusterizada utilizando a *R-tree* como índice sobre o atributo geométrico (*Geom*). Nos experimentos, a base relacional é referenciada como *Waze*.

Com o intuito de rodar experimentos sobre o tamanho da base de dados, foram criados 5 subconjuntos sobre o conjunto original de dados, correspondendo a 20% (B20), 30% (B30), 40% (B40), 50% (B50) e 100% (B100) - Original. Para a criação das bases, a região central de Joinville - SC foi escolhida como a área de interesse inicial para a base de 20%, levando em consideração a maior disponibilidade de eventos de trânsito na região central. As outras bases que se seguem são extensões das bases menores, sendo a B30 a base que contém os eventos da B20 com um adicional de 10% de eventos, expandindo ao longo da borda dos eventos da B20.

Base	MIDM	Relacional Waze
B20	20s425ms	43s149ms
B30	32s734ms	57s958ms
B40	43s962ms	1min21s142ms
B50	56s506ms	1min51s934ms
B100	1m50s055ms	6m15s998ms

Tabela 3.1: Comparação no tempo de processamento MIDM e Waze

Uma das consultas utilizadas para analisar o tempo de processamento e o desempenho do método foi a seguinte: "Quais ruas tiveram engarrafamentos e alertas no mesmo ponto após o primeiro dia de setembro de 2017?". A comparação entre o tempo de processamento desta consulta entre o MIDM e o Relacional Waze é demonstrada na tabela 3.1. É notável que o MIDM tem um desempenho superior ao Relacional Waze. Porém, há uma redução conforme o conjunto de dados aumenta. Para B20, o MIDM reduziu o tempo de consulta em 42%, enquanto para o B100 (conjunto de dados original), essa redução foi de 19%. Após estudar os experimentos, foi notado que para algumas CGs, há uma quantidade massiva de blocos, não sendo possível alocar todos na memória RAM, levando o MIDM a perder desempenho no processamento das consultas. Os resultados obtidos são consistentes com os obtidos anteriormente no trabalho de (Pavlovic et al., 2016), que afirma que um particionamento orientado a espaço tem um desempenho melhor que um particionamento orientado a dados para a junção de um conjunto de dados que possui uma densidade similar. Essa de fato é uma característica dos eventos de trânsito. Alertas como

acidentes de carros ou intervenções nas vias tendem a causar engarrafamentos, e as similaridades espaço-temporais entre os diferentes tipos de eventos são geralmente altas.

Entretanto, a diferença nos tempos de processamento ocorre devido à estratégia de filtrar as CGs. que contém os registros candidatos utilizando operações *bit-a-bit* rápidas e restringindo os operadores da consulta para os pares de eventos que ocorreram na mesma célula. Os resultados demonstram que o MIDM pode reduzir o tempo de processamento das consultas espaço-temporais envolvendo eventos de trânsito.

É importante observar que, embora os resultados tenham sido positivos, o processamento da consulta foi realizado de forma sequencial. O objetivo desse trabalho é desenvolver uma estratégia de execução paralela de consulta, explorando o fato dos registros de eventos já estarem particionados pelos CGs. A nova estratégia é detalhada no próximo capítulo.

4 O MIDM PARALELO

O MIDM foi pensado para ser facilmente adaptável para receber eventos em fluxo de dados, além de realizar o processamento paralelo das consultas. Uma das vantagens que o método possui para ser estendido ao processamento paralelo é o particionamento orientado a espaço, utilizando a técnica de tesselação. Dessa forma, ao dividir a área de interesse em células geográficas e realizar a etapa de filtro para recuperar as CGs que fazem parte da consulta, cada CG pode ser enviada para o processamento em um nó, aplicando os critérios da consulta em uma etapa de refinamento.

Como detalhado anteriormente no capítulo 3, o processador de consultas do MIDM foi implementado em C. A linguagem de programação C traz diversas vantagens quanto ao desempenho do processamento, permitindo uma interação mais próxima do sistema operacional. Dessa forma, a manipulação de arquivos, alocações de memória e outros recursos do sistema que venham a ser utilizados pelo MIDM fica a cargo do programador. Uma estratégia que poderia ter sido adotada para a paralelização do MIDM é utilizando a biblioteca de código aberto OpenMPI (*Open Message Passing Interface*). O OpenMPI oferece um conjunto robusto de ferramentas para a criação e execução de programas paralelos em sistemas distribuídos. No entanto, é importante observar que a programação com o OpenMPI pode ser complexa e requer um conhecimento detalhado do ambiente distribuído, bem como das operações de comunicação entre os processos (Graham et al., 2006).

Quando se trata do desenvolvimento de aplicações para computação distribuída, ferramentas como o *Spark* e *Hadoop HDFS* podem oferecer um nível mais alto de abstração, simplificando o processo de programação paralela em comparação com o OpenMPI. Por exemplo, o *PySpark* fornece uma interface Python para o *Spark*, permitindo que os desenvolvedores escrevam códigos mais simples e expressivos para manipulação de dados em um ambiente distribuído, mantendo a otimização que o *Spark* proporciona. Dada a simplicidade de implementação oferecida, neste trabalho optou-se pela paralelização do MIDM com o *Spark* e *Hadoop HDFS*. Na seção 4.1 é detalhada a arquitetura utilizada no desenvolvimento. A seção 4.4 apresenta a implementação, sendo a seção 4.5 dedicada aos experimentos realizados.

4.1 HDFS

O *Hadoop Distributed File System - HDFS*, um componente do ecossistema do *Apache Hadoop*, que desempenha o papel de armazenamento dos dados provenientes de processamentos conduzidos pelo *Spark* e outros sistemas similares. Este sistema distribui os arquivos entre diversos nós do HDFS, fragmentando-os em blocos de disco. Utilizando a tecnologia *MapReduce*, os arquivos armazenados são processados, embora também possa ser utilizado o processamento do *Spark*.

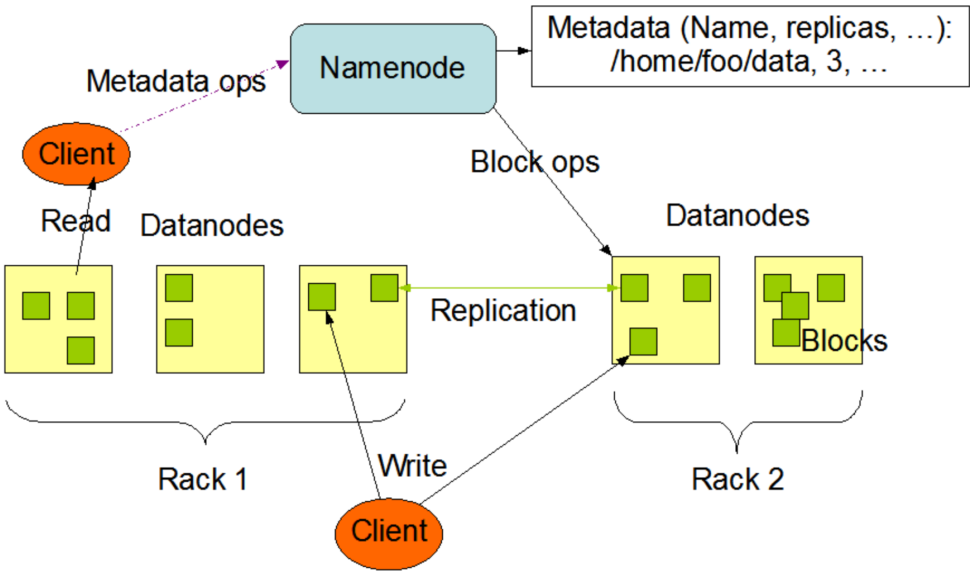


Figura 4.1: Arquitetura do Hadoop HDFS (Apache, 2023)

O *HDFS* organiza os arquivos em formato de diretórios, os quais o *Spark* interpreta como tabelas. Cada *Database* consiste em uma coleção de diretórios contendo as tabelas das bases, seguindo um formato específico de compressão. Para ilustrar a arquitetura proposta pelo HDFS, um exemplo pode ser visualizado na Figura 4.1.

No centro do HDFS está o *NameNode*, o componente responsável por gerenciar o *namespace* do sistema de arquivos e controlar metadados. Ele mantém informações como a árvore de diretórios, atributos de arquivos e a localização física dos blocos de dados nos *DataNodes*. Os DataNodes são os nós de armazenamento reais, onde os dados são fisicamente armazenados em blocos. Eles são responsáveis por ler e gravar dados conforme instruídos pelo cliente ou pelo *NameNode*. Os blocos de dados são replicados em diferentes *DataNodes* para garantir a tolerância a falhas e a disponibilidade dos dados.

Quando um cliente deseja gravar dados no HDFS, ele se comunica com o *NameNode* para obter informações sobre onde os dados devem ser armazenados. A escrita ocorre nos *DataNodes* correspondentes, geralmente replicados em diferentes racks para garantir a redundância e a segurança dos dados. O processo de leitura segue um caminho semelhante: o cliente solicita ao *NameNode* os metadados necessários para localizar os blocos de dados e os *DataNodes* onde estão armazenados. Em seguida, os dados são recuperados dos *DataNodes* relevantes e enviados de volta ao cliente.

A comunicação entre os componentes, como o cliente, o *NameNode* e os *DataNodes*, ocorre por meio de protocolos específicos, como RPC (*Remote Procedure Call*) para solicitar operações e transferir dados entre os nós do *cluster*.

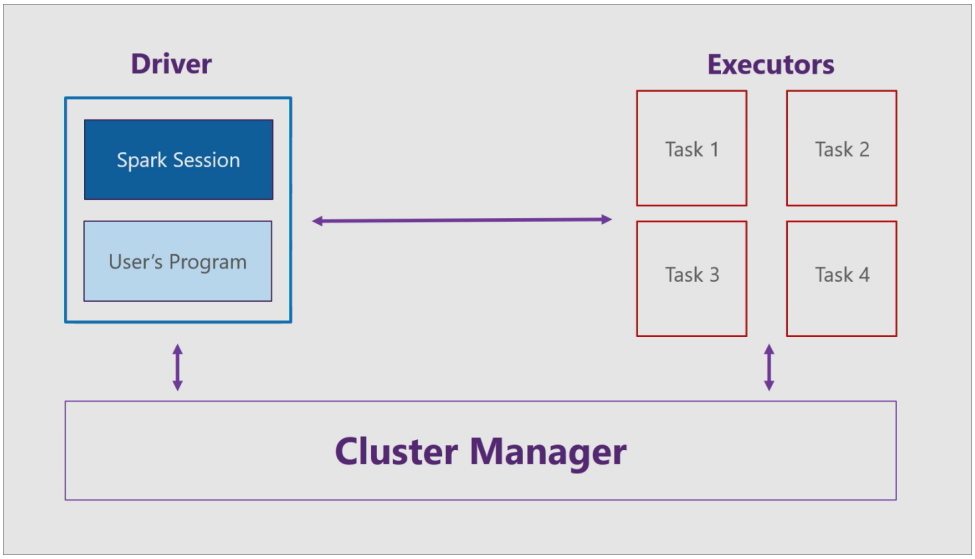


Figura 4.2: Arquitetura do Apache Spark (Habbema, 2023)

4.2 SPARK

O *Apache Spark* é um mecanismo de processamento de dados em memória de código livre criado e mantido pela *Apache*, e projetado para oferecer velocidade e eficiência no processamento de grandes conjuntos de dados. Em comparação com o *Hadoop Distributed File System - HDFS*, o *Apache Spark* se destaca por sua capacidade de processamento em memória e execução de operações complexas de forma mais otimizada. Enquanto o HDFS foca principalmente no armazenamento distribuído e confiável de dados, o *Apache Spark* concentra-se na manipulação eficiente desses dados. O *Spark* opera em cima de diferentes fontes de dados, incluindo o HDFS, mas também suporta outros sistemas de armazenamento, como bancos de dados relacionais, *NoSQL* e sistemas de arquivos locais. Ele oferece suporte a uma ampla biblioteca de funções para a manipulação de dados, compatível também com a linguagem *SQL*, suporte a processamento de *streaming*, aprendizado de máquina e processamento de gráficos.

A arquitetura adotada pelo *Apache Spark* é organizada em torno de um modelo de mestre/escravo. Um dos principais componentes é o *Driver*, responsável pelo controle das operações do usuário através de uma *Spark Session*, que proporciona uma interface para o usuário interagir com o sistema *Spark* e definir as transformações e ações a serem executadas nos dados. Após o usuário submeter um programa para a execução, o *driver* realiza a conversão do programa do usuário para um plano de execução lógico, composto por estágios e tarefas (*Tasks*). O plano de execução é distribuído para os nós ou *Executors* do *Cluster*, que ficam responsáveis pela execução das tarefas de forma coordenada pelo *Driver*. Os executores são processos que estão rodando dentro do *Cluster*, podendo ser local (modo local) através das *Threads* do sistema e compartilhando os mesmos recursos com o *Driver*, ou também remoto (modo *cluster*), de forma que cada executor é uma máquina com recursos próprios. O *Cluster Manager* fica responsável pelo gerenciamento dos recursos como memória e CPU que tanto o *Driver* quanto os *Executors*

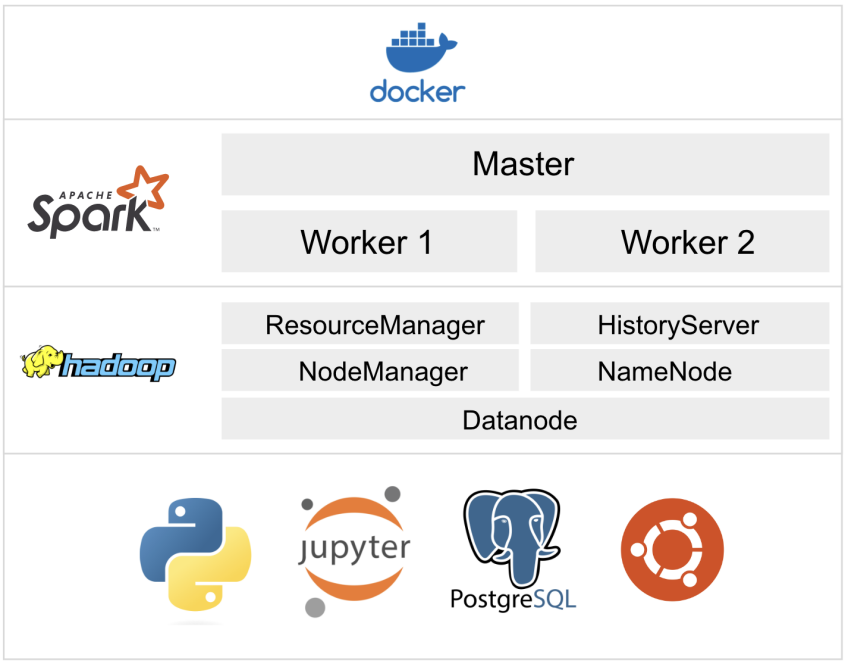


Figura 4.3: Arquitetura do ambiente utilizado

utilizaram para a realização das tarefas. A figura 4.2 ilustra um exemplo de uma arquitetura utilizada pelo *Spark*.

Projetado originalmente utilizando a linguagem de programação *Scala*, o *Spark* possui diversas APIs que dão suporte a outras linguagens como: Java, SQL, R e Python. A interface Python para o *spark* é o *PySpark*, permitindo a utilização do *framework* no ambiente Python, tendo a vantagem da flexibilidade e facilidade da linguagem Python. Ao utilizar o *PySpark*, a própria API do *Spark* irá criar um interpretador Python dentro da própria JVM do *Spark Core* para que o código Python seja executado, tendo um pequeno impacto em seu desempenho de acordo com as funções utilizadas no Python (Karau, 2017).

4.3 AMBIENTE VIRTUALIZADO

Para ser possível a utilização das tecnologias escolhidas para a paralelização do MIDM sem a alocação de grandes recursos, foi optado pela criação de um ambiente virtualizado de *containers*. Essa abordagem traz também a flexibilidade do ambiente ser facilmente migrado para servidores em nuvem com suporte a *Docker*. Uma visão geral da arquitetura utilizada pode ser visualizada na Figura 4.3. Todos os recursos utilizados na criação do ambiente são de código livre e podem ser facilmente encontrados em diversos repositórios, além de uma ampla documentação disponível para realizar as modificações que forem necessárias para uma implementação customizada.

Foram criados 12 *containers* para a execução dos serviços necessários. O *Spark* possui 2 *Workers* que contam com 2 *gigabytes* de memória e realizam a execução das tarefas coordenadas pelo *Master (driver)*, executando em modo *cluster* em todos os testes realizados. A implmentação

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Nov 02 15:14	3	128 MB	_SUCCESS
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=151
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=166
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=171
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=211
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=212
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=232
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=252
drwxr-xr-x	root	supergroup	0 B	Nov 02 15:14	0	0 B	id_cg=273

Figura 4.4: Particionamento das CGs no Hadoop

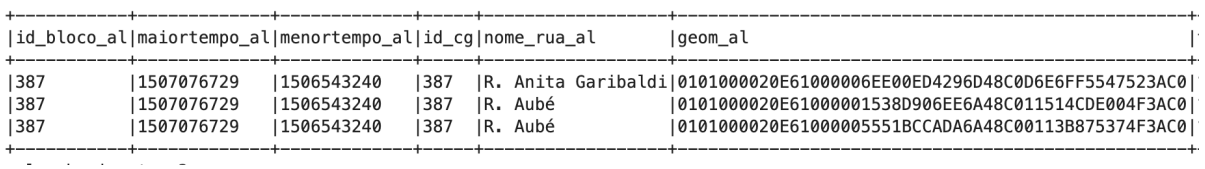
do Hadoop HDFS possui todos os módulos necessários para a utilização completa do *framework*, descritos na seção 4.1. Há também um *Container* específico para a utilização do banco de dados relacional *PostgreSQL*, que possui as extensões necessárias para as operações que utilizam atributos espaciais (PostGIS), além de operações com *bitmaps* comprimidos (*pg_roaringbitmaps*). A implementação em C do MIDM está alocada em um *container* que possui o sistema operacional Linux Ubuntu, contando com as bibliotecas necessárias para a execução e o carregamento das bases de dados.

Para facilitar a realização dos testes e a manipulação dos recursos alocados, foi utilizado o JupyterLab, um ambiente de desenvolvimento interativo acessado através de uma interface *web*, que está alocado em um *container* no ambiente virtual criado. Ele é construído sobre o Jupyter Notebook e oferece uma gama mais ampla de recursos, proporcionando um ambiente flexível para escrever código, criar documentos interativos e realizar as análises necessárias. Todos os *Containers* trabalham em uma rede virtualizada, para permitir que todos os serviços estejam conectados e consigam se comunicar sem que haja nenhum bloqueio, contando com todas as portas dos serviços liberadas.

4.4 IMPLEMENTAÇÃO

Para implementar o MIDM paralelo, foi utilizada a linguagem de programação Python, juntamente com o *framework* PySpark, mencionado na seção anterior. O PySpark nativamente não suporta operações com objetos espaciais. Porém, há bibliotecas como o GeoMesa (Hulbert et al., 2016) que implementam as funções necessárias para o processamento espacial.

Os dados de alertas e engarrafamentos foram extraídos dos arquivos do MIDM e armazenados no Hadoop HDFS utilizando o formato *Parquet*. Para obter vantagem do particionamento espacial, os dados foram particionados no PySpark utilizando a coluna do identificador da CG. Desse modo, ao enviar uma tarefa para ser processada nos *clusters*, os dados que pertencem ao mesmo CG serão processados no mesmo *worker*, evitando um possível *shuffle*, que ocorre



id_bloco_al	maiortempo_al	menortempo_al	id_cg	nome_rua_al	geom_al
387	1507076729	1506543240	387	R. Anita Garibaldi	0101000020E61000006EE00ED4296D48C0D6E6FF5547523AC0
387	1507076729	1506543240	387	R. Aubé	0101000020E61000001538D906EE6A48C011514CDE004F3AC0
387	1507076729	1506543240	387	R. Aubé	0101000020E61000005551BCCADA6A48C00113B875374F3AC0

Figura 4.5: Tabela de alertas visualizada no PySpark

quando os dados precisam ser transferidos de um *worker* para outro. A Figura 4.4 mostra a interface *web* do Hadoop HDFS após a criação das partições dos CGs.

Com os dados espaciais armazenados, é então criada uma sessão PySpark, responsável por realizar a conexão com os componentes do Spark (*Master* e *Workers*). Ao criar uma sessão, o PySpark oferece diversos parâmetros para a configuração do ambiente, sendo os dois principais: a quantidade de memória que o *master* utilizará na orquestração das tarefas junto com recebimento dos dados para a visualização e a quantidade de memória que o *worker* utilizará no processamento das consultas. É importante utilizar uma configuração de sessão adequada ao *hardware* disponibilizado, pois uma má configuração pode gerar problemas com o GC (*Garbage collection*), responsável por liberar os espaços de memória utilizados durantes as operações do *cluster*, culminando em um mau desempenho dos processamentos, ou até um *freezing* completo. Uma configuração adicional necessária nessa implementação, é a submissão dos arquivos *jars* utilizados na execução das funções da biblioteca GeoMesa. As bibliotecas *jars* oferecem uma interface para a implementação de funções em Scala no PySpark.

Após a criação da sessão PySpark, referenciamos o local de armazenamento dos objetos, que pode ser feito da seguinte forma:

```
df = spark.read.parquet("hdfs://hadoop-namenode:9000/midet/alertas.parquet")
```

Utilizando o *namenode* para se conectar ao HDFS, podemos realizar a leitura de todos os objetos armazenados. Nesse caso, podemos especificar a partição na qual será realizada a leitura adicionado "/id_cg=n"ao final do argumento do parquet. Após referenciar o local, teremos um objeto *dataframe* denominado de df. Como o Spark trabalha com o *lazy loading* dos dados, os alertas só serão de fato extraídos e enviados para um *cluster* após o usuário solicitar um método de extração de dados como *show(), collect(), count()...* A Figura 4.5 mostra alguns registros da tabela de alertas após a chamada da função *show()*. Nessa tabela há os atributos: identificador do bloco (MIDM), os *timestamps* referentes ao bloco, conforme ilustrado na 3.2, o identificador da CG e a geometria do ponto representada por um WKB - (*Well-Known Binary*). Os atributos dos engarrafamentos seguem os mesmos da tabela de alertas, diferenciando apenas no tamanho e o conteúdo do WKB, que contém um conjunto de pontos.

Um dos pontos que foi alterado entre a versão sequencial e a versão paralela, foram os filtros realizados com os *bitmaps*. Nessa implementação, optou-se por prosseguir sem as operações de *bitmap*, levando em consideração o particionamento realizado pelo Spark no HDFS. Para a etapa de filtro, é aplicado um *join* pelo atributo de id do cg, que mais tarde, no

processamento, será utilizado para recuperar apenas as partições relevantes para a etapa de refinamento da consulta, conforme o funcionamento das execuções no PySpark.

4.5 EXPERIMENTOS REALIZADOS

Nos experimentos realizados, foram considerados os mesmos conjuntos de dados trabalhados com o MIDM sequencial: alertas e engarrafamentos coletados pelo aplicativo Waze, referente ao período de setembro de 2017 a setembro de 2018. Os dados foram separados em conjuntos de 20% (B20), 50% (B50) e 100% (B100) da base original, como detalhado na seção 3.6.

A Consulta realizada permaneceu a mesma da versão sequencial, com o intuito de analisar e comparar os resultados: "Quais ruas tiveram engarrafamentos e alertas no mesmo ponto após o primeiro dia de setembro de 2017?". A consulta leva em consideração os atributos geométricos das duas tabelas, realizando uma interseção entre o ponto de alerta e o conjunto de pontos do engarrafamento e retornando o nome das ruas distintas. Um exemplo da consulta no PySpark é demonstrado a seguir:

```
df = alerts_20.join(jams_20,
                    ST_Intersects(alerts_20.geom, jams_20.geom),
                    'inner'
                    )
```

Foi utilizado um sistema com 7,5 Gigabytes de memória RAM e 6 núcleos de processamento para os experimentos, distribuídos em todos *containers* utilizados pelos serviços dentro do Docker. Para um melhor desempenho dos testes, foram desativados os *containers* que não estavam mais sendo utilizados para essa etapa de testes (Ubuntu e PostgreSQL). O PySpark foi configurado com 2 *cores*, contando com 1 Gigabyte de memória RAM para cada *worker*, totalizando 2 Gigabytes e mais 1 Gigabyte para o *master*.

Base	MIDM Sequencial	MIDM Paralelo
B20	20s425ms	23s120ms
B50	56s506ms	4min37s435ms
B100	1m50s055ms	9m21s438ms

Tabela 4.1: Comparação da execução sequencia e paralela

Os resultados dos experimentos com o MIDM paralelo em comparação com os resultados já obtidos do MIDM sequencial são apresentados na Tabela 4.1. É possível observar que nessa estratégia de paralelização utilizada, os resultados não foram satisfatórios. Na execução utilizando 20% dos registros, o MIDM paralelo chegou próximo da execução sequencial, porém, ao trabalhar com uma massa maior de dados, o método sequencial acabou se destacando no tempo de processamento. Uma observação importante nos experimentos realizados com o MIDM paralelo, é de que, em alguns momentos ocorreram problemas com o GC do PySpark, pois os recursos alocados para os *clusters* não foram suficientes a medida que os outros serviços também

utilizavam os recursos da virtualização. O PySpark foi projetado e pensado para a execução em memória RAM, sendo uma das melhorias em comparação com o *MapReduce* do Hadoop. Porém, o Spark conta também com uma configuração para realizar escritas em disco ao alcançar a capacidade máxima da memória RAM, realizando um *swapping*. Essa configuração foi ativada durante os testes para que fosse possível a execução do B50 e do B100.

Ao utilizar o PySpark para realizar junções espaciais, como mencionado anteriormente, foi necessário a utilização de bibliotecas que estendem as funções do Spark para funções geométricas, como a *ST_Intersects()*. A biblioteca utilizada foi a GeoMesa, que conta também com recursos para outros serviços da Apache, além de proporcionar um sistema de armazenamento para dados espaciais que não foi explorado no escopo desse trabalho. Também foi explorado o ecossistema de geoprocessamento Apache Sedona, que se mostrou eficiente em trabalhos realizados sobre o processamento de dados espaciais (Yu et al., 2019). Porém, o ecossistema não conta com o Hadoop HDFS, sendo necessário buscar uma alternativa para o particionamento e armazenamento dos dados do MIDM.

Por fim, ao trabalhar com dados espaciais utilizando o PySpark, é necessário realizar implementações utilizando a linguagem de programação nativa Scala para obter o melhor desempenho, sem recorrer a funções definidas pelo usuário diretamente em Python, que necessita a criação de um interpretador na sessão PySpark a cada execução de uma junção, o que também não é recomendado pelas documentações do Spark.

5 CONCLUSÃO

Este trabalho apresentou o **M**étodo de **I**ndexação para **D**ados de **M**obilidade (MIDM), que tem por objetivo criar um índice eficiente para junções de dados espaço-temporais de trânsito, servindo de base para a análise e planejamento de cidades inteligentes. O MIDM explora as familiaridades espaço-temporais de diferentes tipos de eventos para desenvolver um método com particionamento orientado ao espaço introduzido pelo PBSM, com a organização de arquivo similar ao arquivo de *grid*. No MIDM sequencial, há também a indexação *bitmap* para filtrar as CGs que não serão utilizadas na etapa de refinamento, trazendo eficiência ao processamento.

Em uma primeira abordagem para a implementação do método paralelo, foram estudados e explorados os cenários que contemplam uma implementação de forma simples, utilizando o *framework* de processamento paralelo Spark, juntamente com o sistema de armazenamento do Hadoop HDFS. Através de experimentos realizados com dados reais do aplicativo Waze, a implementação paralela realizada não alcançou o desempenho esperado para o MIDM, principalmente por questões ligadas aos recursos alocados e as variáveis de configuração dos *clusters* do PySpark. Entretanto, foram realizados estudos sobre as bibliotecas de processamento paralelo utilizadas atualmente, que darão subsídio a uma pontual implementação paralela do método.

5.1 PUBLICAÇÕES E PRÊMIOS

Ao longo do trabalho com o MIDM, foram produzidos e publicados três artigos em conferências nacionais da área de banco de dados. O primeiro artigo foi publicado e apresentado na ERBD (escola regional de banco de dados) 2021 - Agrupamento de eventos de trânsito baseado em tesselação (Duarte et al., 2021), trazendo uma visão geral da arquitetura proposta para o MIDM. O segundo artigo foi publicado no SBBD (simpósio brasileiro de banco de dados) 2021 - *MIDET: A Method for Indexing Traffic Events* (Pontarolo et al., 2021), neste trabalho publicado, o método passou por algumas modificações e foi de fato implementado, trazendo experimentos com as consultas propostas sobre a base de dados do aplicativo *Waze*, demonstrando que o método é de fato eficaz em otimizar consultas de eventos de trânsito. Neste mesmo simpósio, o artigo *MIDET* foi premiado como *Honorable Mention Award*, uma menção honrosa do artigo dada pela banca após a avaliação de todos os artigos completos, contando com as respectivas apresentações. A última publicação realizada sobre o método, foi a criação de uma interface de consultas interativas (*demo*), denominada *MIDET-IU* no SBBD 2022 (Pontarolo et al., 2022), que através do processamento de consultas do MIDM, retornava graficamente o resultado das consultas realizadas. O trabalho como tema de iniciação científica também foi contemplado como o melhor trabalho da banca, contando com a apresentação realizada no EVINCI/EINTI - UFPR 2022.

5.2 TRABALHOS FUTUROS

Com o intuito de continuar a desenvolver o MIDM, mais experimentos precisam ser realizados sobre o método paralelo implementado, incluindo as porcentagens dos conjuntos de dados que foram incluídos nos testes entre o MIDM e o Waze e incluir novas combinações de configuração para a sessão do PySpark, ou até mesmo migrar para máquinas com um *hardware* mais eficiente.

Além disso, ainda há modificações que precisam ser realizadas no método paralelo para que o desempenho seja otimizado, explorando uma programação a mais baixo nível no *core* do Spark, utilizando RDDs (*Resilient Distributed Dataset*). Outra modificação também seria explorar a possibilidade de incluir a etapa de filtro que o MIDM sequencial possui, sem recorrer a junção pelos CGs, resgatando os blocos físicos do HDFS. Há também a possibilidade de realizar outras implementações com novas bibliotecas de processamento paralelo de dados espaciais, principalmente com o Apache Sedona, um *framework* que foi criado sobre o Spark especificamente para o processamento de dados espaciais.

Os trabalhos futuros também incluem um método para detectar automaticamente o melhor tamanho para a célula geográfica, modificando também a configuração temporal quando a junção estiver excedendo o limite da memória primária. Experimentos com o tamanho das CGs também precisam ser conduzidos para comparar o desempenho, juntamente com a comparação entre outras estruturas de índices espaciais, utilizando outros conjuntos de dados para analisar detalhadamente o impacto da densidade de dados e distribuição ao longo das CGs.

Por fim, um dos tópicos que também será abordado para a extensão do MIDM após mais experimentos conduzidos da extensão paralela, será a extensão do método para o recebimento de eventos de trânsito reportados em fluxo de dados.

REFERÊNCIAS

Aji, A., Vo, H. e Wang, F. (2015). Effective spatial data partitioning for scalable query processing. *CoRR*, abs/1509.00910:12 pages.

Al-Badarneh, A. F., Al-Alaj, A. S. e Mahafzah, B. A. (2013). Multi small index (MSI): A spatial indexing structure. *Journal of Information Science*, 39(5):643–660.

Antoine, E., Ramamohanarao, K., Shao, J. e Zhang, R. (2011). Accelerating spatial join operations using bit-indices. Em *Proc. of the 22nd Australasian Database Conference*, volume 115, página 123–132, AUS. Australian Computer Society, Inc.

Apache (2023). Hdfs architecture. `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`. Acessado em 10/12/2023.

Chaudhry, N., Yousaf, M. M. e Khan, M. T. (2020). Indexing of real time geospatial data by IoT enabled devices: Opportunities, challenges and design considerations. *Journal of Ambient Intelligence and Smart Environments*, 12:281–312.

Duarte, M., Pontarolo, M., Freitas, R. e Hara, C. (2021). Agrupamento de eventos de trânsito baseado em tesselação. Em *Anais da XVI Escola Regional de Banco de Dados*, páginas 91–98, Porto Alegre, RS, Brasil. SBC.

Eldawy, A. e Mokbel, M. F. (2015). Spatialhadoop: A mapreduce framework for spatial data. Em *2015 IEEE 31st International Conference on Data Engineering*, páginas 1352–1363, Seoul, Korea. IEEE.

Gaede, V. e Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231.

Graham, R. L., Woodall, T. S. e Squyres, J. M. (2006). Open mpi: A flexible high performance mpi. Em Wyrzykowski, R., Dongarra, J., Meyer, N. e Waśniewski, J., editores, *Parallel Processing and Applied Mathematics*, páginas 228–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *ACM Sigmod Record*, 14(2):47–57.

Habbema, H. (2023). Pyspark process. `https://medium.com/@habbema/vamos-brincar-com-o-spark-eb3e7b7887a9`. Acessado em 10/12/2023.

Hulbert, A., Kunicki, T., Hughes, J. N., Fox, A. D. e Eichelberger, C. N. (2016). An experimental study of big spatial data systems. Em *2016 IEEE International Conference on Big Data (Big Data)*, páginas 2664–2671.

Karau, H., W.-R. (2017). *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark*. O'Reilly Media, Incorporated.

Lemire, D., Ssi-Yan-Kai, G. e Kaser, O. (2018). Consistently faster and smaller compressed bitmaps with roaring. *Software: Practice and Experience*, 46:1547–1569.

Mahmood, A. R., Punni, S. e Aref, W. G. (2019). Spatio-temporal access methods: a survey (2010 - 2017). *Geoinformatica*, 23:1–36.

Maryland (2022). Reports - maryland department of assessments and taxation. `https://chart.maryland.gov/incidents/index.php`. Acessado em 15/07/2023.

Neto, C. J., Ciferri, R. R. e Santos, M. T. P. (2013). HSTB-index: A hierarchical spatio-temporal bitmap indexing technique. Em *SBBD - Workshop de Teses e Dissertações*.

Nievergelt, J., Hinterberger, H. e Sevcik, K. C. (1984). The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71.

Patel, J. M. e DeWitt, D. J. (1996). Partition based spatial–merge join. Em *Proc. of the 1996 ACM SIGMOD international conference on Management of data*, páginas 259–270.

Patel, J. M., Yu, J., Kabra, N., Tufte, K. A., Nag, B., Burger, J., Hall, N. E., Ramasamy, K., Lueder, R., Ellmann, C. J., Kupsch, J., Guo, S., Larson, J. G., Witt, D. J. D. e Naughton, J. F. (1997). Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. Em *Proc. of the 1997 ACM SIGMOD international conference on Management of data*, página 336–347.

Pavlovic, M., Heinis, T., Tauheed, F., Karras, P. e Ailamaki, A. (2016). Transformers: Robust spatial joins on non-uniform data distributions. Em *Proc. of the 32nd International Conference on Data Engineering (ICDE)*, páginas 673–684.

Pontarolo, M., Duarte, M., Schroeder, R. e Hara, C. (2021). Midet: A method for indexing traffic events. Em *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*, páginas 217–228, Porto Alegre, RS, Brasil. SBC.

Pontarolo, M., Ferreira, M., Schroeder, R. e Hara, C. (2022). Um sistema para consultar eventos de trânsito históricos do waze. Em *Anais Estendidos do XXXVII Simpósio Brasileiro de Bancos de Dados*, páginas 65–70, Porto Alegre, RS, Brasil. SBC.

Sevcik, K. C. e Koudas, N. (1996). Filter trees for managing spatial data over a range of size granularities. Em *Proc. of the 22nd International Conference on Very Large Data Bases (VLDB)*, página 16–27.

Shin, J., Mahmood, A. e Aref, W. (2019). An investigation of grid-enabled tree indexes for spatial query processing. Em *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, páginas 169–178.

Shohdy, S., Su, Y. e Agrawal, G. (2015). Load balancing and accelerating parallel spatial join operations using bitmap indexing. Em *Proc of the IEEE 22nd International Conference on High Performance Computing (HiPC)*, páginas 396–405.

Siqueira, T. L. L., de Aguiar Ciferri, C. D., Times, V. C. e Ciferri, R. R. (2012). The SB-index and the HSB-index: efficient indices for spatial data warehouses. *Geoinformatica*, 16(1):165–205.

Tsitsigkos, D., Bouros, P., Mamoulis, N. e Terrovitis, M. (2019). Parallel in-memory evaluation of spatial joins. Em *Proc. of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, páginas 516–519.

Vo, H., Aji, A. e Wang, F. (2014). SATO: A spatial data partitioning framework for scalable query processing. Em *Proc. of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, página 545–548.

Wang, J., Lin, Y. C. e S., S. (2017). An experimental study of bitmap compression vs. inverted list compression. Em *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, New York, NY.

Yu, J., Wu, J. e Sarwat, M. (2015). Geospark: A cluster computing framework for processing large-scale spatial data. Em *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA. Association for Computing Machinery.

Yu, J., Zhang, Z. e Sarwat, M. (2019). Spatial data management in apache spark: The geospark perspective and beyond. *Geoinformatica*, 23(1):37–78.