UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

THIAGO BORGES DE OLIVEIRA

DSI-RTree

Um Índice R-Tree Distribuído Escalável

Goiânia
2010

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: DSI-RTree – Um Índice R-Tree Distribuído Escalável

Autor(a): Thiago Borges de Oliveira

Goiânia, 15 de Dezembro de 2010.

Thiago Borges de Oliveira – Autor

Dr. Vagner José do Sacramento Rodrigues – Orientador

THIAGO BORGES DE OLIVEIRA

DSI-RTree

Um Índice R-Tree Distribuído Escalável

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Sistemas Distribuídos.

Orientador: Prof. Dr. Vagner José do Sacramento Rodrigues

Goiânia
2010

THIAGO BORGES DE OLIVEIRA

DSI-RTree

Um Índice R-Tree Distribuído Escalável

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 15 de Dezembro de 2010, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Vagner José do Sacramento Rodrigues
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Thierson Couto Rosa
Instituto de Informática – UFG

Prof. Dr. Marco Antonio Casanova
Departamento de Informática – PUC-Rio

Thiago Borges de Oliveira

Graduou–se em Ciência da Computação na FESURV - Universidade de Rio Verde. Na graduação, pesquisou sistemas paralelos e distribuídos para análise microscópica de trânsito urbano. Atualmente é bolsista do laboratório LUPA, onde estuda soluções distribuídas para acesso e processamento de dados GIS. Desde 2002 desenvolve soluções em Banco de Dados e Sistemas Distribuídos para aplicações ERP, na empresa Siagri Sistemas de Gestão.

Aos meus entes queridos e amigos, por entenderem a ausência necessária para me dedicar a este desafio.

Agradecimentos

Muitos tiveram um papel fundamental desde a concepção da ideia até o presente resultado. Meus agradecimentos, aos poucos que aqui me lembro, não estão em nenhuma ordem de importância. Todos tiveram um papel fundamental para a realização deste trabalho, de uma forma ou outra.

Obrigado a meus familiares pelos conselhos e exemplos que me fizeram chegar aqui. Quaisquer outros que fossem, teriam me levado a algum lugar diferente. Estou feliz que tenha chegado onde estou hoje.

À minha esposa, Cristiane Valente, por entender minha "pseudo" ausência, estando em casa e não lhe prestando a devida atenção e companhia.

Meus sinceros agradecimentos ao professor Dr. Vagner Sacramento, meu orientador, que antes de qualquer outro, acreditou que eu seria um aluno digno de suas orientações e projetos.

À Siagri, por me permitir dedicar um tempo precioso às aulas e desenvolvimento do projeto.

Meu amigo Sávio Teles, que me acompanhou durante praticamente todo o mestrado. Sua ajuda foi muito importante no entendimento de artigos científicos e na codificação e depuração de código, desde a primeira versão centralizada do algoritmo.

A todo o time do LUPA que prestou um grande suporte a este trabalho, seja com ideias, artigos, código e ainda ajuda nos testes ou análises de resultados.

A todos que participaram indiretamente, desenvolvendo e disponibilizando livremente para uso as diversas ferramentas utilizadas. Dentre elas Linux, LaTeX, gnuplot, bash script, svn, vim, qgis, a plataforma JVM, biblioteca JTS, biblioteca GeoTools, além de outras, que sem dúvida, fizeram possível este trabalho.

When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem.

Dijkstra, Edsger W.,
EWD 963: Visuals for BP's Venture Research Conference.

Resumo

de Oliveira, Thiago Borges. **DSI-RTree**. Goiânia, 2010. 81p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Em face de um mundo computacional ubíquo cada vez mais possível, tem crescido constantemente a necessidade de sistemas de processamento de dados espaciais que suportem a criação de aplicações massivas para explorar a grande quantidade de dados existente, a fim de auxiliar a vida cotidiana das pessoas e prover novas ferramentas para empresas e governo. Soluções atuais de processamento, em sua maioria, não possuem a escalabilidade necessária para atender esta demanda e novas soluções distribuídas que usam eficientemente os recursos computacionais são necessárias. Este trabalho apresenta o DSI-RTree, um sistema distribuído e escalável, que implementa a indexação e processamento distribuído de dados espaciais em um *cluster* de computadores. Uma avaliação de parâmetros da construção do índice espacial distribuído é realizada, abordando aspectos como o tamanho das partições criadas, a forma de distribuição destas partições e o impacto destas definições na troca de mensagens entre as máquinas do *cluster*. Uma fórmula para cálculo do tamanho das partições conforme o tamanho dos datasets é proposta, a fim de garantir eficiência no processamento de consultas na arquitetura projetada. Testes práticos do sistema mostraram uma escalabilidade maior que linear no processamento de consultas de janela em datasets espaciais de 32 e 158 mil polígonos.

Palavras–chave

Processamento Distribuído, Dados Espaciais, Particionamento de Dados, R-Tree.

Abstract

de Oliveira, Thiago Borges. **DSI-RTree**. Goiânia, 2010. 81p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The demand for spatial data processing systems that support the creation of massive applications has steadily grown in the increasingly ubiquitous computing world. These demands aims to explore the large amount of existing data to assist people's daily lives and provide new tools for business and government. Most of the current solutions to process spatial data do not meet the scalability needed, and thus new solutions that efficiently use distributed computing resources are needed. This work presents a distributed and scalable system called DSI-RTree, which implements a distributed index to process spatial data in a cluster of computers. We also have done a review of details related to the construction of the distributed spatial index, by addressing issues such as the size of data partitions, how that partitions are distributed and the impact of these definitions in the message flow on the cluster. An equation to calculate the size of the partitions based on the size of data sets is proposed, to ensure efficiently query processing on the proposed architecture. We have done some experiments running window queries in spatial data sets of 33,000 and 158,000 polygons and the results showed a scalability greater than linear.

Keywords

Distributed Processing, Spatial Data, Data Partitioning, R-Tree.

Sumário

Lista de Figuras **11**

Lista de Tabelas **13**

1 Introdução **14**

2 Estruturas para Processamento de Dados Espaciais **18**
- 2.1 KD-Tree 18
- 2.2 Hilbert R-Tree 19
- 2.3 R-Tree 20
- 2.4 Termos e Algoritmos Utilizados na R-Tree 22
- 2.5 Extensões da R-Tree 24
- 2.6 Processamento de Dados Espaciais Centralizado, Paralelo e Distribuído 25
- 2.7 Particionamento da R-Tree em Arquiteturas *Shared-Nothing* 27
- 2.8 Considerações Finais 27

3 Arquitetura e Implementação do DSI-RTree **29**
- 3.1 Desafios da Construção de Índices Espaciais Distribuídos 29
- 3.2 Visão Geral da Arquitetura 30
- 3.3 Armazenamento de Partições no Servidor DSI 32
- 3.4 Concorrência na Construção do Índice Distribuído 34
- 3.5 Ajuste do MBR *Top-Down* 35
- 3.6 Escalonador de Partições 35
- 3.7 Outros Detalhes da Implementação 37
- 3.8 Considerações Finais 38

4 Definição do Tamanho das Partições do Índice Distribuído **40**
- 4.1 Valor de M em Sistemas Centralizados 40
- 4.2 Valor de M em Sistemas Distribuídos 41
- 4.3 Impacto do Valor de M em Datasets Reais 43
- 4.4 Definição do Valor de M 46
- 4.5 Considerações Finais 49

5 Avaliação de Desempenho **50**
- 5.1 *Hardware* e *Software* do Ambiente de Execução dos Testes 50
- 5.2 Qualidade do Índice Distribuído 51
- 5.3 Avaliação do Escalonador *Round-Robin* 56
- 5.4 Escalabilidade Horizontal 59
  - 5.4.1 Quantidade de Requisições Processadas 60

5.4.2 Processamento de Grandes Datasets 62
5.5 Considerações Finais 62

6 Trabalhos Correlatos **64**
6.1 Paradise 64
6.2 M-RTree e MC-RTree 66
6.3 *Storing Spatial Data on NOW* 68
6.4 SD-RTree 69
6.5 Hadoop-GIS 71
6.6 Considerações Finais 71

7 Conclusão **73**

Referências Bibliográficas **76**

A Acesso à Folhas e Diretórios do Dataset Arizona com M=80 **80**

Lista de Figuras

2.1 KD-Tree de quatro níveis, com pontos separados em quadrantes à esquerda. A largura de cada linha representa o nível da divisão. À direita é demonstrada a estrutura hierárquica. 19
2.2 Curva de Preenchimento de Espaço de Hilbert. Adaptado de [23]. 19
2.3 *Minimum Bounding Rectangle* (MBR). Retângulo exterior, que engloba o objeto geométrico por completo e com lados paralelos aos eixos x e y. Adaptado de [18]. 20
2.4 Organização de uma R*-Tree. Adaptado de [26]. 21
2.5 Particionamentos possíveis para dados unidimensionais [12]. 26
2.6 Particionamento de uma R-Tree em arquitetura *shared-nothing* [26]. 27

3.1 Visão Geral da Arquitetura do DSI-RTree 31
3.2 Comparação da organização de partições na arquitetura proposta com a representação de uma R-Tree. 33
3.3 Escopo de Bloqueio na técnica *Lock Coupling* [11]. 34
3.4 Acesso a Máquinas de acordo com o agrupamento dos itens/tamanho das partições [23]. 36

4.1 Ilustração dos Datasets e Janelas de Busca: Arizona no canto inferior esquerdo, com ampliação ao seu redor (a), Desmata (b) com ampliação no canto superior direito e Vegeta (c). As áreas em destaque são Janelas de Busca. 45
4.2 Análise de Métricas em cada Dataset, para valores de M variados. 46
4.3 Simulação da baixa quantidade de itens no nó raiz, com M=130, para datasets com quantidade de polígonos entre 10^3 e 10^7. 47
4.4 Sugestão do valor de M para índices de 2, 3 e 4 níveis e datasets com até 10 milhões de polígonos. 48

5.1 Acesso a Folhas e Diretórios do Dataset Arizona, com M=80, por 30 buscas de janela: à esquerda gráfico na escala original, mostrando a quantidade de nós acessados e à direita a parte inferior ampliada. 53
5.2 Percentual de Folhas e Diretório Acessados durante Buscas de Janela, para cada Dataset e Valor de M. 55
5.3 Mapa de calor indicando a quantidade de partições acessadas por cada janela, no dataset Arizona em 5, 10, 15 e 20 máquinas. 58
5.4 Escalabilidade Horizontal do Sistema DSI-RTree, para 5, 10, 15 e 20 máquinas interligadas em *cluster* (a) e Fator de Escalabilidade (b). 61
5.5 Relação entre o tamanho dos datasets testados. 62

6.1 Visão Geral da Arquitetura do Banco de Dados Paradise [29]. 65
6.2 Particionamento na M-RTree: Um servidor mestre mantém os nós internos do índice, e as folhas são divididas entre os servidores clientes (ou escravos) [26]. 66
6.3 Particionamento na MC-RTree: As folhas continuam sendo armazenadas somente nos nós escravos como na M-RTree, mas cada servidor cria uma árvore R-Tree com suas entradas [33]. 67
6.4 Representação da estrutura binária da SD-RTree e a imagem parcial da estrutura na aplicação cliente [28]. 70

Lista de Tabelas

4.1 Mudança de Altura e Preenchimento da Raiz conforme alteração no valor de M. 44

5.1 Percentual médio, mínimo, máximo e desvio padrão da quantidade de polígonos interceptados, em relação ao total de polígonos de cada dataset. 53
5.2 Percentual de Folhas Falsas, em relação ao total de Folhas Acessadas. 55
5.3 Distribuição das partições geradas para cada dataset em 5, 10, 15 e 20 máquinas. 57
5.4 Quantidade de máquinas clientes e requisições por segundo utilizadas na composição total de requisições por segundo enviadas ao sistema. 60

CAPÍTULO 1

Introdução

Se imaginarmos o mundo ubíquo vislumbrado por Mark Weiser [40], no final da década de 70, poderíamos identificar vários cenários que exploram a potencialidade das tecnologias atuais para prover serviços de alto valor agregado para os usuários. Por exemplo, um usuário passeando com sua família, em uma cidade que estão visitando pela primeira vez, recebe, ao longo do trajeto, informações em seu celular sobre ofertas de promoções de lojas que estão próximas de sua localização, ou dicas de segurança dizendo que, na direção em que está indo, a 3 km de distância, há alguma situação de risco tal como um desmoronamento na pista, um arrastão, dentre outras possibilidades. O que ainda falta para tornar esse cenário uma realidade? Será que as tecnologias atuais não são o suficiente?

De fato, estamos na era do mundo ubíquo geográfico, pois há uma grande disponibilidade de dados espaciais e os celulares com GPS/GPRS estão cada vez mais comuns e acessíveis. Entretanto, muitos desafios ainda precisam ser tratados para prover esse tipo de aplicação em larga escala tal como a ineficiência e falta de escalabilidade das plataformas de geoprocessamento que implementam as operações espaciais para encontrar pontos de interesse próximos dos usuários, ou que comparam dois datasets[1] para encontrar registros colocalizados, dentre outras. Este é o problema investigado nesta dissertação.

Uma aplicação LBS usada por milhões de usuários simultaneamente precisaria processar milhões de requisições a cada movimento desses usuários cruzando suas localizações com pontos de interesse, levando suas preferências em consideração. Além disto, o armazenamento e processamento de grandes datasets espaciais têm se tornado um desafio para os Sistemas de Informações Geográficas (SIG) [7] que são a base para as aplicações LBS e aplicações de Análise Espacial que fazem o cruzamento de dados brutos para extrair informações úteis para tomada de decisão.

[1] O termo *dataset* foi empregado no texto para se referir a um conjunto de dados espaciais, como por exemplo um arquivo *shapefile*. Do Inglês *data set*.

Normalmente, os sistemas de geoprocessamento usuais, incluindo banco de dados espaciais e sistemas especialistas SIG, responsáveis por armazenar e realizar o processamento das operações espaciais para as aplicações LBS e de Análise Espacial, são centralizados e estão limitados ao poder de processamento e armazenamento de um único servidor. Mesmo soluções paralelas com réplicas de servidores de grande porte têm a sua escalabilidade de processamento e armazenamento de dados limitada pelas necessidades de processamento de grande quantidade de requisições simultâneas, sincronização, movimentação e armazenamento de grande volume de dados em cada réplica, além de muitas vezes tornar a solução inviável financeiramente.

Trabalhando com servidores de grande porte replicados, há situações em que o desempenho pode deixar a desejar, pois o sistema responsável pelo processamento das consultas espaciais ainda não tira proveito de todo o poder de processamento, dado que uma única consulta complexa não é dividida e processada em paralelo nas réplicas. Para tornar essas aplicações viáveis de serem usadas em larga escala, é necessário trabalhar com soluções distribuídas de geoprocessamento que paralelizam a execução de uma ou mais consultas espaciais em um *cluster* de computadores tirando proveito de todo o poder de processamento paralelo e armazenamento de dados distribuído.

Clusters de máquinas convencionais possuem um custo reduzido em relação a máquinas paralelas de alto desempenho, além de outras facilidades para prover escalabilidade e disponibilidade a um sistema SIG. Por exemplo:

- Os dados espaciais podem ser distribuídos pelas máquinas do *cluster*, de forma que cada uma armazene e processe requisições em parte do dataset;
- É possível aumentar a capacidade de armazenamento e processamento, adicionando ou removendo máquinas ao *cluster* existente, tornando o sistema escalável horizontalmente;
- Possibilita a existência de réplicas pró-ativas, que além de serem utilizadas para prover alta disponibilidade do sistema, podem participar da escalabilidade no balanceamento das requisições;
- Torna o investimento, em *hardware* e energia, proporcional ao aumento da demanda pelo serviço.

Porém, há vários desafios na implementação de soluções eficientes para armazenar e processar grande volume de dados espaciais de forma distribuída. Métodos de divisão devem ser empregados para possibilitar o armazenamento de partes dos dados (partições) em cada uma das máquinas de um *cluster*, de forma a possibilitar a divisão do problema (algoritmo) a ser processado em paralelo de forma eficiente, à medida que mais máquinas são empregadas no cálculo da solução. Esta divisão deve ainda utilizar métodos de organização (índices) que permitam o acesso aos dados de forma eficiente pelos algoritmos, minimizando a necessidade de comunicação entre as máquinas. A distribuição

destas partições (escalonamento de dados entre as máquinas) também é um fator importante a ser considerado, pois distribuições ineficientes podem causar uma baixa utilização do poder de processamento do *cluster*, deixando partes dele ociosas enquanto outras estão sobrecarregadas.

Outros desafios também devem ser considerados, como é o caso da garantia da consistência das informações distribuídas. Manter o conjunto de dados e as respostas dos algoritmos consistentes é uma premissa para qualquer sistema computacional e a distribuição dos dados torna este aspecto um desafio em um sistema distribuído. Construir sistemas distribuídos por si só também é um desafio devido à necessidade de lidar com várias camadas de *software*, protocolos de rede, métodos de sincronização distribuídos, além de outros.

Neste trabalho foi projetado e implementado um sistema de processamento distribuído de dados espaciais, chamado de DSI-Rtree (Distributed Spatial Index-Rtree), que lida com o particionamento e distribuição de dados, a fim de prover escalabilidade no armazenamento e processamento de consultas espaciais em arquitetura de computadores distribuída (*clusters*). Este sistema implementa um algoritmo distribuído de indexação de dados espaciais baseado na R*-Tree [5], que torna viável o armazenamento de dados distribuído e o processamento paralelo de consultas complexas de forma escalável e eficiente.

Testes práticos realizados demonstraram que o sistema implementado provê escalabilidade no processamento de consultas espaciais para tamanhos de datasets crescentes (2 mil, 32 mil e 158 mil itens). Aumentando a quantidade de máquinas de cinco para dez durante a execução de buscas no índice, conseguiu-se um aumento maior que linear na taxa de requisições atendidas por segundo para um dataset com 158 mil polígonos (de 10 reqs/s para 30 reqs/s), e para um dataset com 32 mil polígonos (de 11 reqs/s para 25 reqs/s).

Em função dos resultados obtidos, as principais contribuições científicas deste trabalho podem ser descritas como segue:

- Avaliação do impacto da variação do tamanho de datasets, quantidade de máquinas e configuração do índice distribuído no desempenho de operações de buscas espaciais;
- Um estudo a respeito do tamanho das partições criadas, observando parâmetros como o tamanho dos datasets e os níveis de troca de mensagens entre as máquinas do *cluster*, e a definição de uma fórmula para calcular este tamanho na arquitetura proposta;
- A construção de um índice espacial distribuído para processamento de dados espaciais, que trata do armazenamento e distribuição de forma consistente;

- Desenvolvimento de uma arquitetura distribuída e escalável para o processamento de dados espaciais e investigação científica de algoritmos espaciais distribuídos.

O restante do trabalho está organizado da seguinte forma: O Capítulo 2 faz uma revisão na literatura sobre estruturas e algoritmos propostos para indexação de dados espaciais e formas de divisão destes dados em arquiteturas distribuídas; o Capítulo 3 apresenta a arquitetura e os detalhes de implementação do índice espacial distribuído; o Capítulo 4 apresenta uma análise detalhada para determinar a melhor escolha do tamanho das partições dos dados na arquitetura proposta; o Capítulo 5 apresenta e discute os resultados dos testes de desempenho; o Capítulo 6 discute os trabalhos correlatos e, por fim, o Capítulo 7 apresenta a conclusão e direções para trabalhos futuros.

CAPÍTULO 2

Estruturas para Processamento de Dados Espaciais

Neste capítulo é feita uma revisão da literatura a respeito de estruturas e algoritmos propostos para indexação de dados espaciais.

A indexação de dados espaciais vem sendo pesquisada desde 1975, quando foi proposta a estrutura denominada KD-Tree [6]. A partir de então, outras propostas foram publicadas, como a de Antonin Guttman [14] definindo a R-Tree e a de Kamel [20] definindo a Hilbert R-Tree.

Estas estruturas diferem entre si no tipo de objeto que pode ser armazenado e no conceito base utilizado para organização. Estas particularidades são descritas nas Seções 2.1, 2.2 e 2.3.

Os algoritmos de construção e busca nas árvores R-Trees são descritos na Seção 2.4. Esta foi a estrutura escolhida para implementação do índice distribuído e as definições desta seção ajudarão no entendimento do restante do texto.

A Seção 2.5 apresenta duas variações da R-Tree: A variante mais conhecida e implementada: R*-Tree e a R^0-Tree, a variante proposta mais recentemente na literatura.

A Seção 2.6 discute como é realizado o processamento de dados espaciais na arquitetura centralizada, paralela e distribuída e apresenta o desafio deste trabalho.

Por fim, a Seção 2.8 apresenta a variante da R-Tree escolhida para implementação e os motivos que nortearam esta escolha.

2.1 KD-Tree

A KD-Tree é uma árvore binária para organização de um conjunto finito de pontos em um espaço *k*-dimensional [6].

A Figura 2.1 ilustra uma KD-Tree em um espaço bidimensional. A cada nível da árvore o espaço é dividido em dois subespaços, utilizando uma reta (P1, P2, P3 e P4) perpendicular ao eixo X ou Y. A reta é posicionada na mediana dos valores do eixo X ou Y dos pontos, dividindo o espaço total em subespaços.

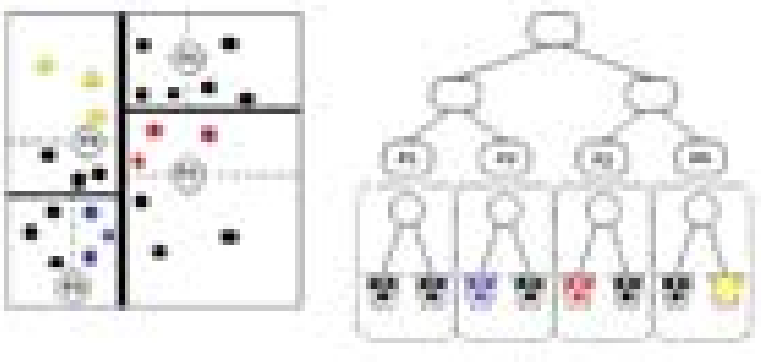

Figura 2.1: *KD-Tree de quatro níveis, com pontos separados em quadrantes à esquerda. A largura de cada linha representa o nível da divisão. À direita é demonstrada a estrutura hierárquica.*

Esta divisão é realizada recursivamente, agrupando os pontos em espaços cada vez menores, de acordo com a quantidade de níveis especificada na criação da árvore.

Um dos problemas observados na construção de uma KD-Tree é que ela depende da ordem em que os itens são inseridos. No pior caso, pode-se obter uma árvore desbalanceada, que apresenta a mesma profundidade do número de elementos inseridos [6]. Neste caso, a árvore iria se assemelhar a uma lista ligada, sendo incapaz de acelerar o processamento de consultas sobre os dados.

2.2 Hilbert R-Tree

A Hilbert R-Tree é uma variante da R-Tree para organização de objetos espaciais. É também tida como variante da B^+-Tree devido a utilização de números para ordenação dos polígonos. Para cada polígono é atribuído um número, obtido através da curva de preenchimento de espaço de Hilbert [20].

Uma Curva de Preenchimento de Espaço (*Space Filling Curve*) é uma curva que começa em um ponto, por exemplo, (0,0) e percorre todos os demais pontos de um espaço, passando por cada ponto uma só vez e utilizando um padrão recorrente. Atribuí-se um número sequencial a cada ponto visitado. Este número é o valor de Hilbert daquele ponto. A Figura 2.2 ilustra a curva de Hilbert.

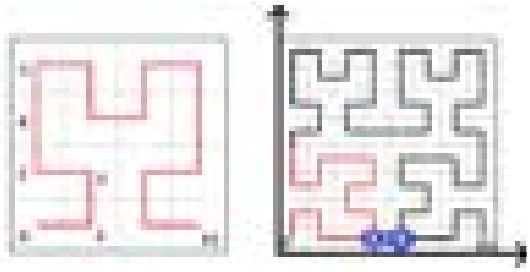

Figura 2.2: *Curva de Preenchimento de Espaço de Hilbert. Adaptado de [23].*

A Hilbert R-Tree utiliza este número para impor uma ordem aos objetos espaciais. Para ordenar objetos com múltiplos pontos, o valor de Hilbert do centro do objeto é

considerado. Dada esta ordem, os objetos são organizados na árvore da forma semelhante à organização de números em uma árvore B.

Devido à utilização do centro das geometrias para encontrar a colocalização entre as mesmas, a Hilbert R-Tree é suscetível a perda de desempenho quando a área das geometrias não é uniforme ou quando existem muitas geometrias com área grande e sobreposta [26].

Devido ao uso de Curvas de Preenchimento de Espaço, alguns objetos na Hilbert R-Tree, apesar de colocalizados, são posicionados em galhos distintos na árvore. Conforme ilustrado na Figura 2.2, o objeto espacial A recebe um valor de Hilbert igual a cinco, enquanto o objeto B tem valor de Hilbert igual a 58. Com estes valores, estes objetos ficam armazenados em galhos diferentes da árvore, apesar de estarem colocalizados. Este comportamento provoca um aumento no tempo de execução de buscas na árvore, pois é necessário visitar mais galhos para encontrar objetos.

2.3 R-Tree

Uma R-Tree é uma estrutura de dados hierárquica, assim como uma árvore B^+, que utiliza retângulos (chamados de MBRs - *Minimum Bounding Rectangle*) para organizar um conjunto dinâmico de objetos geométricos, de maneira que objetos colocalizados fiquem armazenados próximos uns dos outros [14].

Um MBR é o menor retângulo que engloba totalmente um objeto espacial, e que possui lados paralelos aos eixos x e y, conforme demonstrado na Figura 2.3. A área adicional necessária para cobrir o polígono como um todo, mas que não faz parte do mesmo, é chamada de *área morta*.

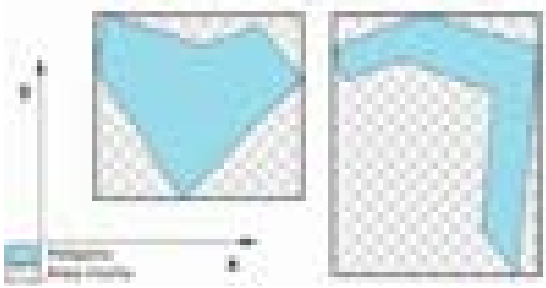

Figura 2.3: Minimum Bounding Rectangle *(MBR). Retângulo exterior, que engloba o objeto geométrico por completo e com lados paralelos aos eixos x e y. Adaptado de [18].*

Conforme ilustrado na Figura 2.4, a estrutura hierárquica da R-Tree possui um nó raiz (N_5, N_6), nós internos ($N_{1..4}$) e um último nível de nós folha ($a..j$). Na parte esquerda da figura tem-se o desenho dos MBR agrupando os objetos espaciais de a a j em subconjuntos, de acordo com sua colocalização. Na parte direita da figura, tem-se a representação da árvore R-Tree.

Os nós internos contêm espacialmente os nós dos níveis inferiores, recursivamente. Por exemplo: $N_5 \supset N_1 \supset a, b, c$. A área de cada MBR reduz-se a cada nível, conforme se observa entre N_5 e N_1. Esta organização acelera a busca de objetos na árvore, pois a cada nível percorrido, descarta-se parte do espaço de busca, assim como na B^+-Tree.

Os nós internos da R-Tree armazenam um conjunto de entradas (mbr, p), sendo *mbr* o MBR que contém espacialmente todas as entradas do nó filho f, e p um ponteiro para f.

Os nós folha armazenam um conjunto de entradas (mbr, oid), sendo *mbr* o MBR do objeto espacial e *oid* o próprio objeto espacial ou um endereço de localização do mesmo.

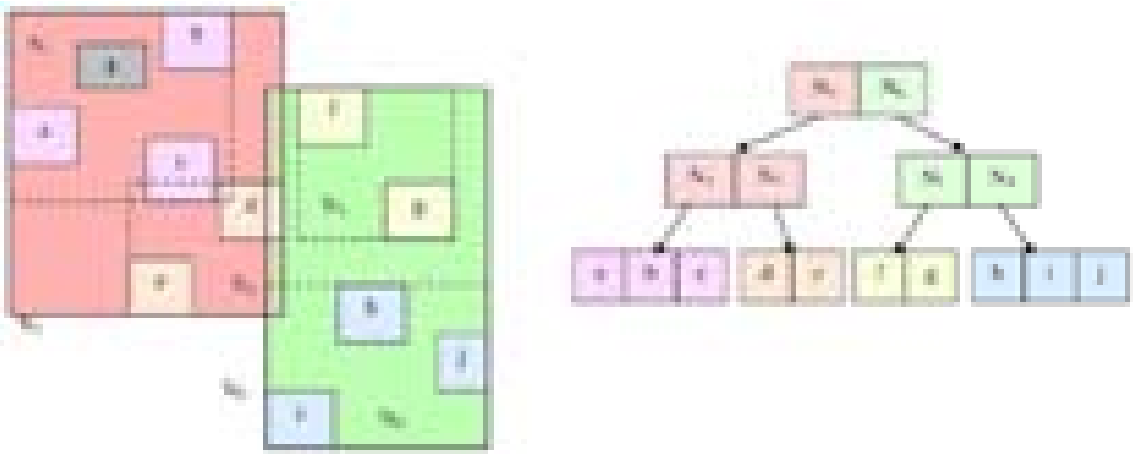

Figura 2.4: *Organização de uma R*-Tree. Adaptado de [26].*

Cada nó armazena no máximo M e no mínimo $m \leq \frac{M}{2}$ entradas [14]. O valor de M é um fator importante na organização de um índice R-Tree. Seus limites superior e inferior é que determinam a quantidade máxima e mínima de polígonos em cada bloco (nó) da R-Tree, e ainda, o grau de agrupamento de geometrias colocalizadas: valores de M muito altos tendem a agrupar geometrias não tão colocalizadas, devido ao limite inferior ser maior e obrigar que mais geometrias sejam armazenadas em cada nó. Isso provoca um aumento no MBR de cada nó da árvore, gerando mais áreas sobrepostas entre os MBRs.

A escolha deste valor pode gerar ainda mais espaço morto e áreas sobrepostas entre os MBRs, o que degrada o desempenho da operação de busca na árvore. *Espaço morto* é a área adicional do MBR necessária para cobrir o polígono como um todo e provoca o caminhamento em subárvores falsas, que não possuem objetos espaciais de resposta para uma busca. Na parte superior da Figura 2.4, a área de N1 não preenchida pelas suas entradas é um exemplo de espaço morto.

Um exemplo de sobreposição é ilustrado na Figura 2.4, entre os nós N_5 e N_6. A sobreposição é a área de interseção entre eles. Ao contrário de árvores B que não possuem este comportamento devido aos dados serem unidimensionais, em árvores R a redução desta sobreposição é bastante estudada. A redução de áreas sobrepostas evita que várias subárvores sejam acessadas durante o caminhamento na estrutura. Na Figura 2.4, a área de sobreposição entre N_5 e N_6 pode obrigar o algoritmo a processar as duas subárvores, o que degrada o desempenho da R-Tree [14, 5].

2.4 Termos e Algoritmos Utilizados na R-Tree

As principais operações utilizadas na construção de uma árvore R-Tree são descritas a seguir:

- **Inserção**: Algoritmo que faz a inserção de objetos na árvore no nível das folhas. Utiliza o algoritmo de escolha da subárvore para decidir onde será inserido o item. Durante sua execução encontra situações onde é necessário fazer o ajuste da árvore (nós internos), realizando divisão de nós e tratamento de *overflow*, conforme descrito a seguir. A inserção em uma R-Tree é similar a inserção em uma B-Tree. O algoritmo é executado em quatro etapas: *i*) É realizada uma escolha de sub-árvore para encontrar o nó folha F onde o novo item n será incluido; *ii*) Se F possui uma entrada livre para acomodar o item n, este é então inserido. Caso contrário, o algoritmo de *divisão de nó* é executado para dividir F em dois nós folha; *iii*) As alterações realizadas (ajuste de MBR devido à inclusão do item, ou o novo nó criado devido à divisão) são notificadas para os nós superiores da árvore, para que os devidos ajustes sejam realizados; *iv*) Se a divisão de nós é propagada até o nível raiz, causando sua divisão, um novo nó raiz é criado com duas entradas correspondentes aos dois nós resultantes da divisão.
- **Escolha da subárvore**: Algoritmo utilizado para encontrar uma subárvore na qual um objeto pode ser inserido ou excluído. Percorre recursivamente a estrutura da árvore, começando pela raiz e descartando partes do espaço de busca até encontrar o nó adequado para a inserção ou exclusão. A seleção das subárvores a serem percorridas é feita através de comparações geométricas entre o MBR de cada nó interno e o MBR do objeto sendo inserido/excluído. Uma subárvore é descartada quando não há interseção entre esses MBRs.
- **Ajuste da árvore: MBR**: Sempre que uma entrada é adicionada em um nó folha, os MBRs dos nós internos da subárvore percorrida devem ser atualizados, de maneira *bottom-up*, para que buscas posteriores encontrem o objeto sendo inserido. A cada nível superior, o novo MBR do nó inferior é comparado com o MBR do nó corrente e ajustado. Se é necessário expandí-lo, o algoritmo continua recursivamente, até que o ajuste não seja mais necessário, ou até que o nó raiz seja ajustado.
- **Divisão de nó**: É o processo de dividir um nó, cuja capacidade M foi ultrapassada, em dois outros, cada qual com uma parte das entradas do nó original. Um algoritmo é empregado para alocar cada entrada em um dos dois nós resultantes. Esta alocação é feita de forma a minimizar o MBR de cada nó resultante e possivelmente evitar sobreposição entre os mesmos.
- **Ajuste da árvore: Divisão**: É o ajuste da árvore realizado após uma divisão de nó folha. Neste ajuste são adicionadas entradas nos nós internos da árvore,

recursivamente, até que, a partir da raiz, seja possível encontrar o nó adicionado. O algoritmo é executado da seguinte forma: Seja A o nó antigo e N o novo nó, resultantes da divisão e com os itens distribuídos entre eles: o algoritmo verifica se há uma entrada livre em P (no nó pai de A): Se existir, uma nova entrada é adicionada, apontando para N e o MBR de P é ajustado. Este ajuste é então propagado para os níveis superiores, executando o algoritmo Ajuste de MBR; Se não existe entrada livre em P, uma nova divisão é executada em P e o processo de ajuste se repete, recursivamente, considerando A e N como os novos nós resultantes da divisão de P.

Após a construção da R-Tree, outros algoritmos são utilizados para realizar buscas de objetos na árvore. Estes algoritmos percorrem a árvore recursivamente – de forma semelhante ao que ocorre na operação escolha da subárvore – comparando características dos objetos com um filtro informado e retornando os que o atendem. As principais buscas são descritas a seguir:

- **Vizinhos Próximos (*Nearest Neighbor*)**: Esta consulta retorna os vizinhos mais próximos de um ponto, dado um ponto p e um raio r. Todos os objetos que estão distantes deste ponto p em no máximo o tamanho do raio r são retornados [32].
- **Busca de Janela (*Range/Window Query*)**: É um dos principais algoritmos de busca na R-Tree. Seu objetivo é encontrar todos os objetos espaciais dentro de uma determinada área, que é especificada através de uma janela (um retângulo r). O algoritmo inicia pelo nó raiz e percorre a árvore comparando a janela com o MBR de cada entrada dos nós internos, seguindo nos ramos onde existe interseção. Nas folhas, a janela de consulta é comparada com cada objeto espacial do nó. São retornados os itens da folha que contém interseção com a janela de consulta.
  [23].
- **Joins**: Um *join* espacial consiste em correlacionar entradas de dois conjuntos de objetos multidimensionais, R e S, no espaço Euclidiano, criando pares de objetos $\{(r,s)|r \in R, s \in S\}$, de acordo com algum critério de filtro. Normalmente, este filtro é a colocalização espacial entre os objetos dos conjuntos. Este algoritmo foi proposto por [9] e posteriormente estendido por [16]. A utilidade de consultas *joins* pode ser evidenciada através do seguinte exemplo: Duas R-Trees, uma mapeando Áreas de Preservação Ambiental (em escala global) e outra mapeando a posição geográfica de onde foram encontrados animais de determinadas espécies. Um *join* entre estas duas relações responde a seguinte pergunta: "Quais animais estão presentes em cada Área de Preservação?"
- **Skyline Query**: Dado um conjunto de objetos espaciais, a consulta Skyline retorna todos os objetos que dominam os demais, observando seus aspectos e dimensões.

Um objeto p_1 domina outro objeto p_2, se $p_1 >= p_2$ em todas as dimensões e $p_1 > p_2$ em uma delas [41]. Como exemplo desta consulta pode-se citar um conjunto de hotéis próximos a uma praia. Cada hotel está a uma determinada distância da praia e possui um preço de diária. A consulta retorna os hotéis que estão mais próximos da praia, com menor preço de diária.

2.5 Extensões da R-Tree

Desde a sua publicação inicial [14], muitas extensões da R-Tree foram propostas [26]. Duas delas são a R*-Tree, implementada por bancos de dados espaciais [2], e a R^0-Tree [41].

A R*-Tree é uma extensão da R-Tree que propõe mecanismos para melhorar o tempo de busca na estrutura de dados [5]. Estes mecanismos são:

1. Redução da sobreposição entre os MBRs, que faz com que as buscas na árvore percorram menos nós falsos, ou seja, que não contêm entradas para compor o resultado das consultas (falsos positivos);
2. Minimização do espaço morto - área do MBR que não é coberta pelos objetos filhos - para evitar que ocorra a sobreposição;
3. Minimização do perímetro do MBR, para deixar o retângulo do MBR o mais quadrado possível, também para reduzir a sobreposição de MBRs;
4. Reinserção de objetos quando ocorre *overflow* em um nó da árvore. Esta técnica escolhe itens menos colocalizados com os demais no nó cheio e tenta uma reinserção dos mesmos na árvore, procurando locais onde eles sejam mais colocalizados e haja espaço para a inserção. Caso isso ocorra, libera-se espaço para a acomodação do item que causaria a divisão, evitando-a.

A R*-Tree é normalmente utilizada como comparativo de outras variações por ter um excelente desempenho. É a estrutura implementada atualmente em extensões espaciais de bancos de dados relacionais como PostgreSQL (Índice GIST), Oracle e MySQL.

A R^0-Tree é uma extensão da R*-Tree. Propõe uma técnica de armazenamento de objetos discrepantes nos nós internos da árvore que reduz consideravelmente o MBR dos nós da árvore. Esta redução tem impacto direto no desempenho das buscas porque resulta em uma menor sobreposição entre MBRs. Reduzir a sobreposição implica em uma maior redução no espaço de busca [41].

2.6 Processamento de Dados Espaciais Centralizado, Paralelo e Distribuído

O foco da pesquisa para processamento de dados espaciais, até meados de 1990, foi a melhoria dos métodos de organização dos dados e redução na complexidade dos passos dos algoritmos para serem executados mais eficientemente em arquitetura centralizadas, com um único processador.

Desde então, o interesse na adequação desses algoritmos para execução em arquiteturas paralelas tem aumentado devido *i*) a demanda por sistemas escaláveis que atendam uma maior quantidade de requisições simultaneamente; *ii*) a necessidade de redução do tempo de reposta de consultas espaciais e *iii*) a maior disponibilidade de *hardware* com componentes paralelos (e.g., multinúcleo).

Alguns métodos diferentes de paralelismo são possíveis, em relação ao compartilhamento de recursos nas arquiteturas paralelas. Uma taxonomia foi proposta em [34]:

- *shared-memory* ou *shared-everything*: Todos os processadores acessam uma memória global, comum a todos, e possuem acessos a todos os discos;
- *shared-disk*: Cada processador tem sua memória privada, mas possui acesso a todos os discos;
- *shared-nothing*: Cada processador tem sua memória e discos privados, e a comunicação entre eles é realizada através de uma rede.

De acordo com [12], as duas primeiras arquiteturas não escalam bem para o processamento de dados devido à concorrência dos processadores pelos recursos compartilhados da arquitetura. É necessário que os recursos compartilhados (memória ou disco) tenham um desempenho compatível com a soma de processadores da arquitetura. Máquinas com estas características, apesar de existirem atualmente (*mainframes*), são mais difíceis e mais caras de se construir, e a complexidade de construí-las aumenta de forma não linear conforme o aumento da quantidade de unidades processadoras.

Ao contrário das duas primeiras, a arquitetura *shared-nothing* não sofre com os problemas de compartilhamento citados, pois nem a memória nem os discos são compartilhados entre as unidades processadoras. Porém, a rede de comunicação entre as máquinas passa a ser o gargalo da arquitetura, cujo uso deve ser controlado e reduzido. Os recentes avanços na criação de redes de alto desempenho e de custo acessível têm mudado este cenário[1].

[1] Em 1995 foi publicada a especificação de redes ethernet 100 Mbit/seg pelo IEEE. Até então, o padrão de redes era de 10 Mbit/seg. Desde então, surgiram redes mais avançadas com vazão de 1 a 10Gbit/seg. Há ainda redes de 100 Gbit/seg sendo avaliadas em cenários específicos.

Algumas estratégias de processamento paralelo de dados foram propostas na literatura para estas arquiteturas. Uma delas é a utilização de *pipeline* de operadores (por exemplo, *select, sort, agregate*), na qual cada operador é executado por uma unidade processadora. Porém, de acordo com [12], vários problemas são difíceis de controlar nesta abordagem, principalmente em relação a não uniformidade do custo computacional entre os operadores.

Os autores em [12] afirmam que a criação de partições dos dados, ou seja, a divisão dos dados em partes menores, juntamente com a distribuição destas partições para as unidades processadoras (escalonamento), é a forma que possibilita as melhores oportunidades de processamento paralelo.

O trabalho em [12] ainda sugere que este particionamento dos dados pode ser realizado de formas diferentes, como ilustrado na Figura 2.5. Cada item do conjunto de dados pode ser alocado em uma partição, de acordo com a definição de intervalos (*range partitioning*), alocando um item em cada partição sequencialmente (*round-robin*) ou alocando cada item através de uma função *hash* que determina a distribuição (*hashing*).

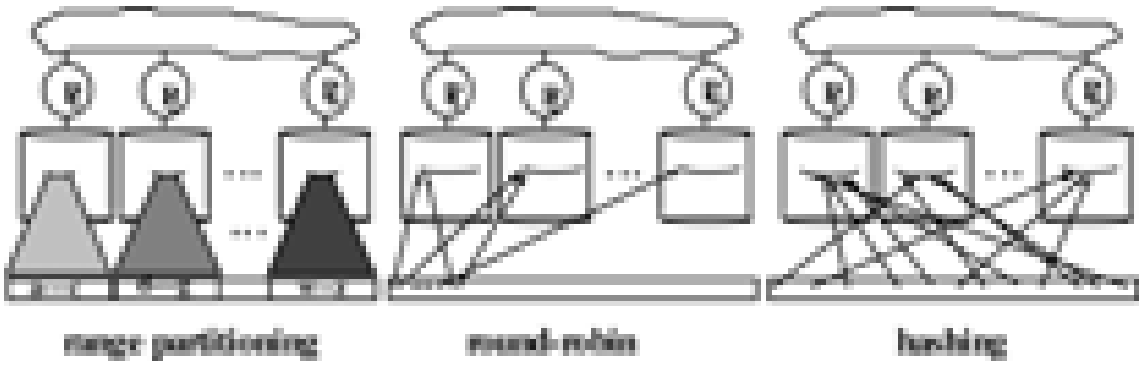

Figura 2.5: *Particionamentos possíveis para dados unidimensionais [12].*

No entanto, estas formas de particionamento parecem ser mais adequadas para dados unidimensionais. Para o caso de dados espaciais, multidimensionais, formas mais adequadas necessitam ser utilizadas, conforme foi discutido por [19], para arquiteturas *shared-disk*. Para arquiteturas *shared-nothing*, a Subseção 2.7 descreve uma abordagem geral de particionamento. Detalhes a respeito do particionamento nesta arquitetura são descritos nos Capítulos 3 e 6.

O termo *arquitetura paralela* foi utilizado no restante do texto para se referir à arquitetura *shared-everything/disk*, e o termo *arquitetura distribuída* ou somente *cluster* foi empregado para se referir à arquitetura *shared-nothing*.

2.7 Particionamento da R-Tree em Arquiteturas *Shared-Nothing*

Para arquiteturas *shared-nothing*, a forma de particionamento e distribuição das partições foi estudada por alguns artigos [23, 33, 4, 28].

De forma geral, a distribuição dos dados espaciais de um dataset considera a colocalização entre os polígonos de forma a acelerar o posterior processamento de algoritmos de busca na estrutura, ao invés da simples divisão dos dados efetuada para dados unidimensionais ilustrada na Figura 2.5.

A Figura 2.6 ilustra a estrutura lógica geral de uma R-Tree distribuída em um *cluster* de computadores. O particionamento dos dados é realizado agrupando os nós da árvore em servidores do *cluster*, de forma indexada segundo a estrutura lógica de uma R-Tree. As linhas ligando os nós na figura representam a troca de mensagens necessária para alcançar as subárvores durante o processamento dos algoritmos.

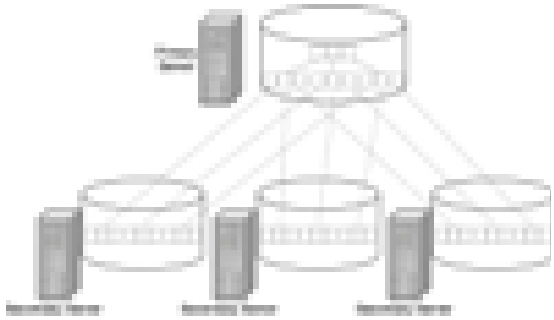

Figura 2.6: *Particionamento de uma R-Tree em arquitetura* shared-nothing *[26].*

Há abordagens, mais utilizadas em datasets estáticos, que constroem a árvore de forma *bottom-up* (das folhas para os nós internos) e outras que constroem a árvore de forma *top-down*, do mesmo modo que a árvore R-Tree centralizada é construída com todas as geometrias dos datasets inseridas através do nó raiz.

Os algoritmos de inserção (e subalgoritmos de ajustes) e as buscas efetuadas na R-Tree distribuída podem ser executados de forma semelhante ao que é realizado na R-Tree centralizada, apenas considerando a troca de mensagens para acessar as partições distribuídas. Porém, os algoritmos na R-Tree distribuída são executados de forma paralela nos servidores e alguns detalhes de concorrência devem ser tratados, conforme será discutido no Capítulo 3.

2.8 Considerações Finais

Neste trabalho, a R*-Tree [5] foi escolhida para implementação distribuída. A KD-Tree, além do problema de desempenho citado na Seção 2.1, é utilizada para organização de pontos espaciais, e neste trabalho pretende-se utilizar objetos geométricos diversos, que representem regiões geográficas como cidades, rios, desmatamentos, dentre outros, que, por natureza, possuem muitos pontos.

A Hilbert R-Tree apresenta os dois problemas discutidos na Seção 2.2, em relação à utilização de Curva de Preenchimento de Espaço e a perda de desempenho com geometrias de área não uniformes. Esta estrutura é mais utilizada para indexação de datasets estáticos conforme foi observado na literatura pesquisada e, no entanto, pretendemos trabalhar com datasets dinâmicos.

A R*-Tree utiliza MBRs para organizar os objetos, forçando que objetos colocalizados fiquem no mesmo galho da árvore, ao contrário da Hilbert R-Tree. É também uma árvore extensamente pesquisada na literatura, conforme nota-se em [26], e implementada em sistemas comerciais de bancos de dados geoespaciais. Pode ser utilizada na organização e indexação de objetos espaciais formados por pontos, linhas e polígonos e é normalmente utilizada para comparações com novas variantes, como em [41], devido a seu reconhecido desempenho.

Apesar dos bons resultados publicados na variante R^0-Tree [41], seu uso foi desconsiderado apenas pela maior complexidade de implementação. As técnicas propostas pelo artigo devem ser consideradas em trabalhos futuros.

No capítulo seguinte é descrita a arquitetura do índice distribuído implementado. Considerando alguns aspectos de propostas de outros índices distribuídos na literatura, criou-se o DSI-RTree, um Índice Espacial Distribuído (*Distributed Spatial Index*) com base na estrutura de dados R*-Tree.

CAPÍTULO 3

Arquitetura e Implementação do DSI-RTree

Este capítulo apresenta a arquitetura e os detalhes de implementação do DSI-RTree. As seções seguintes apresentam os desafios na construção de um sistema distribuído de indexação de dados espaciais e os argumentos para as decisões que foram tomadas diante deles.

A Seção 3.1 discute tais desafios com base em propostas correlatas existentes na literatura, que também discutem a divisão dos dados em partições e a organização destas partições. A seguir, a Seção 3.2 apresenta a arquitetura em camadas proposta, descrevendo os componentes de cada camada e justificando seu propósito.

A Seção 3.3 detalha como é realizado o armazenamento das partições do índice nas máquinas do *cluster*, comparando este armazenamento com os índices centralizados[1] baseados em R-Trees. Os detalhes de implementação relacionados à concorrência são apresentados na Seção 3.4, incluindo as modificações realizadas no algoritmo da R-Tree.

A Seção 3.6 discute como é realizado o processo de escolha do local de armazenamento de cada partição e os impactos advindos desta escolha. Por fim, a Seção 3.7 apresenta outros detalhes de implementação, seguida da Seção 3.8, que apresenta considerações a respeito da arquitetura proposta.

3.1 Desafios da Construção de Índices Espaciais Distribuídos

O interesse pela utilização de *clusters* de computadores para solução de problemas computacionais complexos tem aumentado, devido, principalmente, a seu custo reduzido em relação à máquinas paralelas de alto desempenho. Diante deste interesse, quais são os desafios de implementar algoritmos eficientes, que consigam processar da-

[1] O termo *centralizados* foi utilizado no texto para se referir a implementação de índices baseados na R-Tree não paralelos, e que não utilizam comunicação através de rede.

dos espaciais de forma distribuída e escalável, aproveitando o máximo possível do poder de processamento paralelo e a elasticidade desta arquitetura computacional?

Há na literatura algumas propostas de índices R-Tree distribuídos para *cluster* de computadores [4, 23, 28, 33], apesar de não haver um consenso entre elas a respeito da definição de uma arquitetura de um índice distribuído para dados espaciais.

Estudando tais propostas, observa-se que dois desafios principais são comuns entre elas: *i*) A divisão dos dados em partições, para distribuir os dados entre as máquinas do *cluster* e *ii*) o escalonamento destas partições, que decide em que máquina cada uma delas será armazenada.

A divisão dos dados em partições, apesar de ser o principal fator que proporciona o processamento paralelo dos algoritmos, também causa o principal desafio da construção da arquitetura distribuída: a comunicação necessária entre as máquinas que armazenam as partições.

Para reduzir o impacto da comunicação no tempo de resposta das consultas é utilizado um mecanismo de escalonamento das partições entre as máquinas do *cluster*, organizando-as de forma a reduzir a necessidade de troca de mensagens. No entanto, se este mecanismo tenta reduzir ao máximo esta troca de mensagens, agrupando muitas partições em uma só máquina, provoca a redução de uso do poder de processamento paralelo do *cluster*, que é um dos diferenciais do uso de arquiteturas distribuídas.

Neste trabalho foi projetada e implementada uma arquitetura distribuída para o algoritmo da R-Tree [14], com as alterações proposta para a R*-Tree [5], exceto a técnica de reinserção. Uma única adequação em relação ao ajuste de MBRs nos nós da estrutura foi realizada, com o objetivo de adaptá-la a arquitetura distribuída. Tal adaptação tem relação com a redução da troca de mensagens e organização do índice durante a sua construção em paralelo e é discutida na Seção 3.4.

3.2 Visão Geral da Arquitetura

Conforme a taxonomia definida em [4], o índice distribuído foi contruído com as seguintes características:

- **Unidade de alocação**: Bloco – para cada nó da R-Tree é criada uma partição;
- **Frequência de alocação**: *Overflow* – novas partições são criadas durante a inserção quando há uma divisão de um nó da árvore;
- **Política de distribuição**: Balanceamento – as partições são distribuídas pelas máquinas do *cluster* de forma a manter o balanceamento dos dados.

Uma visão geral da arquitetura distribuída é ilustrada na Figura 3.1. Existem três camadas na arquitetura: *i*) a Camada de Aplicações, *ii*) camada de Armazenamento e Processamento e *iii*) a Camada de Comunicação.

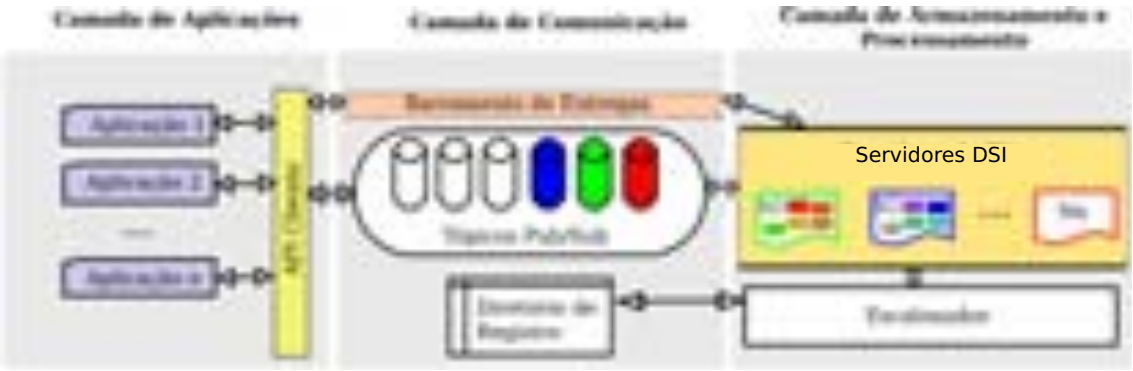

Figura 3.1: *Visão Geral da Arquitetura do DSI-RTree*

A *Camada de Aplicações* é composta pela API Cliente e Aplicações que usam o serviço. A API Cliente tem a função de abstrair os detalhes de conexão e comunicação com as demais camadas, e, com isso, facilitar a implementação das aplicações que utilizam índices distribuídos para prover serviços de valor agregado ao usuário final. Basicamente, a API disponibiliza métodos para inserção de datasets espaciais de forma indexada no *cluster* de computadores e consultas nos dados armazenados no serviço, sem que cada aplicação tenha que lidar com detalhes da distribuição.

Na *Camada de Armazenamento e Processamento* estão os componentes distribuídos, que implementam o armazenamento, o processamento e a gerência da distribuição dos dados. É composta pelos Servidores DSI (N1, N2.. Nn) e o Escalonador de Partições.

Os Servidores DSI ficam distribuídos nas máquinas do *cluster* – um em cada máquina – e são os responsáveis pelo armazenamento e processamento dos dados geoespaciais. Neles são armazenadas as partições do índice e executados os algoritmos da R-Tree.

A decisão do local de armazenamento das partições é efetuada pelo componente *Escalonador*. Sempre que ocorre um *overflow* em uma partição durante a construção do índice, o Servidor DSI que armazena a partição cheia consulta o Escalonador para determinar qual será o local de armazenamento da nova partição criada.

A *Camada de Comunicação* é a camada que *i*) intermedeia as requisições da API Cliente, repassando-as para os Servidores DSI adequados, *ii*) provê um mecanismo de comunicação entre os Servidores DSI e *iii*) disponibiliza o mecanismo de comunicação necessário para a entrega distribuída do resultado do processamento das consultas requisitadas pela API Cliente. É implementada por um *Middleware* Orientado a Mensagens (MOM - *Message Oriented Middleware*), que mantém canais de comunicação com

cada um dos Servidores DSI, chamados de tópicos. Cada tópico é utilizado como canal de comunicação direto entre um servidor específico e os demais componentes da arquitetura.

O envio de mensagens através dos tópicos é feito através do mecanismo de Subscrição e Publicação (*Publish/Subscribe*). De forma geral, para cada Servidor DSI N_n, existe um tópico T_n no MOM. Outro Servidor DSI, interessado em enviar uma mensagem para o servidor N_n, publica uma mensagem no tópico T_n e esta é encaminhada para N_n pelo *middleware*. Para receber as mensagens, o Servidor DSI N_n deve registrar interesse nas mensagens do tópico T_n através de uma subscrição no mesmo, no momento de sua inicialização.

O MOM facilita o desacoplamento entre a implementação dos algoritmos da R-Tree e a comunicação necessária entre os Servidores DSI. O Servidor DSI não necessita lidar diretamente com a responsabilidade da entrega da mensagem ao destinatário. Ele simplesmente publica a mensagem de forma assíncrona e continua suas atividades. O *Middleware* assume a responsabilidade pelo envio da mensagem, que continuará o processamento do algoritmo em outra máquina do *cluster*.

Com o desacoplamento da comunicação, os algoritmos que processam as estruturas de dados podem ser implementados da mesma forma que foram definidos na literatura para R-Trees centralizadas, com a única diferença do envio de mensagem quando é necessário continuar a execução a partir de outro ramo, que porventura está armazenado em outro Servidor DSI. Enquanto numa R-Tree centralizada tem-se ponteiros que direcionam o algoritmo diretamente para as áreas de memória onde estão localizados os ramos do índice necessários, na implementação distribuída utiliza-se o envio de mensagens para realizar este mesmo propósito.

Além da comunicação entre os Servidores DSI, existem outros dois componentes na Camada de Comunicação: o Diretório de Registro e o Barramento de Entregas. O Diretório de Registro contém a lista de Servidores DSI conectados ao serviço, a qual é consultada pelo Escalonador para determinar quais Servidores DSI estão disponíveis para armazenamento de partições. O Barramento de Entregas é o canal de comunicação para a entrega de resultados de consultas requisitadas pela API Cliente. Por ele trafegam os polígonos dos datasets que coincidem com as buscas requisitadas.

3.3 Armazenamento de Partições no Servidor DSI

Cada Servidor DSI armazena uma ou mais partições, de um ou vários datasets distribuídos, como ilustrado na Figura 3.2. Por exemplo, o Servidor DSI N1 armazena as partições do dataset *a* e do dataset *b* representadas pela cor dos retângulos e indicadas

pelas linhas pontilhadas[2]. Correlacionando a cor da partição com os nós na representação dos dois índices R-Tree, percebe-se que vários nós de níveis e datasets diferentes podem ser armazenados no mesmo servidor.

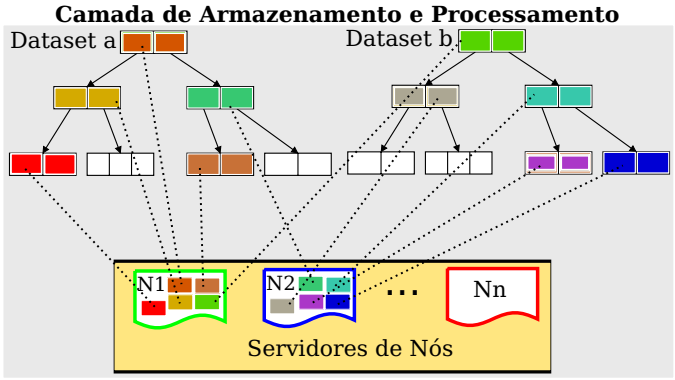

Figura 3.2: *Comparação da organização de partições na arquitetura proposta com a representação de uma R-Tree.*

Considerando a maior disponibilidade de memória primária (RAM - *Random Access Memory*) em *clusters*, não foi criado um mecanismo de paginação dos dados espaciais. Para diminuir o tempo de acesso a disco, o DSI-RTree processa todos os dados inseridos em memória RAM. Cada Servidor DSI armazena as partições no disco rígido apenas para recuperar o estado do sistema, caso haja necessidade de desativá-lo, por exemplo, em caso de manutenção do *cluster*. As partições armazenadas em disco são completamente carregadas para a memória RAM durante a inicialização do sistema.

De certa forma, esta decisão possibilita evitar as latências de leitura em disco, comuns em sistemas centralizados, principalmente em máquinas de 32 bits que possuem capacidade de endereçamento de memória limitada (4GB). No entanto, esta decisão arquitetural pode se tornar um problema para aplicações que necessitam armazenar grandes ou muitos datasets, caso em que uma solução mais adequada deveria ser considerada. Porém, a necessidade de armazenamento de muitos datasets geralmente está acompanhada da necessidade de desempenho no processamento das consultas, e neste caso é mais adequado que seja efetuado o investimento em memória RAM para o *cluster*.

Entende-se ainda, para o DSI-RTree, que o tamanho do dataset diz respeito apenas aos objetos geográficos indexados. Os dados associados aos objetos geográficos podem ser armazenados em outros sistemas específicos para este propósito, quando o armazenamento dos mesmos no índice não for possível devido aos limites de memória RAM do *cluster*. Os mesmos podem ser acessados neste outro sistema a partir de um endereço armazenado juntamente com a geometria no índice distribuído.

[2]Como detalhado anteriormente, na unidade de alocação de bloco, cada partição corresponde a um nó do índice R-Tree.

3.4 Concorrência na Construção do Índice Distribuído

Em uma arquitetura distribuída para indexação de dados espaciais, é fundamental melhorar o desempenho da inserção dos dados de forma a reduzir o tempo de construção do índice. Uma forma de alcançar este objetivo é realizando a inserção das geometrias dos datasets de forma paralela entre as máquinas do *cluster*.

Utilizando o mesmo algoritmo de inserção da R-Tree centralizada, pode-se realizar uma inserção em paralelo das geometrias, formando um *pipeline* de processamento: Todos os itens do dataset são inseridos através do nó raiz do índice, que realiza os ajustes necessários em sua estrutura e envia mensagens para que os outros servidores do *cluster* façam os ajustes nas subárvores do índice. Logo após o envio da mensagem, o nó raiz pode continuar o processamento das próximas geometrias, sem aguardar o ajuste das subárvores.

No entanto, os algoritmos de construção da R-Tree necessitam realizar ajustes nos níveis superiores, de acordo com as mudanças na estrutura que ocorrem nos níveis inferiores. É o caso do algoritmo de divisão de nós que ocorre durante o processo de inserção. Quando um nó no nível inferior do índice é dividido, é necessário comunicar o nó do nível superior, que o contém, a respeito das modificações ocorridas em seu MBR e da nova partição resultante da divisão.

Conforme é possível notar e também já abordado em alguns artigos a respeito de concorrência em estruturas R-Tree [11, 21], o nó do nível superior não deve processar outras requisições na estrutura porque pode tomar decisões sobre um estado da estrutura do índice que não é mais válido. O estado no nível superior é sempre inválido no intervalo entre a modificação da estrutura nos níveis inferiores e a chegada definitiva da mensagem de notificação destas mudanças. Este fato obriga a utilização de bloqueios distribuídos na arquitetura.

Neste trabalho foi implementada a técnica de *lock coupling* conforme descrita em [11] e ilustrada na Figura 3.3. Durante a inserção de uma geometria no índice, sempre que a raiz de qualquer subárvore está totalmente preenchida, um bloqueio é colocado em seu nó pai. Na Figura 3.3a este bloqueio é representado pela letra *w* dentro da circunferência com fundo branco. Toda a subárvore marcada é processada sequencialmente, realizando os ajustes necessários de forma *bottom-up*, da mesma forma realizada pelo algoritmo de divisão das R-Trees. Os bloqueios são removidos à medida que os ajustes vão sendo realizados, até al-

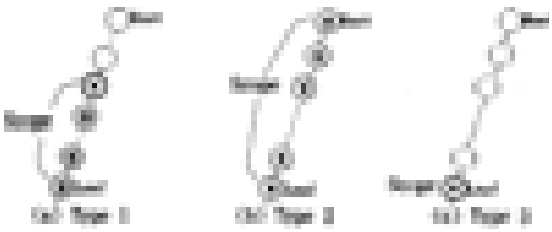

Figura 3.3: *Escopo de Bloqueio na técnica* Lock Coupling *[11].*

cançar o nó raiz da subárvore. A construção em paralelo continua após todos os bloqueios serem removidos.

No entanto o escopo do bloqueio não contempla toda a árvore na Figura 3.3a. A parte superior continua a construção em paralelo, apesar de haver uma subárvore sendo processada sequencialmente. A Figura 3.3b ilustra uma situação na qual a raiz da árvore e subárvore estão totalmente preenchidos. Esta situação ocorre quando a árvore está prestes a criar um novo nível. A Figura 3.3c ilustra a situação onde apenas um nó folha está cheio e a estrutura continua sendo processada em paralelo quase totalmente.

3.5 Ajuste do MBR *Top-Down*

Uma modificação foi realizada no algoritmo de inserção em relação ao ajuste de MBRs executado quando um item é inserido na estrutura. No algoritmo original da R-Tree este ajuste é realizado depois que o item é acomodado em um nó folha, de forma *bottom-up* na estrutura.

Em nossa implementação, este ajuste foi realizado no momento da escolha da subárvore na qual o item será inserido (*top-down*), conforme discutido por [11]. Para plataforma distribuída, esta forma de ajuste é ainda mais importante do que em plataformas paralelas, pois, além de evitar o bloqueio que seria necessário para efetuar este ajuste em todo o ramos acessado, não é necessário o envio de mensagens para notificar os ajustes nos níveis superiores.

Ajustando o MBR antecipadamente, o Servidor DSI pode tomar as mesmas decisões que tomaria se fosse efetuado o bloqueio, no que diz respeito à escolha de subárvores para as próximas mensagens a serem processadas. Isto reduz a quantidade de mensagens na rede e facilita o processo de construção de um índice espacial distribuído.

3.6 Escalonador de Partições

Como discutido anteriormente, o Escalonador é um componente determinante na quantidade de mensagens trocadas na arquitetura, e consequentemente, no desempenho do processamento de requisições. Este componente deve ser implementado de forma que consiga balancear a carga pelo *cluster* através da correta distribuição das partições. Porém, podem ser necessários esquemas de distribuição de partições diferentes, de acordo com o tipo de consulta mais executado no índice.

Para buscas com janelas grandes, por exemplo, que interceptam grande parte dos datasets, o ideal é que as partições estejam bem distribuídas para que a consulta acesse a maior quantidade possível de máquinas, e reduza o tempo de reposta da consulta. Para consultas pequenas, o contrário é mais adequado, já que acessando menos máquinas

reduz-se a necessidade de comunicação [4]. Como ilustrado na Figura 3.4, o dataset (a) pode ser dividido em partições, representadas pelos retângulos preenchidos em (b), que estão armazenadas distribuídas de acordo com a legenda. Ambas as janelas estão acessando três máquinas apesar de serem de tamanhos bem diferentes.

Considerando ainda a existência de outros tipos de consulta sobre dados espaciais, como *joins*, que podem necessitar que sejam comparadas desde uma única partição até todas as partições dos dois índices[3], e outras consultas como a *skyline query*, nota-se que é necessário que o Escalonador seja um componente flexível na arquitetura.

Figura 3.4: *Acesso a Máquinas de acordo com o agrupamento dos itens/tamanho das partições [23].*

Sendo um componente flexível, outras formas de escalonamento podem ser implementadas e conectadas na arquitetura, conforme a necessidade das aplicações. Pode-se por exemplo implementar escalonadores que agrupem nós colocalizados na mesma máquina do *cluster* e acelerar o processamento de consultas *join*, ou ainda, realizar experimentos científicos com outras formas de escalonamento de propósito geral.

Neste trabalho foi implementado um escalonador com o propósito de processamento de consultas de janela seletivas, utilizando o método de distribuição de partições *Round-Robin*. A implementação foi realizada de forma distribuída, ou seja, cada Servidor DSI possui uma instância do escalonador, evitando que *i*) haja um ponto único de falha na arquitetura e *ii*) que ocorra o acesso sequencial ao mesmo pelas várias instâncias de Servidores DSI.

O Escalonador é executado por *overflow*. Quando o algoritmo de inserção verifica que a quantidade de itens no nó é igual a M (ou seja, não há como incluir outro item) a partição é dividida, criando uma nova. Para decidir onde colocar esta nova partição, o Escalonador executa os seguintes passos:

1. Obtém a lista atualizada de servidores disponíveis no Diretório de Registro de Servidores DSI;
2. Encontra o servidor onde a nova partição será armazenada, observando a posição do último servidor utilizado e considerando que a lista é circular;
3. Atualiza o registro do último servidor utilizado para a próxima chamada.

[3]Fato não tão raro, já que o *join* é naturalmente utilizado para encontrar itens colocalizados de dois datasets normalmente sobrepostos.

Para manter um comportamento similar ao de um escalonador centralizado, cada escalonador distribuído, após obter a lista de servidores disponíveis do Diretório de Registro pela primeira vez, encontra a sua própria posição nesta lista, e começa a iterar a partir desta posição. A primeira partição escalonada é colocada no próprio Servidor DSI que solicitou o escalonamento. Desta maneira, mesmo existindo várias instâncias do escalonador, as partições continuam balanceadas entre as máquinas do *cluster*, de forma semelhante ao que aconteceria se existisse uma só instância do mesmo.

O algoritmo *Round-Robin* foi escolhido por ser facilmente implementado em uma infraestrutura distribuída e apresenta bons resultados para consultas de janela seletivas, como demonstrado na seção 5. Entretanto, não é uma boa opção para operações de join espacial entre datasets diferentes, necessária em muitas aplicações GIS.

O escalonador *Round-Robin* não leva em consideração a colocalização de objetos espaciais entre dois ou mais datasets. Isso permite que objetos espaciais de datasets diferentes, porém relacionados, fiquem em máquinas diferentes, gerando muita troca de mensagem para determinar a relação. O Escalonador é um componente flexível e possibilita que novos algoritmos sejam implementados para atender a esta necessidade.

3.7 Outros Detalhes da Implementação

No desenvolvimento do DSI-RTree foi utilizado a linguagem Java e o ambiente de desenvolvimento Eclipse. A escolha da linguagem foi baseada principalmente na existência de bibliotecas para manipulação de dados espaciais de fácil utilização e baseadas em padrões OGC (*Open Geoespatial Consortium*) [17], além da maior quantidade de recursos nativos na linguagem para se trabalhar com comunicação e paralelismo em sistemas distribuídos. As bibliotecas utilizadas foram:

- *JTS - Java Topology Suite*: uma biblioteca para representação de objetos espaciais, robusta e totalmente implementada na linguagem Java;
- *Geotools*: Um conjunto de ferramentas e classes para processamento e visualização de dados geoespaciais, que expande a JTS;
- *Zookeeper*: Coordenador de sistemas distribuídos, também empregado para troca de mensagens entre as máquinas do *cluster*;
- *JUnit*: Para realização de testes unitários no sistema;

Além do paralelismo da arquitetura, descrito na Seção 3.4, internamente, em cada Servidor DSI, foram utilizadas filas de mensagens para cada uma das partições armazenadas. Cada uma destas filas recebe tarefas que devem ser executadas pelas partições, e são processadas por *Thread pools*. Os *Thread pools* podem ser dimensionados

através de parâmetros e isso permite ajustar o sistema para explorar o paralelismo oferecido por computadores multinúcleo ou multiprocessados.

Todas as mensagens enviadas pela API Cliente e pelos Servidores DSI são serializadas através do mecanismo Java padrão de serialização, e compactadas com algoritmo Gzip antes de serem transferidas. Apesar da compactação causar a utilização de mais processamento, o uso de CPU é mais vantajoso em relação ao uso de rede, pois a taxa de transferência da rede é limitada e a sua latência é maior.

Uma particularidade desta serialização é que os objetos espaciais presentes nas mensagens trafegadas pelos tópicos e também pelo Barramento de Entregas foram serializados no formato WKT (*Well Know Text*), definido pela OGC. Neste formato a serialização ocupa menos espaço do que sua contrapartida binária (Objeto Geometry da JTS serializado), e é compactada mais eficientemente pelo algoritmo Gzip. Identificou-se através de experimentos que há uma redução média de 40% no tamanho das mensagens empregando esta técnica.

3.8 Considerações Finais

Neste trabalho, foi projetada e desenvolvida uma arquitetura básica que permitisse realizar as análises e identificar o impacto do tamanho e da distribuição das partições de índices R-Tree em arquiteturas distribuídas, além de possibilitar a verificação do nível de escalabilidade que pode ser alcançado.

Para se ter uma arquitetura completa muitos outros aspectos devem ser abordados. Dentre eles o *Balanceamento de Carga*, que poderia ser obtido por meio de réplicas das partições que estão sendo frequentemente consultadas, e ainda a *Tolerância a Falhas* e *Segurança* do sistema proposto na arquitetura.

A utilização do *Middleware* MOM traz facilidades para a implementação desses serviços no sistema futuramente. Os tópicos podem ter vários interessados subscritos e cada mensagem é entregue para todos eles. As réplicas poderiam utilizar esta característica para se manterem atualizadas de acordo com que ocorrem as alterações no índice.

As consultas requisitadas pela API Cliente poderiam ser enviadas para filas no MOM, ao invés de tópicos. Nas filas, cada mensagem é enviada para apenas um dos interessados subscritos e portanto funcionariam como um roteador de tarefas para as réplicas, fazendo com que houvesse um balanceamento de carga das requisições.

A instanciação das réplicas poderia ainda ser feita de forma dinâmica. À medida que determinada partição fosse mais requisitada, uma réplica da mesma seria instanciada para atender o fluxo de requisições e depois poderia ser removida, de acordo com uma política LRU (*Least Recently Used*), por exemplo.

Alguns dos aspectos citados para os componentes, como, por exemplo, a existência de escalonadores específicos para determinados tipos de consultas no índice distribuído não foram implementados devido ao tempo necessário para conduzir um conjunto de experimentos e análises nos mesmos. Porém, um conjunto de testes foi realizado no Escalonador de propósito geral *Round-Robin* e será apresentado nos próximos capítulos, com evidências da sua utilidade.

O Capítulo 4 faz uma análise para determinar a melhor escolha do tamanho das partições para a arquitetura proposta. Uma comparação foi realizada entre a forma de definição deste tamanho em R-Trees centralizadas e a forma indicada para definição na arquitetura distribuída proposta.

CAPÍTULO 4

Definição do Tamanho das Partições do Índice Distribuído

Este capítulo estuda o impacto do tamanho das partições do índice distribuído (valor de M) no desempenho do sistema. Como detalhado no Capítulo 2, a variável M delimita a quantidade de itens (MBRs ou Polígonos) que cada nó do índice pode armazenar.

Em implementações centralizadas de R-Trees, o valor de M é definido com base no tamanho da página de armazenamento dos dados, a fim de reduzir o fluxo de páginas entre o disco rígido e a memória RAM. A Seção 4.1 realiza um breve estudo a respeito desta forma de definição do valor de M, nessa arquitetura de sistema.

A Seção 4.2 descreve o comportamento do DSI-RTree, de acordo com o valor de M, principalmente no aspecto da troca de mensagens para o processamento das buscas. Resultados práticos coletados são apresentados e discutidos na Seção 4.3.

Finalmente, na Seção 4.4, são discutidas as implicações da definição fixa de M na arquitetura centralizada, e aponta-se uma forma para sua definição na arquitetura do DSI-RTree.

Em todos os artigos revisados, a discussão do valor de M, quando realizada, é feita em relação ao armazenamento paginado do índice em disco. Como discutido no Capítulo 3, a arquitetura proposta não armazena o índice de forma paginada e, portanto, a discussão a seguir é focada no desempenho do índice e na troca de mensagens entre as máquinas do *cluster*, algo ainda não discutido em outros trabalhos correlatos, de acordo com nossa revisão literária.

4.1 Valor de M em Sistemas Centralizados

As implementações e avaliações de desempenho de árvores R-Trees, encontradas na literatura, são normalmente voltadas para uso em sistemas centralizados (como Bancos de Dados Relacionais), e consideram o custo de armazenamento da estrutura em memória secundária (normalmente disco rígido) como o principal fator de medição do desempenho.

Como o acesso e gravação nesta memória é normalmente lento, o desempenho é avaliado com base na quantidade de acessos efetuados para recuperar os dados do índice e levá-los para a memória RAM [15, 24, 31].

Nesses sistemas, a quantidade mínima de bytes lidos ou escritos no disco rígido é limitada pelo tamanho de uma página de dados, também conhecida como Tamanho de Bloco de Dados. Em um Sistema Gerenciador de Banco de Dados (SGBD) seu tamanho é especificado durante a instalação ou na criação de cada banco de dados.

Este tamanho de página é determinado com base na leitura mínima de bloco do subsistema de IO do Sistema Operacional, para evitar o *overhead* da execução de múltiplas chamadas de sistema para leitura de páginas. São comuns, atualmente, tamanhos de páginas de 4, 8 e 16kb em sistemas de 32 bits, e até 32kb em sistemas de 64 bits [37].

O valor de M dos índices é definido conforme o tamanho desta página, de forma que cada nó ocupe aproximadamente uma página completa [15, 24, 31]. Divide-se o tamanho da página (S) pelo tamanho de cada entrada no nó (k) para se obter o valor aproximado de M, ou seja, $M \approx S/k$. A leitura de páginas do disco é reduzida com esta abordagem, pois se observa que:

- $M < S$: Se o valor de M for menor que o tamanho da página, mais que uma folha do índice pode ser armazenada em uma mesma página, e não necessariamente elas conteriam dados colocalizados. Isso provoca a leitura de dados desnecessários para o processamento de uma determinada consulta (presentes em uma só página). A leitura de mais páginas do disco pode também ser necessária para encontrar os dados para a consulta, devido a uma possível fragmentação dos nós nas páginas;
- $M > S$: Com M maior que o tamanho da página, duas ou mais páginas seriam necessariamente lidas do disco a cada acesso a uma folha do índice.

4.2 Valor de M em Sistemas Distribuídos

Ao contrário da arquitetura centralizada, na arquitetura distribuída definida no Capítulo 3, não é empregada paginação no armazenamento do índice. Os dados armazenados em disco são carregados totalmente na memória RAM, durante a inicialização do sistema.

Esta inexistência de paginação possibilita o uso de outros valores de M, e também abre uma discussão a respeito de qual seria o seu valor ideal. Enquanto na arquitetura centralizada procura-se reduzir o acesso ao disco rígido e aproveitar ao máximo os dados em memória, o principal fator de desempenho do índice distribuído é a troca de mensagens entre as máquinas do *cluster*. Quanto maior for à necessidade de comunicação, maior será o tempo gasto para executar buscas no índice, devido à latência da rede entre os nós.

Considerando a forma de particionamento empregada no índice distribuído e o comportamento logarítmico-recursivo das árvores R em geral, notam-se as seguintes relações entre o valor de M e o desempenho do índice distribuído:

- **Índice muito alto**: Quanto menor é o valor de M, maior é a altura do índice. Devido o processamento de buscas no índice ser recursivo, nível a nível, quanto mais níveis a percorrer, maior a quantidade de mensagens trocadas até se alcançar o nível das folhas. Dado que a soma das latências de envio das mensagens entre os níveis domina o tempo de resposta das consultas, quanto maior a quantidade de níveis a serem percorridos, maior é o tempo de resposta de uma consulta;
- **Quantidade de nós**: Quanto menor é o valor de M, menor é a sua capacidade de armazenamento e, consequentemente, mais nós são necessários para armazenar a mesma quantidade de itens. Como uma partição é criada para cada nó do índice, maior é a quantidade de partições acessadas[1] e, com isso, maior a necessidade de comunicação[2]. Pode-se comparar este aspecto com a arquitetura centralizada: enquanto nela há a necessidade de reduzir a quantidade de páginas acessadas, para evitar o IO do disco rígido, na arquitetura distribuída há a necessidade de reduzir a quantidade de partições acessadas, para evitar o IO da rede de comunicação;
- **Quantidade de itens no nó raiz**: Para um mesmo dataset, quanto maior é o valor de M, menor é a quantidade de itens no nó raiz. Um valor de M muito alto pode causar o uso de apenas dois itens na raiz (dois ramos são suficientes para armazenar todos os itens), o que reduz a seletividade do índice, causando a postergação da seleção dos caminhos a serem consultados. Por exemplo, ao se processar uma busca de janela em um nó raiz com apenas dois itens, seleciona-se metade do espaço geográfico do dataset para ser analisado no próximo nível[3]. No mesmo cenário, porém com o nó raiz totalmente preenchido, pode-se selecionar uma fração menor que a metade do espaço já no primeiro nível do índice, e paralelizar mais efetivamente o processamento da consulta, pois, para cada item intersectado no nó raiz envia-se uma mensagem assíncrona para o nível inferior.
- **Índice muito baixo**: Apesar de um índice baixo reduzir a quantidade de mensagens trocadas, se o valor de M é tão alto que causa a existência de apenas um nível, então todos os itens são consultados para cada busca efetuada e o índice apenas introduz

[1] Uma mesma busca de janela que retorna 100 itens do dataset, pode acessar, por exemplo, dez nós folhas com M=10 e apenas um com M=100.

[2] A necessidade de troca de mensagens devido ao particionamento esta relacionada à distribuição feita pelo escalonador empregado. Como foi empregado um escalonador *round-robin*, mesmo as partições com dados colocalizados podem estar intercaladas em máquinas diferentes no *cluster*, e necessitar de comunicação via rede.

[3] Melhor caso possível, onde não há sobreposição entre os dois MBRs e o espaço geográfico foi dividido exatamente ao meio.

um *overhead* no processamento. Como também é criada somente uma partição com o nó raiz, o armazenamento e processamento paralelo da arquitetura não é explorado, já que somente uma máquina é utilizada para processar as consultas. Outro caso extremo também ocorre quando o valor de M provoca a construção de um índice de dois níveis, com o nó raiz pouco preenchido.

- **Tamanho dos MBRs nos níveis superiores**: Quanto menor o valor de M, maiores são os MBRs e o espaço morto nos níveis superiores do índice[4]. Este fato aumenta as chances de buscas no índice percorrerem vários ramos inutilmente, causando uma maior troca de mensagens.

Em resumo, valores de M muito pequenos causam índices muito altos, que necessitam de muita troca de mensagem. Causam ainda aumento nos MBRs e espaço morto nos níveis superiores, provocando o acesso falso a nós do índice (nós que não contêm resultado para as buscas). Por outro lado, valor de M muito alto, apesar de reduzir a quantidade de mensagens trocadas, pode causar índices sem seletividade alguma (com apenas um nível) ou índices com mais de um nível, porém com nó raiz pouco seletivo. A seção a seguir continua esta discussão, com resultados práticos computados com datasets reais.

4.3 Impacto do Valor de M em Datasets Reais

Uma avaliação foi realizada com os datasets descritos a seguir, a fim de mensurar o impacto do valor de M na estrutura do índice e também na quantidade de mensagens trocadas durante o processamento de buscas de janelas. Foram utilizados três datasets geoespaciais reais, disponibilizados gratuitamente na internet, de três tamanhos distintos:

- **Arizona TabBlock**: extraído do banco de dados de censo demográfico dos Estados Unidos (TIGER 2008), com 158.292 polígonos representando áreas delimitadas (quarteirões e propriedades rurais) do estado do Arizona. Este dataset é referenciado no texto e figuras a seguir como *arizona*;
- **Alertas de Desmatamento**: dados do laboratório LAPIG, do Instituto de Geografia da UFG, com 32.578 polígonos, representando alertas de desmatamento no Cerrado brasileiro. É referenciado no texto e figuras a seguir como *desmata*.
- **Vegetação**: Dados nacionais com 2.140 polígonos representando as áreas de vegetação brasileiras. Referenciado no texto e figuras a seguir como *vegeta*.

[4]Beckmann et al. [5] discute esta relação entre o tamanho do MBR e a quantidade de espaço morto: diminuindo-se ambos, provoca-se a escolha dos caminhos a serem percorridos nos níveis superiores e a redução da quantidade de caminhos a serem acessados (p. 323, O1 e O2).

A representação gráfica dos polígonos destes datasets e janelas de busca pode ser visualizada na Figura 4.1. As janelas de busca são de três tamanhos distintos: *i*) para o arizona, as janelas intersectam áreas densas e são bem pequenas se comparadas ao tamanho do dataset; *ii*) para o desmata são proporcionalmente maiores, também em regiões densas; *iii*) para o vegeta são bem maiores que as anteriores e cobrem praticamente todo o espaço mapeado. Nos três casos as janelas são seletivas (acessam apenas parte do dataset), apesar do grau de seletividade mudar de um dataset para outro.

As regiões densas foram escolhidas porque são os locais que mais necessitam de nós para armazenamento, e devido a isso, causam uma maior sobreposição de MBRs. A mudança na seletividade das janelas definidas tem o objetivo de provocar variação na quantidade de nós acessados, bem como na quantidade de áreas de sobreposição e espaço morto intersectados, os quais podem eventualmente provocar uma maior ou menor troca de mensagens.

Os valores de M foram escolhidos de forma a provocar alteração na altura necessária para o armazenamento dos itens. Existem quatro mudanças de altura, conforme é mostrado na Tabela 4.1. Como esperado, aumentando o valor de M diminui-se a altura do índice. No momento que esta redução ocorre, o preenchimento da raiz varia do mínimo possível (dois itens) para um número maior, próximo da capacidade do M. É o que acontece, por exemplo, com o dataset desmata de M=38 para M=42. Com M=38, apenas dois itens existem na raiz e a altura do índice é quatro. Com M=42 tem-se altura igual a três e 41 itens na raiz.

Tabela 4.1: *Mudança de Altura e Preenchimento da Raiz conforme alteração no valor de M.*

Dataset	M	Altura	Preenchimento da Raiz
arizona	50 -> 80	4 -> 3	5 -> 61
desmata	38 -> 42	4 -> 3	2 -> 41
vegeta	17 -> 18	4 -> 3	2 -> 17
vegeta	50 -> 80	3 -> 2	2 -> 42

Outras medidas coletadas são apresentadas no gráfico da Figura 4.2. A escala do eixo y foi usada para duas unidades: *i*) A quantidade média de mensagens trocadas durante o processamento de cada busca de janela e *ii*) o percentual médio de itens intersectados em cada nó acessado. O gráfico está dividido em três partes: à esquerda tem-se o dataset arizona, no meio o dataset desmata e, à direita, a análise mais extensa do dataset vegeta, com uma maior quantidade de valores de M.

Observa-se em todos os datasets que a quantidade de mensagens trocadas diminui, conforme o valor de M aumenta. A única exceção a este comportamento foi o dataset arizona, de M=150 para M=200, na qual a quantidade de mensagens aumentou.

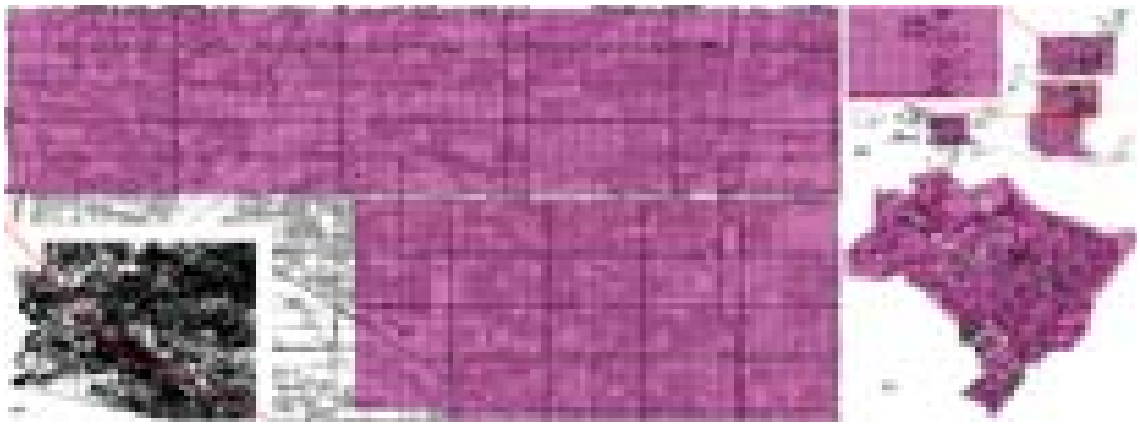

Figura 4.1: *Ilustração dos Datasets e Janelas de Busca: Arizona no canto inferior esquerdo, com ampliação ao seu redor (a), Desmata (b) com ampliação no canto superior direito e Vegeta (c). As áreas em destaque são Janelas de Busca.*

Apesar da mudança de altura para determinados valores de M (Tabela 4.1), no exato intervalo destas mudanças não existe um comportamento diferenciado nas linhas do gráfico da Figura 4.2. A quantidade de mensagens trocadas continua diminuindo, com pequenas oscilações, para todos os valores de M testados. Isto sugere que o fator mais determinante para a redução na troca de mensagens é a diminuição da quantidade de nós no índice, devido à maior capacidade de armazenamento que o valor de M proporciona.

As outras duas linhas no gráfico evidenciam a redução no percentual médio de interseção de itens por nó, conforme discutido na Seção 4.2. Para cada nó do índice, calculou-se a quantidade de itens que são intersectados pela janela, em relação à quantidade de itens existente no mesmo. O percentual no gráfico é a média de todos os valores obtidos, para nós diretórios e folhas separadamente.

Nota-se uma redução contínua do percentual de itens intersectados por nó, à medida que o valor de M aumenta. Em outras palavras, apesar da janela de busca intersectar o MBR do nó, cada vez menos itens contidos no mesmo intersectam com a janela de busca. Para atender o requisito de balanceamento das árvores R (40% de preenchimento mínimo na R*), o nó está agrupando itens não tão colocalizados, e isso provoca um aumento de seu MBR.

No entanto, uma exceção pode ser observada para o dataset arizona, de M=150 para M=200. O percentual de interseção dos nós diretórios aumenta ao invés de diminuir, ao contrário dos demais datasets. Este é o motivo do aumento na quantidade de mensagens no mesmo instante no gráfico: como mais itens são intersectados em cada nó, mais mensagens são enviadas para os níveis inferiores.

Acredita-se que esta variação seja um comportamento semelhante ao que ocorre com o dataset vegeta, com M=24 na Figura 4.2. Mas também pode ser uma particularidade

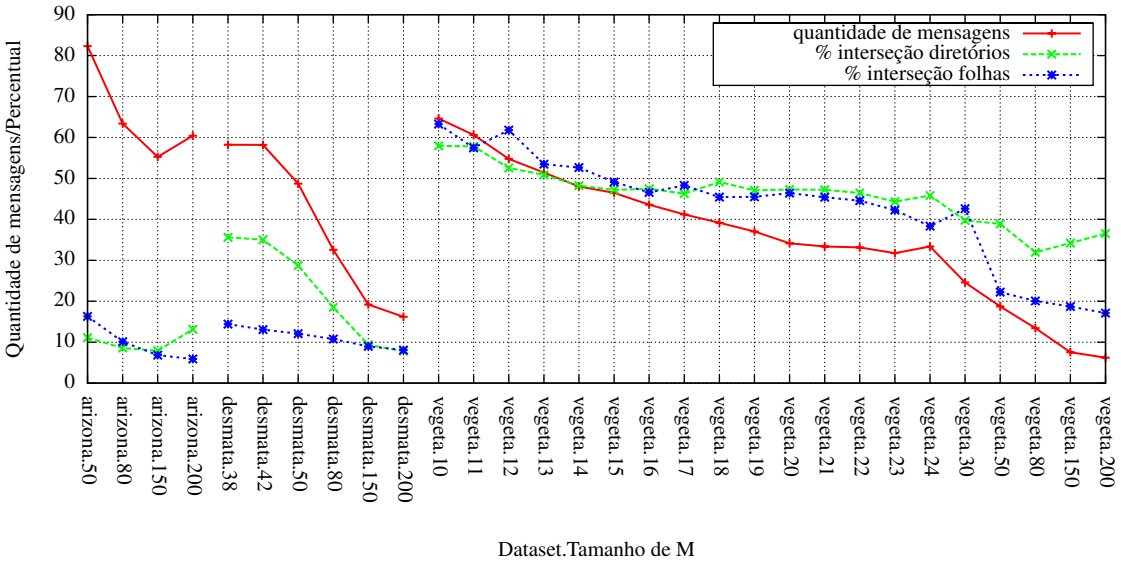

Figura 4.2: *Análise de Métricas em cada Dataset, para valores de M variados.*

do dataset em conjunto com o algoritmo de divisão utilizado que não foi identificada. Testes com valores de M ainda maiores (M=250, 300) e com outros datasets diferentes devem ser realizados em trabalhos futuros para analisar detalhadamente este comportamento.

4.4 Definição do Valor de M

Apesar da definição do valor de M através do tamanho da página ser o ideal para a arquitetura centralizada, observa-se que esta abordagem provoca uma variação no preenchimento do nó raiz do índice, conforme o tamanho do dataset indexado. Indexar datasets pequenos ou grandes[5], com valor de M fixo, causa uma baixa seletividade no nó raiz do índice na maioria dos tamanhos de datasets.

Este comportamento pode ser verificado no gráfico da função de preenchimento do nó raiz, na Figura 4.3. Considera-se no gráfico o valor de M igual a 130, quatro níveis de preenchimentos dos nós (40[6], 60, 80 e 100%) e datasets com quantidade de polígonos entre 10^3 e 10^7. Esses níveis de preenchimento foram utilizados para simular datasets diferentes, devido ao tipo (retângulos, linhas) e a organização de dados (esparso, denso) particular de cada um.

[5]Os termos pequeno ou grande utilizados para qualificar os datasets são empregados em relação a cardinalidade do conjunto – a quantidade de itens (pontos, polígonos) – e não em relação à área geográfica coberta pelos polígonos ou ainda o formato dos mesmos.

[6]O pior nível de preenchimento possível de um nó, considerando o mínimo de 40% estabelecido pelos criadores da R*-Tree.

Em cada uma das linhas no gráfico há um ponto no eixo x para o qual o total de itens no nó raiz é de apenas dois. Este comportamento ocorre devido ao aumento da altura do índice para suportar a quantidade de polígonos. O nó raiz fica minimamente preenchido logo após sua divisão e, conforme a quantidade de polígonos aumenta, seu preenchimento também aumenta. Quando o limiar do valor de M é alcançado novamente, ocorre outra divisão e mais um nível no índice é criado.

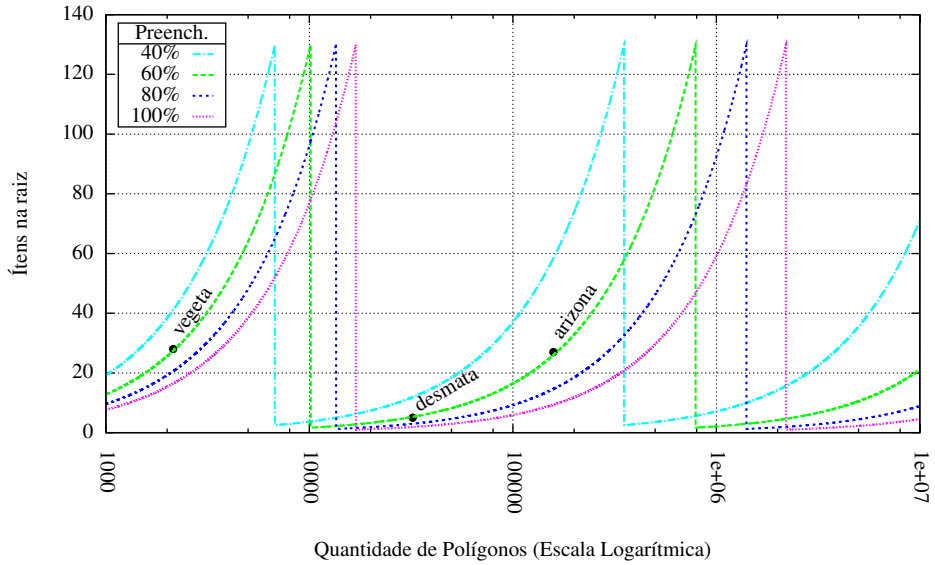

Figura 4.3: *Simulação da baixa quantidade de itens no nó raiz, com M=130, para datasets com quantidade de polígonos entre* 10^3 *e* 10^7.

Em datasets reais, o preenchimento dificilmente alcança 100% nos nós abaixo da raiz, e também não é uniforme, ou seja, podem existir nós com preenchimentos variados, entre 40% e 100%. O fator de preenchimento depende ainda da estratégia de divisão empregada, se é utilizado reinserção para evitar divisões e até da ordem de inserção dos polígonos na criação do índice.

Na prática, porém, na implementação do DSI-RTree, observou-se que os três datasets da Seção 4.3 possuem uma variação mínima de preenchimento para todos os M testados. Ambos ficam com preenchimento entre 59,44 e 72,54%, com desvio padrão de 3,07%. Para um valor de M=130 fixo, encontrado em [31], e o preenchimento de nós real de cada dataset, os três ficariam com preenchimento do nó raiz baixo, conforme nota-se no gráfico da Figura 4.3. Enquanto poderiam estar com aproximadamente 130 itens no nó raiz, todos estão com menos de 30.

Outra desvantagem do valor de M fixo é que ele pode causar índices muito baixos ou muito altos, dependendo do tamanho do dataset. Se o valor de M é baixo, gera índices muito altos para datasets grandes; Se é alto, gera índices muito baixos para datasets menores.

Uma possível forma de evitar este comportamento é calcular um valor de M específico para cada dataset. Dado que na arquitetura do DSI-RTree não é necessário fixar o valor de M, pode-se encontrar o seu valor com base na quantidade de polígonos e no grau de preenchimento, de forma que o nó raiz do índice fique próximo de sua capacidade máxima.

Pode-se reduzir a altura do índice para 2, 3 ou 4 níveis, de acordo com a quantidade de polígonos do dataset, reduzindo a troca de mensagens e a latência do processamento das consultas, garantindo ainda uma boa seletividade e paralelismo, desde o nível raiz do índice. Mesmo com no máximo quatro níveis é possível armazenar datasets de até 10 milhões de polígonos com M=100, aproximadamente.

Para ilustrar esta possibilidade de variação de níveis do índice, é mostrado na Figura 4.4, que existem valores de M adequados para cada tamanho e nível de preenchimento do dataset que fazem com que a raiz fique totalmente preenchida.

Na Figura 4.4, é ilustrado o valor de M (eixo *y*) recomendado para índices com 2, 3 e 4 níveis para os datasets de diferentes tamanhos (eixo *x*), considerando que a raiz está totalmente preenchida. Foram plotadas duas linhas para cada altura do índice: a mais larga indicando o valor de M se o preenchimento dos nós for próximo do limite inferior (40%), e a linha menos espessa indicando o tamanho de M para o preenchimento de 60%. Apenas para comparação, a linha M=100 indica o quanto o tamanho de M fixo fica distante do valores recomendados.

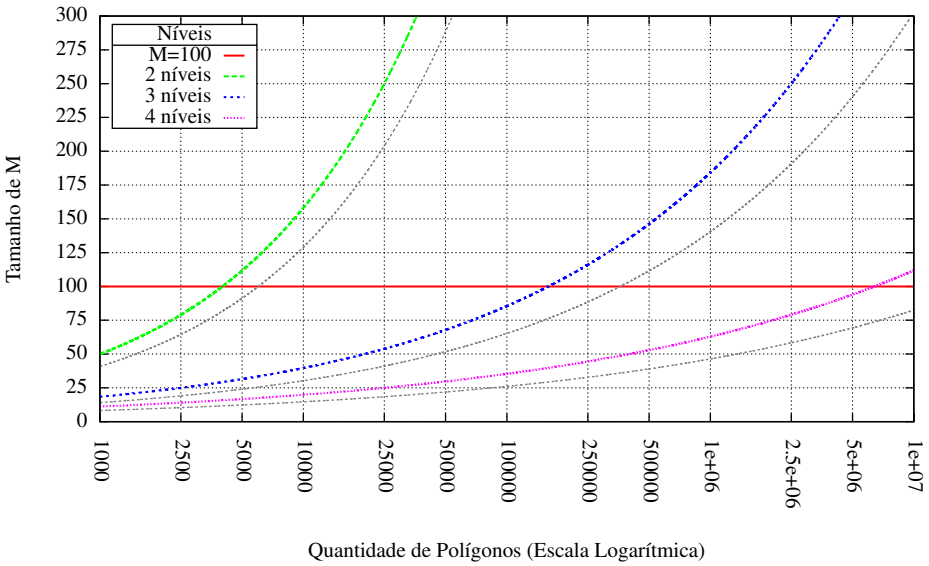

Figura 4.4: *Sugestão do valor de M para índices de 2, 3 e 4 níveis e datasets com até 10 milhões de polígonos.*

O cálculo realizado para plotagem das linhas no gráfico foi o da equação 4-1, onde *a* é a altura do índice desejada, *n* a quantidade de polígonos no dataset e *p* o percentual de preenchimento dos nós (entre 0 e 1). Chegou-se nesta fórmula isolando M

na equação 4-2, a qual computa a quantidade de polígonos que podem ser armazenados para um determinado valor de M e fator de preenchimento de nós. O total de polígonos n é aproximadamente igual ao tamanho de M reduzido por p ($p < 1$) e elevado a quantidade de níveis abaixo da raiz. Como a raiz não participa da redução, a quantidade de nós possíveis é M vezes o valor obtido.

$$M = \sqrt[a]{\frac{n}{p^{a-1}}} \tag{4-1}$$

$$n = M * (M * p)^{a-1} \tag{4-2}$$

4.5 Considerações Finais

A definição do valor de M é um tema pouco explorado na literatura, devido à limitação imposta pela paginação da estrutura para armazenamento em disco.

Em contraste a definição fixa do valor de M com base no tamanho da página, foi discutido nas seções anteriores que removendo esta limitação pode-se determinar o valor de M mais adequadamente para cada tamanho de dataset. Foi discutido também que valores inadequados podem causar o aumento na troca de mensagens em sistemas distribuídos, a criação de índices muito baixos e ineficientes e ainda índices que não exploram eficientemente o paralelismo no processamento das consultas.

Apresentou-se uma fórmula para determinação do valor de M, com base no tamanho do dataset e no preenchimento médio de seus nós. Apesar de não se conhecer antecipadamente esta última medida, obtêm-se um valor de M bem próximo do adequado utilizando o preenchimento mínimo possível dos nós.

A diferença entre o M obtido com preenchimento mínimo e o M adequado pode ser utilizada para garantir espaço adicional para armazenar novos itens em datasets dinâmicos. Para datasets estáticos esta margem não se justifica e seria necessário uma indexação dupla para obter um preenchimento perto do máximo na raiz.

Outra forma possível de determinar o preenchimento médio dos nós é extrair uma amostragem do dataset e simular a criação de um pequeno índice. Alguns trabalhos na literatura exploraram métodos adequados para a extração desta amostragem, como em [36, 1].

O próximo capítulo apresenta os resultados dos testes nos datasets da Seção 4.3, com ênfase no desempenho e na escalabilidade horizontal do sistema. Ou seja, qual é o comportamento do sistema à medida que aumenta a quantidade de máquinas para distribuição das partições e processamento das consultas no índice?

CAPÍTULO 5

Avaliação de Desempenho

Este capítulo apresenta os testes de desempenho executados no DSI-RTree. O objetivo da realização dos testes foi avaliar se o índice distribuído é eficiente para:

- Escalar horizontalmente à medida que mais estações de trabalho são adicionadas ao sistema;
- Maximizar a quantidade de requisições atendidas por segundo e evitar processamento desnecessário (*overhead*);
- Distribuir os dados uniformemente pelas máquinas para o aproveitamento do *hardware* disponível.

No início de cada uma das seções a seguir é descrito o tipo de teste executado e a metodologia empregada, seguido da apresentação dos resultados obtidos e a discussão destes. O ambiente de execução é descrito na Seção 5.1, e os testes de Qualidade do Índice Distribuído são apresentados na Seção 5.2. Estes testes estimam a redução do espaço de busca durante o processamento das consultas.

Na Seção 5.3 é feita uma análise da eficiência do escalonador *Round-Robin* utilizado, medindo o nível de uso das máquinas do *cluster* durante a execução das consultas.

Na Seção 5.4, a Escalabilidade Horizontal do sistema é avaliada observando dois comportamentos: *i*) a quantidade de requisições atendidas por segundo, conforme se aumenta a quantidade de máquinas do *cluster* e *ii*) o quanto o aumento do tamanho do dataset impacta na quantidade de requisições atendidas pelo sistema.

5.1 *Hardware* e *Software* do Ambiente de Execução dos Testes

O *Cluster* de computadores utilizado nos experimentos consiste de 40 máquinas disponíveis em um laboratório de ensino do Instituto de Informática da UFG. O ambiente de execução do *cluster* foi configurado da seguinte forma:

- 20 Máquinas Intel Core 2 Duo 2.4 GHz, com 1 Gb de memória RAM, utilizadas como Servidores DSI;
- 20 Máquinas Intel Core 2 Duo 2.4 GHz, com 1 Gb de memória RAM, utilizadas como simuladores de Clientes do Serviço. Um simulador de cliente é uma máquina que executa requisições ao serviço, em uma determinada taxa por segundo, simulando o que ocorreria em um ambiente de produção;
- 1 Máquina Intel Core 2 Duo 2.4 GHz, com 2 Gb de memória RAM, utilizada para comunicação entre os Servidores DSI;
- Switch Gbit Dell PowerConnect 6248.

Em cada uma das máquinas foi instalada a distribuição Open Suse mínima, versão 11.2, com Kernel Linux 2.6.31.12. A máquina virtual Java utilizada em todas as máquinas foi a Java(TM) SE Runtime Environment (build 1.6.0_20-b02). Nas máquinas simuladoras de clientes, os parâmetros -Xmx e -Xms (*Heap* e *Stack* da JVM – *Java Virtual Machine*) foram ajustados para o máximo possível (650Mb), considerando o uso parcial da memória pelo Sistema Operacional e a fragmentação da memória total disponível (1Gb) causada pelo processo de *boot*.

Durante os testes, notou-se um atraso no relógio das máquinas, principalmente quando sob extremo uso de CPU. Devido a isso, os relógios das máquinas foram sincronizados localmente (com um *host* na própria rede, sem consulta à internet) antes e durante os testes para garantir a corretude das latências coletadas.

Os datasets utilizados nos testes realizados foram os mesmos detalhados no Capítulo 4, Seção 4.3: Arizona, Desmatamento no Cerrado e Vegetação do Brasil.

5.2 Qualidade do Índice Distribuído

A criação de índices é um método empregado para prover eficiência na busca de partes de conjuntos de dados e/ou informações. O índice de um livro impresso é um exemplo prático disso. Consultando o índice geral ou remissivo do livro, espera-se encontrar um determinado tópico de interesse, sem que para isso seja necessário ler o livro todo. O cuidado com que este índice foi escrito pelo autor é que determinará a quantidade de páginas que serão lidas, até que finalmente seja localizado (ou não) o tópico desejado.

Da mesma forma, em sistemas computacionais, o índice é utilizado para procurar dados de interesse eficientemente, sem que seja necessário procurar por todo o conjunto. Exemplos do uso da indexação incluem sistemas de banco de dados relacionais (para acelerar a busca por informações em tabelas) e sistemas de indexação de páginas WEB (para identificar textos que casam com padrões).

A Qualidade do Índice é determinada por dois pontos fundamentais: *i*) a quantidade de dados acessados quando uma busca por um determinado tópico de interesse é

executada (espaço de busca); *ii*) o percentual do espaço de busca acessado que não fará parte da resposta (falsos positivos).

Além do propósito de redução do espaço de busca, o índice distribuído foi empregado como mecanismo de particionamento do conjunto de dados. Durante a construção do índice, partições de tamanho máximo *M* são criadas e distribuídas – por um escalonador – entre as máquinas disponíveis no *cluster*. Cada partição é preenchida de forma que contenha polígonos colocalizados entre si, da mesma forma que em uma árvore R-Tree.

O teste de Qualidade do Índice avalia os seguintes aspectos do comportamento do sistema:

- A fração dos dados acessados (espaço de busca), em relação ao total de dados disponíveis, durante a execução de consultas;
- Na fração acessada, qual o percentual de falsos positivos (dados acessados, que não serão retornados como resposta).

De modo geral, quanto maior for a redução no espaço de busca, melhor é o desempenho do sistema, pois menos acesso físico ao conjunto de dados é necessário. Este acesso físico em sistemas centralizados refere-se ao tempo de acesso à memória secundária (disco rígido, *storage*). Os dados não podem ser colocados totalmente em memória e, então, o sistema operacional realiza paginação em memória virtual para disponibilizá-los à aplicação, conforme discutido na Seção 4.1.

Em sistemas distribuídos, especificamente no DSI-RTree, como os dados estão distribuídos fisicamente nas máquinas do *cluster*, o acesso implica na troca de mensagens. As instâncias do sistema trocam mensagens para encontrar os dados procurados ativando o processamento da consulta em cada Servidor DSI que possui parte da resposta[1]. Reduzindo-se este acesso físico, diminui-se a quantidade de mensagens trocadas para comunicação entre nós e, consequentemente, aumenta-se o desempenho do sistema devido à menor ocorrência de IO na rede.

Os falsos positivos acessados são mensurados para avaliar a quantidade de trabalho extra executado que não produz resultado aproveitável/útil.

Ampliando a descrição dos datasets utilizados na Seção 4.3, as buscas no índice foram efetuadas com janelas seletivas, que interceptam partes do total de polígonos de cada dataset, conforme os percentuais na Tabela 5.1. O objetivo é verificar se cada busca acessa a quantidade mínima necessária do dataset para retornar o resultado esperado, e o quanto este acesso varia, à medida que se aumenta a área de interseção das janelas[2].

[1] Algumas máquinas podem ser acessadas mesmo não tendo parte da resposta, devido a existência dos falsos positivos.

[2] O espaço de busca de janelas maiores pode conter mais falsos positivos do que o de uma janela menor, devido à interseção de mais áreas de sobreposição e espaço morto no índice.

Dataset	Média	Mínimo	Máximo	Desvio padrão
arizona	0,09	0,04	0,14	0,02
desmata	0,29	0,04	1,49	0,26
vegeta	6,53	0,14	19,44	7,29

Tabela 5.1: *Percentual médio, mínimo, máximo e desvio padrão da quantidade de polígonos interceptados, em relação ao total de polígonos de cada dataset.*

A redução do espaço de busca no dataset Arizona, com M igual a oitenta[3], pode ser observada no gráfico da Figura 5.1. À esquerda, tem-se o gráfico exibindo a proporção real do acesso ao nós do índice distribuído e, à direita, sua parte inferior ampliada, com o eixo y ajustado para refletir o intervalo de 0 a 120 da escala original. Para calcular a redução do espaço de busca, utilizou-se a quantidade de folhas e diretórios acessados no processamento de 30 janelas para os datasets arizona e desmata e 19 janelas para o dataset vegeta. São exibidas no gráfico de áreas empilhadas: *i*) a quantidade de folhas e diretórios acessados, necessários para se gerar o resultado; *ii*) a quantidade de folhas e diretórios falsos – que não necessitariam ser consultadas para obter o resultado e *iii*) a quantidade de folhas e diretórios não acessados. Estes resultados são apresentados para cada janela, de 0 a 29, no eixo x.

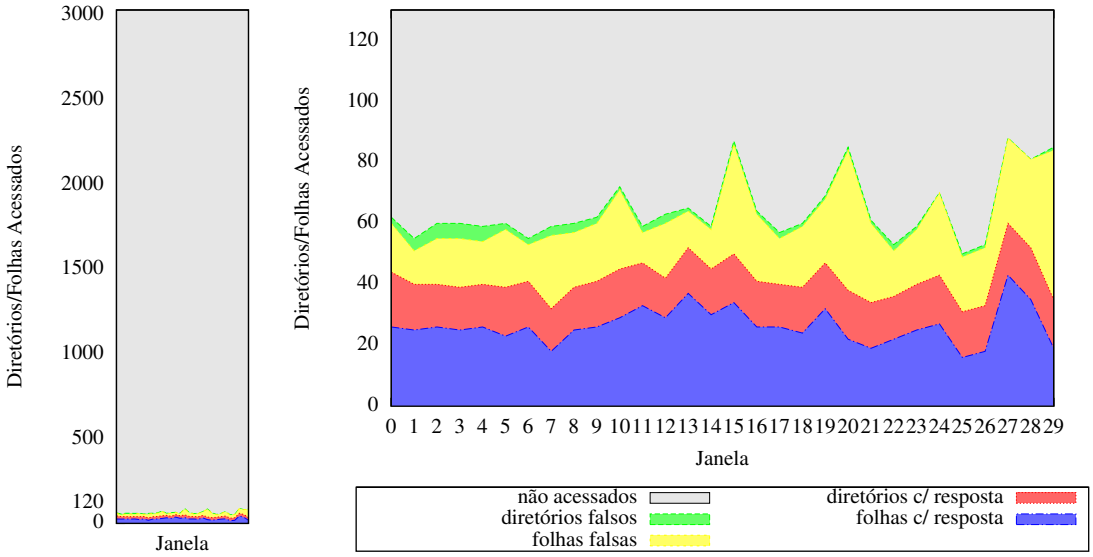

Figura 5.1: *Acesso a Folhas e Diretórios do Dataset Arizona, com M=80, por 30 buscas de janela: à esquerda gráfico na escala original, mostrando a quantidade de nós acessados e à direita a parte inferior ampliada.*

Em relação ao total de folhas ou total de diretórios do dataset, os percentuais

[3]A fórmula do cálculo de M resulta em valor de M=76 para preenchimento igual a 60%.

médios de acesso aos nós do índice durante o processamento das 30 buscas de janela são os seguintes: (Os dados completos de acesso de janelas e diretórios estão no Apêndice A)

- O percentual médio de folhas acessadas do dataset (a soma de folhas falsas ou não) é de 1,60%, com desvio padrão de 0,36% (Mínimo de 1,15% e Máximo de 2,39%);
- Das 2.968 folhas existentes no índice distribuído, 47,5 em média foram consultadas para cada busca;
- O percentual médio de folhas com resultado acessadas é de 0,89%; e de folhas falsas é de 0,71%;
- O percentual médio de acesso a diretórios que levariam a resultados nas folhas é de 27,31%; e de diretórios falsos 2,96%;
- Somando diretórios e folhas acessados, ou seja, o acesso total ao dataset, o percentual médio corresponde a 2,13% do dataset, com desvio de 0,35% (Mínimo de 1,65% e Máximo de 2,90%).

Apesar de poucas folhas falsas serem acessadas (0,71%), elas representam em média 44,38% do total de folhas acessadas, ou seja, de cada duas folhas acessadas, 1,12 contêm resultado e 0,88 é falso positivo. O desvio padrão é de 11,17% (Mínimo de 23,26% na janela 11 e Máximo de 72,06% na janela 29).

Sabe-se que estes valores podem se alterar de um dataset para outro e com a mudança do valor de M. Portanto, as mesmas medidas foram realizadas para os demais datasets e diferentes tamanhos de M. A média de cada um deles é apresentada na Figura 5.2. Os dados foram normalizados em percentuais, devido ao valor de M causar variação na quantidade de nós diretórios e folhas, e devido a essas mesmas quantidades serem diferentes de um dataset para outro. A escala do eixo y foi adaptada de 0 a 40% para melhor visualização. Os demais 60% correspondem à continuação da legenda de não acessados.

Nota-se no gráfico um aumento na quantidade de acesso entre os datasets, principalmente no dataset vegeta. Este aumento está relacionado com a configuração das janelas utilizadas, que interceptam uma maior parte do dataset total. Como as janelas são maiores, ou seja, interceptam uma área maior e maior quantidade de itens do dataset, espera-se o acesso a um maior percentual de nós. Possivelmente, devido à maior interseção de espaço morto e sobreposição, o número de acessos poderia aumentar de forma desproporcional ao aumento da janela, mas isso não ocorreu.

Enquanto as janelas do dataset vegeta são maiores que as do dataset arizona 3,22 vezes, a quantidade de acesso aumenta praticamente na mesma proporção: 3,24 vezes (média para os quatro valores de M's). Entre o dataset vegeta e o dataset desmata a proporção é ainda menor: aumento da janela de 22,5 vezes contra aumento no acesso de 7,63 vezes.

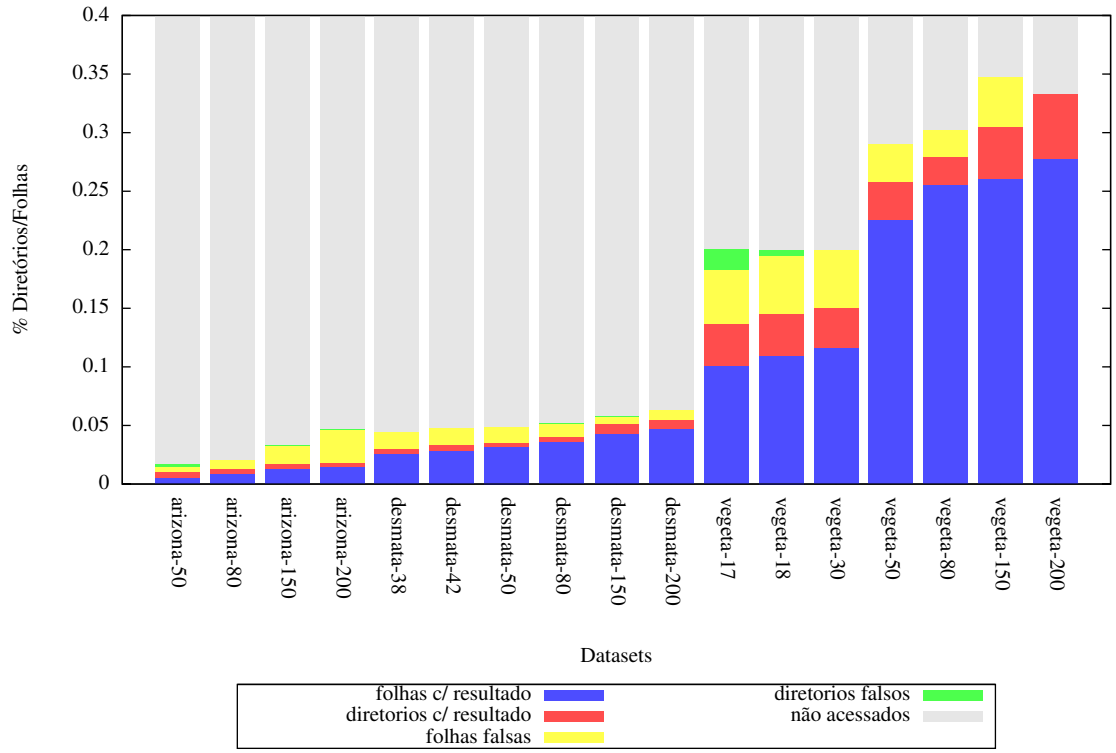

Figura 5.2: *Percentual de Folhas e Diretório Acessados durante Buscas de Janela, para cada Dataset e Valor de M.*

Percebe-se ainda que o percentual da quantidade de folhas falsas em relação ao total de folhas acessadas possui um comportamento diferente entre os datasets, conforme a Tabela 5.2. Enquanto no dataset arizona a tendência é o aumento do percentual conforme o aumento do valor de M, nos outros dois é exatamente o oposto: o percentual de folhas falsas é menor conforme o aumento do valor de M.

Este comportamento indica que o algoritmo de divisão está gerando nós com espaço morto muito grande no dataset arizona, pois apesar da janela da busca interceptar o MBR, não intercepta nenhum polígono no nó folha falso. Esta fato agrava-se ainda mais para o valor de M igual a 150 e 200, onde o percentual de folhas falsas ultrapassa o número de folhas com resultado acessadas, com percentuais de 55,16 e 65,51% respectivamente.

Tabela 5.2: *Percentual de Folhas Falsas, em relação ao total de Folhas Acessadas.*

Dataset	**M=17**	**M=18**	**M=30**	**M=38**	**M=42**	**M=50**	**M=80**	**M=150**	**M=200**
Arizona						47,01	44,38	55,16	65,51
Desmata				34,33	35,03	30,16	26,33	16,44	16,36
Vegeta	32,44	32,96	32,44			15,77	14,09	13,96	11,91

5.3 Avaliação do Escalonador *Round-Robin*

O escalonador do DSI-RTree é o componente que distribui as partições dos índices pelas máquinas do *cluster* para que o processamento de consultas seja realizado de forma paralela e eficiente, utilizando toda a capacidade de processamento do *hardware* disponível.

A distribuição realizada pelo escalonador tem impacto direto na quantidade de requisições atendidas e escalabilidade do sistema. Um mau funcionamento do escalonador, ou ainda, um escalonador que realize a distribuição dos dados de forma inadequada, pode concentrar todo o processamento em uma pequena fração das máquinas, fazendo com que surja um gargalo, mesmo existindo várias outras máquinas ociosas.

A eficiência do escalonador pode ser analisada em relação a um carga de requisições bem distribuída, em toda a região geográfica do dataset, ou uma carga concentrada em apenas uma região, por exemplo, nas áreas mais densas. Um bom escalonador deve lidar com as duas situações provendo o uso gradativo do *cluster* como um todo, sem que haja pontos com concentração de processamento.

As seguintes variáveis foram avaliadas para medir a eficiência do escalonador:

1. A quantidade de nós armazenados em cada máquina utilizada, que deve ser a mais homogênea possível, sem grande diferença na quantidade de partições armazenadas entre uma máquina e outra;
2. O nível de distribuição de processamento que mede a quantidade de máquinas acessadas durante a execução das consultas. O escalonador deve distribuir as partições de forma que mesmo uma grande quantidade de consultas em áreas geográficas específicas utilize o máximo de processamento possível;
3. A quantidade de máquinas acessadas na execução de cada consulta individual: Se uma consulta pode ser executada parcialmente por um subconjunto das máquinas do *cluster*, a latência entre sua requisição e resposta pode ser reduzida, já que mais trabalho é realizado paralelamente.

Item 1: A distribuição das partições para cada um dos datasets pode ser visualizada na Tabela 5.3. O dados coletados são para o dataset arizona com M=80, desmata com M=150 e vegeta com M=30. Nota-se na tabela que as partições estão bem distribuídas entre as máquinas, pois o desvio padrão da distribuição está baixo. No entanto, há uma tendência de aumento do desvio quando o número de máquinas aumenta ou o número de partições diminui (M maior). Os valores de M foram escolhidos de forma a demonstrar este comportamento do desvio padrão. Este aumento ocorre devido a implementação distribuída do método *round-robin* no Escalonador. Se existisse um só contador centralizado para iterar na lista de servidores disponíveis, a máxima diferença entre as máquinas seria de uma partição. Como cada servidor tem o seu próprio contador, algumas máquinas

ficam com mais partições porque cada dataset necessita de divisões em nós diferentes, e então alguns contadores podem ser mais incrementados que outros. Como a variação é pequena, este método distribuído ainda é melhor que criar um contador centralizado, devido ao custo de comunicação e o bloqueio distribuído necessário para incrementá-lo consistentemente.

Tabela 5.3: *Distribuição das partições geradas para cada dataset em 5, 10, 15 e 20 máquinas.*

Máquina	Arizona				Desmata				Vegeta			
	5	10	15	20	5	10	15	20	5	10	15	20
1	607	307	208	157	65	34	13	15	23	9	8	5
2	603	299	211	152	66	34	14	21	26	10	8	5
3	605	304	199	154	65	32	18	19	24	13	6	5
4	607	304	197	149	65	33	26	16	23	12	8	5
5	608	298	202	151	67	34	25	16	24	13	10	7
6		298	202	145		33	27	17		12	10	5
7		306	205	148		30	21	17		13	5	6
8		303	210	146		33	24	13		12	7	5
9		309	196	154		34	27	14		12	9	5
10		302	201	141		31	13	16		14	7	6
11			202	160			23	14			8	8
12			196	148			29	19			10	6
13			203	159			15	17			8	7
14			202	148			26	18			9	5
15			196	158			27	21			7	7
16				159				17				7
17				142				19				6
18				155				11				7
19				156				16				5
20				148				12				8
Desvio p.	2	3,8	4,88	5,76	0,89	1,4	5,74	2,72	1,22	1,49	1,46	1,08

Item 2: Além de deixar as partições distribuídas entre as máquinas, para tamanhos de M e quantidade de máquinas diferentes, é importante que o resultado da distribuição realizada pelo escalonador provoque o uso de todas as máquinas durante o processamento das requisições. Para analisar este comportamento, mediu-se o acesso às máquinas por cada janela consultada no dataset arizona. O resultado é apresentado nos mapas de calor da Figura 5.3. Um ponto preto no mapa indica que determinada janela – eixo x – não acessou nenhum nó na máquina especificada no eixo y. Este nível de acesso varia conforme a legenda à direita de cada mapa, onde a cor branco indica o nível máximo de acesso. Por exemplo, no mapa de 10 máquinas (canto superior direito), a escala mostra um acesso máximo de 12 partições para a janela 19 na máquina três.

Observando a sequência de valores para um determinado y – uma linha completa, da janela 0 a janela 29 – é possível verificar o acesso a uma máquina específica. Se uma

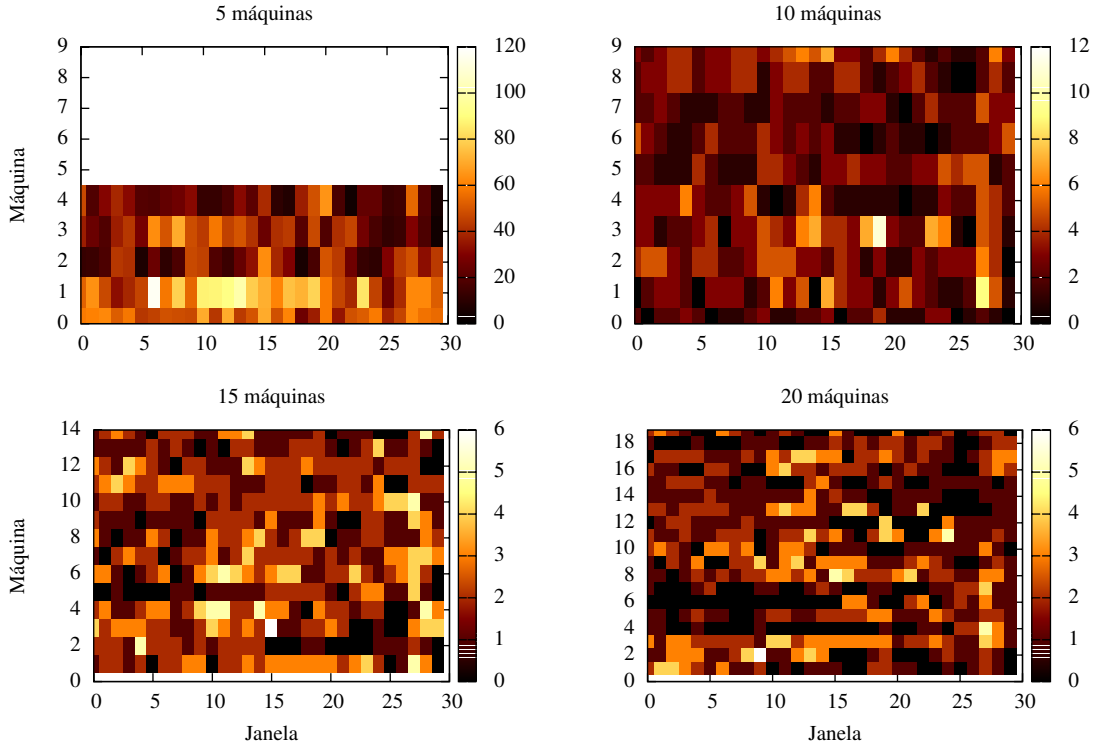

Figura 5.3: *Mapa de calor indicando a quantidade de partições acessadas por cada janela, no dataset Arizona em 5, 10, 15 e 20 máquinas.*

máquina não for acessada por nenhuma busca de janela, existirá uma linha totalmente preta no mapa na posição específica do *y*.

Nota-se que todas as máquinas foram acessadas pelo conjunto de janelas, pois não há, em nenhum dos mapas, uma linha totalmente preta. No entanto, à medida que a quantidade de máquinas aumenta, o uso de algumas delas tende a diminuir, como ocorre no mapa de 20 máquinas (canto inferior direito), máquina seis, janelas de 1 a 15.

Item 3: Ainda nos mapas da Figura 5.3, observando as colunas ao invés das linhas, pode-se identificar quais máquinas processaram cada uma das buscas de janela. Se somente um quadro colorido aparecesse para uma determinada janela, indicaria que a consulta havia sido processada integralmente por apenas uma máquina. Nota-se que o escalonador fez uma distribuição na qual cada janela individual foi processada pela maioria das máquinas. Novamente ocorre a tendência de diminuição no uso conforme o aumento de máquinas. Um exemplo ocorre no mapa de 20 máquinas, janela 17, quando sete máquinas não foram usadas para processar a busca de janela.

5.4 Escalabilidade Horizontal

O Teste de Escalabilidade Horizontal é uma forma de medir o quão eficiente o sistema é em aproveitar novas máquinas a sua disposição, para aumentar sua capacidade ou vazão. Diz-se que um sistema é escalável horizontalmente se ele aumenta proporcionalmente a sua capacidade ou vazão, conforme mais máquinas são adicionadas. Se a proporção é de 1:1, diz-se que ela é linear, ou seja, dobrando-se a capacidade de processamento, dobra-se também a vazão[4]. Uma proporção linear ou superior é desejável, mas nem sempre pode ser atingida devido à limitação física de dispositivos, como é o caso da rede de comunicação e da necessidade de sincronização de tarefas distribuídas no *cluster*.

Conhecer o fator de escalabilidade do sistema é importante para planejar o investimento em recursos computacionais. Dada uma previsão de aumento na demanda de utilização do serviço e conhecendo o fator de escalabilidade do sistema, é possível calcular a necessidade de investimento em novos equipamentos. O fato de o sistema ser escalável horizontalmente também possibilita o investimento gradativo em recursos computacionais, além de reutilizar equipamentos já adquiridos.

No DSI-RTree, este teste mede a eficiência do índice distribuído em utilizar novas unidades processadoras, disponibilizadas de forma não monolítica, ou seja, são adicionadas gradativamente ao sistema, sem substituição de partes do *hardware* existente. Espera-se que o aumento de máquinas, aliado ao particionamento dos dados, leve a uma escalabilidade próxima de linear na quantidade de requisições atendidas.

Foram utilizadas 5, 10, 15 e 20 máquinas para cada um dos datasets. Em cada combinação de datasets *versus* quantidade de máquinas foi executado um conjunto de testes. Em cada teste foi executada uma taxa fixa de requisições por segundo, com início sincronizado entre todos os clientes.

Cada requisição consiste no envio de uma janela pela aplicação cliente ao serviço, que intercepta uma parte do dataset. O sistema procura no índice pelos polígonos que possuem interseção com a mesma, e os envia como resposta para a aplicação cliente. Esta operação é conhecida na literatura como Busca de Janela (*Window Query*). As janelas (Figura 4.1) são colocadas em uma lista circular e enviadas sequencialmente, de acordo com a taxa de requisição por segundo e a duração de cada conjunto de testes.

A quantidade de requisições foi distribuída entre os 20 clientes, conforme a Tabela 5.4. Executou-se sempre uma quantidade total de requisições por segundo possível com algum múltiplo da quantidade de clientes. Para até 20 requisições por segundo, apenas uma requisição é enviada por cliente a cada segundo. A partir de 20, devido ao limite de máquinas no laboratório, combinações de clientes executando duas ou três

[4]O mesmo ocorre para frações ou múltiplos do dobro. Por exemplo, se triplica-se a capacidade de processamento, triplica-se também a vazão e assim por diante.

requisições a cada segundo foram efetuadas, de forma que cada um execute a mesma quantidade de requisições que os demais, evitando com isso distorções nas médias computadas.

Reqs/s	Quant. de Clientes	Reqs p/ cliente
1, 2, 3, .., 20	1, 2, 3, .., 20	1
22, 24, 26, .., 40	11,12,13, .., 20	2
42, 45, 48, .., 60	14,15,16, .., 20	3

Tabela 5.4: *Quantidade de máquinas clientes e requisições por segundo utilizadas na composição total de requisições por segundo enviadas ao sistema.*

Um intervalo entre cada experimento foi empregado para evitar interferência entre os testes, como, por exemplo, finalização da entrega das respostas pelos servidores aos clientes e execução do *Garbage Collector* da JVM.

5.4.1 Quantidade de Requisições Processadas

Os resultados da execução dos testes estão nos gráficos *a* e *b* da Figura 5.4. No Gráfico 5.4(a), a altura de cada barra vertical representa a quantidade de requisições atendidas por segundo e a legenda identifica a quantidade de máquinas utilizadas durante o teste.

No Gráfico 5.4(b), tem-se o fator de escalabilidade, dado pela fórmula: $f = \frac{r_f/r_i - 1}{m_f/m_i - 1}$, onde r é a quantidade de requisições atendidas por segundo inicial e final, e m a quantidade de máquinas utilizadas (inicial e final). Um fator igual a zero indica que não existe ganho de escalabilidade, ou seja, não surtirá efeito algum investir em mais *hardware* (linha nulo). Um fator igual a um indica que, se houver um aumento da demanda pelo serviço de 50%, deve-se aumentar a capacidade de processamento atual também em 50% para atendê-la. Para saber a necessidade de aumento da capacidade de processamento, divide-se o aumento da demanda pelo fator de escalabilidade. Ex: aumento da demanda de 80% e fator de escalabilidade de 0.60 implicam em aumento no processamento de 133.3% ($80\%/0.60 = 133.3\%$).

Há uma margem de erro de 2 requisições/seg para mais no gráfico da Figura 5.4(a). A margem de erro existe porque algumas taxas de requisições por segundo não foram testadas, por não existir um múltiplo exato da quantidade de clientes e quantidade de requisições por cliente.

Observando o gráfico, há um aumento na quantidade de requisições por segundo atendidas pelo sistema, a cada adição de máquinas. Aumentando de cinco para dez máquinas, ou seja dobrando a capacidade de processamento, a escalabilidade é maior que linear para os três datasets, com um fator de escalabilidade entre 1,1 e 1,3 (Figura

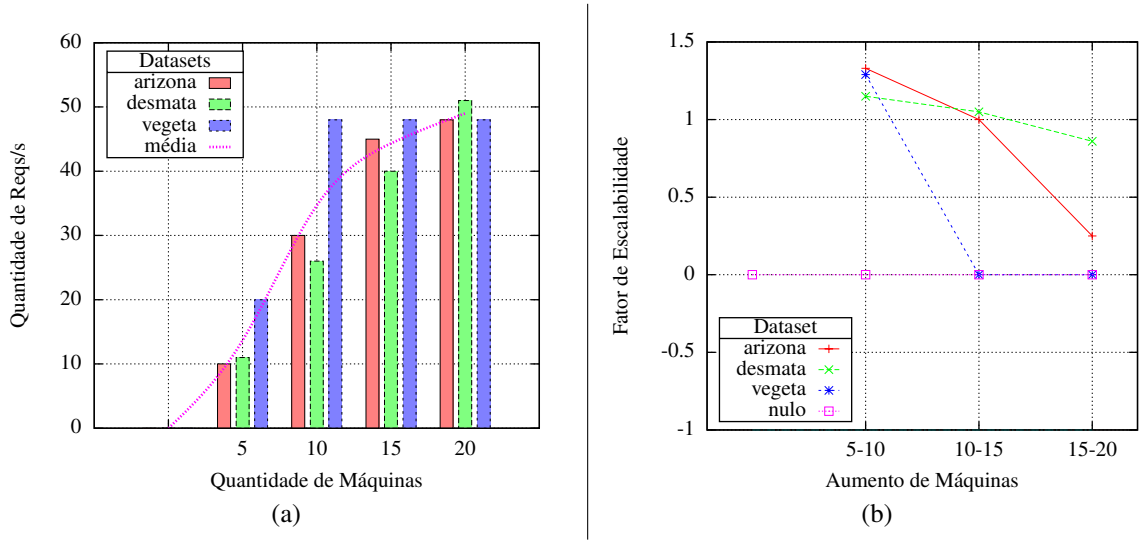

Figura 5.4: *Escalabilidade Horizontal do Sistema DSI-RTree, para 5, 10, 15 e 20 máquinas interligadas em* cluster *(a) e Fator de Escalabilidade (b).*

5.4(b)). A escalabilidade continua maior que linear para os datasets arizona e desmata quando ocorre o aumento de 10 para 15 máquinas, com uma pequena redução em relação ao aumento anterior.

No último aumento de máquina (15 para 20), apesar de ainda haver um ganho de escalabilidade, ele deixa de ser linear. Nota-se que a curva do fator de escalabilidade vai decrescendo à medida que se aumenta a quantidade de máquinas.

Observando várias variáveis durante a execução dos testes, como uso de CPU e rede das máquinas do *cluster*, a distribuição das partições conforme Seção 5.3, o tamanho dos MBRs criados no índice, a latência da troca de mensagens entre as máquinas, não foram identificados motivos para justificar a súbita redução na escalabilidade para o dataset arizona com o aumento de 15 para 20 máquinas. No entanto, foi observado que no dataset desmata esta redução é mais suave. A principal diferença entre os dois é o tipo das geometrias dos mesmos: no desmata as geometrias são bem mais esparsas do que no arizona (geometrias densas e próximas umas das outras). Devido ao complexo conjunto de variáveis, mais testes são necessários em trabalhos futuros para determinar se este comportamento tem relação com o tipo de dados, com a arquitetura proposta ou ainda se é um defeito no código implementado quando o número de máquinas aumenta.

Já o dataset vegeta apresenta um fator de escalabilidade nulo a partir de 10 máquinas. Apesar de o escalonador distribuir bem as partições geradas para o dataset, a pequena quantidade de polígonos é insuficiente para gerar um número significativo de partições para distribuir entre as máquinas, fazendo com que o custo de comunicação domine o tempo de resposta das consultas. Essa é uma indicação de que há um limite de divisão de datasets, o qual é atingido mais rapidamente em datasets pequenos.

5.4.2 Processamento de Grandes Datasets

Outra variável mensurada é a degradação da quantidade de requisições atendidas por segundo, à medida que datasets maiores são utilizados. A Escalabilidade Horizontal aliada à Qualidade do Índice, apresentada na Seção 5.2, deve permitir o processamento de datasets cada vez maiores, mantendo estável ou com pequena degradação a quantidade de requisições processadas por segundo. Em outras palavras, se a seletividade do índice é boa, processar buscas de janela também seletivas em datasets grandes ou pequenos requer um esforço semelhante, já que a redução no espaço de busca é eficiente.

Conforme mostrado no gráfico da Figura 5.4(a), as três barras verticais que representam os datasets testados estão em níveis semelhantes em cada uma das quatro quantidades de máquinas. Em alguns casos (cinco e dez máquinas) a quantidade de requisições processadas para o dataset arizona é maior que o dataset desmata, apesar deste segundo possuir quantidade inferior de polígonos.

Observando o gráfico da Figura 5.5, nota-se que a curva que representa o aumento no tamanho dos datasets testados lembra uma curva exponencial, e, apesar disso, a quantidade de requisições por segundo entre eles permanece praticamente igual em cada conjunto de máquinas testado.

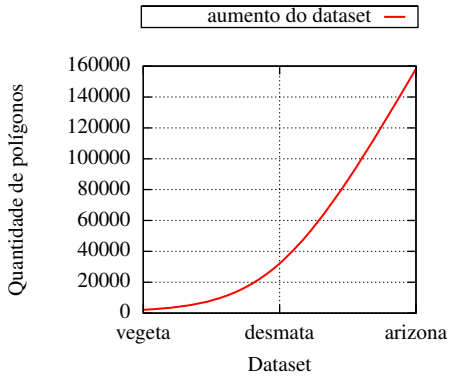

Figura 5.5: *Relação entre o tamanho dos datasets testados.*

5.5 Considerações Finais

Neste capítulo foram apresentados os testes de desempenho do índice distribuído construído. Em todos os aspectos investigados, o índice mostrou um comportamento dentro dos parâmetros esperados para um índice de qualidade, que possibilita a escalabilidade horizontal e que é capaz de processar grandes datasets geoespaciais.

Quanto à qualidade do índice, apesar de terem sido identificadas maneiras de reduzir o acesso aos falsos positivos, nenhuma foi implementada. Duas delas são: *i*) utilizar algoritmo de reinserção [5] e *ii*) implementar a técnica de objetos discrepantes

[41]. Acredita-se que o percentual de acesso a falsos positivos possa ser reduzido com a implementação desses métodos, e que isso melhore o tempo de reposta das consultas.

Apesar do DSI-RTree não implementar nenhum destes métodos, a quantidade de nós acessados é pequena se comparada ao total de nós do índice, mostrando que mesmo o algoritmo original da R-Tree com algumas adaptações da R*-Tree é eficiente para processamento de buscas de janela seletivas. Considera-se portanto a implementação das referidas técnicas como uma otimização possível para o DSI-RTree.

Um fato importante evidenciado é que o particionamento empregado, em conjunto com o escalonador simples *Round-Robin*, foi capaz de prover paralelismo no processamento de uma única consulta. Este fato além de reduzir a latência na entrega da resposta de consultas de janela seletivas é importante para garantir o balanceamento da carga do sistema mesmo se uma grande quantidade de buscas de janela semelhantes são requisitadas no sistema. Com uma mínima variação da janela – o que é mais próximo do caso real de uso – todas as máquinas do *cluster* seriam utilizadas eficientemente.

Apesar do paralelismo no processamento das cada consulta ser importante, sabe-se que ele aumenta o custo de comunicação, que pode prejudicar o desempenho do sistema. É necessário continuar a investigação a este respeito, extrapolando os limites da quantidade de máquinas e quantidade de partições, para verificar qual é o limite adequado de paralelismo. Este limite deve ser levado em conta na implementação do escalonador.

É ainda necessário explorar outros modelos de escalonadores para outros tipos de consultas, como *joins* espaciais. No *join*, o tempo de resposta é dominado pelas cópias necessárias entre dois índices espaciais e, neste caso, o escalonador *Round-Robin* possivelmente causaria uma grande troca de mensagens entre eles.

Apesar de apresentarem um ganho de escalabilidade horizontal, os números de requisições por segundo atendidas foram limitados pela quantidade de máquinas disponíveis no laboratório para simulação de clientes do serviço. O ideal é que mesmo em um teste de 60 requisições por segundo, cada uma seja feita em um computador diferente, pois este cenário é mais próximo do ambiente real de produção. Porém, como é difícil ter um ambiente destes, poder-se-ia continuar executando mais de uma requisição por segundo em cada computador, desde que o mesmo seja uma máquina mais potente, possivelmente com a quantidade de núcleos equivalente à quantidade de requisições por segundo.

Os testes foram executados em datasets densos (arizona, vegeta) e esparsos (desmata), todos eles com polígonos complexos. Acredita-se que este seja um dos piores cenários para este tipo de índice. Porém, particularidades podem aparecer para índices com objetos geométricos diferentes, como, por exemplo, linhas (rios, avenidas) ou pontos (comércios, pontos turísticos). A investigação das variáveis apresentadas também deve ser feita para datasets ainda maiores, a fim de verificar o comportamento do sistema.

CAPÍTULO 6

Trabalhos Correlatos

Neste capítulo foram discutidas as propostas de maior relevância identificadas na literatura, e que possuem uma maior interseção com o DSI-RTree. Procurou-se por trabalhos que enfatizam a declusterização e organização das partições do índice, de forma a obter escalabilidade ou menor tempo de resposta no processamento de consultas em dados espaciais.

Em cada uma das seções seguintes discutiu-se um trabalho específico, destacando sua área de aplicação e os pontos não abordados de maneira a evidenciar as principais contribuições do presente trabalho. Algumas propostas semelhantes foram agrupadas em uma só seção, como é o caso da M-RTree e MC-RTree.

Os artigos originais na língua inglesa utilizam os termos *clustering* e *declustering* para se referir ao agrupamento de polígonos dos datasets em blocos e a distribuição destes blocos entre as máquinas do *cluster*, respectivamente. Como os termos são antônimos, para não dar a impressão de que se estava agrupando e desagrupando em seguida, traduziu-se *clustering* como *particionamento*, ou seja, a criação das partições que serão distribuídas e, *declustering*, como o *escalonamento* destas partições.

Por fim, na Seção 6.6, foi realizada uma discussão geral a respeito do estado da arte de sistemas distribuídos para indexação de dados espaciais e possíveis direções futuras de pesquisa.

6.1 Paradise

Patel et al. [29] descreve a implementação de um ambiente completo de banco de dados paralelo para arquitetura NOW (*Network of Workstations*) chamado Paradise. Este banco de dados é empregado para armazenamento e processamento de dados espaciais e também dados unidimensionais. No entanto, a ênfase do trabalho é o processamento de imagens de satélites (*raster images*), ao contrário do foco deste trabalho que visa o processamento de dados vetoriais.

Os dados são distribuídos em Data Servers que são os componentes que representam os computadores utilizados para armazenar os dados e processar as consultas,

conforme ilustrado na Figura 6.1. Um coordenador mestre chamado de Query Coordinator é empregado para intermediar as requisições de aplicações clientes e controlar a execução das consultas e distribuição dos dados.

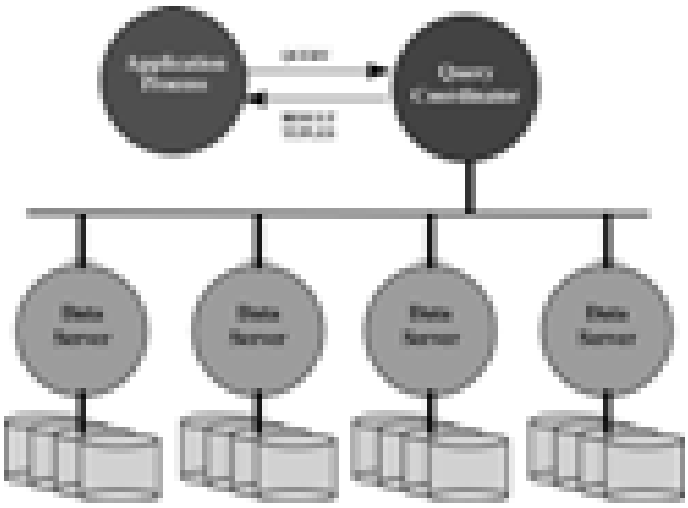

Figura 6.1: *Visão Geral da Arquitetura do Banco de Dados Paradise [29].*

O particionamento dos dados é realizado de forma espacial (*Spatial Declustering*), ou seja, divide-se o espaço geográfico total em múltiplos fragmentos, sem levar em conta os limites dos objetos espaciais. Esta forma de particionamento funciona bem para imagens *raster*, mas para objetos espaciais vetoriais ocorrem alguns problemas, conforme discutido no próprio artigo. São eles:

- Não é possível dividir o dataset de forma que os objetos espaciais mais complexos (retângulos, por exemplo) nunca cruzem as partições. Este fato dificulta o processamento das consultas porque para produzir resultados corretos é necessário replicar os objetos nas partições que o mesmo intercepta (ou consultar todas as partições, o que seria inviável).
- A divisão do espaço sem considerar áreas densas ou esparsas faz com que algumas partições tenham mais objetos que outras, o que provoca um desbalanceamento no processamento das consultas.

Apesar da duplicação dos objetos implicar em uma etapa a mais no processamento para remoção de possíveis respostas duplicadas, caso duas ou mais partições que contenham o objeto sejam consultadas, esta foi a forma implementada pelo artigo para lidar com este problema.

Em relação aos dados esparsos e densos no mesmo dataset, a abordagem implementada foi a de divisão do espaço em micro partições, o que provoca um aumento no problema de cruzamento de partições. Outras possível soluções discutidas em outros ar-

tigos são citadas, mas não foram implementadas e não dizem respeito a outras formas de particionamento que não a adotada (*Spatial Declustering*).

Outro aspecto não discutido no artigo diz respeito à possibilidade de sobrecarga no Query Coordinator, já que todas as requisições feitas pelas aplicações clientes necessitam ser tratadas pelo mesmo.

6.2 M-RTree e MC-RTree

Em [23], os autores propõem a Master RTree (M-RTree) com uma forma de particionamento de dados espaciais diferente da utilizada no Paradise. Nela, um servidor mestre *M* mantém todos os nós internos de uma R-Tree e os nós folha são divididos entre os demais servidores do *cluster* ($C1, C2, C3$), conforme ilustrado na Figura 6.2.

Dois argumentos foram utilizados para justificar o agrupamento de todos os nós internos em um só servidor:

- O espaço físico necessário para armazenamento é pequeno porque são utilizados somente MBRs (dois pontos) nos nós internos, e isso possibilita manter todos eles em um só computador, mesmo para datasets grandes;
- A redução da quantidade de ponteiros entre os servidores minimiza o custo de comunicação.

A curva de preenchimento de espaço de Hilbert é empregada para distribuir os nós folha. Calcula-se o valor de Hilbert dos centros de todos os polígonos para agrupá-los por proximidade, e os grupos são escalonados entre os servidores escravos utilizando um algoritmo *Round-Robin*.

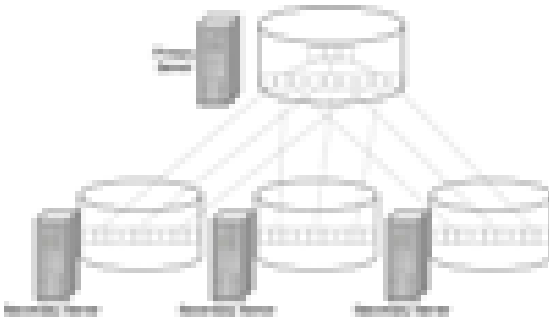

Figura 6.2: *Particionamento na M-RTree: Um servidor mestre mantém os nós internos do índice, e as folhas são divididas entre os servidores clientes (ou escravos) [26].*

Schitzer e Leutenegger [33] propuseram a MC-RTree, uma extensão da M-RTree [23], na qual cada servidor escravo constrói subárvores R-Tree com um subconjunto do dataset, ao invés de armazenar as folhas separadamente, como na M-RTree.

Uma árvore é construída no servidor mestre, da mesma forma que na M-RTree. Porém, há somente um ponteiro para cada servidor escravo, ao invés dos vários ponteiros na M-RTree, cada um apontando para um dos nós folha armazenados nos servidores escravos. O objetivo desta redução de ponteiros é a diminuição de itens para processar

no servidor mestre e a redução da quantidade de mensagens na rede. A Figura 6.3 ilustra a arquitetura da MC-RTree. Cada triângulo na figura representa uma árvore R-Tree.

Além da técnica de escalonamento *Round-Robin* utilizada na M-RTree, algumas outras técnicas são testadas. No entanto, a conclusão do artigo aponta, surpreendentemente, que a escolha de qualquer um deles não produz diferenças significativas no tempo de resposta das consultas de janela testadas na arquitetura proposta.

Certamente é possível armazenar todos os nós diretórios em um só computador, mesmo considerando a indexação de vários datasets, tanto na M-RTree quanto na MC-RTree. Porém, centralizar todas as comparações geométricas de uma grande quantidade de consultas de janela, porventura executadas simultaneamente, faz com que o servidor mestre seja um gargalo de processamento. Principalmente considerando que este mesmo servidor agrega as respostas parciais dos servidores escravos para entregá-las às aplicações clientes.

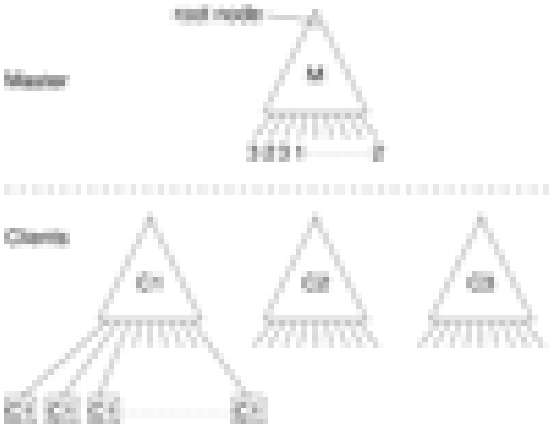

Figura 6.3: *Particionamento na MC-RTree: As folhas continuam sendo armazenadas somente nos nós escravos como na M-RTree, mas cada servidor cria uma árvore R-Tree com suas entradas [33].*

Considerando a execução de consultas mais complexas nestas arquiteturas, como consultas *join*, observa-se que, apesar da haver uma grande redução da comunicação, a maior parte do algoritmo será executada inevitavelmente de forma sequencial, apenas no servidor mestre, não aproveitando o poder de processamento do *cluster*.

Outra limitação das propostas é que a discussão é predominantemente em datasets estáticos, cujos objetos espaciais são totalmente conhecidos previamente. O índice e suas partições são montados de forma *bottom-up*, utilizando técnicas de pré-processamento para particionamento dos objetos (Curva de Hilbert ou STR [25]).

Por um lado, a construção *bottom-up* do índice evita a complexidade de divisões e ajustes da construção da R-Tree original e constrói árvores mais balanceadas. Porém, limita-se o escopo de aplicações que podem se beneficiar do sistema. O índice necessitaria ser reconstruído completamente a cada mudança em um dataset dinâmico, e isto é um obstáculo para aplicações que dependem de respostas em tempo real. Obviamente, alternativas foram propostas para construções de árvores *bottom-up* com posterior uso com datasets dinâmicos. Porém, nenhuma é discutida no artigo.

Na MC-Rtree foi realizada uma avaliação da redução do tempo médio de resposta de consultas de janela, conforme o aumento de máquinas do *cluster*. Os autores

apresentaram um fator de aceleração (*speed-up*) no tempo médio de resposta próximo de linear. Porém, apesar de existir este ganho no tempo de resposta, ele foi obtido somente para consultas de janela que retornam mais de 1000 itens, conforme discutido no trabalho.

Em nenhuma das duas propostas foram realizados testes a respeito da escalabilidade horizontal no processamento de requisições, conforme o aumento de máquinas no *cluster*. Uma avaliação entre a quantidade de mensagens trocadas e as diferentes formas de escalonamento testadas também não foi realizada, deixando este aspecto, bastante citado no texto, sem uma discussão apropriada.

6.3 *Storing Spatial Data on NOW*

Na revisão da literatura realizada, a proposta apresentada por [4] é o trabalho mais completo encontrado, em relação ao conjunto de experimentos realizados e as contribuições propostas.

Os autores exploram a distribuição de uma árvore R-Tree em um *cluster*, propondo uma arquitetura similar à MC-RTree[1], que também agrupa os nós diretórios em um servidor mestre e distribui as folhas em servidores escravos.

No trabalho é proposta uma taxonomia para categorização de possíveis implementações de particionamento e escalonamento de partições em índices espaciais distribuídos:

- Unidade de alocação: Por elemento, por Bloco e por Subárvore;
- Frequência de alocação: Estática ou Overflow;
- Política de distribuição: Clusterização, Declusterização e Balanceamento (Round-Robin).

Os autores testam e apresentam resultados para cinco combinações mais promissoras destes três parâmetros e concluem que o esquema Subárvore/Overflow/Balanceamento é o mais adequado para consultas de janela seletivas ou não seletivas.

O esquema implementado em nosso trabalho não foi testado. Apesar de também implementar frequência de alocação de *overflow*, difere na unidade de alocação empregada e na política de distribuição (forma de escalonamento). Os autores empregam unidade de alocação de Subárvore, na qual são criadas subárvores R-Tree nos servidores escravos, enquanto em nosso trabalho utilizou-se unidade de alocação de bloco, conforme definido na Seção 3.1.

[1] De acordo com os autores e as datas de publicação, os dois trabalhos foram criados em paralelo, na mesma época.

Apesar de afirmarem que a política de distribuição é de balanceamento, nota-se que ela é uma mistura de clusterização e balanceamento, já que é feito um tratamento de colocalização no escalonamento dos polígonos, com a imposição de um parâmetro que limita o aumento máximo de área do MBR do nó raiz das subárvores, no momento da inserção de novos itens. Um número máximo de elementos nas subárvores também é empregado, o que de certa forma justifica a política de balanceamento. Porém, este limite foi definido empiricamente e maiores detalhes a respeito de sua definição não foram discutidos.

Apesar de não considerados nos testes, algumas recomendações a respeito dos esquemas de Bloco são citados em uma versão mais extensa do artigo [3]. Os autores apontaram que o escalonamento nesta frequencia de alocação pode ser feito tanto por clusterização, agrupando partições colocalizadas, quanto por balanceamento, utilizando *round-robin*, e sugerem que este tipo de particionamento pode ser uma alternativa viável em redes de baixa latência, devido a seus maiores custos de comunicação.

Em nosso trabalho discutimos parâmetros de controle deste custo de comunicação baseando na altura do índice e no correto valor de M, além da qualidade da distribuição dos polígonos e o baixo acesso aos nós folha da estrutura durante a execução das consultas. Além disso, uma grande mudança ocorreu nas redes computacionais desde a época de publicação do artigo. Os autores utilizaram rede Ethernet 100 Mbit/seg e uma rede Myrinet 1.28Gbit/seg. Nesta última, conforme discutido no trabalho, os tempos de respostas das consultas foram consideravelmente menores.

Nossa proposta também difere na forma de entrega de respostas, que é feita através de um canal de comunicação exclusivo, ao invés de agregá-las no servidor mestre para, posteriormente, enviar para a aplicação cliente. A utilização deste servidor mestre para processar todos os nós internos e a agregação das entregas de resposta, da mesma forma que na M-RTree e na MC-RTree, pode fazer com que ele se torne um gargalo do sistema.

6.4 SD-RTree

A SD-RTree [28] é uma árvore binária – semelhante à árvore AVL – que usa o princípio de organização dos MBRs das R-Trees. Apesar de ter sido proposta com o mesmo objetivo de indexação de dados espaciais, é uma estrutura bem distinta das demais variantes de árvores R.

Dois problemas principais existem na abordagem binária proposta:

- Ao contrário das R-Trees, que são balanceadas por natureza, em uma árvore binária é necessário implementar algoritmos de balanceamento. Quanto maior os datasets

armazenados, maior a altura da estrutura e consequentemente maior a necessidade de comunicação para rebalancear a árvore durante a construção do índice;
- Devido à altura da estrutura, a quantidade de mensagens necessárias para processamento das consultas também é muito grande;

Para amenizar a sobrecarga de mensagens nos níveis superiores da estrutura, os autores propõem o uso de *cache* parcial da estrutura na aplicação cliente. À medida que são feitas requisições, partes da árvore são enviadas para a aplicação cliente formando um *cache*, conforme ilustrado na Figura 6.4. Antes de enviar novas solicitações, a aplicação consulta este *cache* e redireciona a requisição para um nó possivelmente mais adequado de um nível inferior. Se o mesmo está desatualizado em relação à árvore original, um redirecionamento da requisições é realizado pelos servidores.

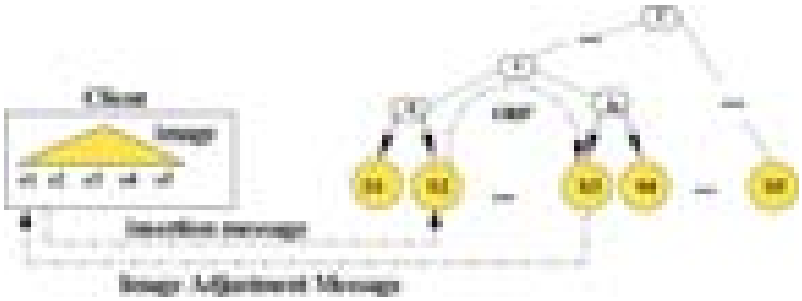

Figura 6.4: *Representação da estrutura binária da SD-RTree e a imagem parcial da estrutura na aplicação cliente [28].*

O uso deste *cache* na aplicação cliente pode ser uma restrição em situações nas quais a aplicação cliente é executada em dispositivos com recursos limitados (celulares, *smartphones*), ou ainda quando a aplicação cliente e os servidores estão em redes distintas.

Não foi discutido no artigo o tamanho do *cache* gerado, mas observa-se que sua eficiência depende de manter a maior quantidade de folhas possíveis no mesmo, já que a quantidade de folhas geradas pelo particionamento binário é grande. O *cache* pode também se tornar ineficiente em datasets dinâmicos, alterados constantemente.

Técnicas de escalonamento de partições também não foram empregadas para redução da troca de mensagens e o agrupamento de nós folha nos servidores também não foram abordados.

Esta proposta não levou em consideração os trabalhos apresentados anteriormente[2]. Acredita-se, devido às discussões anteriores em relação ao controle de concorrência e troca de mensagens, que esta alternativa binária tenha um custo de comunicação muito alto, mesmo para as redes de comunicação atuais, e que mais testes devem ser fei-

[2]Tais artigos foram publicados quase nove anos antes.

tos com datasets reais e ambientes de testes mais hostis, forçando o mecanismo de *cache* implementado em diferentes cenários.

6.5 Hadoop-GIS

O sistema Hadoop-GIS proposto em [22] é uma proposta recente encontrada na literatura, publicada em 2009, e também a que se aplica a um ambiente menos comum aos discutidos anteriormente.

São apresentadas no trabalho duas alternativas para processamento paralelo utilizando as plataformas de processamento distribuído Hadoop[3] e MPI[4]. No Hadoop-GIS é efetuada uma comparação dessas plataformas com o banco de dados PostGis.

A proposta não emprega nenhuma das formas de particionamento discutidas nos trabalhos anteriores e também não emprega nenhum método de indexação dos dados (R-Tree ou similares). Os dados são divididos em partições de tamanho fixo em quantidade de bytes e distribuídos pelo *cluster* de forma homogênea, em relação à quantidade de partições.

Apesar de serem alternativas ao processamento de dados GIS de forma paralela, a ausência de indexação tanto nas duas implementações realizadas, quanto nas relações no banco de dados PostGis, causa uma enorme degradação no desempenho, devido à comparação através de força bruta entre todos os polígonos de todos os datasets.

Ao contrário da afirmação do artigo, acredita-se que este caso se aplique a uma quantidade pequena de aplicações, pois, a maioria das aplicações que realizam consultas em dados GIS vetoriais que temos conhecimento conseguem utilizar índices para acelerar o processamento das consultas que realizam.

Em testes que executamos em laboratório, as mesmas operações, sobre os mesmos dados utilizados no artigo, utilizando o banco de dados Postgis são processadas em menos de um minuto com a correta utilização de índices na estrutura, enquanto levam 55 horas sem a utilização de índices, de acordo com o artigo. Nossos testes foram executados em computadores *desktop* comuns, com processadores Intel Dual Core 2.8 GHz e quatro Gb de memória RAM.

6.6 Considerações Finais

Apesar da grande quantidade de pesquisas na literatura sobre R-Trees em geral, pouca atenção têm sido dedicada a sua utilização em arquiteturas distribuídas. Com base

[3] http://hadoop.apache.org/
[4] http://www.mpi-forum.org/

em nossa revisão bibliográfica e também observando as revisões de literatura feitas em [26, 18], ambas publicadas no ano de 2006, poucos trabalhos têm sido desenvolvidos nesta área.

Existem outros trabalhos na literatura que também discutem arquiteturas distribuídas utilizando árvores R-Tree, que não foram discutidos por darem ênfase a um aspecto diferente: a distribuição de conteúdo em redes P2P. Dentre eles [27, 35, 39, 8].

Um aspecto não discutido nas propostas é a possibilidade de construção do índice de forma paralela e, consequentemente, a necessidade de bloqueios distribuídos para garantir a consistências dos índices gerados. Nenhum dos artigos discutidos possui referências para este aspecto da construção. Nosso trabalho apresenta estes desafios e propõe a utilização de bloqueios distribuídos para controle de concorrência, visando prover mais eficiência na construção de índices para datasets dinâmicos de forma consistente.

A taxonomia proposta em [4] pode ser expandida para evitar as ambiguidades apresentadas anteriormente e para contemplar as novas possibilidades de sistemas distribuídos totalmente em memória, ou ainda que não façam uso de índices no processamento de consultas, como o caso de [22].

Depois da publicação de [4] no ano de 1999, um grande intervalo se passou sem apresentação de publicações que evoluíssem o estado da arte para este tipo de sistema. As duas propostas que encontramos posteriores a este artigo abordaram aspectos muito diferentes dos apresentados anteriormente, sem que fosse feita a confrontação das ideias apresentadas anteriormente com as novas soluções propostas.

A maior parte dos artigos foram publicados em uma época em que a rede era padrão 10 Mbit/seg ou 100 Mbit/seg. Desde então, tanto o *hardware* quanto o *software* evoluíram e novas propostas baseadas na maior disponibilidade de memória e maior velocidade de rede podem ser pesquisadas. Nosso trabalho apresenta novidades em relação a esses aspectos, mas outras possibilidades com certeza existem e podem ser exploradas.

De forma geral, poucos aspectos da construção e processamento do índice tem sido investigados pelos trabalhos publicados, de acordo com nosso conhecimento sobre a literatura existente. Este trabalho apresenta ainda novidades não discutidas na literatura como parâmetros importantes para definição do valor de M e suas implicações na escalabilidade e desempenho do sistema, aspectos do controle da comunicação entre os computadores através do correto dimensionamento da altura dos índices, o balanceamento do índice distribuído através do Escalonador, um componente flexível na arquitetura que pode ser expandido e que possibilita a exploração do maior paralelismo possível da arquitetura.

CAPÍTULO 7

Conclusão

Processar grandes datasets espaciais de forma distribuída é altamente complexo e impacta fortemente no projeto da arquitetura de um sistema de geoprocessamento. Ao contrário de sistemas centralizados, a arquitetura do DSI-RTree foi projetada para tirar o máximo proveito do poder de armazenamento e processamento paralelo de um *cluster* de computadores para atender, de forma escalável e eficiente, o processamento de consultas espaciais. De fato, a principal contribuição deste trabalho consiste de uma plataforma capaz de escalar horizontalmente tanto em número de clientes simultâneos atendidos pelo sistema quanto no tamanho dos datasets manipulados. Os testes realizados em ambiente real de um *cluster* de computadores demonstraram sua capacidade de escalabilidade e a viabilidade em prover serviços de cruzamento de dados espaciais em larga escala.

Para que aplicações SIG e LBS possam ser, de fato, implantadas e utilizadas por muitos usuários é fundamental que os sistemas de geoprocessamento sejam escaláveis. A infra-estrutura física de *cluster* de computadores com muitos nós já está disponível na *Computação nas Nuvens*, e o que ainda falta são infra-estruturas de *softwares* que sejam capazes de explorar esse potencial de processamento e armazenamento para prover serviços de forma geral.

Com este intuito, outros projetos correlatos foram propostos na literatura [23, 33, 4, 28, 22], mas esses, em sua maioria, não discutem o fator de escalabilidade horizontal de suas propostas, as implicações da definição do tamanho de partição dos dados no índice distribuído, inserção ou atualização de dados de forma concorrente no sistema e a qualidade do índice distribuído, conforme apresentado neste trabalho. Até onde conhecemos, não encontramos nenhum outro trabalho na literatura que investiga o impacto da definição do tamanho de partições para distribuição dos dados em um índice espacial distribuído com unidade de alocação de bloco. Isto é uma questão crucial em um sistema espacial distribuído e gera implicações diretas na quantidade de mensagens trocadas na rede, na qualidade do índice e, consequentemente, no desempenho e escalabilidade do sistema. A maioria desses trabalhos são implementações "parciais" com propósito de pesquisa. Muitas questões de implementação são negligenciadas e não são considerados controle de concorrência, estresse de protocolos do kernel, da suíte TCP/IP, dispositivos

de rede, dentre outros. Essas questões não são tratadas de forma integral em uma única plataforma de geoprocessamento distribuído [26]. Muitos dos trabalhos correlatos simplificaram ou simularam parte de seu projeto e não aprofundaram seus experimentos em ambientes de *cluster* reais. Isto dificulta e compromete a interpretação de seus resultados e limita a aplicabilidade das soluções propostas. Ao contrário da maioria desses trabalhos, os resultados do DSI-RTree foram obtidos em ambientes reais de experimentação considerando as limitações e restrições dos ambientes de execução utilizados.

Apesar dos resultados apresentados neste trabalho confirmarem o potencial de escalabilidade[1], eles ficaram limitados pela capacidade de processamento das máquinas e da rede utilizados. Em testes preliminares realizados em um *cluster* de máquinas HPC nas *Nuvens da Amazon*, foram obtidos excelentes resultados, com consultas de janela para um dataset da Navteq de Pontos de Interesse (e.g., Restaurantes, Postos de Gasolina) da cidade de São Paulo, conseguindo atender até 1500 requisições por segundo com um tempo médio de resposta de 1.81 segundo.

Inovações tecnológicas radicais geradas por grande corporações (e.g., Google, Yahoo) mostram que, para prover serviços que possam tirar proveito das tecnologias e demandas instaladas é necessário projetar e desenvolver sistemas altamente escaláveis que possam explorar o potencial de mineração e cruzamento de dados de grande quantidade de usuários. Apesar de ser algo desejável, essa possibilidade não é comumente explorada devido à alta complexidade de desenvolvimento de sistemas distribuídos avançados que permitam experimentar e avaliar novas ideias que possam evoluir o estado da arte da computação e ditar a criação de novos algoritmos.

Por isso, uma contribuição real deste trabalho consiste da implementação do núcleo do DSI-RTree, pois este, além dos resultados demonstrados, permitirá a experimentação de diferentes algoritmos de processamento de dados espaciais em larga escala, viabilizando a criação e avaliação de novos algoritmos de escalonamento e distribuição de dados espaciais, algoritmos de *join* distribuídos, processamento de operações espaciais de forma paralela, processamento de *workflow* com muitas operações espaciais encadeadas, dentre outros. De fato, esta plataforma nos permitirá verticalizar e desenvolver, de forma prática, sistemas reais que permitam experimentar novos conceitos e ideias relacionadas a geoprocessamento distribuído em grande escala. No entanto, muito ainda pode ser feito para evoluir o núcleo do DSI-RTree. Alguns desses trabalhos futuros são discutidos na Seção a seguir.

Devido à abrangência do escopo e a quantidade de desafios envolvidos no desenvolvimento de uma arquitetura de processamento de dados espaciais distribuída, muitos requisitos importantes não foram tratados devido à complexidade de sua implementação

[1] Testes realizados em um *cluster* de 40 máquinas - clientes e servidoras.

e a restrição de tempo para o desenvolvimento deste trabalho.

Algumas análises apresentadas nos capítulos de resultados devem ser estendidas para completar a verificação de comportamentos apresentados, como é o caso da redução da escalabilidade observada com o aumento de máquinas no *cluster* (Seção 5.4.1), e do aumento de interseção a nós diretórios da Seção 4.3. Além disto, é necessário investigar o desempenho do sistema no processamento de datasets maiores e com tipos de dados diferentes dos utilizados nos experimentos.

Técnicas apresentadas recentemente com extensões da R-Tree original também devem ser consideradas, como é o caso da R^0-Tree que propõe ideias para reduzir o espaço morto através do tratamento de geometrias discrepantes nos nós internos da árvore. É necessário ainda avaliar artigos [13, 38] que propõem algoritmos de divisão mais eficientes para redução de acesso aos nós falsos do índice. Outra possibilidade complementar é o uso de geometrias complexas nos nós internos, ao invés de MBRs: técnicas como o algoritmo *Convex Hull* podem reduzir significativamente o espaço morto e sobreposição nos nós para datasets com polígonos muito extensos, como é o caso de datasets com rios e rodovias.

Métodos de amostragem devem ser implementados para obter o nível de preenchimento médio dos nós do índice, e com isso proporcionar a correta definição do tamanho das partições para um dataset dinâmico já na sua primeira indexação no sistema.

Outros algoritmos de busca também devem ser implementados na plataforma, juntamente com escalonadores que acelerem a sua execução. Exemplos importantes de consultas que devem ser implementadas são os *joins* [16, 9, 10, 42, 30] e *skyline query* [41].

Para atender outras demandas reais de aplicações SIG e LBS, deve-se implementar mecanismos de tolerância a falhas e criação de réplicas dinâmicas baseada no nível de sobrecarga do sistema para garantir requisitos de qualidade de serviço e escalabilidade sob cenários extremos.

Para atender a possibilidade de inserção e consultas simultâneas no sistema, deve-se investigar protocolos de *commit* (e.g., (*two-phase-commit*)) que garantam a consistência dos resultados retornados pelos algoritmos de busca no índice distribuído.

Referências Bibliográficas

[1] ACHARYA, S.; POOSALA, V.; RAMASWAMY, S. **Selectivity estimation in spatial databases**, Mar. 5 2002. US Patent 6,353,832.

[2] AN, N.; KANTH, R.; KOTHURI, V.; RAVADA, S. **Improving performance with bulk-inserts in Oracle R-trees**. In: *Proceedings of the 29th international conference on Very large data bases-Volume 29*, p. 951. VLDB Endowment, 2003.

[3] AN, N.; LU, R.; QIAN, L.; SIVASUBRAMANIAM, A.; KEEFE, T. **Storing spatial data on a network of workstations**. *Technical Report CSE-98-006 - The Pennsylvania State University*, p. 13, may 1998.

[4] AN, N.; LU, R.; QIAN, L.; SIVASUBRAMANIAM, A.; KEEFE, T. **Storing spatial data on a network of workstations**. *Cluster Computing*, 2(4):259–270, 1999.

[5] BECKMANN, N.; KRIEGEL, H.; SCHNEIDER, R.; SEEGER, B. **The R*-tree: an efficient and robust access method for points and rectangles**. *ACM SIGMOD Record*, 19(2):322–331, 1990.

[6] BENTLEY, J. **Multidimensional binary search trees used for associative searching**. *Communications of the ACM*, 18(9):517, 1975.

[7] BERNHARDSEN, T. **Geographic information systems: an introduction**. Wiley, 2002.

[8] BIANCHI, S.; FELBER, P.; POTOP-BUTUCARU, M. **Stabilizing Distributed R-trees for Peer-to-Peer Content Routing**. *IEEE Transactions on Parallel and Distributed Systems*, 2009.

[9] BRINKHOFF, T.; KRIEGEL, H.; SEEGER, B. **Efficient processing of spatial joins using R-trees**. In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, p. 237–246. ACM New York, NY, USA, 1993.

[10] BRINKHOFF, T.; KRIEGEL, H.; SEEGER, B. **Parallel processing of spatial joins using R-trees**. In: *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON*

DATA ENGINEERING, p. 258–265. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 1996.

[11] CHEN, Y.; OTHERS. **A study of concurrent operations on R-trees**. *Information Sciences*, 98(1-4):263–300, 1997.

[12] DEWITT, D.; GRAY, J. **Parallel database systems: the future of high performance database systems**. *Communications of the ACM*, 35(6):85–98, 1992.

[13] GONG, J.; KE, S. **A new node-split algorithm in R-tree based on spatial clustering**. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 5, p. 2081–2085. IEEE, 2010.

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

[15] HOEL, E.; SAMET, H. **A qualitative comparison study of data structures for large line segment databases**. In: *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, p. 205–214. ACM, 1992.

[16] HUANG, Y.; JING, N.; RUNDENSTEINER, E. **Spatial joins using R-trees: Breadth-first traversal with global optimizations**. In: *Proceedings of the International Conference on Very Large Data Bases*, p. 396–405. INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS (IEEE), 1997.

[17] INC, O. G. C. **OGC Standards and Specifications**, 2010. `http://www.opengeospatial.org/standards`.

[18] JACOX, E. H.; SAMET, H. **Spatial join techniques**. *ACM Trans. Database Syst.*, 32(1):7, 2007.

[19] KAMEL, I.; FALOUTSOS, C. **Parallel R-trees**. In: *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, p. 195–204. ACM, 1992.

[20] KAMEL, I.; FALOUTSOS, C. **Hilbert R-tree: An Improved R-tree using Fractals**. In: *Proceedings of the 20th International Conference on Very Large Data Bases*, p. 509. Morgan Kaufmann Publishers Inc., 1994.

[21] KANTH, R.; SERENA, D.; SINGH, A. **Improved concurrency control techniques for multi-dimensional index structures**. In: *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*, p. 580–586. IEEE, 2002.

[22] KERR, N. **Alternative Approaches to Parallel GIS Processing**. *Arizona State University - Master Thesis*, 2009.

[23] KOUDAS, N.; FALOUTSOS, C.; KAMEL, I. **Declustering spatial databases on a multi-computer architecture**. *Advances in Database Technology EDBT'96*, p. 592–614, 1996.

[24] LANG, C.; SINGH, A. **Modeling high-dimensional index structures using sampling**. *ACM SIGMOD Record*, 30(2):389–400, 2001.

[25] LEUTENEGGER, S.; LOPEZ, M.; EDGINGTON, J. **STR: A simple and efficient algorithm for R-tree packing**. In: *Data Engineering, 1997. Proceedings. 13th International Conference on*, p. 497–506, 1997.

[26] MANOLOPOULOS, Y.; NANOPOULOS, A.; THEODORIDIS, Y. **R-trees: Theory and Applications**. Springer Verlag, 2006.

[27] MONDAL, A.; LIFU, Y.; KITSUREGAWA, M. **P2pr-tree: An r-tree-based spatial index for peer-to-peer environments**. *Current Trends in Database Technology-EDBT 2004 Workshops*, p. 516–516, 2005.

[28] MOUZA, C.; LITWIN, W.; RIGAUX, P. **SD-Rtree: A Scalable Distributed Rtree**. In: *Proc. of IEEE ICDE Conference, Istanbul, Turkey*, p. 296–305, 2007.

[29] PATEL, J.; YU, J.; KABRA, N.; TUFTE, K.; NAG, B.; BURGER, J.; HALL, N.; RAMASAMY, K.; LUEDER, R.; ELLMANN, C.; OTHERS. **Building a scaleable geo-spatial DBMS: technology, implementation, and evaluation**. In: *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, p. 347. ACM, 1997.

[30] PATEL, J.; DEWITT, D. **Clone join and shadow join: two parallel spatial join algorithms**. In: *Proceedings of the 8th ACM international symposium on Advances in geographic information systems*, p. 54–61. ACM New York, NY, USA, 2000.

[31] PELOUX, J.; REYNAL, G.; SCHOLL, M. **Evaluation of spatial indices implemented with the DBMS O2**. *Ing eni erie des Syst emes d'Information*, 3(4):517–554, 1995.

[32] ROUSSOPOULOS, N.; KELLEY, S.; VINCENT, F. **Nearest neighbor queries**. In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, p. 71–79. ACM, 1995.

[33] SCHNITZER, B.; LEUTENEGGER, S. **Master-client R-trees: A new parallel R-tree architecture**. In: *ssdbm*, p. 68. Published by the IEEE Computer Society, 1999.

[34] STONEBRAKER, M.; BERKELEY, C. **The Case for Shared Nothing**. *Database engineering*, p. 4, 1986.

[35] TANIN, E.; HARWOOD, A.; SAMET, H. **A distributed quadtree index for peer-to-peer settings**. In: *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON DATA ENGINEERING*, volume 21, p. 254. Citeseer, 2005.

[36] THEODORIDIS, Y.; SELLIS, T. **A model for the prediction of R-tree performance**. In: *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, p. 161–171. ACM, 1996.

[37] VAIDYANATHA, G. **Oracle Database Performance Management**. *Quest Software. White Paper. Pub*, 1999.

[38] WANG, L.; YU, S.; CHEN, F. **A new solution of node splitting to the R Tree algorithm**. In: *Intelligent Control and Information Processing (ICICIP), 2010 International Conference on*, p. 611–614. IEEE, 2010.

[39] WEI, X.; SEZAKI, K. **DHR-Trees: A Distributed Multidimensional Indexing Structure for P2P Systems**. In: *Parallel and Distributed Computing, 2006. ISPDC'06. The Fifth International Symposium on*, p. 281–290, 2006.

[40] WEISER, M. **The computer for the 21st century**. *Scientific American*, 272(3):78–89, 1995.

[41] XIA, T.; ZHANG, D. **Improving the R*-tree with outlier handling techniques**. In: *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, p. 125–134. ACM New York, NY, USA, 2005.

[42] ZHOU, X.; ABEL, D.; TRUFFET, D. **Data partitioning for parallel spatial join processing**. *GeoInformatica*, 2(2):175–204, 1998.

APÊNDICE A

Acesso à Folhas e Diretórios do Dataset Arizona com M=80

Total de nós: 3030
Total de folhas: 2968
Total de diretórios: 62

Janela	fol	dir	ff	df	%fol	%ff	%df	fac	%fac	dac	%dac	%atot
J00	26	18	16	2	0,88%	0,54%	3,23%	42	1,42%	20	32,26%	2,05%
J01	25	15	11	4	0,84%	0,37%	6,45%	36	1,21%	19	30,65%	1,82%
J02	26	14	15	5	0,88%	0,51%	8,06%	41	1,38%	19	30,65%	1,98%
J03	25	14	16	5	0,84%	0,54%	8,06%	41	1,38%	19	30,65%	1,98%
J04	26	14	14	5	0,88%	0,47%	8,06%	40	1,35%	19	30,65%	1,95%
J05	23	16	19	2	0,77%	0,64%	3,23%	42	1,42%	18	29,03%	1,98%
J06	26	15	12	2	0,88%	0,40%	3,23%	38	1,28%	17	27,42%	1,82%
J07	18	14	24	3	0,61%	0,81%	4,84%	42	1,42%	17	27,42%	1,95%
J08	25	14	18	3	0,84%	0,61%	4,84%	43	1,45%	17	27,42%	1,98%
J09	26	15	19	2	0,88%	0,64%	3,23%	45	1,52%	17	27,42%	2,05%
J10	29	16	26	1	0,98%	0,88%	1,61%	55	1,85%	17	27,42%	2,38%
J11	33	14	10	2	1,11%	0,34%	3,23%	43	1,45%	16	25,81%	1,95%
J12	29	13	18	3	0,98%	0,61%	4,84%	47	1,58%	16	25,81%	2,08%
J13	37	15	12	1	1,25%	0,40%	1,61%	49	1,65%	16	25,81%	2,15%
J14	30	15	13	1	1,01%	0,44%	1,61%	43	1,45%	16	25,81%	1,95%
J15	34	16	36	1	1,15%	1,21%	1,61%	70	2,36%	17	27,42%	2,87%
J16	26	15	22	1	0,88%	0,74%	1,61%	48	1,62%	16	25,81%	2,11%
J17	26	14	15	2	0,88%	0,51%	3,23%	41	1,38%	16	25,81%	1,88%
J18	24	15	20	1	0,81%	0,67%	1,61%	44	1,48%	16	25,81%	1,98%
J19	32	15	21	1	1,08%	0,71%	1,61%	53	1,79%	16	25,81%	2,28%
J20	22	16	46	1	0,74%	1,55%	1,61%	68	2,29%	17	27,42%	2,81%
J21	19	15	26	1	0,64%	0,88%	1,61%	45	1,52%	16	25,81%	2,01%
J22	22	14	15	2	0,74%	0,51%	3,23%	37	1,25%	16	25,81%	1,75%
J23	25	15	18	1	0,84%	0,61%	1,61%	43	1,45%	16	25,81%	1,95%
J24	27	16	27	0	0,91%	0,91%	0,00%	54	1,82%	16	25,81%	2,31%
J25	16	15	18	1	0,54%	0,61%	1,61%	34	1,15%	16	25,81%	1,65%
J26	18	15	19	1	0,61%	0,64%	1,61%	37	1,25%	16	25,81%	1,75%
J27	43	17	28	0	1,45%	0,94%	0,00%	71	2,39%	17	27,42%	2,90%
J28	35	17	29	0	1,18%	0,98%	0,00%	64	2,16%	17	27,42%	2,67%
J29	19	16	49	1	0,64%	1,65%	1,61%	68	2,29%	17	27,42%	2,81%
média	26,4	15,1	21,1	1,8	0,89%	0,71%	2,96%	47,5	1,60%	16,9	27,31%	2,13%
mínimo	16,0	13,0	10,0	0,0	0,54%	0,34%	0,00%	34,0	1,15%	16,0	25,81%	1,65%
máximo	43,0	18,0	49,0	5,0	1,45%	1,65%	8,06%	71,0	2,39%	20,0	32,26%	2,90%
desvio padrão	6,0	1,1	9,4	1,4	0,20%	0,32%	2,28%	10,7	0,36%	1,2	1,89%	0,35%

Legenda:

fol	Folhas c/ resultado
dir	Diretórios c/ resultado
ff	Folhas falsas
df	Diretórios falsos
%fol	Percentual de folhas c/ resultado acessadas (fol / total de folhas * 100)
%ff	Percentual de folhas falsas acessadas (ff / total de folhas * 100)
%df	Percentual de diretórios falsos acessados (df / total de diretórios * 100)
fac	Folhas acessadas (fol + ff)
%fac	Percentual de folhas acessadas (fac / total de folhas * 100)
dac	Diretórios acessados (dir + df)
%dac	Percentual de diretórios acessados (dac / total de diretórios * 100)
atot	% de acesso total