TESE DE DOUTORADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Processamento Eficiente de Junção Espacial em Ambiente Paralelo e Distribuído Baseado em *SpatialHadoop*"

ALUNO: Eduardo Fernando Mendes
ORIENTADOR: Prof. Dr. Ricardo Rodrigues Ciferri

São Carlos
Fevereiro/2017

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROCESSAMENTO EFICIENTE DE JUNÇÃO ESPACIAL EM AMBIENTE PARALELO E DISTRIBUÍDO BASEADO EM SPATIALHADOOP

EDUARDO FERNANDO MENDES

ORIENTADOR: PROF. DR. RICARDO RODRIGUES CIFERRI

São Carlos - SP
Fevereiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROCESSAMENTO EFICIENTE DE JUNÇÃO ESPACIAL EM AMBIENTE PARALELO E DISTRIBUÍDO BASEADO EM SPATIALHADOOP

EDUARDO FERNANDO MENDES

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Engenharia de Software / Banco de Dados / Interação Humano-Computador.
Orientador: Prof. Dr. Ricardo Rodrigues Ciferri.

São Carlos - SP
Fevereiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Tese de Doutorado do candidato Eduardo Fernando Mendes, realizada em 17/12/2017.

Prof. Dr. Ricardo Rodrigues Ciferri
(UFSCar)

Prof^a. Dr^a. Marcela Xavier Ribeiro
(UFSCar)

Prof^a. Dr^a. Marilde Terezinha Prado Santos
(UFSCar)

Prof. Dr. André Santanchè
(UNICAMP)

Prof^a. Dr^a. Valéria Cesário Times
(UFPE)

André Santanchè e Valéria Cesário Times , depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Eduardo Fernando Mendes.

Prof. Dr. Ricardo Rodrigues Ciferri
Presidente da Comissão Examinadora
(UFSCar)

Aos meus familiares, meus amigos e meus professores, que

tanto contribuíram para a realização deste trabalho.

AGRADECIMENTOS

Aos meus pais, Joaquim Moreira Mendes e Nésia de Carvalho Mendes, pelo exemplo de vida, cuidados por toda vida e amor incondicional.

A minha esposa Ana Beatriz Lopes Françoso, pelos incentivos, compreensão e parceria.

As minhas filhas Karen Martineli Mendes e Karine Martineli Mendes, pela compreensão devido a minha frequente ausência durante todo este período.

Aos meus orientadores e colaboradores, professores Ricardo Rodrigues Ciferri, Cristina Dutra de Aguiar Ciferri e Renato Bueno, pelos ensinamentos, pelo apoio e pela dedicação.

Aos meus amigos Adriano dos Santos Fernandes, Marcos Rodrigo Bernardes Ferreira, Denise Cristina Luzzi Gomes e Catarina Albardeiro Bahia pelas diversas sugestões que tornaram melhor a qualidade da escrita desta tese de doutorado.

A Profa. Dra. Maria Olívia Garcia Ribeiro de Arruda pela revisão ortográfica e incentivo.

A todos os amigos que compartilharam deste sonho e apoiaram-me para que se transformasse em realidade. Todos muito especiais.

AGRADECIMENTO ESPECIAL

Algumas vezes na vida, você encontra uma pessoa especial. Alguém que muda a sua vida simplesmente por estar nela. Alguém que te faz ir além até você não poder mais parar. Alguém que faz acreditar que realmente tem algo bom no mundo. Alguém que te convence que lá tem uma porta destrancada só esperando você abri-la. Alguém que te ajuda nas horas difíceis, tristes e confusas. Alguém que te põe no caminho certo quando você perde seu caminho. Muito obrigado Dr. Valdir Ferracin Pasotto.

Se você conta com alguém que tem menos qualidades que você,

isso levará à sua degeneração.

Se você conta com alguém com qualidades iguais às suas,

você permanece onde está.

Somente quando conta com alguém cujas qualidades são superiores às suas

é que você atinge uma condição sublime.

Dalai Lama

RESUMO

A explosão no volume de dados espaciais gerados e disponibilizados nos últimos anos, provenientes de diferentes fontes, por exemplo, sensoriamento remoto, telefones inteligentes, telescópios espaciais e satélites, motivaram pesquisadores e profissionais em todo o mundo a encontrar uma forma de processar de forma eficiente esse grande volume de dados espaciais. Sistemas baseados no paradigma de programação *MapReduce*, como exemplo *Hadoop*, provaram ser durante anos um *framework* eficiente para o processamento de enormes volumes de dados em muitas aplicações. No entanto, o *Hadoop* demonstrou não ser adequado no suporte nativo a dados espaciais devido a sua estrutura central não ter conhecimento das características espaciais desses dados. A solução para este problema deu origem ao *SpatialHadoop*, uma extensão do *Hadoop*, com suporte nativo para dados espaciais. Entretanto o *SpatialHadoop* não é capaz de alocar conjuntamente dados espaciais relacionados e também não leva em consideração qualquer característica dos dados no processo de escalonamento das tarefas para processamento nos nós de um *cluster* de computadores. Diante deste cenário, esta tese tem por objetivo propor novas estratégias para melhorar o desempenho do processamento das operações de junção espacial para grandes volumes de dados usando o *SpatialHadoop.* Para tanto, as soluções propostas exploram a alocação conjunta dos dados espaciais relacionados e a estratégia de escalonamento de tarefas *MapReduce* para dados espaciais relacionados também alocados de forma conjunta. Acredita-se que o acesso eficiente aos dados é um passo essencial para alcançar um melhor desempenho durante o processamento de consultas. Desta forma, as soluções propostas permitem a redução do tráfego de rede e operações de Entrada/Saída para o disco e consequentemente melhoram o desempenho no processamento de junção espacial usando *SpatialHadoop*. Por meio de testes de desempenho experimentais foi possível comprovar que as novas políticas de alocação de dados e escalonamento de tarefas de fato melhoram o tempo total de processamento das operações de junção espacial. O ganho de desempenho variou de 14,7% a 23,6% com relação ao *baseline* proposto por *CoS-HDFS* e variou de 8,3% a 65% com relação ao suporte nativo do *SpatialHadoop*.

Palavras-chave: Banco de Dados Espaciais, Processamento de Consulta, Junção Espacial, Processamento Paralelo e Distribuído, *Clusters* de Computadores, *Hadoop, MapReduce, SpatialHadoop*.

ABSTRACT

The huge volume of spatial data generated and made available in recent years from different sources, such as remote sensing, smart phones, space telescopes, and satellites, has motivated researchers and practitioners around the world to find out a way to process efficiently this huge volume of spatial data. Systems based on the MapReduce programming paradigm, such as Hadoop, have proven to be an efficient framework for processing huge volumes of data in many applications. However, Hadoop has showed not to be adequate in native support for spatial data due to its central structure is not aware of the spatial characteristics of such data. The solution to this problem gave rise to SpatialHadoop, which is a Hadoop extension with native support for spatial data. However, SpatialHadoop does not enable to jointly allocate related spatial data and also does not take into account any characteristics of the data in the process of task scheduler for processing on the nodes of a cluster of computers. Given this scenario, this PhD dissertation aims to propose new strategies to improve the performance of the processing of the spatial join operations for huge volumes of data using SpatialHadoop. For this purpose, the proposed solutions explore the joint allocation of related spatial data and the scheduling strategy of MapReduce for related spatial data also allocated in a jointly form. The efficient data access is an essential step in achieving better performance during query processing. Therefore, the proposed solutions allow the reduction of network traffic and I/O operations to the disk and consequently improve the performance of spatial join processing by using SpatialHadoop. By means of experimental evaluations, it was possible to show that the novel data allocation policies and scheduling tasks actually improve the total processing time of the spatial join operations. The performance gain varied from 14.7% to 23.6% if compared to the baseline proposed by CoS-HDFS and varied from 8.3% to 65% if compared to the native support of SpatialHadoop.

Keywords: *Spatial Databases, Query Processing, Spatial Join, Parallel and Distributed Processing, Cluster Computing, Hadoop, MapReduce, SpatialHadoop.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

B&M - *Algorithm Build&Match*

CPU - *Central Processing Unit*

DJ - *Distributed Join*

GB - *Gigabytes*

GFS - *Google File System*

HDFS - *Hadoop Distributed File System*

IaaS - *Infrastructure as a Service*

IP - *Internet Protocol*

KNN - K vizinhos mais próximos

MAM - Métodos de Acesso Multidimensionais

MB - *Megabytes*

MBR - *Minimum Bounding Rectangle*

MR - *MapReduce*

NIST - *National Institute of Standards and Technology*

OGC - *Open Geospatial Consortium*

PaaS - *Platform as a Service*

PBSM - *Partition Based Spatial Merge Join*

SaaS - *Software as a Service*

SGBD - Sistema Gerenciador de Bando de Dados

SHJ - *Spatial Hash Join*

SIG - Sistema de Informações Geográficas

SJMR - *Spatial Join with MapReduce*

SSBSJ - *Sort Sweep-Based Spatial Join*

STT - *Synchronized Tree Transversal*

TB - *Terabytes*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta a contextualização em que este trabalho está inserido e o problema de pesquisa juntamente com a motivação que deu origem a esta tese de doutorado. Em seguida, são apresentadas as hipóteses e os objetivos traçados assim como os principais resultados obtidos com o desenvolvimento deste trabalho. Por fim, é apresentada a organização desta tese.

1.1 Contextualização

Nos últimos anos, pesquisadores e profissionais em todo o mundo têm trabalhado com o objetivo de encontrar uma forma eficiente de processar consultas espaciais diante da explosão no volume de dados espaciais gerados e disponibilizados por meio de diferentes fontes, por exemplo, sensoriamento remoto, telefones inteligentes, telescópios espaciais e satélites.

Sistemas baseados no paradigma de programação *MapReduce* (DEAN; GHEMAWAT, 2008), como exemplo, *Hadoop* (SHVACHKO et al., 2010), provaram ser eficientes para o processamento de enormes volumes de dados em muitas aplicações. As tentativas de processar consultas espaciais em *Hadoop* concentravam-se principalmente em tipos específicos de operações e dados espaciais, tais como processamento dos K-vizinhos mais próximos (LU et al., 2012; ZHANG; LI; JESTES, 2012) com dados de trajetórias (MA et al., 2009). Porém, a eficiência de tais operações fica limitada devido ao núcleo do sistema *Hadoop* não ter conhecimento das características espaciais dos dados (ELDAWY; MOKBEL, 2015a).

Diversas alternativas foram propostas para o processamento eficiente de consultas espaciais, tanto na área acadêmica quanto na indústria. Na academia, foram desenvolvidos três protótipos de sistema:

(1) *Parallel-Secondo* (LU; GUTING, 2012) como um sistema gerenciador de banco de dados (SGBD) espacial e paralelo que usa *Hadoop* como um escalonador de tarefas distribuídas;

(2) *MD-HBase* (NISHIMURA et al., 2013) que estende o *HBase* (APACHE, 2013), um SGBD não relacional, que roda no topo de um sistema *Hadoop*, para apoiar índices multidimensionais;

(3) *Hadoop-GIS* (AJI et al., 2013) que estende o *Hive* (THUSOO et al., 2009), uma infraestrutura de *Data Warehouse,* construído sobre o *Hadoop* com um *Grid Index* uniforme para permitir o processamento de consultas dos tipos *Range Queries* e *Self-Joins*.

Nos últimos anos, a academia propôs o *Yarn,* também chamado de *MapReduce* 2.0*,* (VAVILAPALLI et al., 2013) uma evolução do *Hadoop,* considerado como a próxima geração da plataforma computacional, separando o modelo de programação da infraestrutura de gerenciamento de recursos, delegando muitas funções de escalonamento de tarefas para componentes de cada aplicação (VAVILAPALLI et al., 2013). Esta separação de componentes propostas no *Yarn* promove uma grande flexibilidade na escolha da plataforma de programação, pois o *MapReduce* é apenas um dos modelos de programação que é permitido pelo *Yarn* (VAVILAPALLI et al., 2013). Além do *MapReduce*, o *Yarn* permite também os modelos de programação: *Dryad* (ISARD et al., 2007), *REEF* (CHUN et al., 2013), *Spark* (ZAHARIA et al., 2010a), *Storm* (APACHE, 2014a), *Tez* (APACHE, 2014b) e *Giraph* (APACHE, 2014c).

Na indústria, a *ESRI* lançou um conjunto de ferramentas em *Hadoop* para trabalhar com o seu Sistema de Informações Geográficas (SIG), conhecido como *ArcGIS* (ESRI, 2013).

As principais desvantagens dos sistemas, *Parallel-Secondo, MD-HBase, Hadoop-GIS e ESRI* são a falta de integração com o núcleo do *Hadoop* e a ausência de suporte nativo para dados espaciais (ELDAWY, 2014; ELDAWY; MOKBEL, 2013, 2015a). Por exemplo, no *Parallel-Secondo* todo o armazenamento e processamento são realizados por um *SGBD* espacial e o *Hadoop* é usado apenas para escalonar as tarefas distribuídas. O *MD-HBase* e *Hadoop-GIS compartilham* do mesmo

inconveniente, ou seja, adicionam uma camada de *software* em sua arquitetura incluindo o *HBase* e *Hive* respectivamente, fazendo que os programas de *MapReduce* tradicionais não obtenham benefícios, pois funcionam diretamente em *Hadoop* e *HDFS*.

A solução para estes problemas foi a motivação para um novo projeto criado pelo *Data Management Lab da University of Minnesota,* em 2013, por *Ahmed Eldawy* sob a supervisão de *Mohamed Mokbel*, dando origem ao *SpatialHadoop* (ELDAWY; MOKBEL, 2013, 2015a), uma extensão do *Hadoop*, com suporte nativo para dados espaciais, tornando-se o primeiro *framework MapReduce* que implementa construções espaciais e a consciência de dados espaciais dentro do código base do *Hadoop* (ELDAWY; MOKBEL, 2015a).

O SpatialHadoop tem aproximadamente 45.000 mil linhas de código e estende o *Hadoop,* ou seja*,* implementa *as* construções espaciais no *Hadoop,* tornando-se mais eficiente para o processamento de consultas espaciais. Este é o ponto chave por trás do poder e da eficiência de *SpatialHadoop* em relação aos outros sistemas anteriormente identificados. Além disso, o *SpatialHadoop* apresenta índices espaciais, componentes padrões e *MapReduce* que permitem aos pesquisadores e desenvolvedores implementarem novas operações espaciais. Tudo isso está em contraste com *Parallel-Secondo, MD-HBase* e *Hadoop-GIS* que não permitem esse tipo de flexibilidade e, portanto, são limitados para a criação de novas funções.

As opções de indexação nos sistemas anteriores são limitadas. Por exemplo, o *Parallel-Secondo* não implementa indexação, *MD-HBase* implementa *Quad-Tree* (SAMET, 1984) *e o KD-Tree* (BENTLEY, 1975)*, Hadoop-GIS* implementa apenas o *Grid Index,* enquanto *SpatialHadoop* oferece mais opções, incluindo *Grid File* (NIEVERGELT; HINTERBERGER; SEVCIK, 1984), *R-Tree* (GUTTMAN, 1984) e *R^+-Tree* (SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987). Além disso, o *SpatialHadoop*, segundo os seus autores e publicações relacionadas (ELDAWY; MOKBEL, 2015a), propicia uma melhora significativa no desempenho do processamento de consultas espaciais em sistemas baseados em *Hadoop*. Por exemplo, para 70 MB de objetos espaciais em um *cluster* de 20 nós, o *Hadoop* executa uma consulta em 193 segundos, enquanto o *SpatialHadoop* executa a mesma consulta em 2 segundos (ELDAWY; MOKBEL, 2015a).

No núcleo do *SpatialHadoop*, os índices *Grid File*, *R-Tree* e *R+-Tree* são adaptados para o sistema de arquivos *Hadoop* (*HDFS*), construídos como índices globais que particionam os dados entre os nós de um *cluster* de computadores e vários índices locais para organizar os registros dentro de cada nó. Os novos índices ficam acessíveis a programas *MapReduce* através da introdução de dois novos componentes, *SpatialFileSplitter* e *SpatialRecordReader* que são usados para acessar os índices globais e locais (ELDAWY; MOKBEL, 2015a). O novo projeto físico de índices espaciais permite uma infinidade de operações espaciais a serem implementadas de forma eficiente em *SpatialHadoop*. Três operações básicas já são implementadas: consulta por faixa (*Range Query)*, K vizinhos mais próximos (*KNN query)* e junção espacial (*Spatial Join)* (ELDAWY; MOKBEL, 2015a), além do *CG_Hadoop* (ELDAWY et al., 2013), um conjunto de operações de geometria computacional em *SpatialHadoop* que faz uso de índices espaciais. Nesta tese de doutorado o foco será nas operações de junção espacial.

O *SpatialHadoop* também é usado em três sistemas ativos. *MNTG* (MOKBEL et al., 2014), um serviço *web* para a geração de dados de tráfego. *TAREEG* (ALARABI et al., 2014), uma ferramenta baseada na *web* de extração de dados do *OpenStreetMap* (CURRAN; CRUMLISH; FISHER, 2012), e *Shahed* (ELDAWY et al., 2015a), um sistema de análise de dados de satélite da *NASA*. Com o intuito de permitir o processamento de dados espaço-temporal, foi proposto o sistema *Hadoop* espaço-temporal como uma extensão que leva a dimensão de tempo em consideração (ELDAWY, 2014; ELDAWY et al., 2015a), porém dados espaço-temporais não são o foco desta tese de doutorado.

1.2 Problema

Segundo (ELDAWY; MOKBEL, 2015a), a ideia central do *SpatialHadoop* é que o mesmo funcione como um veículo de pesquisa, permitindo contribuições da comunidade acadêmica para ampliar ainda mais as suas funcionalidades, proporcionando um rico sistema que será amplamente utilizado por desenvolvedores, profissionais e pesquisadores. O *SpatialHadoop* está disponível como O*pen Source* e já foi baixado mais de 80.000 vezes desde o seu lançamento.

Ele tem sido usado por vários laboratórios de pesquisa e indústrias em todo o mundo. Mesmo sendo mais eficiente do que os outros sistemas alguns problemas foram identificados no *SpatialHadoop,* tornando-o ineficiente no processamento das operações de junção espacial.

O primeiro problema está relacionado a sua incapacidade de tratar da alocação conjunta dos dados espaciais relacionados. O *SpatialHadoop* adota a política padrão de alocação do *HDFS*, que não leva em consideração qualquer característica dos dados espaciais no processo de alocação dos dados nos nós de um *cluster* de computadores. Isto se deve pelo fato que o *Hadoop* não foi idealizado para trabalhar com dois conjuntos de dados relacionados. O foco da política padrão de alocação do *HDFS* é obter o balanceamento de carga, distribuindo os dados uniformemente entre os nós de um *cluster* de computadores, independentemente da utilização prevista para os dados espaciais (ZHAO et al., 2012). Esta política simples de alocação de dados funciona bem com a maioria das aplicações *Hadoop* que acessam apenas um único arquivo, mas as aplicações que processam dados de diferentes arquivos, como a junção espacial pode obter um aumento significativo no desempenho com estratégias personalizadas (ELTABAKH et al., 2011). Evidências recentes mostram que a organização dos dados pode proporcionar aumento no desempenho de aplicações *Hadoop* e no processamento dos algoritmos de consulta (JIONG XIE et al., 2010; WEI et al., 2010; ZHOU; XU, 2011).

Seguindo a mesma linha de raciocínio, (CHENT; SHAO; LI, 2016) afirma que as aplicações de *Big Data* exigem alto desempenho de operações de E/S (Entrada e Saída de Dados), uma vez que enormes volumes de dados precisam ser transferidos entre a memória e os dispositivos de armazenamento para o processamento. Embora o desempenho dos dispositivos de E/S tenha sido constantemente melhorado, os sistemas de armazenamento de E/S ainda permanecem como um dos principais gargalos dos sistemas que lidam com grande volume de dados (AWAD; KETTERING; SOLIHIN, 2015; CAULFIELD et al., 2009; CAULFIELD; SWANSON, 2013; JEON; EL MAGHRAOUI; KANDIRAJU, 2013; JIA et al., 2014; KOCBERBER et al., 2013).

Outro problema identificado no *SpatialHadoop*, que também prejudica a execução da junção espacial, está relacionado com o escalonamento de tarefas em um *cluster* de computadores. O *SpatialHadoop* utiliza o escalonador de tarefas padrão do *Hadoop*, que não leva em consideração a localidade dos dados. Com o

escalonador de tarefas faz-se com que o processamento ocorra no mesmo nó onde os dados foram alocados, para minimizar a transferência de dados entres os nós (SENTHILKUMAR; ILANGO, 2016). Para o escalonador de tarefa ser eficiente, é necessário que os dados a serem processados estejam alocados no mesmo nó. Essa situação torna-se complexa em relação ao tamanho do *cluster* (HASHEM et al., 2016). Quanto maior o *cluster* de computadores*,* maior é a dispersão dos dados. Desta forma, o escalonador de tarefas torna-se ineficiente (RASOOLI; DOWN, 2014).

Apesar desses problemas serem reconhecidos, ainda existe pouco entendimento sobre a interferência da localidade dos dados e do escalonamento de tarefas no desempenho das operações espaciais. Alguns trabalhos avaliam o ganho de se alocar o processamento exatamente no mesmo nó que contém os dados, porém apenas o trabalho de (FAHMY; ELGHANDOUR; NAGI, 2016) trata de dados espaciais relacionados e operações de junção espacial. A solução destes problemas no *SpatialHadoop*, a alocação e escalonamento, tende a melhorar o desempenho de várias operações espaciais, proporcionando acesso rápido aos dados e evitando congestionamento da rede e diminuindo o custo de E/S de disco.

Diante do exposto foi definido o seguinte tema de pesquisa: **Processamento de junção espacial em *clusters* de computadores baseado em *SpatialHadoop***.

Dentro do tema estabelecido foi investigada a solução para o seguinte problema: **Quais os impactos da alocação conjunta dos dados espaciais relacionados e do escalonamento de tarefas para dados espaciais relacionados sobre o desempenho da execução das operações de junção espacial em *SpatialHadoop*?**

1.3 Motivação

A junção espacial consiste no relacionamento entre dois conjuntos de dados espaciais, combinando suas geometrias de acordo com algum predicado espacial, como exemplo a interseção entre dois objetos espaciais. Existem na literatura diversos algoritmos para realizar a operação de junção espacial, geralmente com um

alto custo de processamento e de E/S em disco, devido ao volume de dados e a complexidade dos dados espaciais (JACOX; SAMET, 2007).

Várias soluções foram propostas, como as centralizadas, limitadas pelo poder de processamento e armazenamento. Recentemente as soluções de processamento paralelo e distribuído, em que um conjunto de computadores interconectados em um *cluster* de computadores coopera para a realização da operação, aproveitando as características do *Hadoop* e *MapReduce,* tornando-se a forma mais eficiente de tratar esse problema (ELDAWY, 2014).

Neste tipo de solução, o grande gargalo de desempenho do *Hadoop* é a sua falta de capacidade de alocar dados e escalonar tarefas para dados espaciais relacionados no mesmo conjunto de nós dentro de um *cluster Hadoop*. A alocação e o escalonamento podem ser utilizados para melhorar a eficiência das várias operações espaciais, incluindo junção espacial (ELTABAKH et al., 2011). O acesso eficiente aos dados é um passo essencial para alcançar um melhor desempenho durante o processamento de consultas (HASHEM et al., 2016). Desta forma**,** a sua utilização precisa ser feita de modo eficaz. Surge então o problema de levar as aplicações para perto dos dados**.** Caso isso não seja feito, o preço de mover os dados pelo sistema pode ser muito alto, prejudicando o desempenho final da solução. Para garantir um melhor desempenho, é importante que o processamento ocorra próximo aos dados. As soluções atuais buscam localidade apenas no nível da rede local, não de nó. (ZHAO et al., 2012). A alocação conjunta dos dados espaciais relacionados pode proporcionar uma redução de até 47% do tráfego de rede (FAHMY; ELGHANDOUR; NAGI, 2016).

Pesquisas em bancos de dados paralelos (GRAY et al., 2009), bem como trabalhos recentes sobre *Hadoop* (ABOUZEID et al., 2009; DITTRICH et al., 2010), mostram que a organização cuidadosa dos dados permite o processamento eficiente de algoritmos de consultas. Transferir essa ideia para *Hadoop* não é uma tarefa simples. Apesar do particionamento de dados em blocos ser fácil de conseguir em *Hadoop* (JIANG et al., 2010), a localização das partições não é (ABOUZEID et al., 2009; DITTRICH et al., 2010). Isso ocorre porque o *Hadoop* não fornece qualquer maneira para as aplicações controlarem onde os dados serão armazenados (ELTABAKH et al., 2011).

Diversos trabalhos atacam os problemas de alocação de dados e escalonamento de tarefas no *Hadoop*, como (ABOUZEID et al., 2009;

ANANTHANARAYANAN et al., 2011a; BARROSO; CLIDARAS; HÖLZLE, 2013; CASTILLO; SPREITZER; STEINDER, 2011; DITTRICH et al., 2010, 2012; ELDAWY, 2014; ELDAWY; ALARABI; MOKBEL, 2015; ELDAWY; MOKBEL, 2015b, 2013, 2015a; ELTABAKH et al., 2011; FAHMY; ELGHANDOUR; NAGI, 2016; GUO; FOX; ZHOU, 2012; HASHEM et al., 2016; HOTT; ROCHA; GUEDES, 2016; IBRAHIM et al., 2012; ISARD et al., 2007; JIANG; TUNG; CHEN, 2011; JIONG XIE et al., 2010; LI et al., 2016; NISHANTH S et al., 2013; OUSTERHOUT et al., 2010; PALANISAMY et al., 2011; PAVLO et al., 2009; RICHTER et al., 2012; YAO et al., 2014; ZAHARIA et al., 2010a, 2010b; ZHAO et al., 2012; ZHOU et al., 2016). Pelo conhecimento do autor, existe um único trabalho publicado em dezembro de 2016 que trata de um dos assuntos abordados nesta tese, que é a alocação conjunta dos dados espaciais relacionados, utilizando técnicas diferentes desta pesquisa. Diante deste contexto, as questões principais que motivaram a iniciativa do presente trabalho foram:

I. Tornar as operações de junção espacial em *SpatialHadoop* mais eficientes afinal, atualmente a junção espacial é considerada o principal gargalo em termos de desempenho do *SpatialHadoop.*
II. No cenário acadêmico, o estudo da junção espacial tem sua relevância reconhecida, considerando-se os numerosos artigos científicos que destacam esta área na atualidade.
III. Na indústria, é considerada uma das operações mais importantes no suporte à dados espaciais, e a busca por um melhor desempenho é fundamental para o continuo crescimento de sua utilização.

1.4 Hipóteses

Diversos trabalhos versam sobre a importância da organização dos dados espaciais para um melhor desempenho das operações espaciais. Nesse contexto, o presente trabalho propõe as seguintes hipóteses:

I. O processamento das operações de junção espacial, quando ocorrem no mesmo nó onde estão armazenados os dois conjuntos de dados espaciais*,* é mais eficiente do que o processamento dos dados espalhados em nós de armazenamento distintos;

II. A alocação conjunta dos dados espaciais relacionados e o escalonamento de tarefas para dados espaciais relacionados reduzem o tráfego de rede e o custo de E/S de disco e consequentemente melhoram o desempenho no processamento de junção espacial em *SpatialHadoop.*

1.5 Objetivos

De acordo com as hipóteses apresentadas anteriormente, esta tese de doutorado tem como principal objetivo:

Melhorar o desempenho do processamento de operações de junção espacial em *clusters* de computadores usando o *SpatialHadoop,* considerando grandes volumes de dados. Neste sentido, as soluções propostas exploram a alocação conjunta de dados espaciais relacionados e uma estratégia customizada de escalonamento de tarefas *MapReduce* para dados espaciais relacionados, alocados de forma conjunta.

Para apoiar isso, foi desenvolvida uma aplicação *web* que facilita para os usuários finais a execução das operações de junção espacial com as novas políticas de alocação conjunta de dados espaciais relacionados e escalonamento de tarefas para dados espaciais relacionados em *SpatialHadoop*.

Para alcançar o objetivo principal, foram traçados os seguintes objetivos específicos:

- Desenvolver um algoritmo para realizar a alocação conjunta de dados espaciais relacionados visando o processamento eficiente de junção espacial;
- Desenvolver um algoritmo para realizar o escalonamento de tarefas *MapReduce* de acordo com a alocação conjunta de dados espaciais relacionados;
- Desenvolver uma aplicação *web* para a execução dos algoritmos das novas políticas de alocação conjunta de dados espaciais relacionados e do escalonamento de tarefas para dados espaciais relacionados no *SpatialHadoop*;

- Comparar o desempenho das novas políticas de alocação e escalonamento tarefas em relação às atuais políticas do *SpatialHadoop* e ao estado da arte no processamento de junção espacial em *cluster* de computadores, a saber *CoS-HDFS* (FAHMY; ELGHANDOUR; NAGI, 2016), por meio de testes de desempenho experimentais.

1.6 Organização do trabalho

Esta tese de doutorado está organizada da seguinte forma:

- **Capítulo 2** – é descrita a fundamentação teórica necessária para o entendimento desta tese. Neste capítulo, são discutidos os principais conceitos relacionados a bancos de dados espaciais; tipos de dados espaciais, objetos espaciais, abstração das geometrias, métodos de acesso, estruturas de indexação e operações espaciais.
- **Capítulo 3** – são apresentados os conceitos de *Big Data*, processamento paralelo e distribuído em *clusters* de computadores usando *MapReduce*, *Hadoop* e *SpatialHadoop*.
- **Capítulo 4** – são descritos trabalhos correlatos encontrados na literatura, com maior detalhamento daqueles que focam na alocação de dados e no escalonamento de tarefas.
- **Capítulo 5** – são descritas as contribuições desta tese de doutorado. São detalhados os algoritmos de alocação conjunta de dados espaciais relacionados e o novo algoritmo de escalonamento de tarefas para a execução de operações de junção espacial em *SpatialHadoop*.
- **Capítulo 6** – são detalhados os resultados obtidos em testes de desempenho experimentais, a fim de validar a eficiência das novas políticas de alocação conjunta de dados espaciais relacionados e escalonamento de tarefas no *SpatialHadoop* para a execução de operações de junção espacial.
- **Capítulo 7** – finalmente são destacadas as conclusões obtidas nesta tese de doutorado, validadas as hipóteses, destacadas as principais contribuições e possibilidade de trabalhos futuros.

Capítulo 2

DADOS ESPACIAIS

Este capítulo descreve os principais conceitos relacionados a esta pesquisa de doutorado com relação aos dados espaciais, tais como: os tipos de dados espaciais, objetos espaciais, relacionamentos topológicos, predicados topológicos, abstração de geometrias, estruturas de indexação e operações espaciais.

2.1 Tipos de dados espaciais

Os tipos de dados espaciais estão formalizados por especificações produzidas pelo consórcio *Open Geospatial Consortium* (OGC) (OGC, 2014). Este consórcio é formado por mais de 250 companhias, agências governamentais e universidades, sendo responsável por promover e desenvolver tecnologias que facilitem a interoperabilidade entre sistemas que envolvem dados espaciais (GARDELS, 1996).

Segundo (BORGES, 2011; CÂMARA, 1996; CIFERRI, 2002; OGC, 2014), os tipos de dados espaciais básicos definidos pelo consórcio são:

- **Ponto:** unidade mínima representativa de um determinado objeto espacial. O objeto representado por um ponto não possui dimensionalidade, ou seja, é 0-dimensional, como exemplo, uma cidade em um mapa. É representada textualmente da seguinte forma: *Point* (0 0);
- **Linha:** é formada por uma sequência de pontos conectados. O objeto representado por uma linha é considerado 1-dimensional, como exemplo, um rio. É representada textualmente da seguinte forma: *LineString* (0 0, 1 1, 2 2);

- **Polígono:** este objeto é representado por uma sequência de linhas conectadas, sendo que o primeiro ponto deve coincidir com o último, formando uma sequência fechada. O objeto representado por esse tipo de dados é considerado 2-dimensional, como exemplo, um bairro. É definido por apenas um limite exterior e por nenhum ou vários limites interiores representadas por: *Polygon* (0 0, 4 0, 4 4, 0 4 0, 0 0).

A Figura 2.1 ilustra os tipos de dados espaciais básicos.

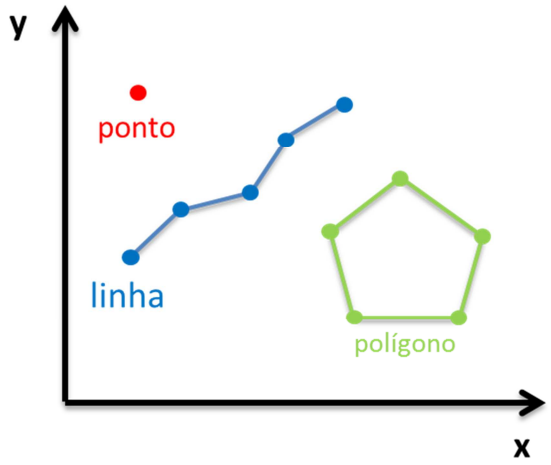


Figura 2.1 - Tipos de dados espaciais básicos. Reproduzida de (CIFERRI, 2002).

Um conjunto similar de tipos de objetos básicos agrupados forma outros tipos de objetos compostos chamados de coleção de geometrias.

- Coleção de Geometrias: **c**oleção de uma ou mais geometrias de pontos, linhas e polígonos, tais como:
  - **Multipontos:** representado por uma coleção de pontos, como exemplo, *Multipoint* (0 0, 0 4);
  - **Multilinhas:** uma coleção de linhas pode representar o objeto, como exemplo, *MultiLineString* ((0 0, 1 1, 2 2), (4 4, 5 5, 6 6)).
  - **Multipolígonos:** formando por uma coleção de polígonos, como exemplo, *MultiPolygon* (((0 0, 4 0, 4 4, 0 4, 0 0),(... ,(... ,(...)));

Pela combinação de tipos de dados espaciais básicos, é possível formar outros tipos de dados mais complexos, tais como:

- **Polígonos complexos:** este objeto é considerado 2-dimensional e pode conter buracos ou serem formados por diversas partes disjuntas chamadas ilhas, por exemplo, o estado do Alaska nos Estados Unidos da

América é um exemplo de ilha, enquanto a Lagoa Rodrigo de Freitas é considerada um buraco no polígono que representa a cidade do Rio de Janeiro.

A Figura 2.2 ilustra os vários tipos de dados espaciais complexos.

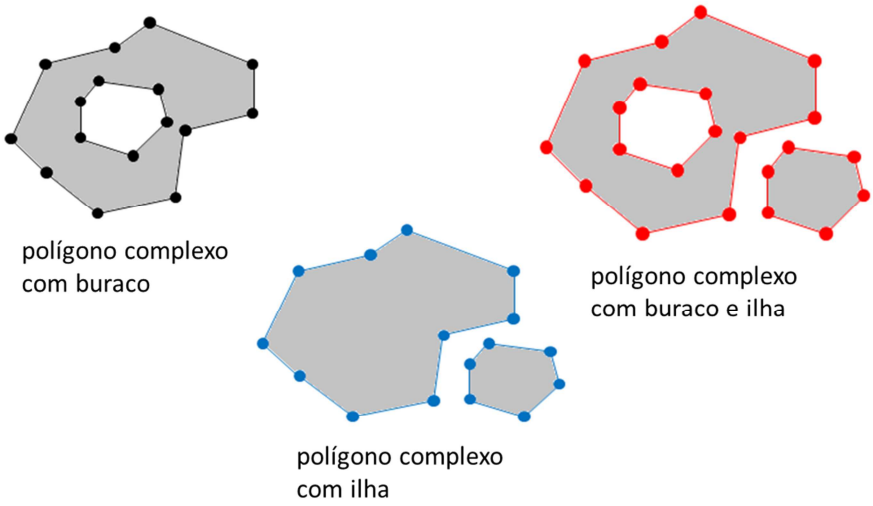


Figura 2.2 - Tipos de dados espaciais complexos. Reproduzida de (CIFERRI, 2002).

Os tipos de dados espaciais são utilizados para a criação dos objetos espaciais.

2.2 Objetos espaciais

Segundo (BORGES, 2011; CÂMARA, 1996; CIFERRI, 2002; OGC, 2014) uma determinada representação do mundo real é formada pela combinação de tipos de objetos espaciais dando origem ao conceito de objeto espacial.

Esses objetos, segundo (CIFERRI, 2002), estão associados a um determinado atributo espacial representado por coordenadas que definem a localização e a sua extensão dentro do espaço Euclidiano *d-dimensional*. Em geral, os objetos espaciais são representados por vários atributos espaciais contendo várias representações. A Figura 2.3 ilustra as diversas coordenadas que representam retângulos em um espaço bidimensional.

Cell #0 Rectangle: (-0.26752020000000004,49.058542499999994)-(5.9347001,62.6677811)
Cell #0 Rectangle: (21.5381913,55.5624771)-(179.9336535,74.9837666)
Cell #0 Rectangle: (-176.043354,-84.41596309999997)-(-65.24779420000002,34.411125000000006)
Cell #0 Rectangle: (-178.44359300000002,-55.04331)-(-65.039,34.57998760000001)
Cell #0 Rectangle: (5.182719,-13.534279799999998)-(13.300513,47.4322459)
Cell #0 Rectangle: (5.7455750000000005,-2.9107437)-(12.7963605,47.404393899999995)
Cell #0 Rectangle: (-124.41512519999999,34.0202574)-(-73.886652,40.80648)
Cell #0 Rectangle: (-124.134889,34.189167000000005)-(-73.88506,41.1842534)
Cell #0 Rectangle: (5.737884799999999,47.321473)-(12.568252600000001,50.1416966)
Cell #0 Rectangle: (5.8018404,47.37654479999999)-(12.4992094,50.124537200000006)
Cell #0 Rectangle: (-178.95728100000002,40.00247)-(-64.5174759,74.12826959999998)
Cell #0 Rectangle: (-179.231086,40.392422)-(-65.7934321,79.1299817)
Cell #0 Rectangle: (5.5140601,50.0482148)-(12.8364528,67.7852025)
Cell #0 Rectangle: (-67.5599855,-46.000000899999996)-(0.38387099999999996,38.452909)
Cell #0 Rectangle: (11.3747358,-34.767249)-(23.056669000000003,48.21986009999999)
Cell #0 Rectangle: (-66.00742819999999,38.270847200000006)-(-0.1413684,48.1974198)

Figura 2.3 - Coordenadas de retângulos que serão utilizadas nos testes.

O enfoque desta pesquisa de doutorado é em objetos espaciais no espaço Euclidiano 2-dimensional e, portanto, que comporta apenas objetos espaciais 0-dimensional (pontos simples e complexos), 1-dimensional (linhas simples e complexas) e 2-dimensional (polígonos simples e complexos).

Os objetos espaciais podem se relacionar com outros objetos espaciais através de relacionamentos topológicos.

2.3 Relacionamentos topológicos

Por meio de operadores topológicos, que têm por objetivo descrever as relações entre os objetos espaciais, podem-se identificar os relacionamentos entre os objetos. Segundo (EGENHOFER; HERRING, 1990), a matriz de 4-interseções é utilizada como modelo para definir os possíveis relacionamentos topológicos. A matriz é composta por oito relações topológicas binárias, representando a interseção entre as fronteiras e interiores de duas geometrias. Na Figura 2.4, pode-se visualizar a definição formal da matriz de 4-interseções e suas relações: disjunção (*Disjoint*), toda (*Meet*), contem (*Contains*), cobre (*Covers*), igual (*Equal*), sobreposição (*Overlap*), dentro de (*Inside*) e coberto por (*CoverBy*) (EGENHOFER; CLEMENTINI; DI FELICE, 1994).

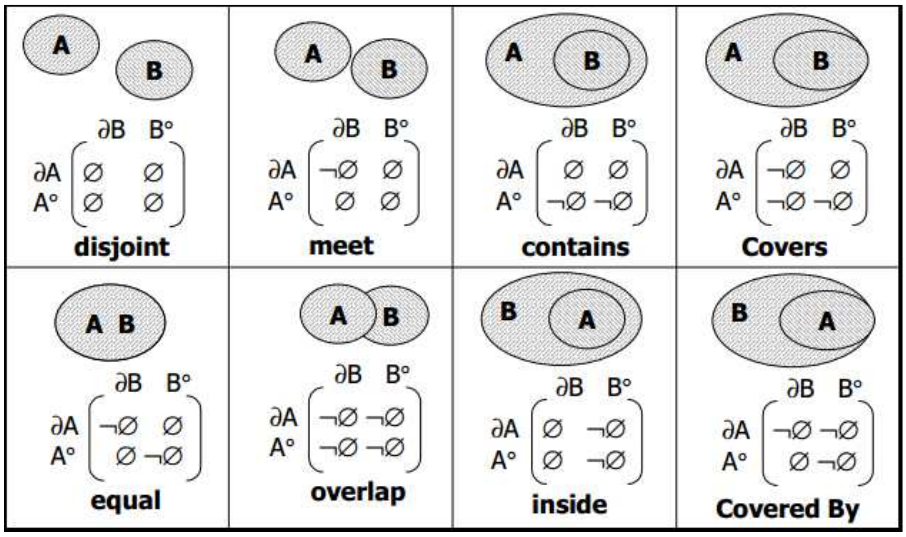


Figura 2.4 - Matriz de 4-interseções. Reproduzida de (EGENHOFER; CLEMENTINI; DI FELICE, 1994).

O trabalho de (EGENHOFER, 1993) propõe a extensão da matriz 4-interseções para melhor diferenciar os relacionamentos topológicos entre geometrias que possuem estruturas mais complexas, como exemplo, regiões com ilhas. A Figura 2.5 apresenta o novo modelo, uma extensão da matriz de 4-interseções que é nomeado matriz de 9-interseções (PAIVA, 1998). Ela, por sua vez, considera o resultado da interseção entre as fronteiras, interiores e exteriores de duas geometrias.

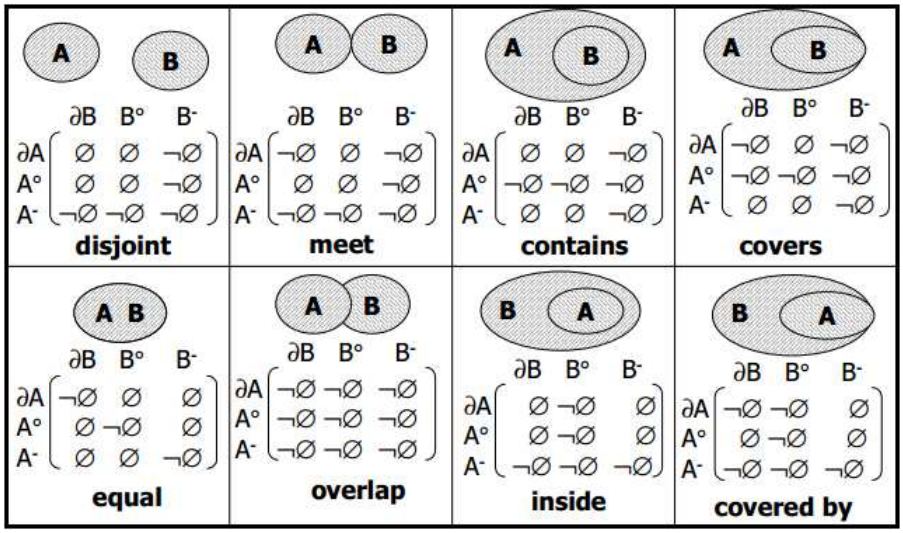


Figura 2.5 - Matriz de 9-interseções. Reproduzida de (EGENHOFER, 1993).

O conjunto dos 52 relacionamentos topológicos, englobando os oito da matriz 4-interseções, foi agrupado em cinco relacionamentos gerais, sendo eles: *touch*, *in*, *cross*, *overlap e disjoint*, que podem ser usados indistintamente para: pontos, linhas e polígonos são apresentados na Tabela 2.1.

Tabela 2.1 - Relacionamentos topológicos. Reproduzida de (CLEMENTINI; FELICE; OOSTEROM, 1993)**.**

touch	Aplica-se a pares de geometrias dos tipos região/região, linha/linha, linha/região, ponto/região e ponto/linha.
	$<A,touch,B> \Leftrightarrow (A^\circ \cap B^\circ = \emptyset) \wedge ((\partial A \cap B^\circ \neq \emptyset) \vee (A^\circ \cap \partial B \neq \emptyset) \vee (\partial A \cap \partial B \neq \emptyset))$
in	Aplica-se a pares de geometrias com qualquer combinação de tipos.
	$<A,in,B> \Leftrightarrow (A^\circ \cap B^\circ \neq \emptyset) \wedge (A^\circ \cap B^- = \emptyset) \wedge (\partial A \cap B^- = \emptyset)$
cross	Aplica-se a pares de geometrias dos tipos linha/linha e linha/região.
	No caso de linha/linha tem-se: $<A,cross,B> \Leftrightarrow \dim (A^\circ \cap B^\circ = \emptyset)$ No caso de linha/região tem-se: $<A,cross,B> \Leftrightarrow (A^\circ \cap B^\circ \neq \emptyset) \wedge (A^\circ \cap B^- \neq \emptyset)$
overlap	Aplica-se a pares de geometrias dos tipos região/região e linha/linha.
	No caso de região/região tem-se: $<A,overlap,B> \Leftrightarrow (A^\circ \cap B^\circ \neq \emptyset) \wedge (A^\circ \cap B^- \neq \emptyset) \wedge (A^- \cap B^\circ \neq \emptyset))$ No caso de linha/linha tem-se: $<A,overlap,B> \Leftrightarrow (\dim (A^\circ \cap B^\circ) = 1) \wedge (A^\circ \cap B^- \neq \emptyset) \wedge (A^- \cap B^\circ \neq \emptyset)$
disjoint	Aplica-se a pares de geometrias com qualquer combinação de tipos.
	$<A,disjoint,B> \Leftrightarrow (A^\circ \cap B^\circ = \emptyset) \wedge (\partial A \cap B^\circ = \emptyset) \wedge (A^\circ \cap \partial B = \emptyset) \wedge (\partial A \cap \partial B = \emptyset))$

A representação de toda a variedade de objetos geográficos do mundo real é comprometida pela limitação das estruturas simples formada pelos pontos, linhas e polígonos. Recentemente, no trabalho de (MCKENNEY; SCHNEIDER, 2008) é apresentada uma ampliação do conjunto de estruturas para representar objetos complexos.

Este trabalho concentra-se na operação de junção espacial, que combina dois conjuntos de objetos espaciais com base em um predicado topológico, mais comumente o relacionamento topológico de interseção. Esta operação está entre as mais importantes e mais difíceis de serem realizadas, devido a diversos

relacionamentos espaciais que podem ser utilizados para definir uma junção espacial (FORNARI, 2006). O relacionamento topológico de interseção é definido formalmente na equação da Figura 2.6 e consiste em ter pelo menos um ponto em comum entre dois objetos espaciais. Ou seja, o predicado de interseção ocorre quando não é válido o relacionamento espacial *Disjoint*. Em outras palavras, o predicado de interseção ocorre se for valido pelo menos um dos relacionamentos de *Mett*, *Contains*, *Covers*, *Equal*, *Overlap*, *Inside* e *CoverBy*.

$$IRQ\ (R, dataset) = \{\ o\ \mid\ o \in dataset \wedge o.G \cap R \neq \varnothing\ \}$$

Figura 2.6 – Formalismo topológico de interseção. Reproduzida de (CIFERRI, 2002).

Os predicados espaciais em uma consulta espacial promovem a relação entre as coordenadas dos objetos espaciais, que são excessivamente dispendiosas na indexação em termos de espaço de armazenamento e complexidade na avaliação de predicados espaciais, tornando custosa a avaliação do predicado espacial porque objetos espaciais podem conter milhares de tipos de objetos (CÂMARA, 1996; FORNARI, 2006). Desta forma é necessário estabelecer uma estratégia para armazenamento adequado das geometrias, assunto abordado na próxima seção.

2.4 Abstração das geometrias

Para o armazenamento das diversas coordenadas que descrevem a geometria exata de um objeto espacial, por exemplo, de um polígono, é necessária uma grande quantidade de espaço em memória secundária. A solução para este problema é proposta através de duas estratégias, uma focada no armazenamento de objetos espaciais em memória secundária e a outra focada em um armazenamento mais simples para melhorar o processo de recuperação destes objetos espaciais. O foco no armazenamento da geometria exata do objeto permite que não haja perda de precisão de sua representação; a outra é focada na abstração das geometrias dos objetos através de aproximações, criando um nível de armazenamento intermediário mais simples, diminuindo o custo de armazenamento

e o custo para processar os predicados espaciais (CÂMARA, 1996; FORNARI, 2006).

Conhecidas como aproximações simplificadoras, também chamadas de conservativas (CIFERRI, 2002), são utilizadas como estratégia para representação dos objetos espaciais. O *Minimum Bounding Rectangle (MBR*), ou retângulo envolvente mínimo, é a aproximação conservativa mais amplamente usada na literatura e consiste no menor retângulo que envolve completamente um determinado objeto espacial.

A Figura 2.7 ilustra alguns objetos e suas abstrações, através de seus *MBRs*.

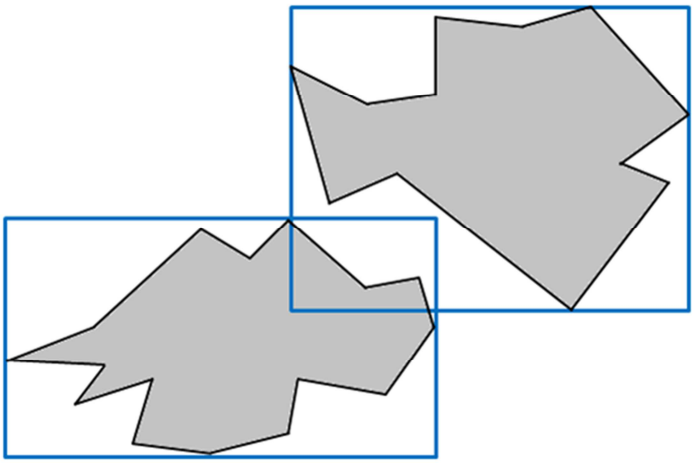


Figura 2.7 - Objetos e suas abstrações através de *MBRs*. Reproduzida de (CIFERRI, 2002).

Essa forma de representar as geometrias dos objetos espaciais causa a perda de precisão na representação delas, tornando-as imprecisas, porém facilita o processo de avaliação dos predicados espaciais. A imprecisão está no sentido de não ser muitas vezes possível retornar a resposta final a uma consulta espacial sobre a avaliação dos seus *MBRs.* Isto ocorre devido à criação de *dead space*, também chamado de área morta, área adicional que não pertence ao objeto espacial. Desta forma, é necessário mais uma fase de processamento para obter a resposta final de uma determinada consulta espacial. Esta outra fase é realizada sobre as geometrias exatas dos objetos, fazendo com que o processamento seja dividido em duas fases (CIFERRI, 2002).

Na Figura 2.8, a janela de consulta toca o *MBR* de um objeto espacial representado por uma linha diagonal. A área superposta é chamada de *dead space*.

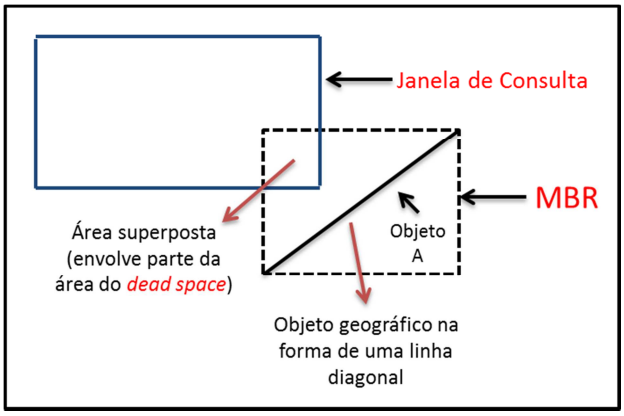


Figura 2.8 - *Dead Space*. Reproduzida de (CIFERRI, 2002).

O resultado do processamento da primeira fase pode recuperar objetos espaciais considerados como “falsos candidatos”, pois os mesmos não satisfazem completamente o predicado espacial. Para solucionar este problema, uma segunda fase torna-se necessária, fazendo a análise da geometria exata dos objetos selecionados durante a primeira fase, retornando desta forma a resposta correta. Com isso, pode-se concluir que a utilização de aproximações *MBR* torna o processamento de consultas espaciais impreciso, conforme é ilustrado na Figura 2.9 (CIFERRI, 2002).

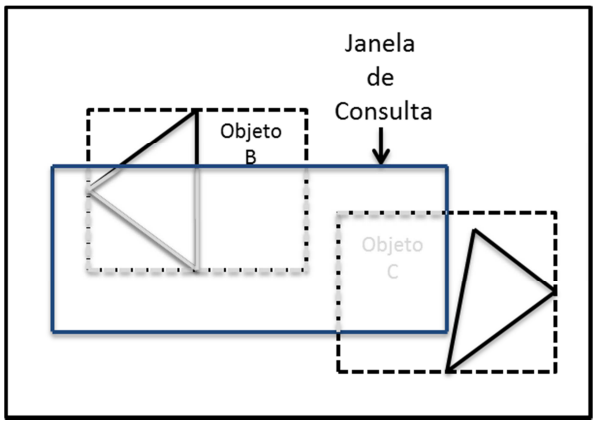


Figura 2.9 - Imprecisão das aproximações simplificadoras. Reproduzida de (CIFERRI, 2002).

Analisando a janela de consulta da Figura 2.9, para um predicado espacial que procura selecionar todos os objetos espaciais que intersectam a janela de consulta, retornariam como resultado preliminar os objetos b e c, pois os *MBRs* destes objetos intersectam a janela de consulta. Em uma análise mais detalhada sobre a geometria exata destes objetos, pode-se verificar que o objeto b realmente intersecta a janela de consulta, porém o objeto c não, ou seja, ele é considerado um falso candidato na seleção preliminar dos objetos que satisfazem ao predicado espacial. Este é um exemplo do problema provocado pelo *dead space*, a seleção de objetos que não satisfazem o predicado espacial.

O primeiro a propor uma técnica para resolver esse problema foi (ORENSTEIN, 1986), executando a operação em duas fases:

- **A primeira fase denominada filtragem:** são selecionados os objetos candidatos através da utilização do *MBR* que satisfazem um determinado predicado espacial;
- **A segunda fase denominada refinamento:** os candidatos provenientes da primeira fase de filtragem podem então ser avaliados através da sua geometria exata, devido à possível existência dos *dead spaces*. Os objetos que satisfazem o relacionamento topológico são devolvidos no conjunto resposta da avaliação final.

Com abstração das geometrias é possível obter um melhor desempenho na execução das operações espaciais. Isso é implementado com o uso de métodos de acesso multidimensionais, conhecidos como MAM, assunto da próxima seção.

2.5 Métodos de acesso

São estruturas de dados e algoritmos de pesquisa que auxiliam o processamento das operações espaciais tornando-as mais eficientes, também conhecidos como métodos de acesso multidimensionais, os quais criam um caminho otimizado, permitindo o acesso aos dados espaciais de forma mais eficiente. O espaço indexado é organizado de modo a recuperar os objetos espaciais de uma determinada área. Para recuperá-los, devem-se acessar os objetos próximos a ela. O desempenho dos MAM varia significativamente em função como, por exemplo,

das características dos dados espaciais e da escolha do tipo de consulta espacial. Em outras palavras, a relação de desempenho entre um conjunto de MAM pode não ser a mesma quando considerados diferentes conjuntos de dados e diferentes tipos de consultas espaciais (CIFERRI, 2002).

Podem-se identificar na literatura diversos métodos de acesso multidimensionais, sendo que um método bastante conhecido é a *R-Tree* (GUTTMAN, 1984), destinada a indexar dados espaciais que são representados por geometrias compostas por um conjunto de coordenadas. Na próxima seção serão apresentados os principais índices espaciais.

2.6 Principais índices espaciais

Historicamente, desde 1975, diversas pesquisas vêm sendo realizadas com o objetivo de melhorar as formas de indexação dos dados espaciais, como exemplo, as árvores de pesquisa. Estas têm como principal característica o balanceamento, ou seja, possuem o mesmo comprimento do nó raiz até o nó folha, conforme ilustrado na Figura 2.10.

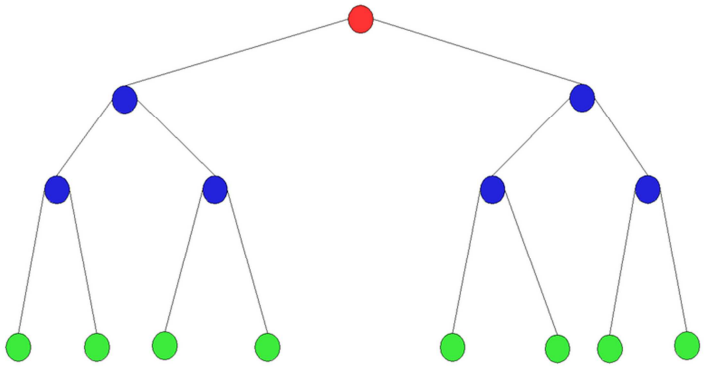


Figura 2.10 - Árvore balanceada.

Outras estruturas utilizadas na indexação dos dados podem ser encontradas na literatura, sendo as principais: *KD-Tree* (BENTLEY, 1975), *R-Tree* (GUTTMAN, 1984) e a *Hilbert R-Tree* (KAMEL; FALOUTSOS, 1994). Cada uma dessas estruturas é diferente com relação ao tipo de objeto que podem armazenar e no

conceito base utilizado na organização dos dados espacias (GAEDE; GÜNTHER, 1998). A escolha da melhor estrutura é uma tarefa complicada, pois cada uma se comporta de uma maneira como, por exemplo, uma estrutura pode ser eficiente em consultas sobre pontos, mas ineficiente em relação a polígonos.

Devido à complexidade dos dados, os métodos de indexação utilizam uma forma simplificadora para facilitar o acesso, sendo o mais conhecido o MBR (GAEDE; GÜNTHER, 1998). O presente trabalho é baseado no *SpatialHadoop* que utiliza os métodos de indexação de objetos espaciais derivados de árvores como: *Grid File* (NIEVERGELT; HINTERBERGER; SEVCIK, 1984), *R-Tree* (GUTTMAN, 1984) e R^{+}*-Tree* (SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987). Na sequência serão apresentados os métodos de indexação utilizados pelo *SpatialHadoop*.

2.6.1 *R-Tree*

As limitações encontradas nos mecanismos de indexação que não possibilitavam pesquisas no espaço multidimensional tais como a *B-Tree* (GAEDE; GÜNTHER, 1998), provocaram a criação de uma nova estrutura de indexação multidimensional *R-Tree*, que foi considerada um marco no desenvolvimento de índices espaciais (MANOLOPOULOS et al., 2006).

Segundo (GUTTMAN, 1984), *R-Tree* é uma estrutura hierárquica dinâmica que utiliza os *MBRs* para organizar objetos espaciais de forma que os objetos espacialmente próximos são armazenados próximos uns dos outros, consequentemente, reduzindo o espaço de busca em cada nível de árvore. Ela é uma árvore balanceada por altura com ponteiros para os objetos espaciais nos nós folhas. Como pode-se observar na Figura 2.11, os *MBRs* (1, 2, 3, 4) estão agrupando os objetos espaciais 1 (A, B), 2 (C, D, E), 3 (F, G) e 4(H, I, J), conforme a localização destes objetos.

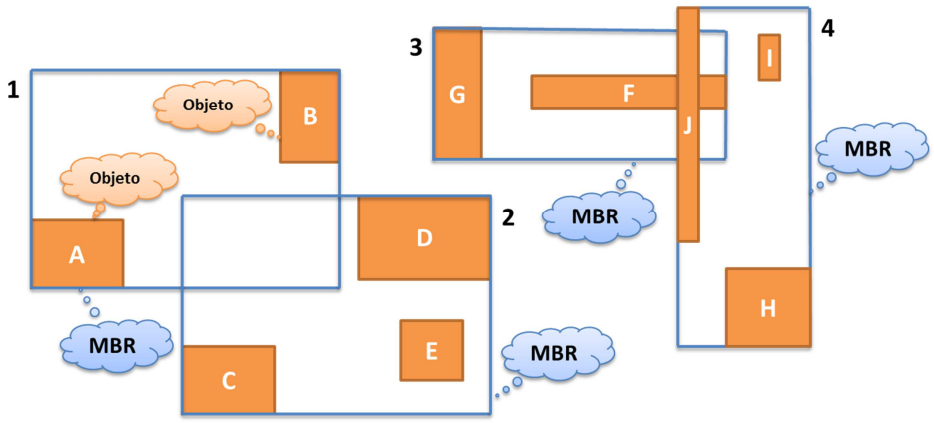


Figura 2.11 - *MBRs* agrupando os objetos. Adaptada de (MANOLOPOULOS et al., 2006).

Uma *R-Tree* possui uma estrutura de dados dividida em dois tipos de nós:

- **Nós folhas:** armazenam informações sobre os objetos indexados na forma (I, id) onde:
  - **I – *MBR*:** contorna o objeto indexado;
  - **Id:** Identificador do objeto que é uma referência a um endereço de memória que possui os dados do objeto.
- **Nós internos:** contêm informações da forma *(I, p)* onde:
  - **I – *MBR***: cobre todos os nós dos níveis inferiores;
  - **p:** ponteiro para um nó de nível inferior.

Cada nó é armazenado em uma página de disco cuja capacidade máxima é *M* e a capacidade mínima $m < M/2$ de entradas e devem satisfazer as seguintes propriedades:

- Cada nó folha contém entre *m* e *M* elementos, a menos que ele seja uma raiz;
- Para cada elemento em um nó folha, *I* é o menor *MBR* que o contém espacialmente;
- Cada nó interno tem entre *m* e *M* elementos, a menos que ele seja o nó raiz;
- O nó raiz tem pelo menos dois filhos, exceto ele seja um nó folha;
- Todos os nós folhas estão no mesmo nível da estrutura.

Na Figura 2.12, a estrutura hierárquica da *R-Tree* é formada por um nó raiz (1, 2, 3, 4) *e quatro n*ós folhas *(A, B)*, *(C, D, E)*, *(F, G)* e *(H, I, J)*.

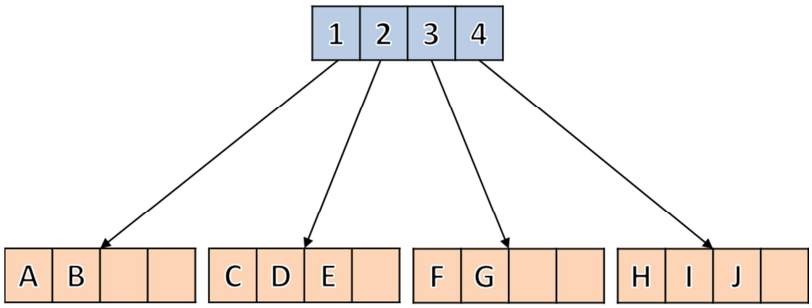


Figura 2.12 - Estrutura de indexação *R-Tree*. Adaptada de (MANOLOPOULOS et al., 2006).

A principal função da *R-Tree* é reduzir o espaço de consulta, descartando os objetos que não fazem parte do predicado espacial selecionado. Um grande benefício proporcionado pela *R-Tree* é que sua estrutura permite que vários nós, que não fazem parte do critério de busca, sejam descartados, diminuindo o acesso a disco. Porém, a sobreposição de nós internos pode fazer com que a busca percorra vários caminhos do nó raiz até os nós folhas.

2.6.1.1 Inserção

O processo de inserção de um novo elemento em uma *R-Tree* inicia no nó raiz e vai percorrendo a árvore até encontrar o nó folha mais apropriado, pois os dados devem ser inseridos em um nó folha. A Figura 2.13 ilustra a inserção de um novo objeto espacial L em uma árvore *R-Tree*.

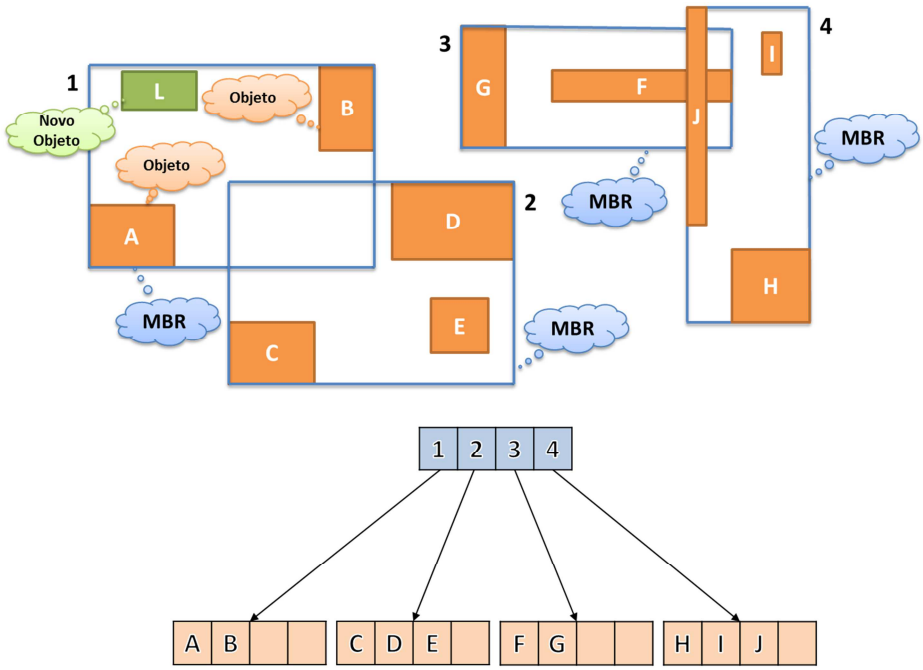


Figura 2.13 - Inserção de um novo objeto em uma *R-Tree*. Adaptada de (MANOLOPOULOS et al., 2006).

Esse processo é executado através do algoritmo *ChooseLeaf.* A partir do nó raiz o algoritmo percorre-se a árvore em busca de uma entrada cujo *MBR* irá necessitar do menor aumento de área, conforme ilustrado na Figura 2.14.

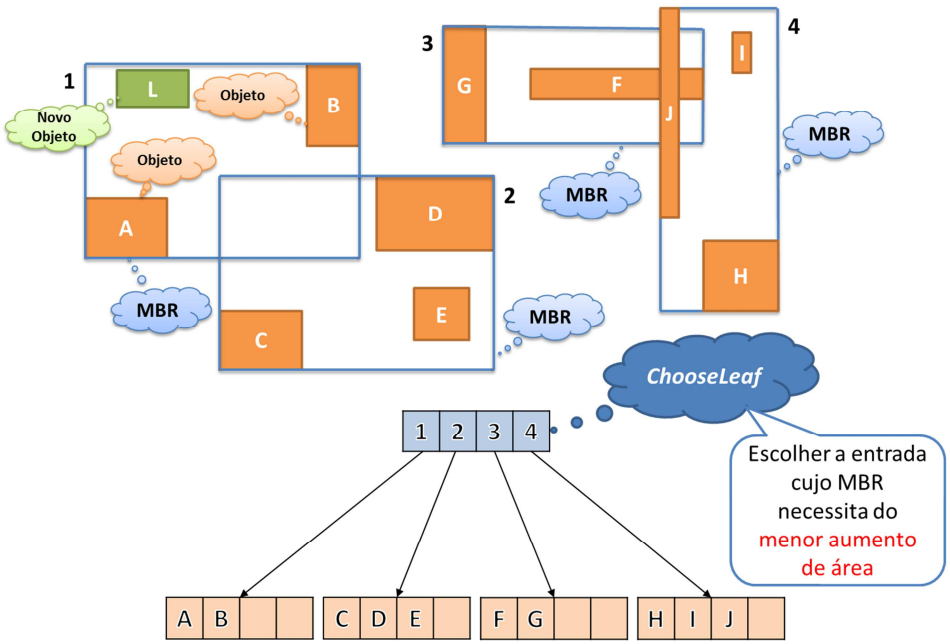


Figura 2.14 - Escolhe a entrada com menor aumento de área. Adaptada de (MANOLOPOULOS et al., 2006).

Caso o algoritmo identifique muitos nós que atendam aos requisitos, a próxima fase é identificar, dentro de todos os *MBRs* escolhidos, aquele que possui a menor área. Neste caso, será escolhida a entrada 1, pois seu *MBR* exige o menor aumento de área, conforme ilustrado na Figura 2.15.

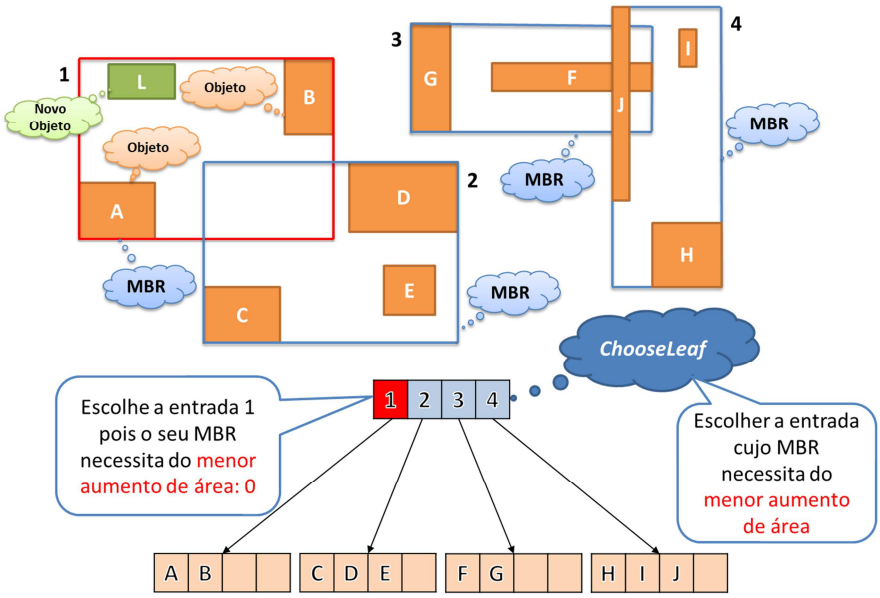


Figura 2.15 - *MBR* 1 com o menor aumento de área. Adaptada de (MANOLOPOULOS et al., 2006).

Após todos os requisitos serem cumpridos, deve-se verificar se o nó folha possui espaço para o novo objeto, caso possua, este é inserido e o processo se encerra. Neste caso, é necessário o ajuste da área do *MBR* do nó no qual o objeto foi inserido e isso pode se propagar até o nó raiz. Caso contrário, o processo se repete até que seja alcançado um nó folha, que é o nó em que o novo objeto deve ser inserido, conforme ilustrado na Figura 2.16.

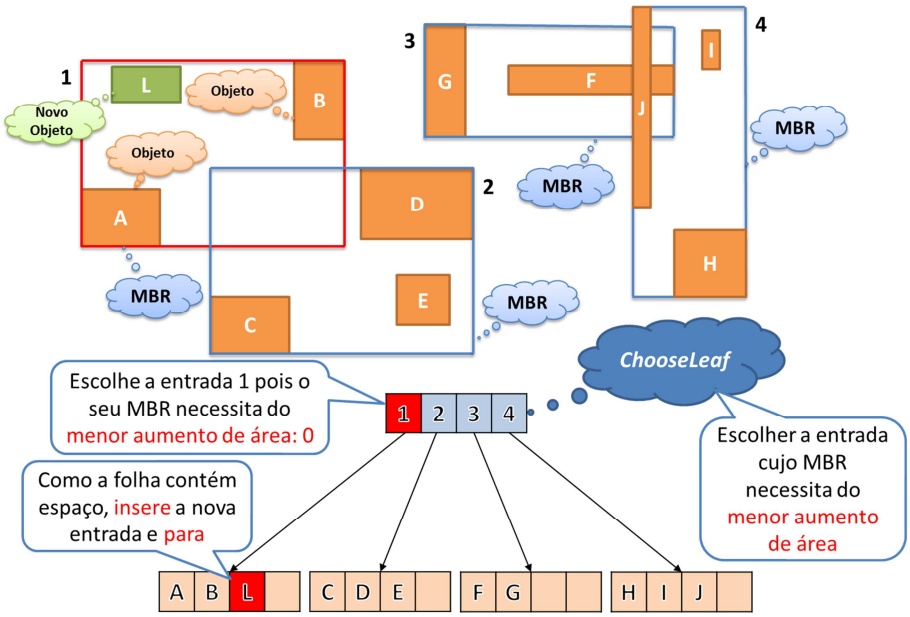


Figura 2.16 - Inserção do objeto em um nó folha. Adaptada de (MANOLOPOULOS et al., 2006).

Se não for encontrado nenhum nó para a inserção do objeto, um novo processo é acionado através do algoritmo *SplitNode,* que tem a função de dividir o nó em dois nós para que o objeto possa ser inserido. Se ocorrer a divisão, a estrutura deve ser reorganizada, verificando se a divisão se propagou nos níveis superiores. Finalmente, se o nó raiz se dividiu, é criada então uma nova raiz cujos filhos são os dois nós resultantes do último processo de divisão e a estrutura cresce um nível. As políticas utilizadas para divisão são as seguintes:

- **Algoritmo exaustivo** – é o método que testa todas as combinações possíveis de áreas de nós após a divisão para que se obtenha a área mínima após o processo;
- **Algoritmo de custo quadrático** – tenta encontrar uma área de divisão do nó que seja pequena, não necessariamente a menor possível. Ele funciona escolhendo duas das *M+1* entradas para serem os primeiros elementos dos dois novos grupos de tal modo que o par desperdiçaria a maior parte da área se ambos fossem colocados no mesmo grupo. A cada passo a área da expansão requerida para adicionar cada entrada restante a cada grupo é calculada e a entrada atribuída é aquela que possui a maior diferença de área entre os dois grupos;

- **Algoritmo de custo linear** – esse algoritmo é uma simplificação do algoritmo de custo quadrático e escolhe qualquer entrada do conjunto de entradas restantes para a sua execução.

2.6.1.2 Pesquisa

A pesquisa de um determinado objeto espacial é realizada em uma *R-Tree* pelo nó raiz e vai percorrendo a árvore até encontrar o nó folha onde os dados estão inseridos conforme o predicado espacial. Na Figura 2.17, a busca dos objetos contidos em uma determinada janela de consulta X em uma árvore *R-Tree*. A janela de busca X é representada por um retângulo com lados paralelos aos eixos.

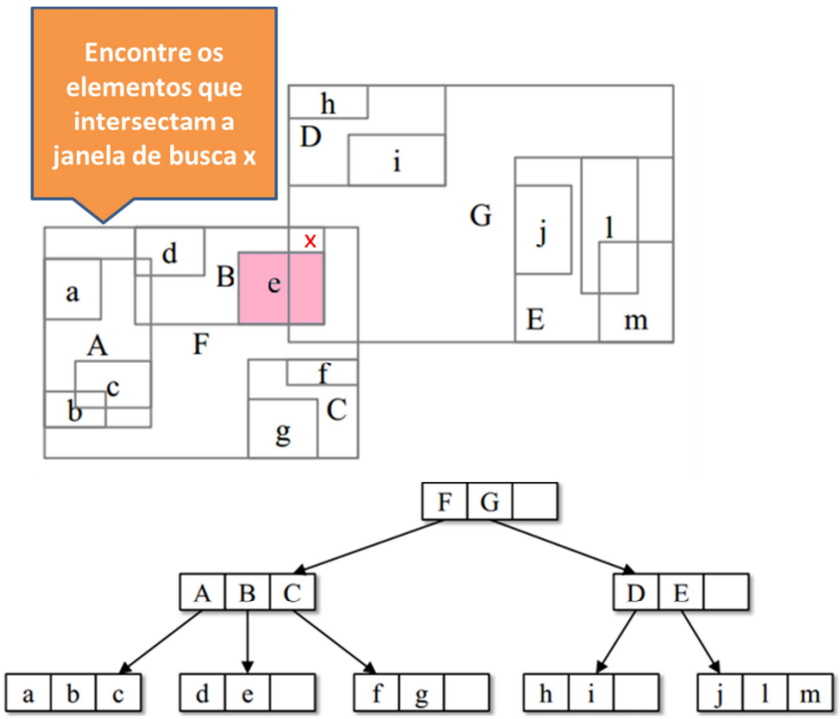


Figura 2.6 - Busca pelos objetos que intersectam a janela de busca X. Adaptada de (MANOLOPOULOS et al., 2006).

O primeiro passo é pesquisar no nó raiz as entradas F e G para verificar se elas intersectam a janela de busca, determinada pelo predicado espacial. Neste caso, conforme ilustrado na Figura 2.18, pode-se notar que as entradas F e G intersectam a janela de busca, então a subárvore delas deverá ser analisada.

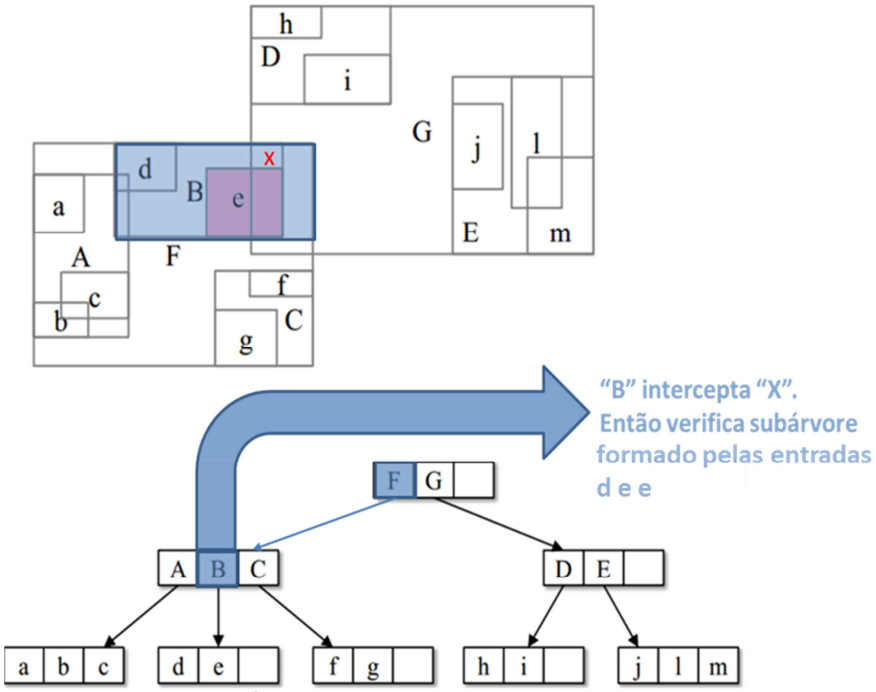


Figura 2.78 - Avaliação se as subárvores F e G intersectam a janela de busca X. Adaptada de (MANOLOPOULOS et al., 2006).

Ainda no processo de pesquisa das subárvores, para saber se estas intersectam a janela de busca, deve-se pesquisar a subárvore do nó F, que são os nós que possuem as entradas A, B e C, para saber se estas intersectam a janela de busca. Neste caso, a entrada A não intersecta a janela de busca, então ela é descartada. Já a entrada B intersecta a janela de busca, fazendo com que esta seja selecionada para uma nova pesquisa em seus elementos. Já a entrada C não intersecta a janela de busca. Esse processo deve ser repetido para todas as entradas e no final obtém-se um conjunto de possíveis entradas que deverão ser pesquisadas, conforme ilustrado na Figura 2.19.

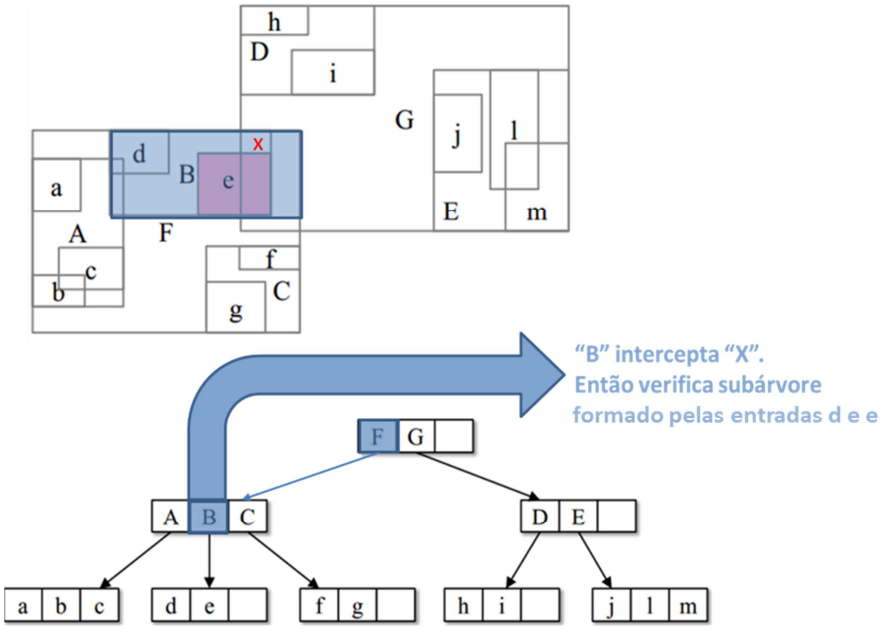


Figura 2.89 - Avaliação se as subárvores A, B e C intersectam a janela de busca. Adaptada de (MANOLOPOULOS et al., 2006).

Por fim, avaliando a entrada B, um nó folha cujos objetos nele armazenados devem ter, cada um, sua geometria analisada para determinar se há sobreposição entre a janela de busca e os objetos do nó, se o resultado for verdadeiro, então a entrada é adicionada ao conjunto de resposta, conforme ilustrado na Figura 2.20.

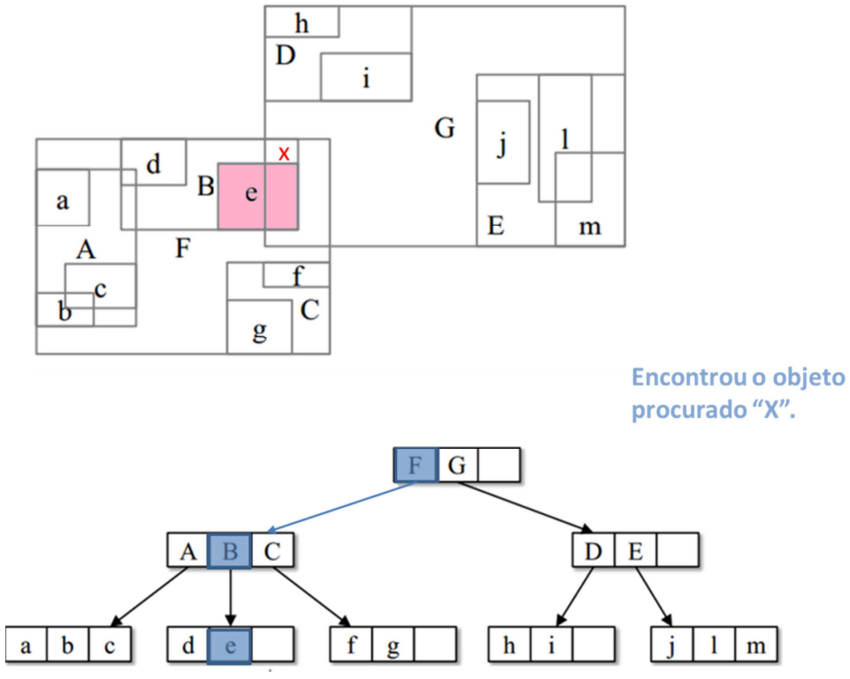


Figura 2.20 - Seleção do objeto que intersecta a janela de busca X. Adaptada de (MANOLOPOULOS et al., 2006).

2.6.1.3 Remoção

O processo de remoção de um objeto espacial em uma *R-Tree* inicia-se no nó raiz e vai percorrendo a árvore até encontrar o nó folha, onde o dado está inserido, ou seja, o mesmo procedimento de pesquisa. A Figura 2.21 ilustra a busca de um objeto espacial em uma árvore *R-Tree. A*pós a localização do objeto, este será removido.

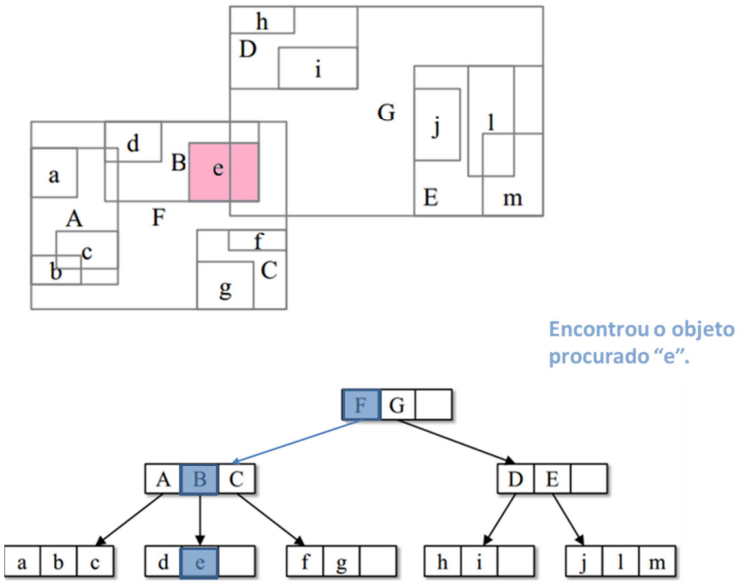


Figura 2.21 - Busca pelo objeto e que será removido. Adaptada de (MANOLOPOULOS et al., 2006).

Após a eliminação do objeto, é necessário verificar se a árvore precisa ser balanceada, caso o número de entradas do nó folha seja menor que as restrições exigidas para a construção da árvore, ou seja, a capacidade máxima de um determinado nó é *M* e a capacidade mínima *m < M/2*. Neste caso, se a capacidade mínima for menor do que a exigida, esta situação é denominada como *underflow*, então a árvore deverá ser balanceada em todos os seus níveis até a raiz, se necessário.

O processo de balanceamento de uma árvore, após a remoção de um objeto, ocorre da seguinte forma:

- Após detectar o evento de *underflow*, os objetos restantes nos nós folhas serão atribuídos a um conjunto temporário. Esse processo é executado em

todos os níveis até a raiz e todos os conjuntos afetados são atribuídos ao conjunto temporário;

- Após percorrer toda a árvore e identificar todos os níveis e objetos afetados, começa o processo de reinserção. Neste processo são utilizados os objetos atribuídos ao conjunto temporário e a árvore é balanceada utilizando o processo de inserção.

Para (GUTTMAN, 1984) o processo de reinserção utilizado na remoção de um objeto justifica-se por duas razões:

- Porque realiza a mesma função e é mais simples de se implementar, uma vez que a rotina de inserção pode ser reutilizada para este fim. O desempenho nesta operação não é afetado, as páginas visitadas durante a reinserção já estão na memória, pois foram utilizadas durante a busca pela entrada;
- A reinserção vai aos poucos refinando a estrutura espacial da árvore, evitando a deterioração gradual que poderia ocorrer se cada entrada estivesse sempre sob o mesmo nó pai.

Existem diversas variações da R-Tree, por exemplo, no trabalho de (FEDOTOVSKY et al., 2013) são destacadas as X-Tree (DESPAIN; PATTERSON, 1978), KDB tree (ROBINSON, 1981), Octree (YAMAGUCHI et al., 1984), RUM-tree (SILVA; XIONG; AREF, 2009) e R+-Tree (SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987), cada uma tentando melhorar um aspecto diferente.

2.6.2 *R+-Tree*

Conforme ilustrado na Figura 2.22, existe a possibilidade, na *R-Tree,* que os *MBRs* estejam sobrepostos. Neste caso, na busca de um determinado objeto espacial faz-se necessário percorrer os dois *MBRs* para encontrar o objeto, pois o mesmo objeto espacial pode estar contido em mais de um *MBR*. No exemplo, (CIFERRI, 2002; SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987), para a busca pelo objeto espacial H será necessário verificar os nós A e depois o B, prejudicando o tempo da busca.

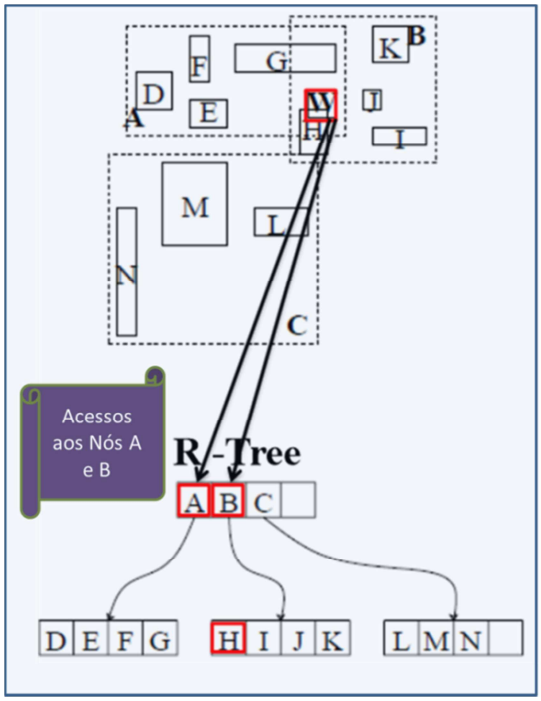


Figura 2.22 - Busca de um objeto espacial H em *MBRs* com interseção - *R-Tree.* Reproduzida de (FORNARI, 2006).

Para contornar esse problema, uma variação da *R-Tree* foi proposta, chamada *R*$^{+}$*-Tree*. O principal objetivo desta nova árvore é melhorar o tempo de busca de um determinando objeto espacial, principalmente no processamento de *Point Queries* (uma *Range Query* degenerada quando a janela de consulta corresponde a um ponto). Isso é obtido através da eliminação da interseção de *MBRs,* fazendo com que um determinando objeto espacial pertença a apenas um *MBR* (CIFERRI, 2002; FORNARI, 2006; SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987). A Figura 2.23 ilustra a situação, cujo resultado é a redução no tempo de busca de um objeto espacial.

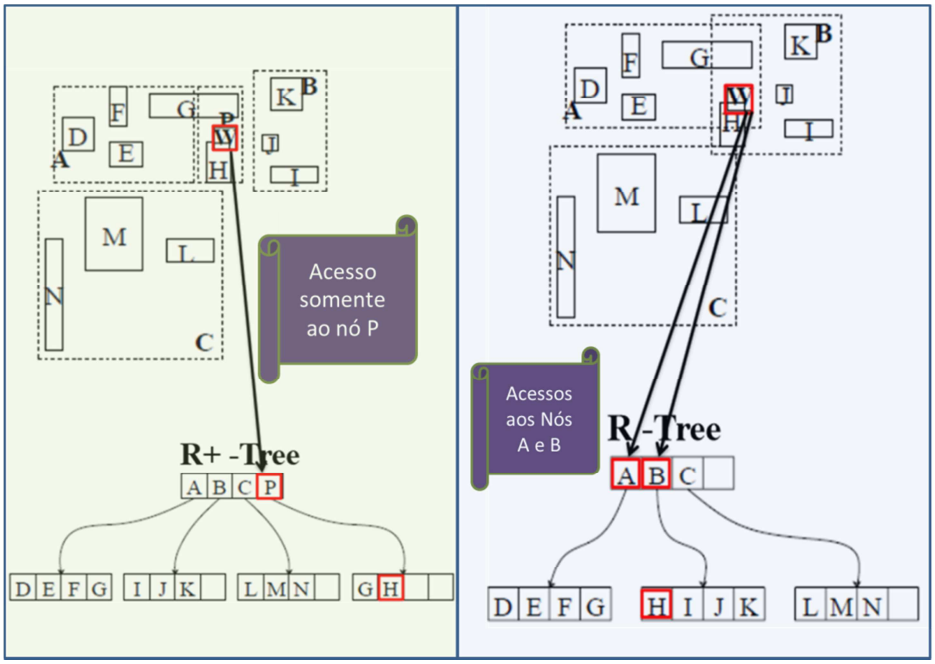


Figura 2.23 - Busca de um objeto espacial H em *MBRs* sem interseção *R+-Tree.* Reproduzida de (FORNARI, 2006).

A redução de acessos ao disco, em consultas de pontos, proporciona um melhor desempenho para a *R⁺-Tree*. No processamento de retângulos, para evitar que os mesmos se interseccionem, a *R⁺-Tree* causa divisões nos *MBRs* gerando um número maior de nós e maior espaço ocupado pelo índice (CIFERRI, 2002; FORNARI, 2006; SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987).

Segundo (CIFERRI, 2002; FORNARI, 2006; GAEDE; GÜNTHER, 1998), o desempenho de atualizações e inserções na *R⁺-Tree* é afetado pela necessidade de alterar os *MBRs* existentes e rebalancear a árvore. As Figuras 2.24a e 2.24b mostram exemplos de sobreposição de MBRs para as árvores *R-Tree* e *R⁺-Tree.*

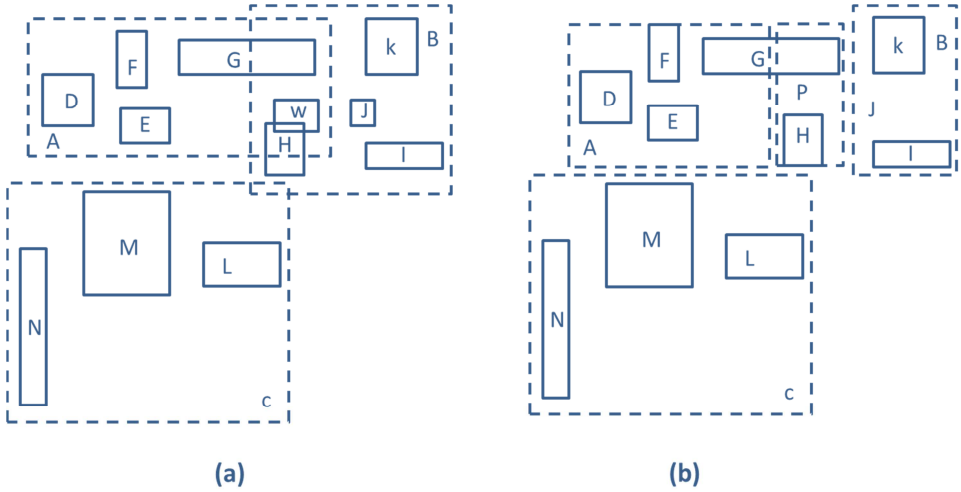


Figura 2.24 - Sobreposição na *R-Tree* e a *R+-Tree*. Reproduzida de (FORNARI, 2006).

A Figura 2.25 apresenta as *R-Tree* e *R*$^{+}$*-Tree* resultantes da divisão do espaço, como apresentado na Figura 2.24 acima.

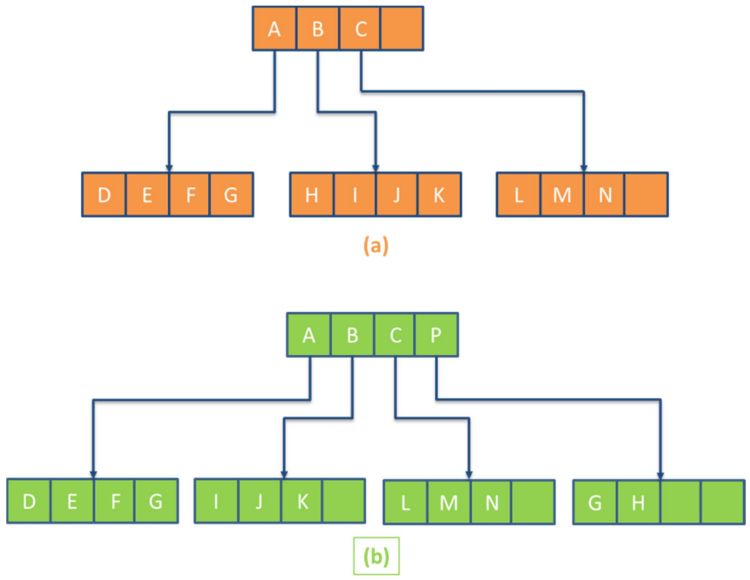


Figura 2.25 - Sobreposição na *R-Tree* e a *R+-Tree*. Reproduzida de (FORNARI, 2006).

A escolha de uma ou de outra variação da árvore está diretamente relacionada ao contexto da aplicação.

2.6.3 *Grid File*

Segundo (NIEVERGELT; HINTERBERGER; SEVCIK, 1984), o *Grid File* faz a divisão do espaço em células que são chamadas de *buckets*. É semelhante a imaginar quadrados sobre a superfície da terra no caso de utilização de dados geográficos. Esta estrutura é projetada para gerenciar blocos de armazenamento de tamanho fixo em disco. Os *buckets* têm capacidade de armazenar certa quantidade de registros, limitado pelo tamanho da memória principal. A Figura 2.26 mostra um *Grid File* armazenando objetos espaciais em um espaço bidimensional.

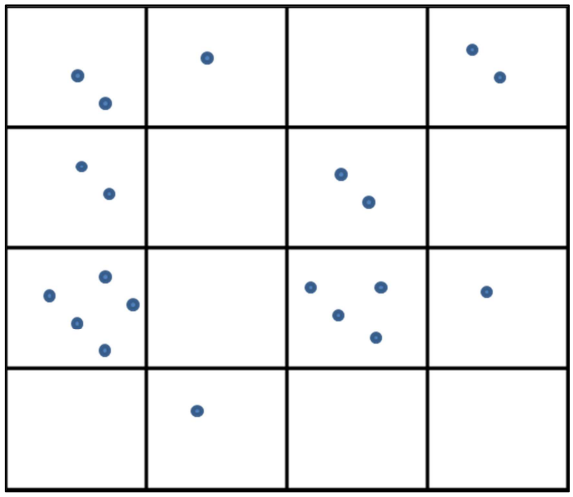


Figura 2.26 - *Grid File* armazenando objetos espaciais. Reproduzida de (NIEVERGELT; HINTERBERGER; SEVCIK, 1984).

Estrutura de acesso aos dados é composta de dois elementos: diretório e escala:

- **Diretório:** é um *array* bidimensional com ponteiros para *buckets*.
- **Escala Linear**: duas matrizes unidimensionais usadas para acessar a matriz de grade.

Na Figura 2.27 pode-se visualizar a representação dos elementos.

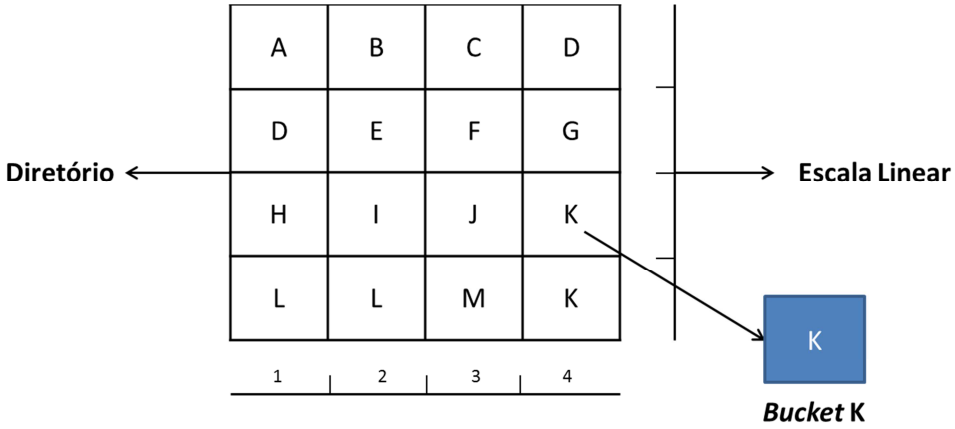


Figura 2.27 – Representação dos elementos. Reproduzida de (NIEVERGELT; HINTERBERGER; SEVCIK, 1984).

A inserção de uma nova entrada inicia-se com uma consulta na tabela de escala para identificar o *bucket* que possui as mesmas coordenadas do elemento que esta sendo inserido. Se o *bucket* selecionado tem espaço para receber a entrada o mesmo é inserido e o processo é finalizado. Se o *bucket* selecionado não possui espaço é realizada a divisão, dando origem a um novo *bucket*. Neste caso, os elementos de diretório e escala precisam ser atualizados para contemplar a inserção da nova entrada.

Para realizar a exclusão de um elemento deve-se inicialmente identificar o *bucket* que está associado ao elemento. A exclusão provoca a redução da utilização do espaço, necessitando que os *buckets* sejam desalocados e o conteúdo dividido. Assim como ocorreu na inserção os elementos de diretório e escala precisam ser atualizados.

Uma consulta do tipo *point queries* é executada através de dois acessos ao disco. Por meio das coordenadas do elemento a ser consultado é possível identificar o *bucket* a ser pesquisado, sendo esse o primeiro acesso ao disco. O segundo acesso é realizado no momento em que o *bucket* é lido para obter os dados armazenados.

2.7 Junção espacial

A combinação das estruturas de indexação tais como: *Grid File* (NIEVERGELT; HINTERBERGER; SEVCIK, 1984), *R-Tree* (GUTTMAN, 1984) *e* R^+-*Tree* (SELLIS; ROUSSOPOULOS; FALOUTSOS, 1987) com as estratégias de representação dos objetos espaciais através do *MBR* permite realizar de forma eficiente muitas operações espaciais (Consulta por faixa (*Range Query)*, K-vizinhos mais próximos (*KNN)* e junção espacial (*Spatial Join)*). Nesta seção, descreve-se uma das operações mais importantes à junção espacial e os seus respectivos algoritmos.

Segundo (JACOX; SAMET, 2007), a junção espacial pode ser conceitualmente definida a partir de duas relações R e S, tal que $R = \{r_1, ..., r_n\}$ e $S = \{s_1, ..., s_m\}$, onde r_i e s_j são objetos espaciais, $1 \leq i \leq n$ e $1 \leq j \leq m$. A operação verifica todos os pares (r_i, s_j) que satisfazem a um determinado predicado espacial, por exemplo o relacionamento topológico de interseção, isto é, $r_1 \cap s_j \neq \varnothing$. Esta operação é usada para combinar dois ou mais conjuntos de objetos com os seus respectivos predicados espaciais. Tipicamente, pode-se adotar como exemplo de junção espacial a seguinte situação: “Encontrar todos os rios e cidades que se intersectam”. Por exemplo, na Figura 2.28, o resultado da junção entre o conjunto de rios $R = \{r_1, r_2\}$ e cidades $C = \{c_1, c_2, c_3, c_4, c_5\}$, onde $R \cap C$ é $\{r_1, c_2\}$ e $\{r_2, c_3\}$ (DEBNATH, 2005; JACOX; SAMET, 2007).

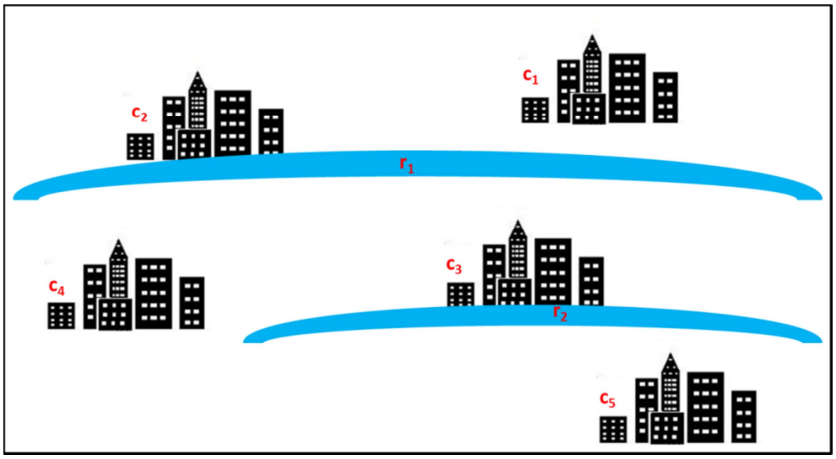


Figura 2.98 - Exemplo de junção espacial. Reproduzida de (DEBNATH, 2005).

Essa operação é de fundamental importância para seleção de informações a partir do cruzamento entre dois ou mais conjuntos de dados espaciais baseados em um predicado de seleção espacial. Porém essa operação possui um alto custo de processamento devido à necessidade de realização de um produto cartesiano dos 2 conjuntos e por isto, são utilizadas várias estratégias no intuito de melhorar o seu desempenho. Na literatura, podem-se encontrar pesquisas nesta área desde o ano de 1975.

Os algoritmos de junção espacial executam o processamento em duas fases, filtragem e refinamento, conforme Figura 2.29 (DEBNATH, 2005; ORENSTEIN, 1986).

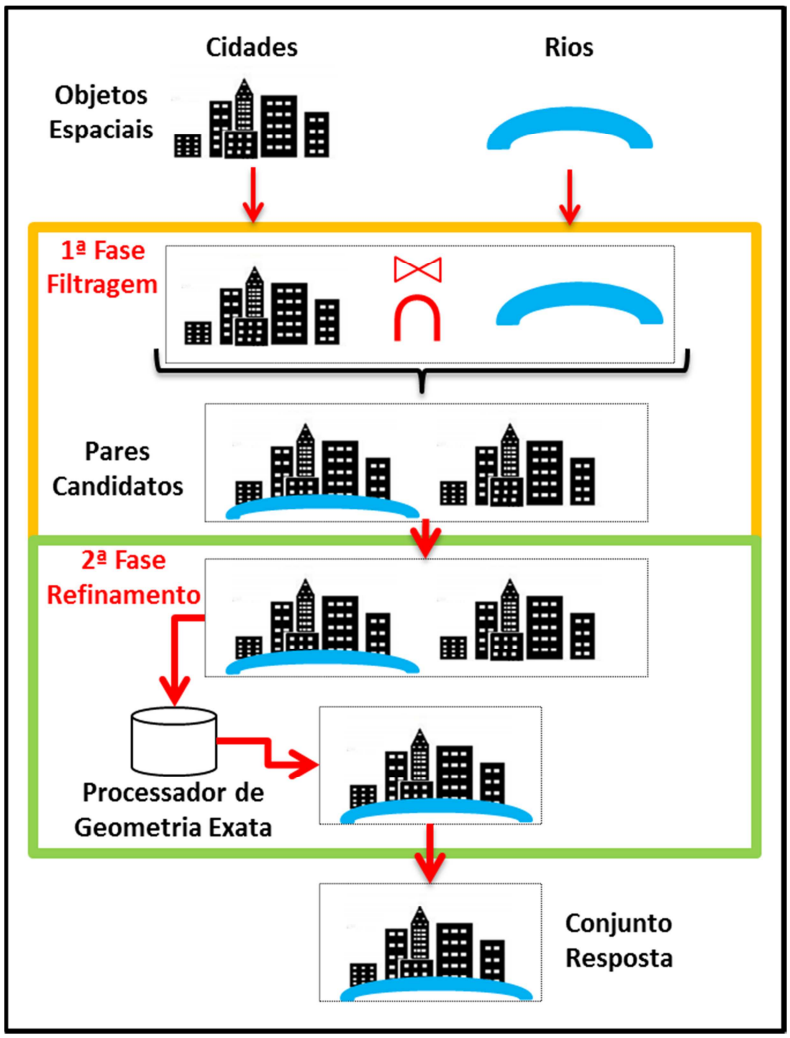


Figura 2.29 - Fases de filtragem e refinamento. Adaptada de (DEBNATH, 2005).

O processo como um todo tem um ganho de desempenho, pois a primeira fase é menos custosa. Nela, a quantidade de geometrias analisadas é muito menor

e permite podar – eliminar – todas as geometrias que não fazem parte do predicado analisado. Para a segunda fase do processo, será processada a geometria exata apenas dos objetos que passarem na primeira fase, tornando o processo, como um todo, mais rápido. Este ganho é permitido apenas se a fase de filtragem for extremamente eficiente na eliminação das geometrias que não fazem parte do conjunto resposta (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003).

2.7.1 Memória interna

Uma junção espacial é realizada nas aproximações dos objetos durante o estágio de filtro. Se o sistema possuir memória suficiente, ela será realizada inteiramente na memória interna. Caso contrário, a memória externa deverá ser usada para armazenar todos os dados ou partes deles durante o processamento. Existem também algoritmos de junção espacial em memória externa, que reduzem o tamanho do conjunto de dados, a fim de que eles possam ser processados usando as técnicas de memória interna (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003).

2.7.1.1 *Plane Sweep*

Para a etapa de processamento em memória primária, a junção espacial utiliza a interseção. Nesta etapa, são testados os predicados espaciais sobre os pares de objetos que estão em memória primária. Como exemplo, pode-se supor o teste de um predicado espacial de interseção sobre os *MBRs* dos objetos representados na Figura 2.30 (a, b e c). As situações a e b resultam em verdadeiro para o predicado espacial de interseção, enquanto que a situação c resulta em falso (FORNARI, 2006; JACOX; SAMET, 2007).

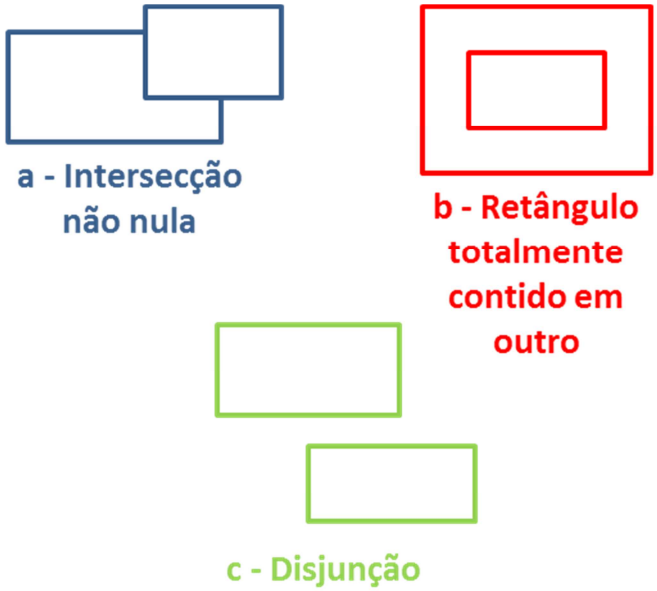


Figura 2.30 - Situações possíveis com o teste de retângulos. Reproduzida de (FORNARI, 2006).

Segundo (FORNARI, 2006; JACOX; SAMET, 2007), em situações mais complexas, em que se tem um conjunto de objetos em memória e precisa-se encontrar os pares de objetos que se intersectam, é proposto o algoritmo *Plane Sweep.* Ele baseia-se em uma linha L, chamada de *Sweep Line,* que desloca em um dos eixos x ou y. A Figura 2.31 ilustra o algoritmo para um conjunto de quatro objetos representados por seus envelopes.

Desta forma, encontra-se o início de um objeto utilizando o eixo x, conforme apresentando na Figura 2.31. Por ser o primeiro objeto, ele é inserido na tabela temporária *E. A seguir a* linha percorre até as coordenadas do próximo objeto. Novamente é testado se existe algum objeto na tabela temporária. Neste caso, o objeto o1 foi encontrado. Então é realizado um novo teste para saber se o novo objeto encontrado é o2 e se a resposta for positiva o par de objetos é colocado na lista de resposta R. Deve-se testar se o objeto possui interseção e se o algoritmo funciona com uma linha que percorre os eixos x e y.

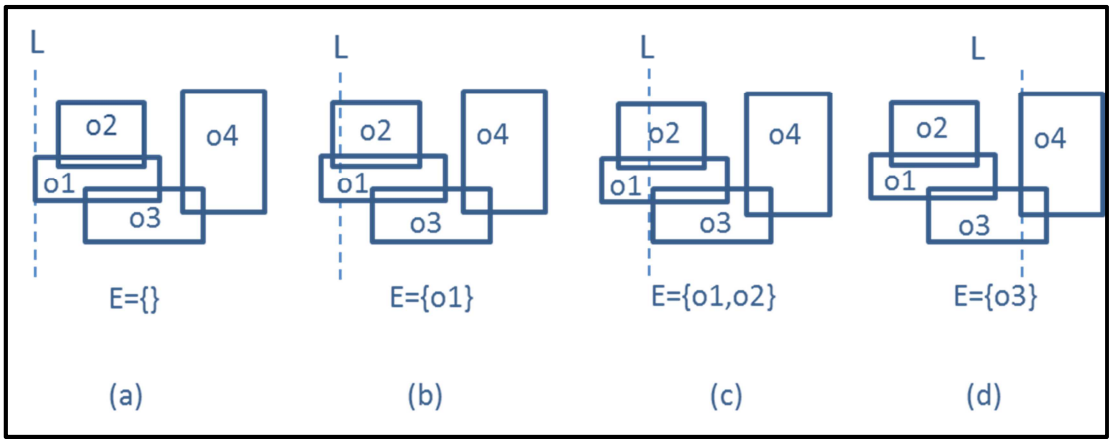


Figura 2.31 - Funcionamento do algoritmo *Plane Sweep*. Reproduzida de (FORNARI, 2006).

2.7.1.2 Algoritmos

Na literatura existem diversas propostas de algoritmos de junção espacial. Para facilitar o entendimento, dividem-se os algoritmos conforme a estratégia utilizada para acessar os dados, ou seja, pela disponibilidade de índices espaciais (BRITO, 2012; FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003), criando os seguintes grupos: arquivos sequenciais não indexados, arquivos ordenados, arquivos indexados e apenas um arquivo indexado (GUNTHER et al., 1998).

Para os arquivos sequencias não indexados, existem duas propostas de algoritmos: laços aninhados e subdivisão de espaço (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003). Os laços aninhados são uma adaptação do algoritmo tradicional de junção espacial de bancos de dados relacionais, adicionando a ideia do *plane-sweep* para analisar os pares de objetos em memória (KORTH; SILBERSCHATZ, 1999). A subdivisão de espaço divide este espaço em subespaços e para cada subespaço que contêm os objetos espaciais, é realizado o processamento da junção espacial. Pode-se citar, como exemplo de algoritmos que utilizam a subdivisão do espaço, o chamado *Partition Based Spatial Merge Join* (PBSM) (PATEL; DEWITT, 1996). Em (LO; RAVISHANKAR, 1996), é proposto o algoritmo *Spatial Hash Join* (SHJ), em que cada objeto é alocado à partição em que está contido ou á partição mais próxima. (KOUDAS; SEVCIK, 1997) propuseram o algoritmo que é denominado *Size Separation Spatial Join*, cuja divisão resultante é um conjunto de grades regulares, sendo uma para cada nível da árvore,

aumentando o número de linhas e colunas da raiz para as folhas. A junção é realizada percorrendo, sequencialmente, todos os arquivos de todos os níveis. Cada bloco de disco de cada arquivo é lido uma única vez e a memória deverá ser suficiente para conter um conjunto destes blocos em um mesmo instante de tempo.

Algoritmos baseados em arquivos ordenados ordenam cada conjunto de objetos separadamente, antes de realizar o *plane-sweep*, constituindo-se assim uma adaptação do algoritmo de *sort-merge* utilizado para a junção de relações (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA; COMPUTAÇÃO). Há dois algoritmos para este grupo que se diferenciam no tratamento dado para o caso do número de objetos ativos excederem a capacidade de memória. O algoritmo *Scalable Sweeping-based Spatial Join* propõe dividir o espaço em faixas (*strips*) (ARGE et al., 1998). Para (JACOX; SAMET, 2003) ao identificar no *Iterative Spatial Join* a impossibilidade de inserir um objeto na lista de ativos, gravam em um arquivo auxiliar de objetos a serem processados. Quando a *sweep line* atinge o limite final do espaço, o algoritmo de *plane-sweep* é reinicializado, processando apenas os objetos do arquivo auxiliar.

O terceiro grupo de algoritmos baseia-se em árvores espaciais para realizar a operação de junção. Uma árvore espacial é uma estrutura de dados utilizada para indexar objetos espaciais, sendo árvores-R as de uso mais amplo, merecendo maior atenção (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003). Em geral, os algoritmos de junção espacial baseados em árvores-R podem ser aplicados a quaisquer variações de árvore-R; o mais conhecido é o *Synchronized Tree* Transversal (STT) (BRINKHOFF; KRIEGEL; SEEGER, 1993). O STT percorre as duas árvores de índice a partir do nodo raiz de cada uma delas, de modo sincronizado. Para cada nó filho, na árvore do conjunto A, são identificados os nós filhos da árvore do conjunto B que atendem ao predicado de junção. Em função disso, uma lista de possíveis pares é criada. O algoritmo *Priority Queue Driven Process* (PQDP) é uma adaptação do *plane-sweep*, sendo as duas árvores percorridas por uma *sweep-line* que identifica os nós filhos a serem processados.

A quarta estratégia é utilizada quando apenas um dos conjuntos estiver indexado por uma árvore-R (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003). O algoritmo *Scan Index* utiliza, para cada objeto do conjunto B, seu *MBR* como “janela" e realiza uma consulta por ela na árvore de índice do conjunto A para localizar objetos que atendam ao predicado de junção. O algoritmo

Build&Match (B&M), consiste em construir a árvore-R correspondente ao conjunto B como uma primeira etapa do processamento, e, depois, aplicar o algoritmo STT. O algoritmo *Seeded-Join*, proposto por (LO; RAVISHANKAR, 1994), realiza a indexação do conjunto B, construindo uma estrutura específica, a *Seeded-Tree*, que é uma adaptação de árvore-R e em seguida, aplica-se o algoritmo STT. Já o algoritmo *Sort Sweep-Based Spatial Join* (SSBSJ) de (GURRET; RIGAUX, 2000) baseia-se em uma abordagem de ordenação. Primeiro os envelopes dos nós folhas da árvore-R existentes são ordenados por uma coordenada. O algoritmo *Priority Queue-Driven Process* (ARGE et al., 2000) pode ser adaptado para o caso de um dos conjuntos possuírem uma árvore-R. Para tanto, os objetos não indexados do conjunto B são ordenados por um eixo.

Também pode-se encontrar diversos trabalhos na literatura voltados à proposição de índices e algoritmos de junção espacial. O trabalho (BRITO, 2012) identifica os principais trabalhos voltados à criação de índices e algoritmos de junção espacial. Grande parte desses algoritmos dedica-se à fase de filtragem, que corresponde à seleção de pares candidatos. Dentre esse grande número de trabalhos destaca-se:

- (BAE; ALKOBAISI; LEUTENEGGER, 2006, 2010; BRINKHOFF; KRIEGEL; SEEGER, 1993) trabalham com os índices da família *R-Tree*;
- (JACOX; SAMET, 2007; MANOULIS, 2009; MANOULIS; THEODORIDIS; PAPADIAS, 2005) focam no custo de processamento e técnicas de otimização que podem ser utilizadas pelos algoritmos de junção espacial;
- (PAPADOPOULOS; RIGAUX; SCHOLL, 1999), produziram um estudo detalhado sobre o plano de execução das consultas de junção espacial;
- Em (GUNTHER, 1993), é realizado um estudo sobre a utilização dos índices espaciais nas operações de junção espacial;
- No trabalho de (BRINKHOFF et al., 1994) são utilizadas as aproximações progressivas dos objetos espaciais; e

(AREF, 2007; BACK et al., 2010; CHEN; CHANG, 2009; JACOX; SAMET, 2007) baseiam-se em outros tipos de estruturas, tais como *Quadtree* (GARGANTINI, 1982), *Double transformation* (DOT) (SHOWCASE et al., 1991), *NA-trees* (CHANG; LIAO; CHEN, 2003).

Esta tese de doutorado usou algoritmos do terceiro grupo, ou seja, os dois arquivos indexados, baseando-se em árvores espaciais para realizar a operação de junção espacial. Os algoritmos utilizados foram adaptados para o processamento em *clusters* de computadores, conforme descrito na seção 3.6.2.

2.7.2 Desempenho

Segundo (JACOX; SAMET, 2007), a velocidade do processador, o custo de E/S em memória secundária, a indexação dos dados, o tamanho dos dados e o processamento em memória interna são as principais características que podem influenciar o desempenho dos algoritmos de junção espacial, consequentemente afetando diretamente em sua especificação. Os algoritmos mais recentes de junção espacial são afetados diretamente pelo custo de E/S, fazendo com que o foco na melhoria dos algoritmos tenha por objetivo reduzir a quantidade de dados lidos e escritos para a memória interna.

Porém, nos últimos anos, algumas pesquisas demonstraram que o processamento em memória primária do computador é responsável por até 95% do tempo de resposta do algoritmo de junção espacial (SUN; AGRAWAL; EL ABBADI, 2003). Essa informação exige que os novos algoritmos de junção espacial foquem também no processamento em memória interna e não se restrinja apenas ao número de acessos a disco (FORNARI, 2006; GATTI, 2000; JACOX; SAMET, 2007; ROCHA, 2003).

2.8 Considerações finais

Neste capítulo, foram apresentados os principais conceitos relacionados a está pesquisa de doutorado com destaque para os tipos de dados espaciais que são combinados para formar os objetos espaciais, que se relacionam com outros através dos relacionamentos topológicos. Este trabalho concentra-se na operação de junção espacial, que combina dois conjuntos de objetos espaciais com base no relacionamento topológico de interseção. Para o processamento das operações espaciais é utilizado a abstração das geometrias para simplificar e melhorar o

desempenho conhecidos como *MBR*. Essa forma de representar as geometrias dos objetos causa a perda de precisão na representação delas, tornando-as imprecisas, porém facilita o processo de avaliação dos predicados espaciais. A melhoria do desempenho é obtida com o uso de estruturas de indexação com ênfase nos índices de *Grid File* e *R-Tree* que serão utilizados no decorrer deste trabalho.

Na próxima seção será apresentado o *SpatialHadoop* o primeiro *framework MapReduce* com suporte nativo para dados espaciais.

Capítulo 3

SPATIALHADOOP

Devido à obtenção de enormes volumes de dados espaciais provenientes de diferentes fontes, há uma demanda crescente para explorar a eficiência computacional da estrutura Hadoop MapReduce no processamento de consultas espaciais em clusters de computadores. No entanto, o Hadoop é inadequado para processar consultas espaciais de forma eficiente, pois seu núcleo não tem conhecimento das propriedades desses dados espaciais. O SpatialHadoop é um Framework MapReduce completo, com suporte nativo para dados espaciais. SpatialHadoop é uma extensão abrangente para Hadoop que permite o tratamento de dados espaciais nas principais camadas do Hadoop, disponibilizando uma linguagem simples e de alto nível para os tipos de dados espaciais e operações espaciais. Seu núcleo implementa tradicionais índices espaciais, como Grid, R-Tree e R^+-Tree, que são adaptados ao ambiente MapReduce. Além disso, é equipado com diversas operações-padrão de geometria computacional. O SpatialHadoop é utilizado como componente principal em três sistemas MNTG, TAREEG e Shahed. No entanto, o processamento eficiente de junção espacial em SpatialHadoop ainda é um desafio devido ao fato que este framework não trata adequadamente a alocação conjunta de dados espaciais relacionados, tampouco trata do escalonamento de tarefas adaptado a alocação de dados espaciais relacionados.

3.1 *Big Data*

Na última década, observam-se inúmeras revoluções tecnológicas e culturais na sociedade; dentre elas a que mais vem chamando a atenção é a explosão no volume de dados coletado, disponibilizado e publicado nas mais diversas mídias e na *internet*. Este fenômeno tecnológico é conhecido como *Big Data*. O conceito de

Big Data é abstrato, surgiu em meados de 2010, para designar a tendência tecnológica de gerar enormes volumes de dados, de diferentes origens e formatos, que são extremamente difíceis de armazenar integralmente, processar e analisar com as tecnologias de banco de dados tradicionais (ANTÔNIO; MACÊDO; MACHADO, 2013; PHILIP CHEN; ZHANG, 2014). O relatório da *Mckinsey Global Institute* (MANYIKA et al., 2011)*,* define *big data* como "grupo de dados do qual o tamanho está além da habilidade de captura, armazenamento e análise por um típico *software* de banco de dados".

O termo *"Big Data*" é usado principalmente para descrever o volume, variedade e velocidade de dados, chamados de "3 Vs de *Big Data*" (CHEN; WANG; LU, 2012; PHILIP CHEN; ZHANG, 2014). O enorme volume de dados é a principal e mais importante característica já estampada na nomenclatura "*big*". A utilização de diferentes fontes e tipos de dados é denominada de variedade e a geração extremamente rápida, constante e em tempo real destes dados é chamada de velocidade (PHILIP CHEN; ZHANG, 2014). Demchenko e outros (DEMCHENKO et al., 2013) agregam outras duas características em sua teoria para o trabalho com *Big Data,* chamadas de "5 Vs": a veracidade e o valor. Segundo os autores, a veracidade trata da necessidade de que os dados coletados tenham origem comprovada e reputação, ou seja, dados confiáveis. O valor é trabalhar com dados que de fato tenham significado e que sejam fontes de valor agregado para a tomada de decisão.

Existe uma grande quantidade de pesquisas em curso sobre a melhor forma de lidar e processar estes enormes volumes de dados.

3.2 *MapReduce*

Uma das soluções encontradas para o processamento de enormes volumes de dados é o *MapReduce* do *Google*. O *MapReduce* foi introduzido pelos engenheiros do *Google* - Jeffrey Dean e Sanjay Ghemawat. Segundo (DEAN; GHEMAWAT, 2008), o *MapReduce* é um modelo de programação criado e implementado pela empresa Google (GHEMAWAT; GOBIOFF; LEUNG, 2003) para processamento de grandes volumes de dados não estruturados, amplamente

utilizados por pesquisadores e organizações para a computação em *clusters* de computadores. O processamento é conceitualmente simples, porém a entrada de dados é geralmente muito extensa e requer o processamento distribuído em milhares de nós, a fim de obter um tempo de processamento aceitável.

3.2.1 Arquitetura

O processamento *MapReduce* é disponibilizado através do uso de uma biblioteca de funções que recebem um conjunto de chaves e valores, processam e devolvem outro conjunto de chaves e valores. Tudo isto é basicamente realizado através do uso de duas funções (DEAN; GHEMAWAT, 2008):

- ***Map:*** O usuário especifica uma função que irá processar os dados de entrada. Este processamento irá gerar um conjunto intermediário de chaves e valores.
- ***Reduce:*** Após a execução da função *Map*, será feita uma mescla para identificar os valores que compartilham a mesma chave com objetivo de combinar os dados derivados apropriadamente em um conjunto menor de dados.

Para (SOUSA; MOREIRA; MACHADO, 2009), a função de Map, recebe como entrada um arquivo e gera como saída um conjunto de registros no formato de chave-valor. A função de Reduce recebe a saída do processamento da função de Mapeamento no formato de chave-valor e gera como resultado os registros que são armazenados em arquivos de saída, conforme ilustrado na Figura 3.1.

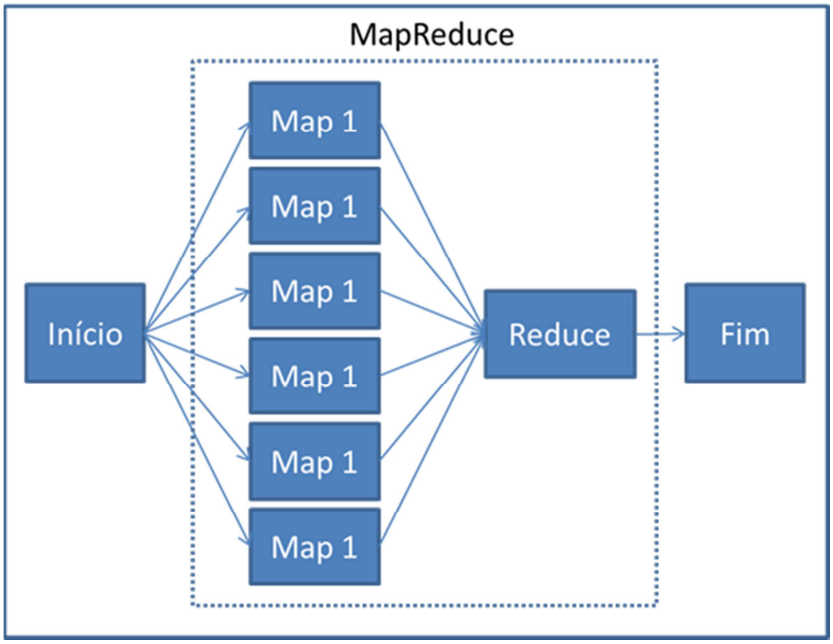


Figura 3.1 - Fases do *MapReduce*. Reproduzida de (SOUSA; MOREIRA; MACHADO, 2009).

O processamento no sistema *MapReduce* é gerenciado por um processo *Master*, cuja função é coordenar o processamento, gerenciar o processo de agrupamento de registros e distribuir os blocos para os *DataNodes* de forma equilibrada. Programadores sem experiência em sistemas distribuídos e computação paralela podem facilmente utilizar os recursos destes sistemas, pois a programação *MapReduce* abstrai toda a dificuldade de utilizar computação paralela, distribuir dados, balancear cargas, obter tolerância a falhas e ainda fornecer um alto desempenho de processamento (DEAN; GHEMAWAT, 2008).

3.2.2 Estrutura de dados no *master*

O Master mantém várias estruturas de dados de controle (DEAN; GHEMAWAT, 2008):

- Para cada trabalho de *map* e *reduce* seu *status* é armazenado (parado, em execução ou concluído) como também a identificação do nó escravo.
- Controle de armazenamento dos arquivos intermediários que devem ser repassados aos trabalhos de *reduce* após a conclusão dos trabalhos de Map.

- Controle de envio incremental para nós escravos com tarefas *reduce* em execução.

3.2.3 Tolerância a falhas

A tolerância a falhas é prevista pela biblioteca de *MapReduce* e os seguintes tipos de falhas (DEAN; GHEMAWAT, 2008) são previstos:

- Falha no nó escravo*:*
  - O *master* verifica periodicamente os escravos, se não tiver qualquer resposta em um tempo pré-determinado, o processo é marcado como falho.
  - Se o escravo falhar no processo de mapeamento, todo o processo é reiniciado desde o início. Isto ocorre porque o resultado de uma função *map* é armazenado no disco local do nó com falha.
  - Trabalhos concluídos de *reduce* não precisam ser reiniciados, pois o resultado é armazenado no sistema de arquivos global.
- Falha do Master:
  - Em caso de falha do processo *master*, um novo processo poderá ser iniciado a partir do último *checkpoint*.
  - Como existe apenas um *master*, uma falha destas é indesejada e irá abortar todo o processamento *MapReduce*.
  - Os usuários podem verificar a falha e reiniciar o processamento após a restauração do *master*.

3.2.4 Localização

A largura de banda de rede é economizada no modelo de programação *MapReduce*, pois se tira vantagem de que os dados de entrada são armazenados nos discos locais dos nós que compõem o *clusters* de computadores. O sistema de arquivos distribuídos *Google File System (GFS)* (GHEMAWAT; GOBIOFF; LEUNG, 2003), que é utilizado no modelo, divide cada arquivo em pedaços de 64 MB e armazena de forma redundante por padrão: três cópias de cada um destes pedaços em nós diferentes dos *clusters* de computadores. Este armazenamento é relevante para o *master* que irá iniciar o trabalho de *map* se possível em um nó que possui

uma réplica do arquivo de entrada. Desta forma, a tendência é não consumir nenhuma banda de rede (DEAN; GHEMAWAT, 2008).

3.2.5 Granularidade de tarefas

O trabalho de *map* é subdividido em várias partes, assim como o *reduce*. Desta forma, o trabalho é escalonado em vários processos *worker*. Isto cria uma granularidade de processamento dinâmico capaz de balancear a carga de trabalho. Além disto, em caso de falha de algum *worker* este pode ser realocado para outros nós rapidamente (DEAN; GHEMAWAT, 2008).

3.3 *Hadoop*

O Hadoop (SHVACHKO et al., 2010) é um sistema de arquivos distribuídos, criado por *Doug Cutting,* em 2005, no *Yahoo* (APACHE, 2014d)*,* fornece uma estrutura básica para análise e processamento de volumes massivos de dados utilizando a programação *MapReduce* (DEAN; GHEMAWAT, 2008). Ele tem como uma das principais características, o particionamento dos dados em milhares de nós e o processamento distribuído em *clusters* de computadores. Um *cluster Hadoop* possibilita o aumento da capacidade computacional, o armazenamento e a comunicação, simplesmente adicionando nós de baixo custo. Na pratica, o *Hadoop* é uma combinação de dois projetos, conforme ilustrado na Figura 3.2, que são:

- ***MapReduce* (*MR*):** é um *framework* para processamento paralelo e distribuído de volumes massivos de dados em *clusters* de computadores;
- ***Hadoop Distributed File System* (*HDFS*):** sistema de arquivo para o armazenamento distribuído de volumes massivos de dados em *clusters* de computadores.

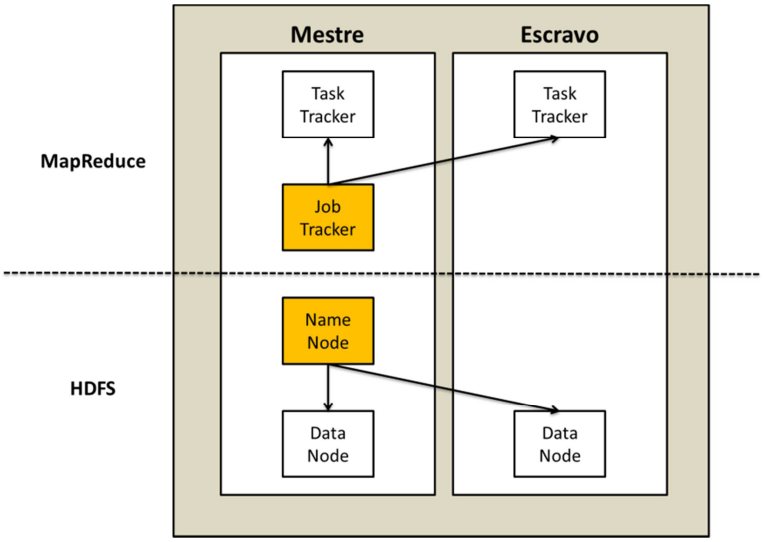


Figura 3.2 - A combinação dos dois projetos do *Hadoop.* Reproduzida de (BORTHAKUR, 2007).

O *HDFS* é o sistema de arquivos distribuído do *Hadoop que a*rmazena os metadados e os dados separadamente. Os metadados são armazenados em um servidor dedicado, chamado de *NameNode e o*s dados são armazenados em outros nós chamados de *DataNodes.* Inspirado no Google File System (GFS) (GHEMAWAT; GOBIOFF; LEUNG, 2003), projetado para ser implantado em *hardware* de baixo custo, organizados dentro de um *cluster*, onde o conteúdo dos arquivos são replicados em múltiplos *DataNodes* para fins de confiabilidade, garantindo a durabilidade dos dados (BORTHAKUR, 2007; SHVACHKO et al., 2010; WHITE, 2012). A Figura 3.3 ilustra a arquitetura do *Hadoop* e seus principais componentes.

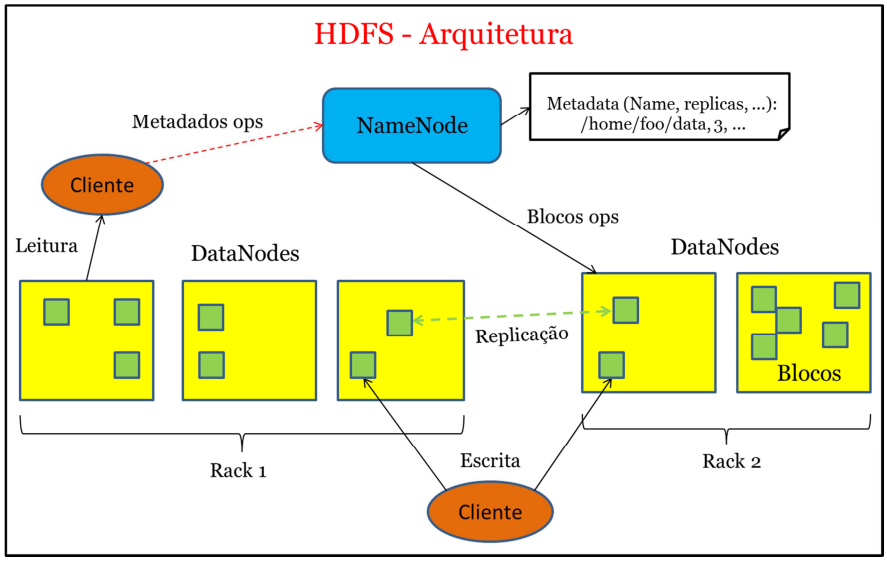


Figura 3.3 - Arquitetura do *HDFS*. Reproduzida de (BORTHAKUR, 2007).

3.3.1 *NameNode*

Segundo (SHVACHKO et al., 2010; WHITE, 2012), cada *cluster* contém um único *NameNode*, que é o componente de *software* responsável por armazenar em memória os metadados referentes à localização dos arquivos gerenciados, conforme ilustrado na Figura 3.4. O *namespace* de um *HDFS* é uma hierarquia de arquivos e diretórios. Estes arquivos e diretórios são representados no *NameNode* como *inodes;* estes registram informações como permissões, controle de acesso, modificações e quotas de disco. O conteúdo do arquivo é dividido em grandes blocos (por padrão 64 MB) e cada bloco de arquivo é independente e poderá ser replicado para múltiplos *DataNodes, local de armazenamento dos dados,* (por padrão três réplicas). O *namespace* é mantido pelo *NameNode*, assim como o mapeamento dos blocos de arquivos para os *DataNodes*. Um cliente *HDFS* que precisa fazer a leitura de um arquivo, primeiramente comunica-se com o NameNode para obter a localização dos blocos de dados que compõem o arquivo e então faz a leitura nos *DataNodes* mais próximos. O processo de gravação dos dados é iniciado através de uma requisição ao *NameNode,* que designa três *DataNodes* para armazenar os blocos de dados. O cliente então realiza a gravação dos dados nos nós selecionados.

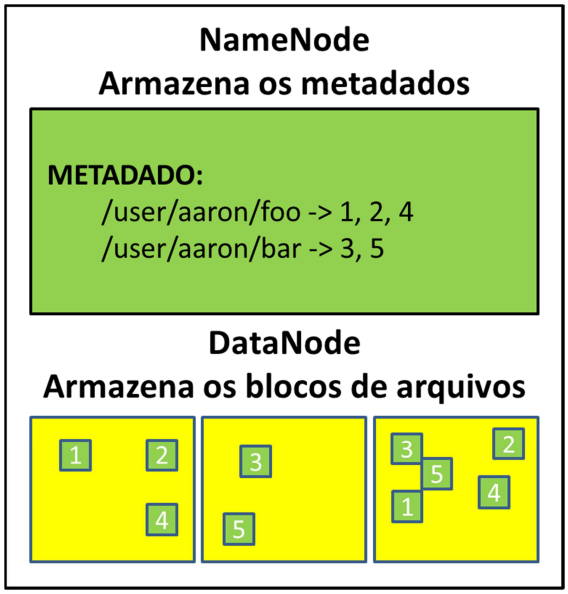


Figura 3.4 - Armazenamento dos Metadados. Reproduzida de (BORTHAKUR, 2007).

O *namespace* do *HDFS* é armazenado na memória principal. Os dados de *inode* e a lista de blocos que compõem cada arquivo formam os metadados denominados *image*. A gravação persiste na *image* armazenada no disco local do servidor, que é chamado de *checkpoint*. Também são armazenadas pelo *NameNode* as modificações da *image* em um registro de alterações denominado *journal,* armazenado no disco local. Com o objetivo de durabilidade e confiabilidade dos dados, cópias redundantes dos *checkpoints* e do *journal* são feitas para outros nós (GOLDMAN et al., 2012; SHVACHKO et al., 2010; WHITE, 2012).

3.3.2 *DataNode*

Segundo (SHVACHKO et al., 2010; WHITE, 2012), o *DataNode* armazena cada bloco de réplica em dois arquivos gravados no disco local do servidor. O primeiro arquivo contém os dados propriamente ditos e o segundo são os seus metadados, com informações de geração e integridade do arquivo. O tamanho do arquivo de dados é igual ao tamanho do bloco e não necessita de espaço extra de arredondamento para o tamanho nominal do bloco, como é feito nos sistemas de arquivos tradicionais. Na inicialização do *DataNode,* uma conexão com o *NameNode* é aberta, a fim de realizar um processo denominado *handshake*, que é uma validação da identificação do *namespace* e da versão de software do *DataNode*. Se ambos não forem iguais aos armazenados no *NameNode,* o *DataNode* é

automaticamente desligado. *DataNodes* recém-criados e que não possuam nenhuma identificação de *namespace* são permitidos a se unir ao *cluster* e, desta forma, recebem a identificação de *namespace* do sistema. A identificação do *namespace* é armazenada de forma persistente em todos os nós que compõem o *cluster* e é obtida quando o sistema de arquivos é iniciado. Nós com identificações incompatíveis não podem integrar o *cluster*, pois isto colocaria em risco a integridade do sistema de arquivos.

A versão do *software* também é importante, pois versões incompatíveis podem levar à perda dos dados. Após todas as validações, o *DataNode* é então registrado ao *NameNode* e recebe a sua identificação de armazenamento (*Storage ID*). A identificação de armazenamento é única, persistente e é um identificador interno do *DataNode* que fará com que ele seja identificado até mesmo se ele for reiniciado com outro endereço *Internet Protocol* (*IP*) ou porta diferentes. Um relatório de armazenamento contendo todas as réplicas armazenadas no *DataNode* é enviado ao *NameNode*. O primeiro é enviado assim que o *DataNode* é registrado. Relatórios posteriores são enviados a cada hora. Este relatório contém informações de identificação e tamanho de blocos, informações de data e hora de criação. O objetivo deste relatório é manter o *NameNode* atualizado de onde e como os blocos estão armazenados no *cluster*.

Os *DataNodes* enviam a cada três segundos informações de controle denominados *heartbeats* para informar ao *NameNode* que eles estão em funcionamento. Se estas informações não forem enviadas em um tempo máximo de dez minutos, o *NameNode* considera este *DataNode* falho e inicia o processo de reposição das réplicas que eram mantidas naquele nó em outros *DataNodes* em funcionamento. O *NameNode* utiliza a resposta ao *heartbeat* para enviar, se necessário, as seguintes instruções aos *DataNodes* (WHITE, 2012):

- Replicar blocos para outros nós;
- Remover réplicas locais;
- Registrar novamente ou desligo nó;
- Enviar imediatamente relatório de armazenamento.

Estas informações são muito importantes para manter a integridade do sistema e é crítico manter os *heartbeats* até em grandes *clusters*. O *NameNode*

pode processar milhares de *heartbeats* por segundo sem afetar as suas outras operações.

3.3.3 Cliente

Segundo (BORTHAKUR, 2007; SHVACHKO et al., 2010; WHITE, 2012), o acesso ao *HDFS* é realizado através de uma biblioteca (*HDFS Client*), que fornece a *interface* necessária para as aplicações dos usuários acessarem o sistema de arquivos. Assim como nos sistemas de arquivos convencionais, são permitidas operações como leitura e gravação de arquivos e diretórios. O usuário irá referenciar apenas arquivos e diretórios nos caminhos do *namespace*. A interface disponibilizada abstrai toda a dificuldade de gerenciamento de metadados, réplicas, nós para armazenamento distribuído, etc. Em uma operação de leitura e escrita, que ocorrem de forma similar, o *HDFS Client* solicita ao *NameNode* a lista de *DataNodes* que possuem as réplicas do arquivo necessário. Ele então acessa diretamente o *DataNode* e requisita a transferência dos blocos desejados. Já em uma operação de gravação, são solicitados ao *NameNode* os *DataNodes* para armazenarem as réplicas do primeiro bloco do arquivo. Assim como na leitura, o envio dos dados é direto ao *DataNode*. Quando o primeiro bloco estiver cheio, é então feita uma nova requisição de *DataNodes* para armazenarem o próximo bloco e uma nova conexão direta é aberta para / com novos *DataNodes* fornecidos. A Figura 3.5 ilustra as interações do *Cliente*, *NameNode* e *DataNode*:

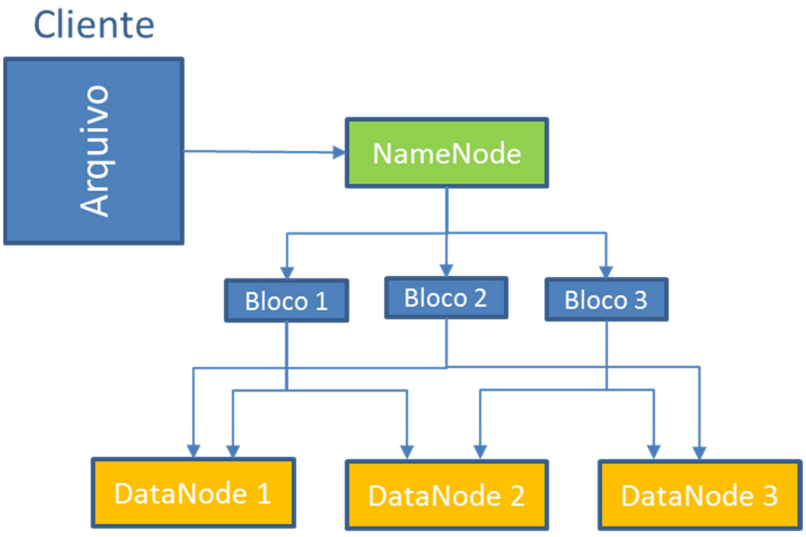


Figura 3.5 - Interações do cliente, *NameNode* e *DataNode*. Reproduzida de (BORTHAKUR, 2007).

Em uma biblioteca disponibilizada no *HDFS,* é exposta a localização dos blocos dos arquivos e isto permite às aplicações de *MapReduce* escalonar tarefas nos nós onde os dados estão armazenados e isto aumenta muito o desempenho de leitura do sistema (SHVACHKO et al., 2010).

3.3.4 Operações e gerenciamento de réplicas

Segundo (SHVACHKO et al., 2010; WHITE, 2012), no *HDFS* os dados são inseridos através de aplicações que criam novos arquivos e gravam os dados. Após o arquivo ser salvo, os dados não podem ser mais alterados ou removidos, exceto quando novos dados são inseridos no mesmo arquivo após ele ser reaberto. O modelo utilizado no *HDFS* é gravar uma vez e ler várias. O *HDFS Client,* quando abre um arquivo para gravação, recebe uma permissão para executar a operação e nenhum outro usuário poderá alterar aquele arquivo. Quando do término da gravação dos dados, a permissão é revogada.

A leitura não é influenciada por esta permissão de gravação e muitos usuários concorrentes podem ler o arquivo. Em um *cluster* de milhares de nós, é comum ocorrerem falhas de nós ou falha em dispositivos de armazenamento dos nós e, desta forma, uma réplica armazenada na estrutura falha torna-se corrompida. Para tratar este tipo de problema, o *HDFS* gera e armazena em arquivos separados uma verificação para cada bloco de arquivo no *HDFS*. Na leitura de arquivos do *HDFS*, cada bloco de dados e seus respectivos arquivos de checagem são enviados para o *DataNode* que, por sua vez, irá computar a checagem dos dados recebidos e comparar com os armazenados e caso não sejam iguais, o *NameNode* é notificado pelo cliente, que irá fazer a leitura de outro *DataNode*. O cliente recebe a listagem dos *DataNodes* ordenados pela menor distância entre a localização física dos dados e o cliente. A leitura será realizada nesta sequência para ser mais eficiente.

Uma boa prática em grandes *clusters* é a distribuição em vários *racks*. Nós de um mesmo *rack* compartilham os mesmos equipamentos de rede e acabam se tornando pontos de falhas. O local de armazenamento das réplicas é crítico para o *HDFS.* O *HDFS* possui uma política que tenta minimizar o custo de gravação,

maximizar a confiabilidade dos dados, disponibilizar e agregar banda de leitura. Quando um novo bloco é criado, o *HDFS* armazena a primeira réplica onde o processo cliente de gravação está localizado, a segunda e a terceira réplica é armazenada em um *rack* diferente da primeira. Este recurso é configurado pelo administrador do *HDFS* em um script que será executado no *NameNode* quando um *DataNode* se unir ao *cluster*, identificando a sua localização física. Caso este *script* não seja configurado, o *NameNode* assume que todos os nós do *cluster* estão no mesmo nível e, por consequência, no mesmo *rack* (SHVACHKO et al., 2010; WHITE, 2012).

Uma das funções do *NameNode* é controlar o número de réplicas necessárias para cada bloco de arquivo. Desta forma, quando chegam os relatórios de armazenamento enviados pelos *DataNodes,* é possível identificar os blocos que estão com mais réplicas que o necessário ou se está faltando alguma réplica para algum bloco. Ajustes são realizados pelo *NameNode* para adicionar réplicas que estiverem faltando ou remover réplicas desnecessárias, porém sempre com critérios para manter a melhor condição de armazenamento no *cluster*. O administrador do *HDFS* também possui uma ferramenta para balancear o espaço em disco alocado nos *DataNodes,* caso algum desbalanceamento ocorra, por exemplo, a adição de um novo nó ao *cluster*. Os *DataNodes* também possuem uma ferramenta denominada *Block Scanner*, que periodicamente executa leituras em seus blocos para confrontar com os seus respectivos arquivos de checagem e caso seja detectado algum bloco corrompido, é comunicado ao *NameNode* para providenciar a criação de uma nova réplica (SHVACHKO et al., 2010; WHITE, 2012).

O processo de retirada, assim como a inclusão de um novo *DataNode* do *cluster*, é realizado através de uma lista de inclusão ou exclusão, contendo os endereços dos nós. Ao incluir o endereço de um *DataNode* na lista de exclusão, ele é marcado para ser excluído e os seus dados começam a ser replicados para outros *DataNodes*. Após todos os dados serem migrados, é seguro retirar o *DataNode* sem causar problemas com a disponibilidade dos dados (SHVACHKO et al., 2010).

3.3.5 *Yarn*

Segundo (VAVILAPALLI et al., 2013), a concepção inicial do *Hadoop* previa a utilização para processamento de grandes volumes de dados, porém com a adoção

em massa de diversas companhias e para uso em fins diversos, isto acabou causando certos questionamentos à especificação inicial e demonstrou duas situações claras e chaves para o *Hadoop*:

1. Um modelo de programação específico para a concepção inicial que forçava os desenvolvedores a utilizar de uma forma não ideal as rotinas de *Map* e *Reduce* e

2. Uma arquitetura centralizada de gerenciamento e controle de fluxo de tarefas que não previa em sua concepção a possibilidade de escalonamento.

O *Yarn* ou *Hadoop* 2.0 foi proposto como a próxima geração da plataforma computacional do *Hadoop*. A nova arquitetura proposta no *Yarn* separa o modelo de programação da infraestrutura de gerenciamento de recursos e delega muitas funções de escalonamento de tarefas para componentes de cada aplicação (VAVILAPALLI et al., 2013).

Esta separação de componentes propostas no *Yarn* promove uma grande flexibilidade de escolha da plataforma de programação, pois o *MapReduce* é apenas um dos modelos de programação que é permitido pelo Yarn (VAVILAPALLI et al., 2013). Além do *MapReduce,* o *Yarn* permite também os modelos de programação: Dryad (ISARD et al., 2007), REEF (CHUN et al., 2013), Spark (ZAHARIA et al., 2010a), Storm (APACHE, 2014a), Tez (APACHE, 2014b), Giraph (APACHE, 2014c).

As plataformas de programação executadas no *Yarn* conseguem coordenar comunicações entre as aplicações, controle de fluxo além de otimizações dinâmicas em tempo de execução, que possibilitam gigantescos aumentos de desempenho nas execuções das tarefas (VAVILAPALLI et al., 2013). Na Figura 3.6, podem-se visualizar as principais diferenças entre o *Hadoop* e o *Yarn.*

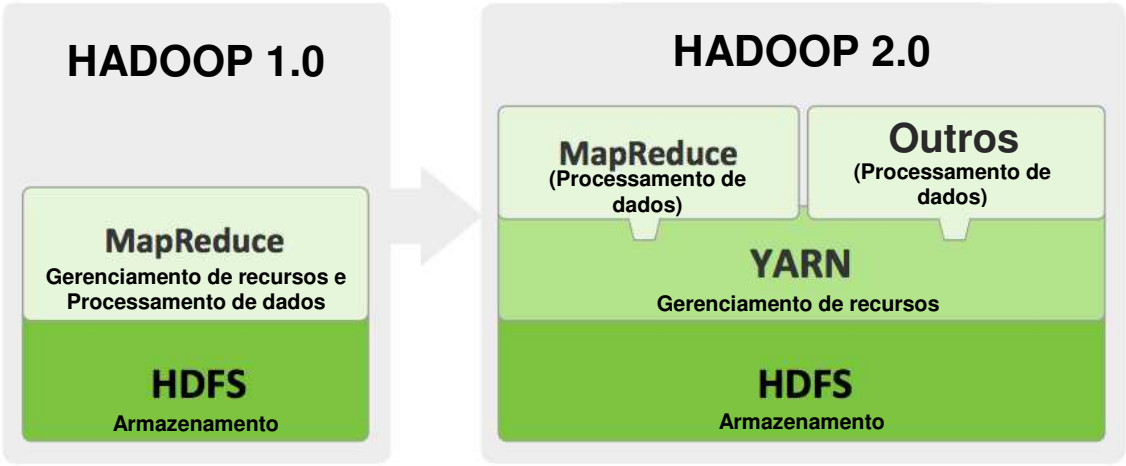


Figura 3.6 - Interações do cliente, *NameNode* e *DataNode*. Reproduzida de (VAVILAPALLI et al., 2013).

3.4 *SpatialHadoop*

O conteúdo desta seção está fortemente embasado na escrita dos seguintes trabalhos (ELDAWY, 2014; ELDAWY et al., 2015b; ELDAWY; MOKBEL, 2013, 2015a, 2015b) do *SpatialHadoop.*

Incorporado ao Hadoop, o *SpatialHadoop* implementa funcionalidades espaciais no interior do núcleo do *Hadoop,* tornando-se o primeiro *framework MapReduce* com suporte nativo para dados espaciais. Além disso, oferece índices espaciais, componentes padrões e *MapReduce* que permitem aos pesquisadores e desenvolvedores implementar novas operações espaciais mais eficientes. Tudo isso está em contraste com *Hadoop-GIS* e outros sistemas que não oferecem este tipo de suporte e, portanto, são limitados para a criação de novas funções.

O núcleo de *SpatialHadoop* foi concebido para servir como uma espinha dorsal para aplicações que lidam com o processamento de dados em grande escala. Existem muitas aplicações que utilizam *SpatialHadoop* como um componente central para lidar com grandes volumes de dados espaciais, incluindo *Shahed* (ELDAWY et al., 2015a, 2015b; INTERFACE, 2015), um sistema para consulta e visualização de dados via satélite espaço-temporal; *TAREEG* (ALARABI et al., 2014), um extrator baseado na *Web* para dados *OpenStreetMap*; MNTG (MOKBEL et al., 2013, 2014), um serviço *web* para gerar objetos em movimento de um conjunto de dados; e *GISQF* (AL-NAAMI; SEKER; KHAN, 2016; NAAMI; SEKER; KHAN, 2014), um analisador de dados para eventos mundiais.

3.4.1 Arquitetura

A Figura 3.7 apresenta uma visão geral da arquitetura do *SpatialHadoop*. Existem três tipos de usuários que interagem com *SpatialHadoop*. Usuários casuais usam o sistema através dos aplicativos disponíveis. Os desenvolvedores são capazes de criar novas operações via *MapReduce*. Os **a**dministradores do sistema

têm profundo conhecimento de problemas e podem ajustar o sistema através dos arquivos de configuração.

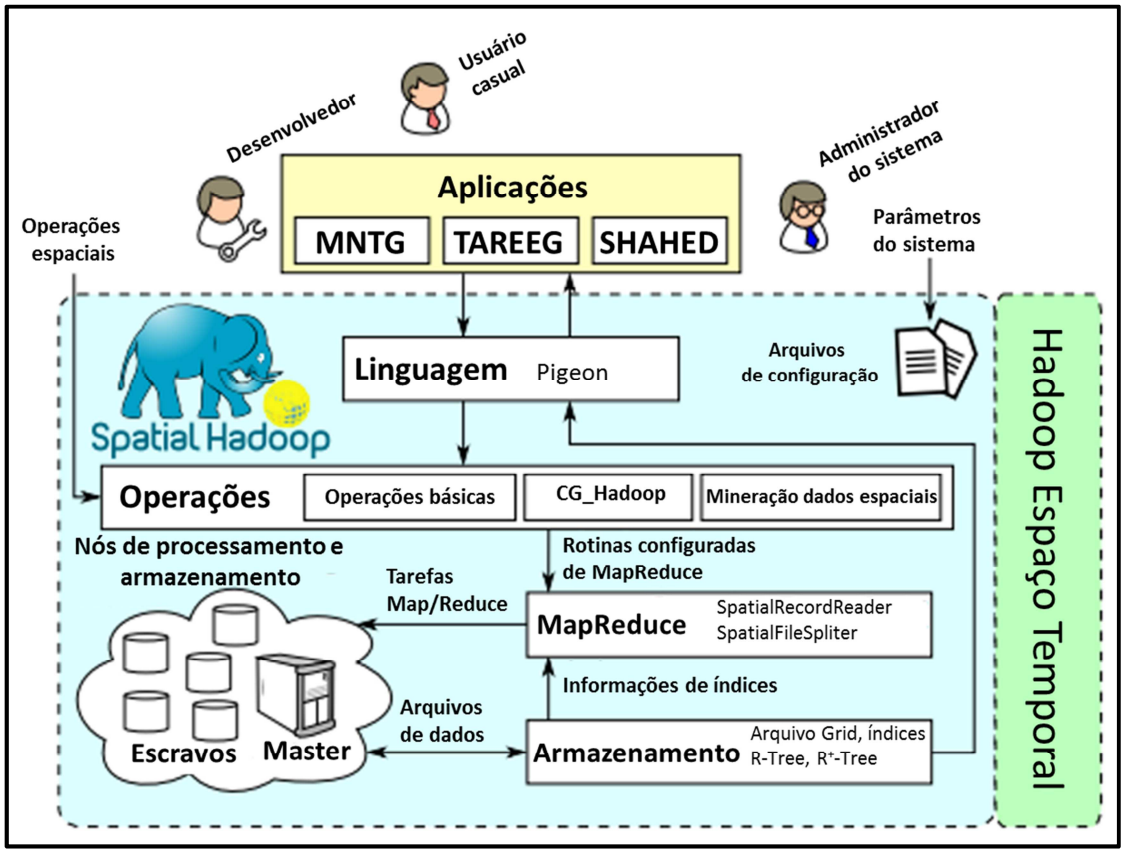


Figura 3.7 - Visão geral do *SpatialHadoop*. Reproduzida de (ELDAWY; MOKBEL, 2013).

O núcleo de *SpatialHadoop* consiste de quatro camadas, ilustradas na Figura 3.8.

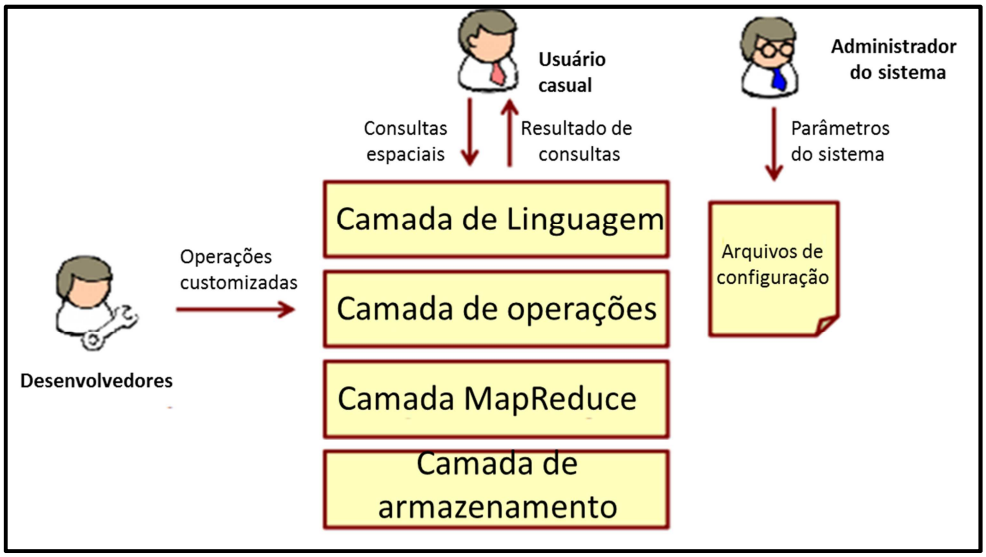


Figura 3.8 - Arquitetura do *SpatialHadoop*. Reproduzida de (ELDAWY; MOKBEL, 2013).

1. A camada de linguagem contém *Pigeon*, uma linguagem de alto nível, com tipos de dados espaciais compatíveis com o padrão *OGC* e funções.
2. A camada de armazenamento contém três índices espaciais, *Grid Files*, *R-Tree* e *R^+-Tree*, todos implementados dentro do *Hadoop Distributed File System* (*HDFS*), permitindo operações espaciais muito mais rápidas em comparação a arquivos *heap* em *Hadoop*. Os índices são organizados em duas camadas: um índice global, com os dados das partições dos nós, e vários índices locais para organizar os registros dentro de cada nó.
3. A camada de *MapReduce* conta com dois componentes, nomeados *SpatialFileSplitter* e *SpatialRecordReader*, que permitem às operações espaciais acessarem os índices construídos.
4. A camada de operações encapsula as operações espaciais oferecidas pelo *SpatialHadoop*. Operações básicas contêm três operações espaciais padrões, Consulta por faixa (*Range Query)*, K vizinhos mais próximos (*KNN)* e Junção Espacial (*Spatial Join)*. *CG_Hadoop* (ELDAWY et al., 2013) é um conjunto de operações fundamentais de geometria computacional.

3.4.2 Camada de linguagem

Como o paradigma *MapReduce* exige grandes esforços de codificação (CHAIKEN et al., 2008; OLSTON et al., 2008; ZHOU et al., 2012), um conjunto de linguagens SQL declarativas têm sido propostas, por exemplo, *HiveQL* (THUSOO et al., 2009), *Pig Latin* (OLSTON et al., 2008), e *Scope* (CHAIKEN et al., 2008; ZHOU et al., 2012). O *SpatialHadoop* não fornece uma linguagem completamente nova. Em vez disso, ele fornece *Pigeon* (ELDAWY; MOKBEL, 2014), uma extensão para a linguagem *Pig Latin* (OLSTON et al., 2008) e adiciona tipos de dados espaciais, funções e operações que estão em conformidade com o padrão *Open Geospatial Consortium* (OGC) (BORGES, 2011; OGC, 2014).

Segundo (ELDAWY; MOKBEL, 2013), o *Pigeon* é uma linguagem simples de alto nível, para facilitar a interação de usuários não técnicos com o sistema,

fornecendo suporte para tipos de dados espaciais e funções primitivas, conforme ilustrado na Figura 3.9.

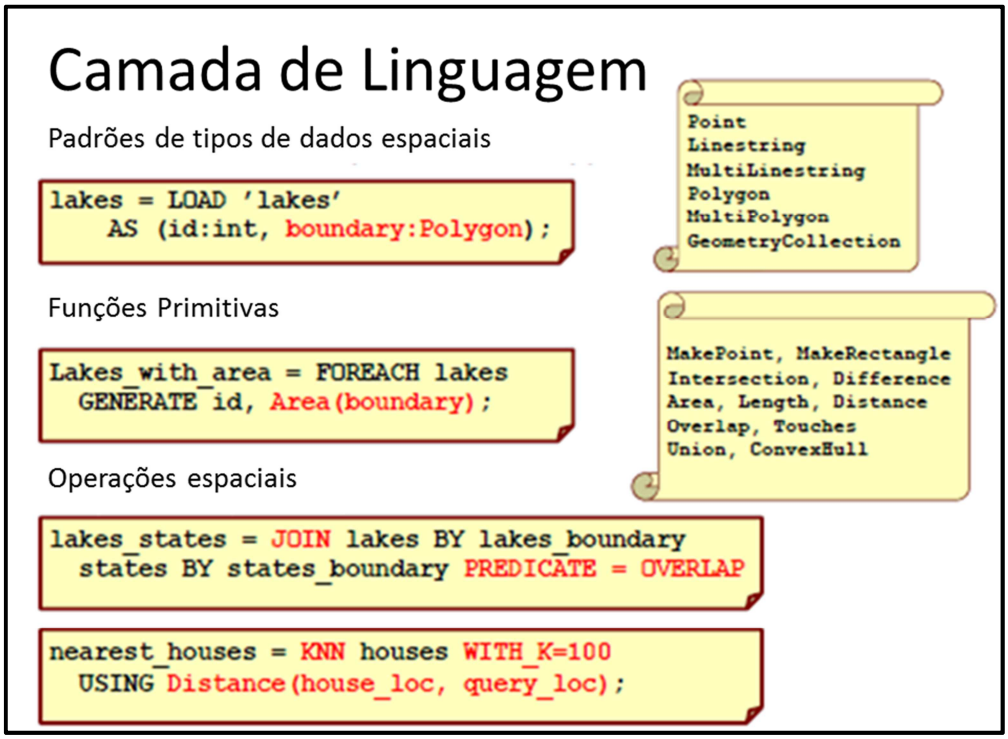


Figura 3.9 - Camada de Linguagem. Reproduzida de (ELDAWY; MOKBEL, 2013).

3.4.3 Camada de armazenamento

Na camada de armazenamento, o *SpatialHadoop* adiciona índices espaciais adaptados ao paradigma *MapReduce*. Esses índices superam uma limitação do *Hadoop*, que provê suporte apenas para arquivos não indexados do tipo *heap*. Existem dois desafios que impedem índices espaciais tradicionais de serem usados com o *Hadoop:*

1. Primeiro, os índices tradicionais são projetados para o paradigma de programação procedural, enquanto *Hadoop* utiliza programação *MapReduce*;
2. Segundo, os índices tradicionais são projetados para sistemas de arquivos locais, enquanto *Hadoop* usa o *Hadoop Distributed File System* (*HDFS*), que é limitado em arquivos que podem ser escritos apenas uma vez e depois não podem ser alterados.

Para superar estes desafios, *SpatialHadoop* organiza o seu índice em dois níveis.

1. O índice global particiona os dados através de nós no *cluster*;
2. O índice local organiza os dados em cada nó escravo.

O índice global é mantido na memória principal do nó principal (NameNode), enquanto cada índice local é armazenado como bloco de uma imagem num nó escravo. Um índice é construído em *SpatialHadoop* através de um trabalho de *MapReduce* que é executado em três fases:

1. **Particionamento** - na etapa de particionamento, os dados são distribuídos entre os nós de tal forma que cada partição se encaixe dentro de um bloco de 64 MB no *HDFS*. Para dados distribuídos uniformemente, uma grade uniforme é usada, enquanto para dados dispersos uma compartimentação *R-Tree* é utilizada. Um registro (por exemplo, polígono) pode ser replicado caso, ele sobreponha várias partições. Esses registros duplicados são tratados mais tarde no processamento de consultas para evitar a produção de resultados duplicados;
2. **Indexação local** - na etapa de indexação local, cada partição é indexada localmente como uma *R-Tree* e armazenada em um arquivo separado. Na fase de indexação local e de acordo com o tipo de índice que está sendo construído, um índice local é criado para cada partição separada e levado para um arquivo com um bloco *HDFS,* que é anotado pela *MBR* da partição. Uma vez que cada partição tem um tamanho fixo (64 MB), os índices locais são construídos em memória antes de serem gravados no disco;
3. **Indexação global** – a terceira e última fase é a indexação global, em que os arquivos contendo índices locais são concatenados em um arquivo e um índice global é construído para indexar todas as partições usando seus *MBRs* como chaves e armazenados na memória principal do nó mestre. Em caso de falha do sistema, o índice global é reconstruído a partir do bloco *MBR* apenas quando necessário.

A Figura 3.10 ilustra as três etapas de criação de um índice espacial no *SpatialHadoop*.

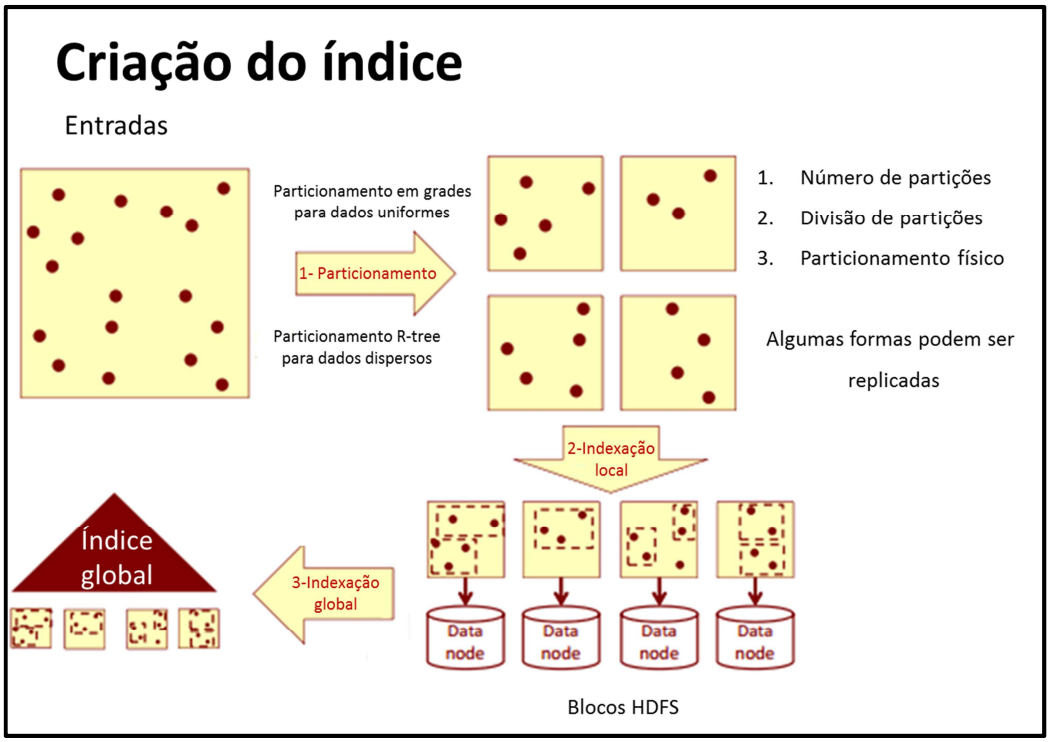


Figura 3.10 - Etapas da criação dos índices espaciais. Reproduzida de (ELDAWY; MOKBEL, 2013).

O *SpatialHadoop* implementa três classes de índices, ou seja, *Grid File*, *R-Tree* e *R*$^+$*-Tree*. *Grid File* é usado para dados distribuídos uniformemente, enquanto *R-Tree* e *R*$^+$*-Tree* são utilizados para dados com distribuição não uniformes. Na *R-Tree*, os registros não são replicados, o que faz com que as partições se sobreponham. Isso torna mais eficiente *Range Query,* onde partições que estão completamente contidas na faixa de consulta podem ser copiadas para a saída e nenhum passo de "desduplicação" é necessário. *R*$^+$*-Tree* garante que as partições sejam disjuntas, mas alguns registros precisam ser replicados.

3.4.4 Camada *MapReduce*

Segundo (ELDAWY, 2014; ELDAWY; MOKBEL, 2013), a camada de *MapReduce* no *Hadoop* tradicional é projetada para trabalhar com arquivos *heap* não indexados como entrada. No entanto, as operações espaciais em *SpatialHadoop* recebem arquivos indexados espacialmente como entrada, o que requer um tratamento diferente. Além disso, algumas operações espaciais, por exemplo, junção espacial, são operações binárias que tomam dois arquivos como entrada. Para ser capaz de lidar com arquivos indexados espacialmente, o

SpatialHadoop introduz dois novos componentes na camada de *MapReduce*, ou seja, *SpatialFileSplitter* e *SpatialRecordReader,* que exploram os índices globais e locais, respectivamente, para o acesso aos dados com mais eficiência, conforme ilustrado na Figura 3.11.

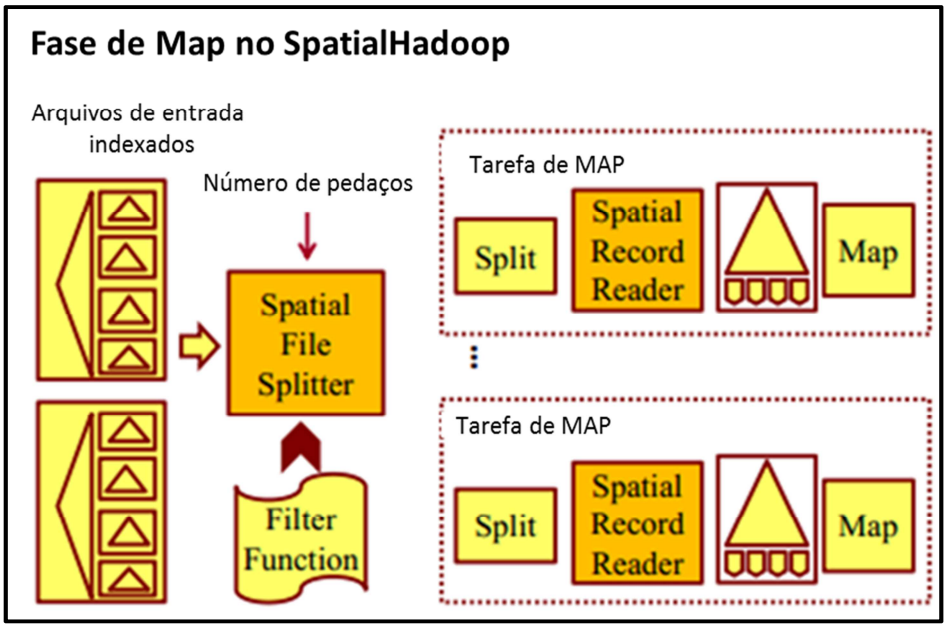


Figura 3.11 - Camada de *MapReduce*. Reproduzida de (ELDAWY; MOKBEL, 2013).

O *SpatialFileSplitter* recebe a entrada com um ou dois arquivos indexados espacialmente, além de uma função de filtro fornecida pelo usuário, conforme ilustrado na Figura 3.12.

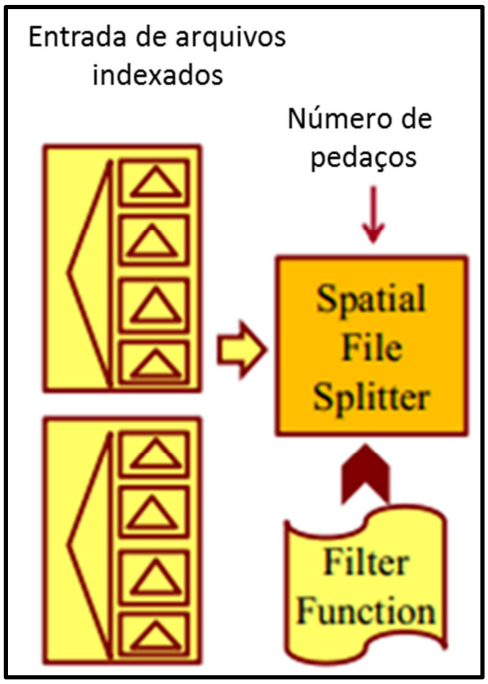


Figura 3.12 - Execução do *SpatialFileSplitter*. Reproduzida de (ELDAWY; MOKBEL, 2013).

Em seguida, ele usa o índice global para separar blocos de arquivos que não contribuam para a resposta da consulta (por exemplo, faixa de consulta externa), com base nos retângulos de limite mínimo que lhes foram atribuídos quando o índice foi criado. No caso de operações binárias, onde há dois arquivos de entrada, o *SpatialFileSplitter* usa dois índices globais para selecionar pares de blocos de arquivos que precisam ser processados juntos (por exemplo, blocos sobrepostos na Junção Espacial).

O *SpatialRecordReader* utiliza o índice local, permitindo que os registros em um bloco possam ser localizados através do índice local em vez de iteração sobre todos os registros, um por um, conforme ilustrado na Figura 3.13.

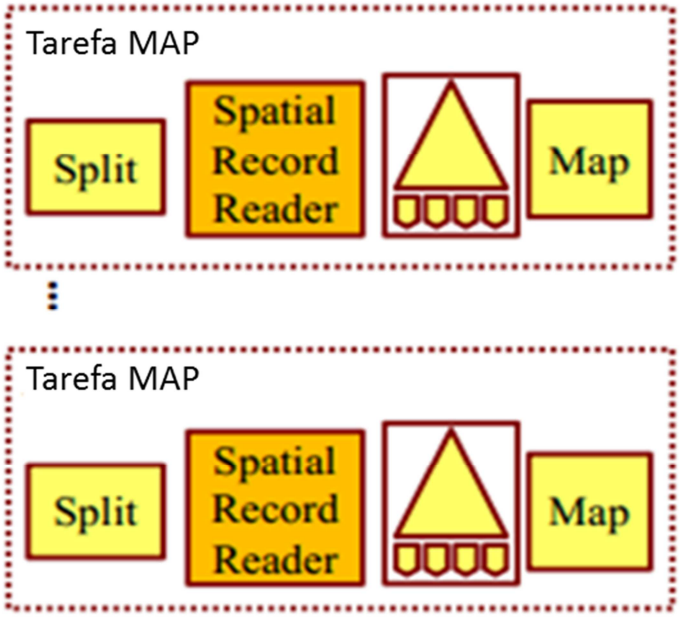


Figura 3.13 - Execução do *SpatialRecordReader*. Reproduzida de (ELDAWY; MOKBEL, 2013).

O índice local é lido a partir do bloco e lhe é atribuído um ponteiro para a função de mapeamento que utilizará este índice para selecionar os registros processados sem a necessidade de iterar sobre todos os registros.

Juntos, *SpatialFileSplitter* e *SpatialRecordReader* ajudam os desenvolvedores a escrever muitas operações espaciais com programas *MapReduce.*

3.4.5 Camada de operações

Segundo (ELDAWY, 2014; ELDAWY; MOKBEL, 2013), a camada de operações encapsula a implementação de diversas operações espaciais que utilizam os índices espaciais e os novos componentes na camada de *MapReduce*. O *SpatialHadoop* é equipado com implementação para três operações espaciais básicas, Consulta por faixa (*Range Query)*, K vizinhos mais próximos (*KNN)* e Junção Espacial (*Spatial Join)*, conforme ilustrado na Figura 3.14.

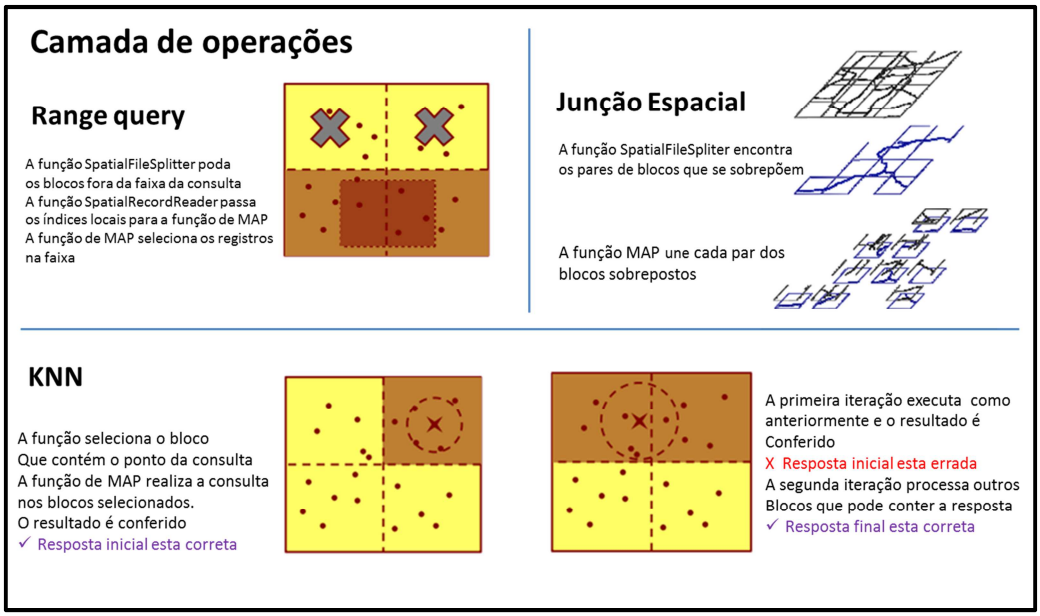


Figura 3.14 - Camada de Operações. Reproduzida de (ELDAWY; MOKBEL, 2013).

3.4.5.1 Consulta por faixa

Uma consulta por faixa tem um conjunto de registros *R* com dados espaciais e uma janela de consulta *A* como entrada, e retorna os registros que se intersectam com *A,* conforme ilustrado na Figura 3.15. Em *SpatialHadoop*, o *SpatialFileSplitter* lê o índice global e usa uma função de filtro para selecionar as partições que se intersectem a janela de consulta. As partições que estão completamente fora da janela de consulta são podadas, pois não contribuem para o conjunto resposta. Cada partição selecionada que se intersecta a área de consulta é processada em uma tarefa *map*, onde o *SpatialRecordReader* extrai seu índice local e o processa com uma consulta por faixa para retornar registros correspondentes à janela de consulta. Finalmente, como alguns registros são duplicados durante a indexação, para evitar resultados duplicados um novo passo é executado para filtrá-los.

Figura 3.15 - Execução de uma *Range Query*. Reproduzida de (ELDAWY; MOKBEL, 2013).

3.4.5.2 K vizinhos mais próximos

A operação de K vizinhos mais próximos é realizada em duas iterações. Na primeira iteração, o *SpatialFileSplitter* utiliza o índice global para selecionar a partição com o ponto de consulta. O índice local da partição do ponto de consulta, executado na primeira interação, é extraído pela *SpatialRecordReader* e usado para encontrar o *KNN* desta partição, através de uma função *map*. O resultado obtido após as duas iterações é conferido. Para testar se a resposta está correta ou não, um círculo de teste é desenhado com o ponto de consulta no centro. Se o círculo de teste se encaixa completamente na partição processada, a resposta está correta. Em caso contrário, uma segunda iteração é realizada para processar estas partições sobrepostas, conforme ilustrado na Figura 3.16.

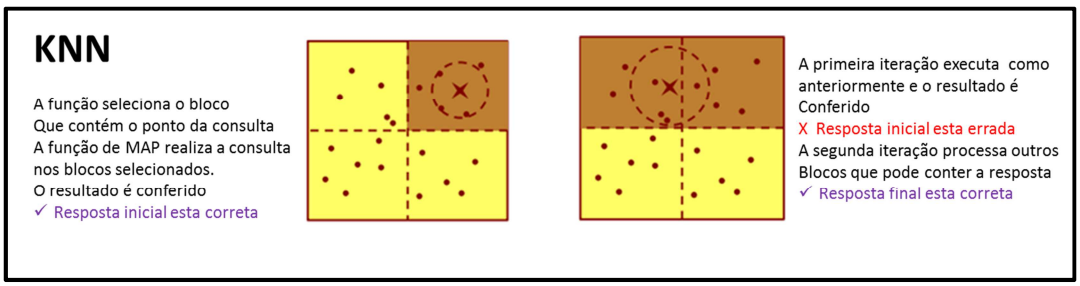


Figura 3.16 - Execução *KNN*. Reproduzida de (ELDAWY; MOKBEL, 2013).

3.4.5.3 Junção espacial

Para implementar a operação de junção espacial no *SpatialHadoop*, usa-se o *SpatialFileSplitter* com os índices globais de ambos os arquivos para que o mesmo

possa selecionar os pares de partições que se intersectam. Cada par de partições que se intersectem é processado pelo *SpatialRecordReader*, que utiliza os índices locais em ambos os arquivos para retornar todos os pares sobrepostos. Finalmente, para evitar registros duplicados, utiliza-se a técnica de ponto de referência para eliminar a duplicação na resposta causada pela replicação do índice, conforme ilustrado na Figura 3.17.

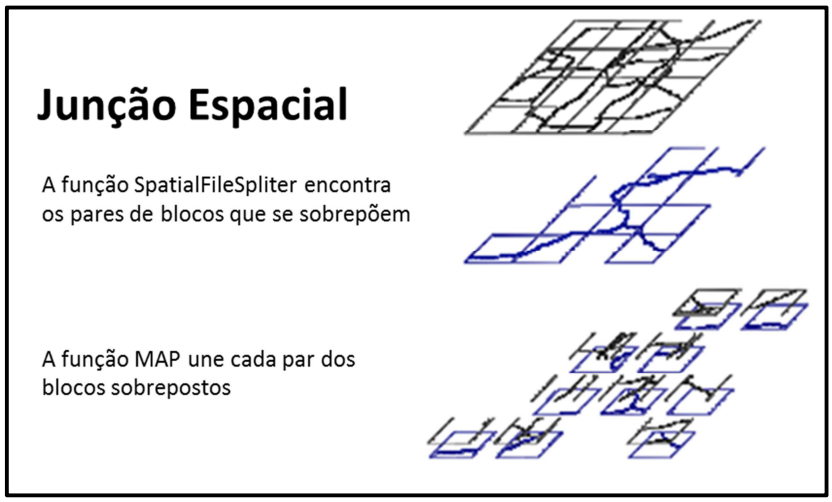


Figura 3.17 - Execução junção espacial. Reproduzida de (ELDAWY; MOKBEL, 2013).

3.4.6 Algoritmos de junção espacial do *SpatialHadoop*

Diversos algoritmos foram propostos para processar junção espacial, sendo que a grande maioria é direcionada principalmente ao passo de filtragem. A maioria dos recentes algoritmos foi baseada em índices espaciais, usados em ambos os conjuntos de dados de entrada. O *SpatialHadoop* implementa dois desses algoritmos o *Distributed Join* (*DJ*) e a Junção Espacial *with MapReduce* (*SJMR*) (ZHANG et al., 2009).

3.4.6.1 Distributed Join (DJ)

Segundo (ELDAWY; MOKBEL, 2013), o algoritmo de *Distributed Join* (*DJ*) realiza uma Junção Espacial entre dois arquivos. Se esses arquivos são indexados, a *DJ* une cada par de intersecções de partições, que é muito escalável e eficiente para arquivos grandes. Se um dos arquivos não é indexado, o algoritmo se reduz a

um simples bloco de *loop* aninhado que funciona em um ambiente distribuído em *SpatialHadoop*. Se ambos os arquivos não são indexados, o algoritmo pode escolher automaticamente o reparticionamento deles para combinar as partições usadas pelo outro arquivo que, em alguns casos, pode aumentar o desempenho da operação, reduzindo o número total de tarefas do *map*.

3.4.6.2 Spatial Join with MapReduce (SJMR)

Segundo (ELDAWY; MOKBEL, 2013), a operação *Spatial Join with MapReduce (*SJMR) (ZHANG et al., 2009) é a implementação *MapReduce* da partição com base na junção espacial. Esta operação é projetada para processar Junção Espacial eficiente para arquivos não indexados. Esse algoritmo particiona ambos os arquivos de acordo com uma grade uniforme e junta cada par de correspondências. As dimensões da grade de particionamento são automaticamente determinadas com base em tamanhos de arquivo de entrada de tal forma que o tamanho médio de partição é igual ao tamanho do bloco *HDFS* (por exemplo, 64 MB). Se um ou os dois arquivos de entrada são não uniforme, o desempenho deste algoritmo pode degradar-se.

3.4.7 *CG_Hadoop*

Segundo (ELDAWY et al., 2013), o *CG_Hadoop* é um conjunto de operações de geometria computacional para *MapReduce*, conforme ilustrado na Figura 3.18, que provê suporte para cinco operações fundamentais de geometria computacional, ou seja, a união poligonal, *skyline*, *convex hul*, par mais distante e par mais próximo, todas implementadas no paradigma de programação *MapReduce*.

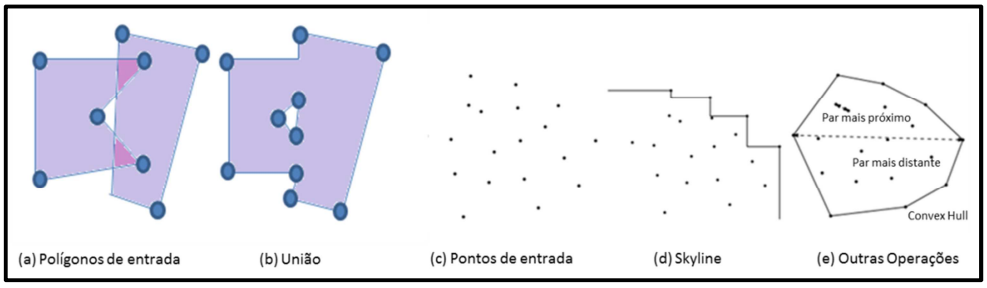


Figura 3.18 - Operações de geometria computacional do *CG_Hadoop*. Reproduzida de (ELDAWY et al., 2013).

Embora existam algoritmos de geometria computacional bem estabelecidos para esses problemas (DE BERG et al., 2008; PREPARATA; SHAMOS, 1985), esses não se ajustam ao lidar com conjuntos de dados espaciais que podem conter bilhões de pontos. Por exemplo, calcular um *convex hull* de um conjunto de dados de quatro bilhões de pontos, utilizando um algoritmo tradicional pode levar até três horas, enquanto o cálculo da *polygon union* de um conjunto de cinco milhões de pontos leva cerca de uma hora e falha com uma exceção de memória para dados de conjuntos maiores. O *CG_Hadoop* atinge um desempenho melhor do que os algoritmos de geometria computacional tradicionais ao lidar com dados espaciais de grande escala (ELDAWY et al., 2013).

A principal ideia por trás de todos os algoritmos em *CG_Hadoop* é aproveitar a natureza de dividir e conquistar de muitos algoritmos de geometria computacional. A propriedade de dividir e conquistar aplica-se ao ambiente *MapReduce*, onde a maior parte do trabalho pode ser paralelizado para trabalhar em vários nós em um *cluster*. No entanto, o *CG_Hadoop* tem de adaptar algoritmos computacionais tradicionais para trabalhar melhor no ambiente *MapReduce*. Por exemplo, ao contrário de algoritmos tradicionais que normalmente dividem a entrada ao meio e fazem várias rodadas, o *CG_Hadoop* divide a entrada em pedaços menores para garantir que a resposta seja computada em uma rodada *MapReduce*, que é preferível tanto para *Hadoop* quanto para *SpatialHadoop*. Além disso, usa índices espaciais distribuídos, sempre que possível, para acelerar os cálculos por pedaços de entrada iniciais que não contribuem para a resposta da operação de geometria computacional (ELDAWY et al., 2013).

No geral, o *CG_Hadoop* forma um núcleo de uma biblioteca *MapReduce* abrangente de operações de geometria computacional que atinge de 29 vezes a 260 vezes um melhor desempenho do que os algoritmos tradicionais quando são utilizados sistemas *Hadoop* e *SpatialHadoop*, respectivamente (ELDAWY et al., 2013).

3.4.8 Adicionando arquivos no *SpatialHadoop*

Um arquivo é adicionado no *SpatialHadoop* através de um trabalho *MapReduce*. Por exemplo, para adicionar um arquivo de 300 MB composto por

dados sintéticos, o processo ocorre em três fases: 1 – particionamento, 2 – indexação local, 3 – indexação global.

1. **Particionamento:** Esta fase irá dividir o arquivo em **n** partições, executando três passos:
   I. **Número de partições:** primeiramente é calculado o número de partições necessárias para armazenar o arquivo através da equação:

$$n = \left\lceil \frac{S(1+\alpha)}{B} \right\rceil$$

   Onde **S** é o tamanho do arquivo de entrada, **B** o tamanho do bloco do *HDFS,* por padrão 64 MB, e **α** é uma relação de sobrecarga, por padrão é 20%. Para armazenar um arquivo de 300 MB no *HDFS* com tamanho do bloco de 64 MB serão necessárias 6 partições. A Figura 3.19 demostra a fórmula e o cálculo da quantidade de partições.

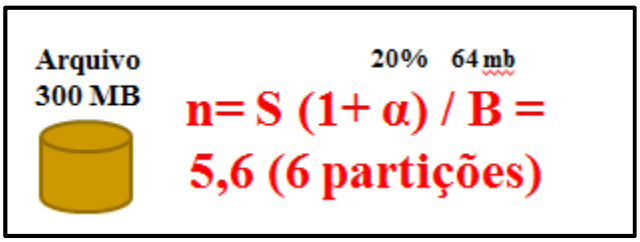


Figura 3.19 - Cálculo das partições.

   II. **Limites das partições:** O próximo passo será definir os limites das 6 partições criadas na etapa anterior. O resultado final será um conjunto de 6 retângulos que representam os limites das 6 partições (*MBRs*). Cada partição possui as coordenadas que irão delimitar os dados, conforme mostra Figura 3.20.

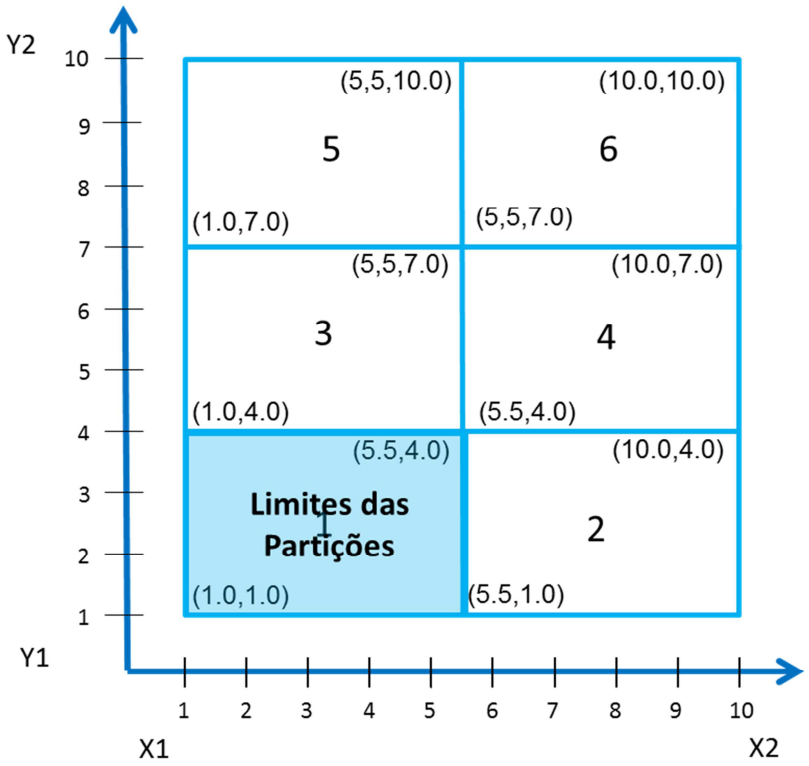


Figura 3.20 - Limites das partições.

A **part_00001_data_0001** é formada por todos os dados em que as coordenadas **x1=1.0, x2=5.5, y1=1.0 e y2=4.0** e todos os registros dentro desta partição possuem essas coordenadas como limite. As coordenadas de limites das partições irão formar o índice global desta partição, conforme pode-se visualizar na Figura 3.21.

X1	Y1	X2	Y2	Partição
1.0	1.0	5.5	4.0	part-00001_data_00001
5.5	1.0	10.0	4.0	part-00002_data_00002
1.0	4.0	5.5	7.0	part-00003_data_00003
5.5	4.0	10.0	7.0	part-00004_data_00004
1.0	7.0	5.5	10.0	part-00005_data_00005
5.5	7.0	10.0	10.0	part-00000_data_00006

Figura 3.21 - Índice global das partições.

III. **Particionamento físico:** O último passo irá particionar fisicamente o arquivo em blocos de dados, por padrão 64 MB. Cada registro do arquivo com suas respectivas coordenadas espaciais será lido e atribuído a uma das 6 partições, conforme Figura 3.22

Figura 3.22 - Particionamento físico.

2. **Indexação Local:** a segunda fase será executada somente se for informado pelo usuário que é necessária a criação de um índice para o arquivo adicionado. Desta forma, além dos dados de cada partição física, é construída também uma estrutura de índice. Nesta tese usa-se a *R-Tree* para a indexação local, conforme mostra Figura 3.23.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	3	64 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
_logs	dir				2015-09-22 15:55	rwxr-xr-x	root	supergroup
_master.rtree	file	1.59 KB	3	64 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00000_data_00009	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00000_data_00009_1	file	2.23 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00001_data_00001	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00001_data_00001_1	file	2.53 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00002_data_00002	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00002_data_00002_1	file	2.33 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00003_data_00003	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00003_data_00003_1	file	2.21 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00004_data_00004	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00004_data_00004_1	file	2.34 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00005_data_00005	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00005_data_00005_1	file	2.56 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00006_data_00006	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00006_data_00006_1	file	1.64 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00007_data_00007	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00007_data_00007_1	file	1.7 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00008_data_00008	file	32 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup
part-00008_data_00008_1	file	1.68 MB	3	32 MB	2015-09-22 15:55	rw-r--r--	root	supergroup

Figura 3.23 - Indexação local.

3. **Indexação Global:** Esta última fase tem por objetivo construir o índice global que irá indexar as seis partições criadas. O nó mestre constrói um índice global em memória que indexa todos os blocos de arquivos usando seus limites das partições como chave de índice, conforme mostra Figura 3.24.

Name	Type	Size	Replication	Block Size
_SUCCESS	file	0 KB	3	64 MB
_logs	dir			
_master.grid	file	0.33 KB	3	64 MB
part-00000_data_00006	file	49.29 MB	3	64 MB
part-00001_data_00001	file	50.65 MB	3	64 MB
part-00002_data_00002	file	50.13 MB	3	64 MB
part-00003_data_00003	file	50.29 MB	3	64 MB
part-00004_data_00004	file	49.57 MB	3	64 MB
part-00005_data_00005	file	50.07 MB	3	64 MB


```
0,5.5,7.0,10.0,10.0,part-00000_data_00006
0,1.0,1.0,5.5,4.0,part-00001_data_00001
0,5.5,1.0,10.0,4.0,part-00002_data_00002
0,1.0,4.0,5.5,7.0,part-00003_data_00003
0,5.5,4.0,10.0,7.0,part-00004_data_00004
0,1.0,7.0,5.5,10.0,part-00005_data_00005
```

Figura 3.24 - Indexação global.

Assim que o arquivo é adicionado no *HDFS* é executada a política padrão de alocação dos dados como pode-se visualizar no mapa de alocação da Figura 3.25.

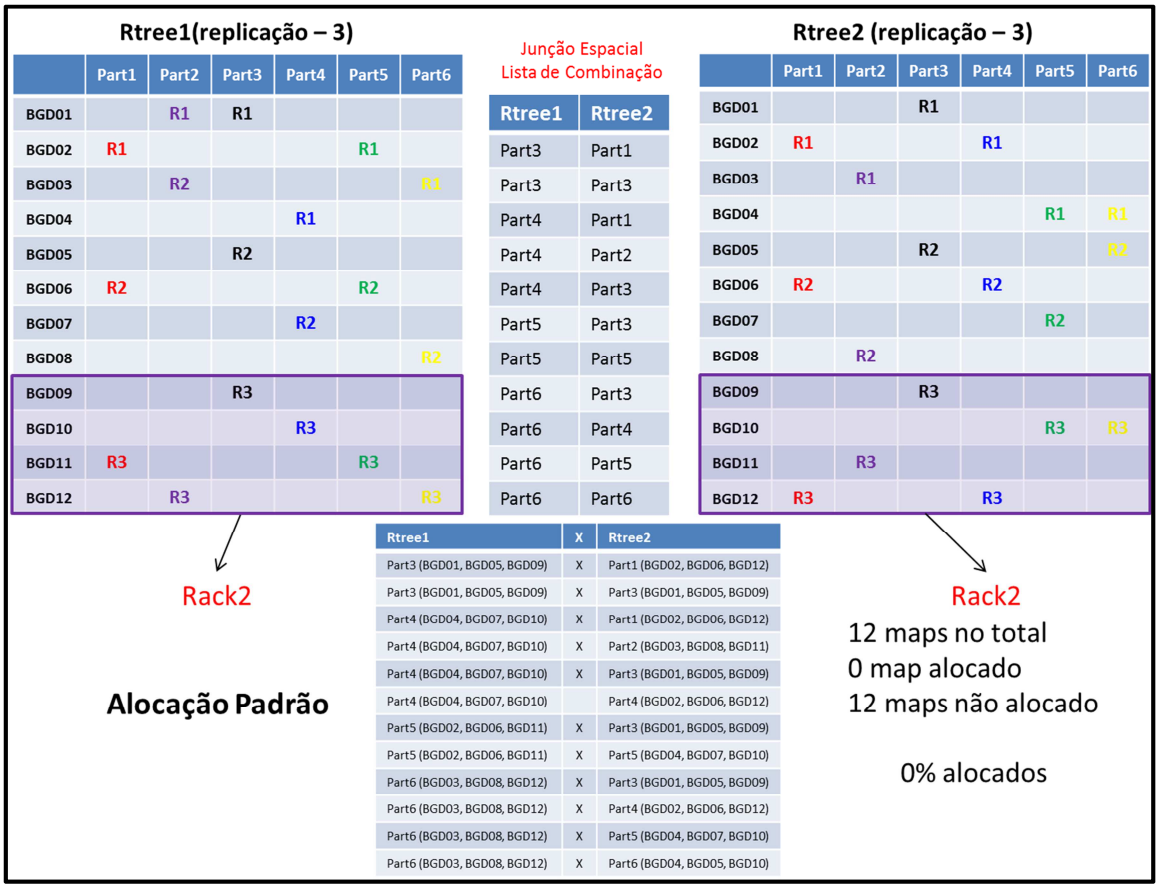


Figura 3.25 – Mapa de alocação padrão do *HDFS*.

O foco da política padrão é o balanceamento de carga no *cluster*. O *Hadoop* foi concebido para trabalhar com apenas um conjunto de dados de entrada, o qual tem como padrão 3 réplicas para cada bloco de dados que compõe o arquivo.

3.4.9 Execução da junção espacial no *SpatialHadoop*

O processamento da junção espacial em *SpatialHadoop,* utilizando o algoritmo de junção espacial *Spatial Join with MapReduce (*SJMR) (ZHANG et al., 2009) é realizado em quatro etapas: 1 – Pré-processamento, 2 – Junção Global, 3 – Junção Local e 4 – Eliminação dos duplicados.

Na etapa de pré-processamento, o algoritmo recebe como entrada dois conjuntos de dados que serão processados pela junção espacial e também o

relacionamento topológico de interseção como predicado espacial, conforme mostra Figura 3.26. Nesta etapa, é executado o algoritmo de reparticionamento dos conjuntos de dados de entrada para reduzir o número de interseções entre as partições. Ao final desta etapa têm-se os dois conjuntos de dados pré-processados.

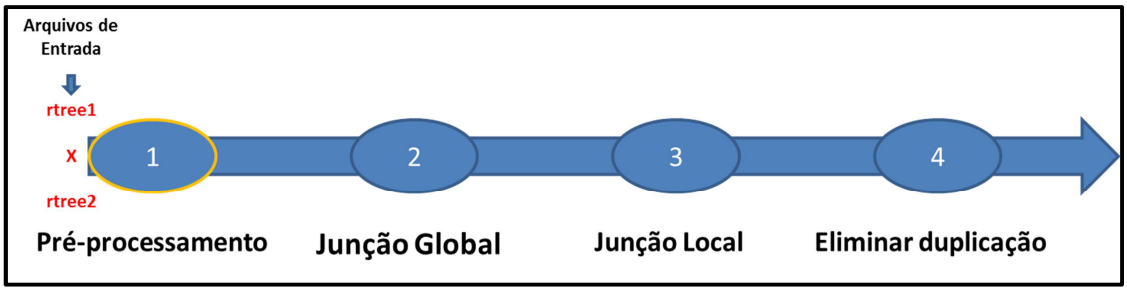


Figura 3.26 - Etapas de processamento da junção espacial.

A próxima etapa é a junção global que recebe como entrada os arquivos da etapa anterior de pré-processamento e tem por objetivo selecionar todos os pares de blocos cujos *MBRs* se intersectam, conforme mostra Figura 3.27.

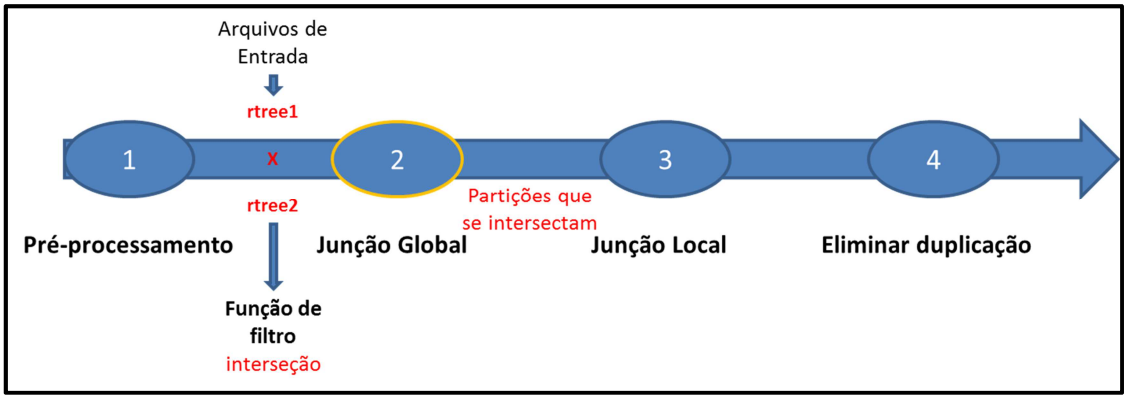


Figura 3.27 - Etapas de processamento da junção espacial.

Para o processamento da próxima etapa é utilizado o índice global de cada um dos conjuntos de dados que foi processado na etapa anterior e sobre eles é executado o algoritmo de junção espacial tradicional com o predicado de interseção, gerando como resultado uma lista de combinação de todas as partições que se intersectam, conforme Figura 3.28.

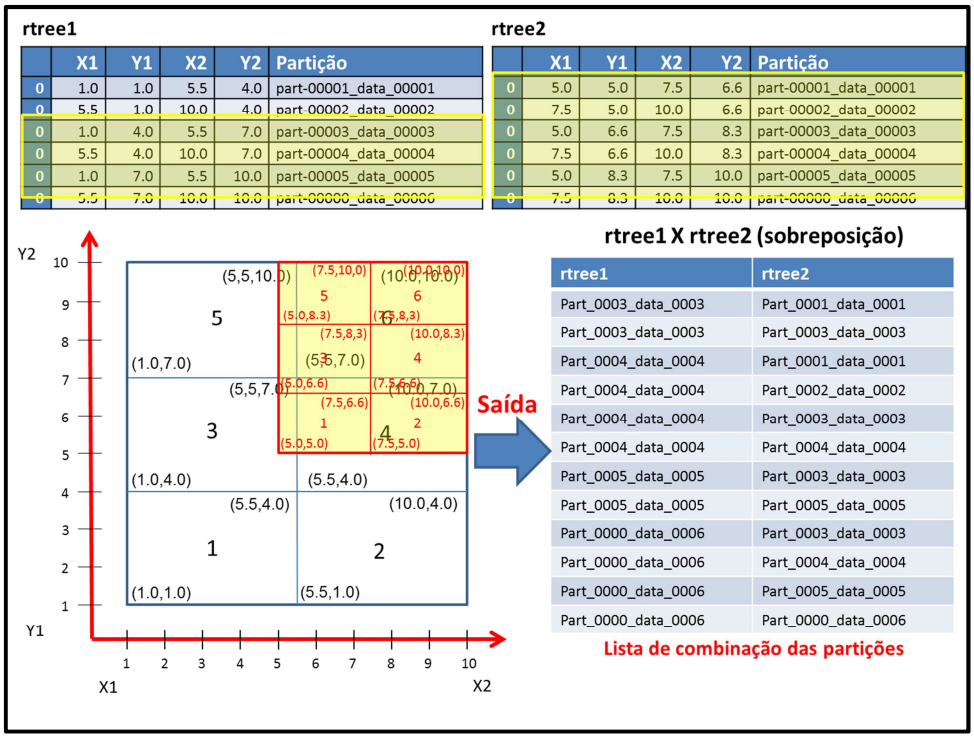


Figura 3.28 - Junção espacial tradicional sobre o índice global dos dois conjuntos.

Analisando o resultado da lista de combinações das partições da Figura 3.28, é possível concluir que, após a execução do algoritmo de junção espacial, a partição **Part_0003_data_0003** do arquivo **rtree1** intersecta a **partição Part_0001_data_0001** do arquivo **rtree2**. A mesma conclusão pode ser obtida para o restante da lista de combinações.

A terceira etapa, que é a junção local, recebe como entrada para o processamento, a lista de combinações de partições que se intersectaram, uma com as outras, proveniente do processamento da etapa anterior e tem por objetivo juntar os registros dos dois blocos para produzir os pares de registros que se intersectam. Neste caso, deverá ser executado o algoritmo de junção espacial sobre os dados das duas partições, conforme mostra Figura 3.29.

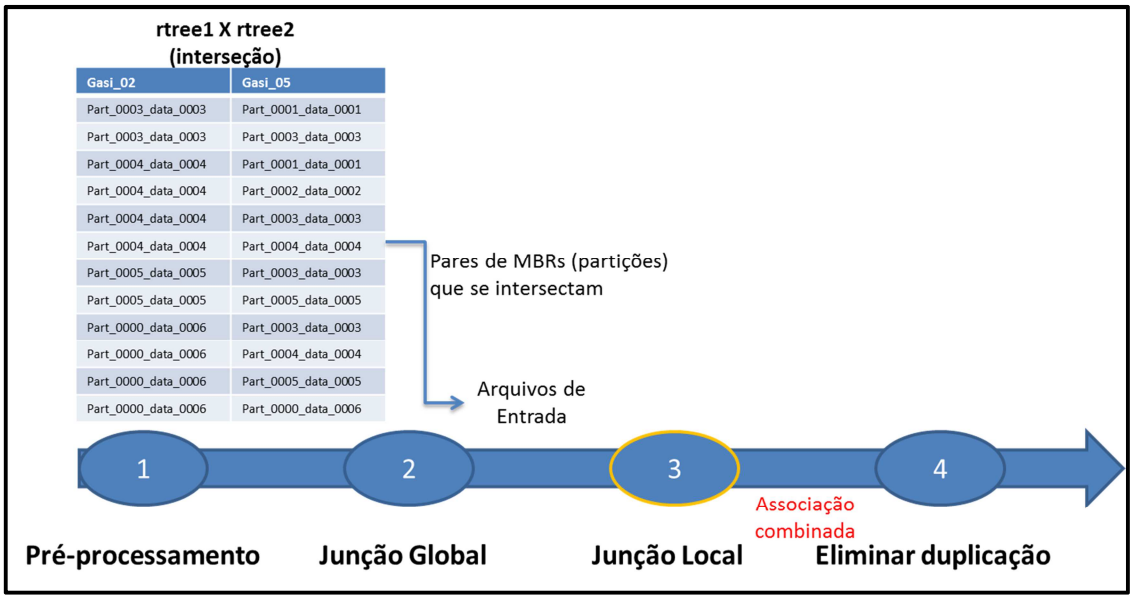


Figura 3.29 - Etapas de processamento da junção espacial.

Neste ponto, antes de executar o processamento, o escalonador de tarefas do *Hadoop* distribui as tarefas entre os nós do *cluster*. Cada linha da lista de combinações corresponde a uma função *MapReduce* que deverá ser escalonada para processar no *cluster*. Para realizar o escalonamento, uma solicitação de execução de tarefas é enviada para o nó principal definir em qual nó do *cluster* a tarefa será executada, baseado na política de escalonamento padrão do *Hadoop*. A política recebe um conjunto de tarefas que devem ser escalonadas e retorna o mapeamento das tarefas e dos respectivos nós que deverão executá-las.

Após a execução da política de escalonamento, é processado o algoritmo de junção espacial. Nesta etapa, acredita-se que o escalonador, quando cabível, atribui o processamento para o mesmo nó dos dados. Então é utilizado o índice local de cada conjunto de dados e é executado o algoritmo de junção espacial chamado *Spatial Join with MapReduce (*SJMR) (ZHANG et al., 2009), com o predicado topológico de interseção, gerando como resultado à combinação de todos os registros que atendam ao predicado espacial, conforme mostra Figura 3.30.

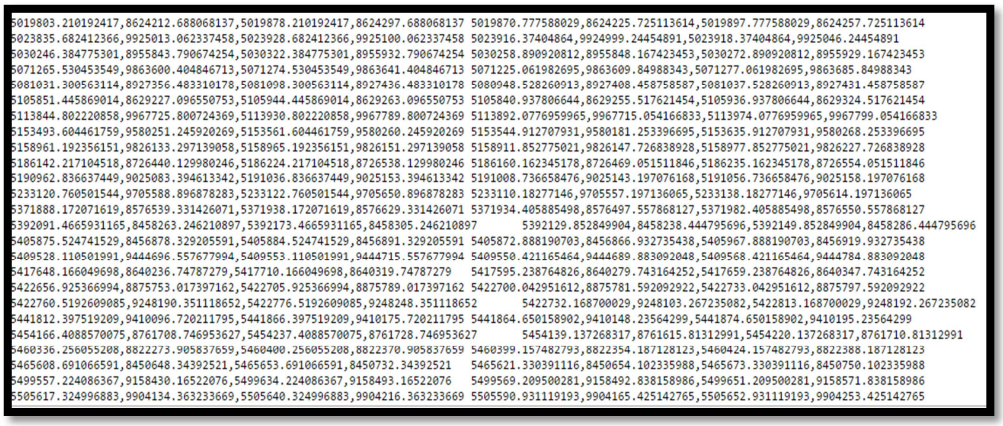


Figura 3.30 - Associação combinada.

A última etapa de processamento da junção espacial é a etapa de eliminação dos registros duplicados, que recebe como entrada a combinação de todos os registros provenientes do processamento da etapa anterior e tem por objetivo eliminar os registros duplicados e gerar o conjunto resposta que atenda ao predicado topológico de interseção, conforme mostra Figura 3.31.

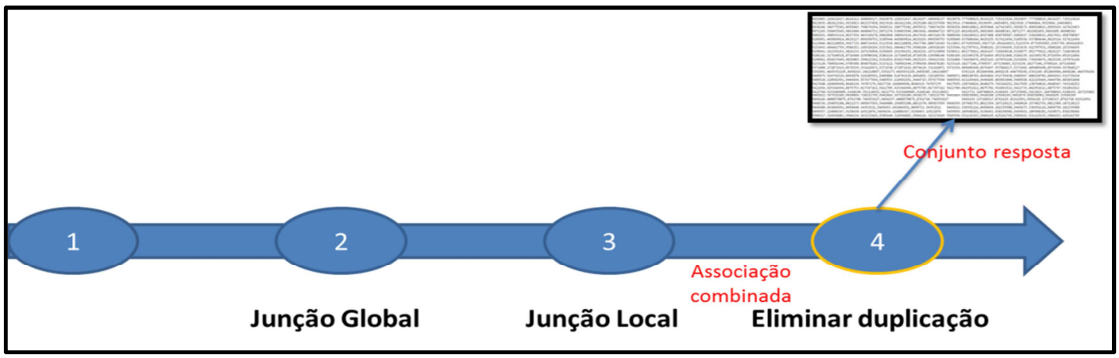


Figura 3.31 - Etapas de processamento da junção espacial.

3.5 Considerações finais

Nesta seção, foram discutidos os principais conceitos sobre *Big Data*, *MapReduce*, *Hadoop* e *SpatialHadoop* que serão abordados nesta tese de doutorado. Com a explosão no volume de dados nos últimos anos surge um novo conceito conhecido como *Big Data*. As soluções propostas para o processamento destes enormes volumes de dados é o *MapReduce.* É neste cenário que surge o

Hadoop, podendo ser considerado uma das maiores invenções para o gerenciamento de volumes massivos de dados não estruturados para o processamento em *clusters* de computadores. Em 2013, foi proposto o *SpatialHadoop, é* uma extensão do *Hadoop* com suporte nativo para dados espaciais, tornando-se o primeiro *framework MapReduce* com essas características. O processamento das soluções propostas nesta tese utilizam o *SpatialHadoop*.

No próximo capítulo, serão apresentados os trabalhos relacionados, encontrados na literatura que tratam do tema abordado nesta tese de doutorado.

Capítulo 4

TRABALHOS RELACIONADOS

Este capítulo descreve os principais trabalhos existentes na literatura que estejam relacionados ao assunto de pesquisa principal investigado nesta tese de doutorado, sendo organizado da seguinte forma: considerações iniciais, alocação de dados, escalonamento de tarefas, CoS-HDFS e considerações finais.

4.1 Considerações iniciais

Atualmente, os dados espaciais gerados por diversos dispositivos estão distribuídos em vários nós em um *cluster* de computadores para facilitar o seu processamento, devido ao volume ser muito grande. O processamento desses dados é executado por um dos *frameworks* mais populares, chamado *Hadoop*, principalmente devido a suas características mais importantes que incluem escalabilidade, tolerância a falhas, facilidade de programação em ambiente distribuído e flexibilidade. No entanto, apesar de seus méritos, o *Hadoop* tem limitações de desempenho evidentes em tarefas diversas e isto deu origem a um conjunto significativo de pesquisas, que visam melhorar a sua eficiência, mantendo as suas propriedades desejáveis (DOULKERIDIS; NØRVAG, 2014). Para alcançar um melhor desempenho durante o processamento de consultas espaciais é essencial ter acesso eficiente aos dados espaciais (HASHEM et al., 2016). É importante que o processamento ocorra próximo aos dados e as soluções atuais buscam localidade apenas no nível da rede local, não de nó (ZHAO et al., 2012). Rede de banda larga é um recurso relativamente escasso no ambiente típico de *Hadoop.* Desta forma a sua utilização precisa ser feita de modo eficaz. Essa

situação gera a necessidade de levar as aplicações para perto dos dados e, caso não seja possível, o preço de mover os dados pelo sistema pode ser muito alto e prejudicar bastante o desempenho final da aplicação.

Apesar desses problemas serem reconhecidos, ainda existe pouco entendimento sobre a interferência da localidade destes dados no desempenho do *Hadoop*. Diversos trabalhos (ABOUZEID et al., 2009; ANANTHANARAYANAN et al., 2011a; BARROSO; CLIDARAS; HÖLZLE, 2013; CASTILLO; SPREITZER; STEINDER, 2011; DITTRICH et al., 2010, 2012; ELDAWY, 2014; ELDAWY; ALARABI; MOKBEL, 2015; ELDAWY; MOKBEL, 2015b, 2013, 2015a; ELTABAKH et al., 2011; FAHMY; ELGHANDOUR; NAGI, 2016; GUO; FOX; ZHOU, 2012; HASHEM et al., 2016; HOTT; ROCHA; GUEDES, 2016; IBRAHIM et al., 2012; ISARD et al., 2007; JIANG; TUNG; CHEN, 2011; JIONG XIE et al., 2010; LI et al., 2016; NISHANTH S et al., 2013; OUSTERHOUT et al., 2010; PALANISAMY et al., 2011; PAVLO et al., 2009; RICHTER et al., 2012; YAO et al., 2014; ZAHARIA et al., 2010a, 2010b; ZHAO et al., 2012; ZHOU et al., 2016) avaliam o ganho de se alocar o processamento exatamente no mesmo nó que contém os dados, proporcionando acesso rápido, evitando congestionamento da rede. Os trabalhos desenvolvidos nesta linha procuram: (i) alocar os dados conforme alguma característica dos dados, denominada de alocação de dados nesta tese; ou (ii) buscam escalonar as tarefas para os nós onde esses dados estão, denominada escalonamento de tarefas.

Na sequência, destacam-se os principais trabalhos dentro de cada uma destas áreas: alocação de dados e escalonamento de tarefas em *clusters* de computadores com uso de *Hadoop*.

4.2 Alocação de dados

Neste tópico, relacionam-se alguns trabalhos encontrados na literatura que abordam a alocação de dados no *Hadoop* como forma de melhorar o desempenho das operações espaciais.

Conceitualmente pode-se definir que a distância entre o nó de dados que detém a entrada para uma função *Map* e o nó de tarefa onde será executado o processamento da função *Map* é chamada de localidade. O custo de transferência

de dados depende da distância entre o nó de entrada de dados e o nó de computação. Se o nó de entrada de dados é muito próximo, então o custo de transferência de dados torna-se baixo, afetando positivamente o desempenho da operação. Muitas vezes o *cluster Hadoop* não é capaz de alcançar a localidade, prejudicando a eficiência do processamento das operações. O ideal é evitar o custo de transferência de dados, alocando os dados no mesmo nó onde irá ocorrer o processamento (SENTHILKUMAR; ILANGO, 2016).

O trabalho de (GOLDMAN et al., 2012) propõe um modelo para facilitar a compreensão dos custos de obter os dados em um *cluster*. O custo é considerado D(0) quando os dois conjuntos estão localizados no mesmo nó do processamento, melhor situação para o processamento das operações espaciais. O custo dobra D(2) quando o processamento ocorre no mesmo nó onde se encontra um dos conjuntos de dados e precisa obter o segundo conjunto que está localizado em outro nó do *cluster* no mesmo *rack* do primeiro conjunto. A situação mais critica é quando o custo atinge D(4), ou seja, o segundo conjunto está localizado em um *rack* diferente do primeiro conjunto onde esta ocorrendo o processamento. A Figura 4.1 apresenta o modelo de custo para obter dados em um *cluster*.

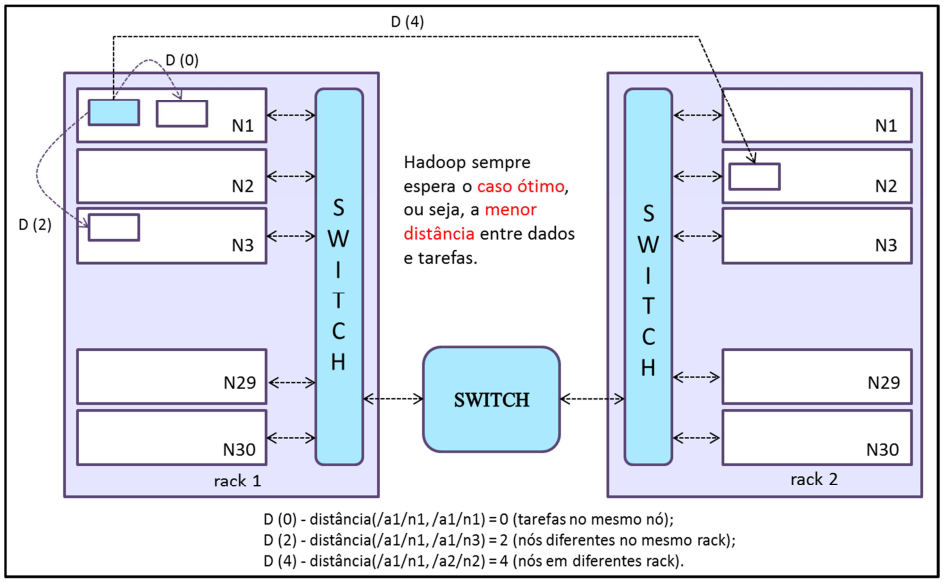


Figura 4.1 - Custo para obter dados em um *cluster*. Reproduzida de (GOLDMAN et al., 2012).

Pesquisas em bancos de dados paralelos (GRAY et al., 2009), bem como trabalhos sobre *Hadoop* (ABOUZEID et al., 2009; DITTRICH et al., 2010) relatam

que a organização dos dados permite o processamento eficiente de algoritmos de consultas. Transferir essa ideia para *Hadoop* não é uma tarefa simples. Apesar do particionamento de dados ser fácil de conseguir em *Hadoop* (JIANG et al., 2010), a localização não é fácil de obter (ABOUZEID et al., 2009; DITTRICH et al., 2010). Isto ocorre porque o *Hadoop* não fornece qualquer recurso para as aplicações controlarem onde os dados serão armazenados (ELTABAKH et al., 2011). Em vários sistemas, a localidade dos dados é um dos principais indicadores de desempenho utilizados para avaliar a eficiência (DEAN; GHEMAWAT, 2008; ISARD et al., 2007; KUMAR et al., 2010). Encontram-se na literatura diversos trabalhos abordando o assunto, porém não existe um consenso sobre a questão de localidade dos dados e sua influência no desempenho das operações espaciais (HOTT; ROCHA; GUEDES, 2016). Entender sobre a melhor forma de alocar os dados é fundamental para a execução eficiente de operações espaciais.

Em *Hadoop,* os dados são armazenados no *HDFS* que não foi projetado para tratar o armazenamento conjunto dos dados. O *HDFS* foi projetado para replicar arquivos, proporcionando uma rápida recuperação em caso de falhas. A política utilizada de alocação de dados do *HDFS* é muito simples, um nó do *cluster* é selecionado de forma aleatória para armazenar os blocos e réplicas de um determinado arquivo. A política padrão de *HDFS* aloca a primeira cópia de um bloco recém-criado no nó local, no qual o bloco é criado, desde que haja espaço suficiente. O *HDFS*, então, tenta selecionar um nó em um *rack* diferente da primeira cópia para armazenar a segunda e a terceira cópia do bloco, conforme ilustra Figura 4.2.

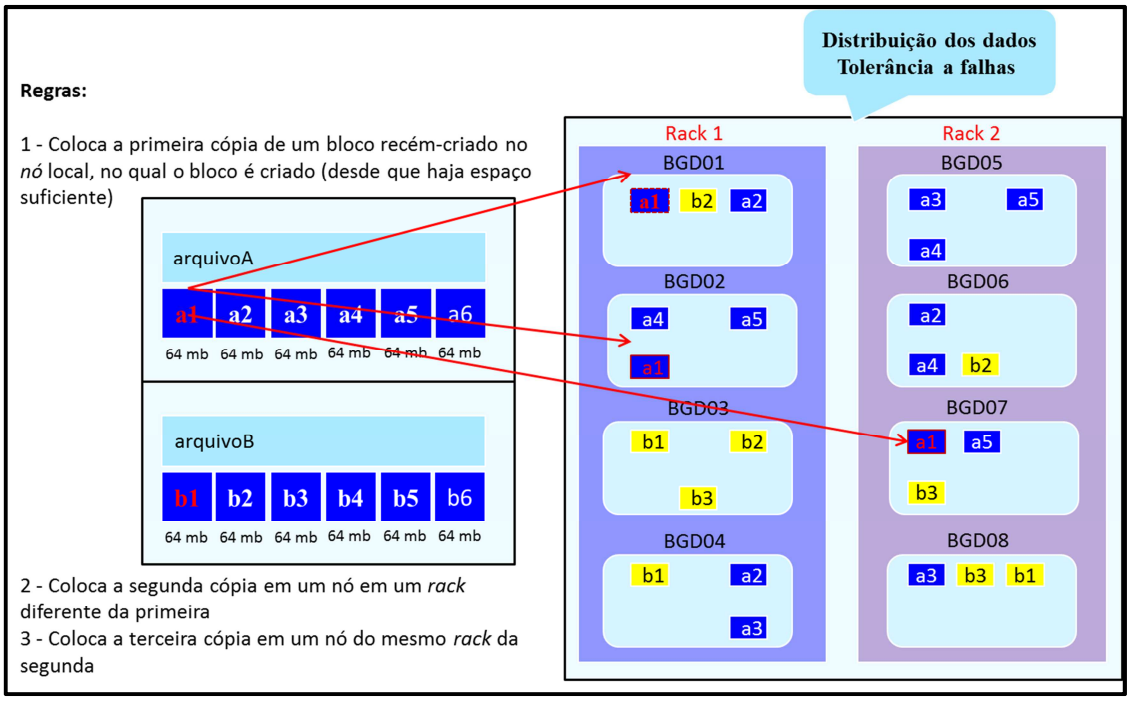


Figura 4.2 – Política padrão de alocação de dados do *HDFS*.

O objetivo desta política de alocação de dados do *HDFS* é alcançar o balanceamento de carga, distribuindo os dados uniformemente entre os nós de um *cluster*, independentemente da utilização prevista para os dados. Esta política de alocação de dados funciona bem para a maioria das aplicações *Hadoop* que acessam apenas um único arquivo. Porém, as aplicações que processam dados de diferentes arquivos podem obter um aumento significativo no desempenho com as estratégias personalizadas (ELTABAKH et al., 2011). Isso ocorre pelo fato da política padrão de alocação de blocos em um *cluster* do *HDFS* não levar em consideração qualquer tipo de relacionamento que possa existir entre os dados. Desconsiderar essa questão pode aumentar o tráfego de dados na rede e prejudicar o desempenho da operação. Evidências recentes mostram que a organização cuidadosa dos dados pode proporcionar aumento no desempenho de aplicações *Hadoop* e no processamento dos algoritmos de consulta. Desta forma, torna-se vital criar uma política de alocação, onde as características entre os dados possam ser levadas em consideração (JIONG XIE et al., 2010; WEI et al., 2010; ZHOU; XU, 2011).

No trabalho de (ELTABAKH et al., 2011) foi identificado que a política de alocação dos dados do *HDFS* tenta balancear a carga, alocando os blocos de forma aleatória; não levando em consideração quaisquer características dos dados. Em particular, o *HDFS* não fornece qualquer meio para alocar os dados relacionados no

mesmo conjunto de nós. Para superar esse problema é proposto *CoHadoop*, uma extensão do *Hadoop* com um mecanismo que permite que os aplicativos controlem onde os dados serão alocados nos nós de um *cluster*. O *CoHadoop* tenta alocar esses arquivos para melhorar a eficiência de tal forma que, as propriedades de tolerância a falhas do *Hadoop* sejam mantidas. Ele atribui uma chave para cada partição e aloca as partições que têm a mesma chave em um mesmo conjunto de nó, removendo as limitações apresentadas por *Hadoop++* (DITTRICH et al., 2010). No entanto, a seleção inicial de nós para uma chave recém-atribuída a uma partição é realizada de acordo com a política de posicionamento padrão de blocos do *Hadoop*. Ele não leva em consideração a possível carga que será criada para aqueles conjuntos de nós de dados devido à localização. Além disso, *CoHadoop* não considera a heterogeneidade na capacidade de armazenamento de cada nó de dados. Estes problemas não são abordados no *CoHadoop*.

Em (NISHANTH S et al., 2013), o sistema proposto supera os problemas do *CoHadoop* (ELTABAKH et al., 2011), assegurando simultaneamente a alocação das partições relacionadas e o balanceamento de carga através do uso de um algoritmo. No algoritmo proposto quando são selecionados nós de dados para uma nova chave de agrupamento, os nós que são potenciais candidatos de carga excessiva são excluídos da política de seleção. Isso garante que o nó de dados seja atribuído a várias chaves de agrupamento, quando nós de dados não atribuídos estão disponíveis. O *CoHadoop++* garante que a propriedade de tolerância a falhas do *Hadoop* não seja comprometida ao excluir os nós da política de seleção de nó.

Para (GUO; FOX; ZHOU, 2012), a localidade de dados é uma vantagem significativa de sistemas de dados paralelos em relação aos sistemas tradicionais. A boa localidade de dados reduz o tráfego de rede, um dos gargalos na computação intensiva de dados. Nesse trabalho, é feita uma investigação profunda sobre os efeitos da localidade dos dados em tempo de processamento. Para isso, foi construído um modelo matemático de programação em *MapReduce* e analisado o impacto da localidade dos fatores de configuração, como os números de nós e tarefas. Um novo escalonador de tarefas foi proposto para agendar várias tarefas simultaneamente. Diversos experimentos foram produzidos para quantificar a melhoria do desempenho dos algoritmos propostos.

A distribuição dos dados é uma questão-chave em sistemas de armazenamento distribuídos em larga escala, para colocar *petabytes* de dados ou

até mesmo além, entre dezenas ou centenas de milhares de dispositivos de armazenamento. Algoritmos de distribuição de dados atuais podem alcançar mapeamento eficiente, escalável e equilibrado, mas não distinguem diferentes características dos dispositivos heterogêneos. O algoritmo *Suora*, sistema de armazenamento heterogêneo, aborda os desafios de equilíbrio de desempenho, capacidade e custo para a alocação de dados, tomando vantagens de características do dispositivo heterogêneo. Ao combinar o desempenho com a capacidade e eficiência econômica de diferentes dispositivos, o algoritmo *Suora* constrói uma solução de armazenamento unificado e adaptável para um ambiente heterogêneo (ZHOU et al., 2016).

Para (JIONG XIE et al., 2010), a implementação *Hadoop* assume que os nós de computação em um *cluster* são homogêneos e a localidade dos dados não são levadas em consideração, durante o escalonamento de tarefas. Para manter o balanceamento de carga, o *Hadoop* distribui os dados para os vários nós com base na disponibilidade de espaço em disco. Tal estratégia de alocação de dados é muito prática e eficiente para um ambiente onde os nós são homogêneos, idênticos em termos de computação e capacidade de disco. A localidade dos dados é um fator determinante para um melhor desempenho *MapReduce*, desta forma, ignorar as questões relativas à localidade de dados em ambientes heterogêneos pode visivelmente reduzir o desempenho *MapReduce*. Este trabalho aborda o problema de como alocar dados entre os nós de forma que cada nó tenha uma carga de processamento de dados equilibrada. O novo mecanismo distribui fragmentos de um arquivo de entrada para nós heterogêneos com base na sua capacidade de computação. Os resultados experimentais mostram que a estratégia de posicionamento de dados pode melhorar o desempenho *MapReduce*, através de um equilíbrio dos dados entre os nós antes de executar uma aplicação de uso intensivo de dados em um *cluster Hadoop* heterogêneo.

O *Scarlett* é um sistema que replica os blocos de dados em um *cluster* com base na popularidade deles. Ao prever com precisão a popularidade do arquivo e trabalhar dentro de limites rígidos sobre o armazenamento adicional, o *Scarlett* faz uma interferência mínima na execução dos trabalhos. Simulações e experimentos conduzidos nos populares *frameworks Hadoop/MapReduce* mostraram que o *Scarlett* pode acelerar os trabalhos em até 20% (ANANTHANARAYANAN et al., 2011b).

Na Tabela 4.1, pode-se visualizar o resumo dos principais trabalhos encontrados na literatura que tratam da alocação de dados. Nenhum destes trabalhos trata da alocação de dados espaciais, ou seja, eles são genéricos e, portanto não consideram as características implícitas dos dados especiais, também não consideram as características da junção espacial que necessita da alocação conjunta de dados relacionados, com exceção ao CoS-HDFS, que será tratado na seção 4.4 desta tese.

Tabela 4.1 - Trabalhos encontrados na literatura sobre alocação de dados.

Autores	Alocação de Dados	Dados Espaciais	Junção Espacial
(ABOUZEID et al., 2009)	Sim	Não	Não
(ANANTHANARAYANAN et al., 2011a)	Sim	Não	Não
(ANANTHANARAYANAN et al., 2011b)	Sim	Não	Não
(DITTRICH et al., 2010)	Sim	Não	Não
(ELTABAKH et al., 2011)	Sim	Não	Não
(GUO; FOX; ZHOU, 2012)	Sim	Não	Não
(HOTT; ROCHA; GUEDES, 2016)	Sim	Não	Não
(JIONG XIE et al., 2010)	Sim	Não	Não
(NISHANTH S et al., 2013)	Sim	Não	Não
(WEI et al., 2010)	Sim	Não	Não
(ZHOU et al., 2016)	Sim	Não	Não
(ZHOU; XU, 2011)	Sim	Não	Não

4.3 Escalonamento de tarefas

Segundo (ZHAO et al., 2012), o escalonamento de tarefas no *Hadoop* é realizado por um nó mestre chamado *JobTracker*, que gerencia um número de escravos chamados *TaskTracker*. Cada *TaskTracker* tem um número fixo de *slots* de *Map* e *Reduce* nos quais ele pode executar tarefas. Normalmente, os administradores definem o número de *slots* a um ou dois por núcleo. O *JobTracker* atribui tarefas em resposta a pulsações enviadas pelos *TaskTracker* a cada poucos segundos, que por sua vez, relatam o número de *slots Map* e *Reduce* no *TaskTracker*. Assim, pode-se definir o escalonamento como: atribuição de tarefas a

um *TaskTracker* com a restrição de que para cada trabalho, todas as tarefas *Map* devem ser atribuídas antes de suas tarefas *Reduce*.

O gerenciamento de *cluster* do *Hadoop* com múltiplas tarefas *MapReduce* em múltiplos nós precisa de políticas eficazes e eficientes de escalonamento de tarefas para alcançar desempenho e a melhor utilização dos recursos disponíveis. O desempenho de um cluster *Hadoop* é afetado por algumas questões como energia gasta para o processamento dos trabalhos, imparcialidade no compartilhamento dos recursos do *cluster* para a execução dos trabalhos, localidade de dados e sincronização entre a saída de uma função *Map* e a entrada para a função *Reduce* (SENTHILKUMAR; ILANGO, 2016).

Por padrão, as aplicações decidem aleatoriamente em qual nó será processada uma determinada tarefa que será monitorada até o seu encerramento (HOTT; ROCHA; GUEDES, 2016). Os estudos revelam que o escalonador de tarefas padrão do *Hadoop* executa de forma ineficiente o escalonamento (IBRAHIM et al., 2012). O compartilhamento de recursos entre *MapReduce* de um *cluster* é desafiado, especialmente quando requer um controle de alocação de recursos que se adapta automaticamente para diferentes aplicações para alcançar suas metas de desempenho (HASHEM et al., 2016).

A maioria dos trabalhos desenvolvidos nos últimos anos com o objetivo de melhorar o desempenho do *framework Hadoop* tiveram como foco o escalonamento de tarefas. Desta forma, são propostos escalonadores de tarefas sensíveis à localidade dos dados que são integrados ao sistema de arquivos *HDFS*. Essa situação torna-se complexa em relação ao tamanho do *cluster*. Quanto maior o *cluster,* maior é a dispersão dos dados, tornando o papel do escalonador uma tarefa ineficiente.

Alguns trabalhos publicados, com foco na criação de novos escalonadores de tarefas, tais como: (BARROSO; CLIDARAS; HÖLZLE, 2013), que foram os primeiros pesquisadores a tratar do assunto sobre a influência da localidade dos dados no desempenho das operações espaciais, destacam como os diferentes recursos (disco, memória e rede), com características distintas em termos de velocidade de acesso, podem afetar diretamente a eficiência da execução de uma operação. Essas características tendem a ser menos eficientes quando envolvem nós em diferentes *racks* de rede. O custo para obter os dados que serão utilizados no processo pesa no resultado final da operação.

No trabalho de (HOTT; ROCHA; GUEDES, 2016) é avaliado quais os ganhos obtidos quando se executa o processamento no mesmo nó que contêm os dados. Os autores propõem um escalonador de tarefas sensível à localidade dos dados, ou seja, leva as aplicações para próximo dos dados, gerando como resultado uma melhora significativa no desempenho das aplicações. De acordo com a localidade, o acesso aos dados pelas aplicações para a execução do processamento pode ser realizado por disco do nó local, memória local, *caching* ou memória de outro nó que esteja no *cluster* e através da rede. Cada uma das formas de acesso tem suas vantagens e desvantagens. Em busca de uma melhor eficiência no processamento das operações espaciais, a escolha do melhor acesso aos dados tem impacto direto no desempenho da operação, porém essa escolha não é uma tarefa simples.

O *COSHH* (RASOOLI; DOWN, 2014) é um escalonador de tarefas para melhorar o desempenho do *Hadoop*, quando executado em um ambiente de cluster heterogêneo, questão negligenciada na maioria dos escalonadores. O principal objetivo do *COSHH* é melhorar o tempo médio de conclusão dos trabalhos. A abordagem principal do sistema de escalonamento *COSHH* é usar informações do sistema para tomar melhores decisões de escalonamento de tarefas, o que leva à melhoria do desempenho. O *COSHH* consiste em dois principais processos, em que cada processo é desencadeado ao receber uma determinada mensagem. Ao receber um novo trabalho, o escalonador executa o enfileiramento do processo para armazenar o trabalho de entrada em fila, de forma adequada. Ao receber uma mensagem de pulsação, o escalonador desencadeia o processo de roteamento para atribuir um trabalho para o recurso livre.

O escalonador *Quincy* (ISARD et al., 2009), utilizado na plataforma *Dryad* (ISARD et al., 2007), aborda o problema do escalonamento de trabalhos simultâneos em *clusters*, onde os dados do aplicativo são armazenados nos nós de computação. Ele introduz um novo, poderoso e flexível *framework* para escalonamento de trabalhos distribuídos simultâneos, com compartilhamento de recursos. Utiliza-se de grafos de fluxos para resolver as questões do escalonamento, considerando as questões de localidade de dados e balanceamento de carga. Nas avaliações de *Quincy*, demonstrou ser mais eficiente do que várias outras políticas de escalonamento.

K%-Fair Scheduling (ZHAO et al., 2012) é uma estratégia de escalonamento de tarefas mais flexível com base em várias filas de tarefas em nível de nó, levando

em consideração a equidade e localidade dos dados para definir a melhor opção de escalonamento. Os escalonadores de *MapReduce* (como o JT no *Hadoop*) podem utilizar este conhecimento a priori para escalonar tarefas e economizar custos de rede. O escalonador tem a informação de localização dos blocos de entrada em conta para agendar uma tarefa *Map* em um *nó* que contém uma réplica dos dados de entrada correspondente, que é chamado de localidade no *Hadoop*. Se falhar, ele tenta escalonar uma tarefa *Map* em um nó que está no mesmo *switch* de rede dos *nós* que contêm os dados, chamado localidade *rack* no *Hadoop*.

No trabalho de (YAO et al., 2014) é proposto um escalonador que foi desenvolvido especificamente para aplicações *MapReduce,* chamado de *HaSTE,* que é integrado ao *YARN*, com o objetivo de reduzir o tempo de processamento de aplicações que estão sendo executadas de forma concorrente, além de proporcionar uma utilização eficiente dos recursos disponíveis no *cluster.* Considerando os recursos requisitados, capacidade do cluster e as dependências entre as tarefas de cada aplicação.

Segundo (ZAHARIA et al., 2010b), Um escalonador deve levar em consideração a equidade e localidade dos dados. No entanto, há um conflito entre equidade e localidade dos dados no escalonamento de tarefas. Para resolver este conflito é proposto um algoritmo simples chamado *Delay Scheduling*: quando o trabalho que deve ser escalonado não consegue ser agendado para o mesmo nó onde seus dados de entrada residem, ele espera por um pequeno período de tempo, melhorando significativamente a localização dos dados. Com esse algoritmo é possível alcançar quase 100% de localidade, simplesmente atrasando o escalamento de tarefas, além de obter uma melhora nos tempos de resposta, para pequenos trabalhos, em 5 vezes.

Em (IBRAHIM et al., 2012), em seus estudos com *Hadoop*, demonstra que uma fonte de tráfego excessivo de rede e o elevado número de execuções de tarefas *Map* que processam dados remotos. Isso leva a um excessivo número de execuções especulativas inúteis de tarefas *Map* e uma execução desequilibrada dessas tarefas *Map* em diferentes nós. Esses fatores produzem uma degradação de desempenho. Os autores propõem um algoritmo de escalonamento de tarefas denominado *Maestro*, projetado para melhorar o desempenho da computação *MapReduce*, maximizando no número de tarefas *Map* locais.

O trabalho desenvolvido por (ANANTHANARAYANAN et al., 2011a) contrapõe as ideias apresentadas nesta tese. Para o autor, a questão de localidade dos dados será resolvida no futuro com o avanço dos diversos recursos tecnológicos, tornando-se irrelevante. No trabalho, os autores afirmam que a localização dos discos é irrelevante e irão questionar os dois itens da abordagem popular que são utilizados para justificar o uso dos discos locais: 1) a banda dos discos excede a banda de redes; 2) o custo de E/S de disco constitui uma fração considerável do ciclo de vida de uma tarefa. A tecnologia de redes evolui muito mais que a tecnologia empregada em discos. Atualmente, realizar a leitura de discos locais é apenas 8% mais rápido que realizar a leitura de um disco remoto via rede. Qualquer diferença encontrada, por menor que seja, produz resultados grandiosos devido ao tamanho dos dados processados.

Na Tabela 4.2 pode-se visualizar o resumo dos principais trabalhos encontrados na literatura que tratam de escalonamento de tarefas. Nenhum destes trabalhos trata do escalonamento de tarefas para dados espaciais e junção espacial que é um dos objetivos desta pesquisa.

Tabela 4.2 - Trabalhos encontrados na literatura sobre escalonamento de tarefas.

Autores	Escalonamento de tarefas	Dados Espaciais	Junção Espacial
(ANANTHANARAYANAN et al., 2011a)	Sim	Não	Não
(BARROSO; CLIDARAS; HÖLZLE, 2013)	Sim	Não	Não
(HASHEM et al., 2016)	Sim	Não	Não
(HOTT; ROCHA; GUEDES, 2016)	Sim	Não	Não
(IBRAHIM et al., 2012)	Sim	Não	Não
(ISARD et al., 2007)	Sim	Não	Não
(RASOOLI; DOWN, 2014)	Sim	Não	Não
(SENTHILKUMAR; ILANGO, 2016)	Sim	Não	Não
(YAO et al., 2014)	Sim	Não	Não
(ZAHARIA et al., 2010a)	Sim	Não	Não
(ZHAO et al., 2012)	Sim	Não	Não

4.4 CoS-HDFS

Todo o conteúdo desta seção está fortemente embasado no artigo *CoS-HDFS: Co-Locating Geo-Distributed Spatial Data in Hadoop Distributed File System* (FAHMY; ELGHANDOUR; NAGI, 2016), publicado no dia 13 de dezembro de 2016 no evento *IEEE/ACM 3rd International Conference on Big Data Computing.* O *CoS-HDFS* ataca um dos objetivos desta tese que é a alocação conjunta de dados espaciais relacionados com foco em dados geo-distribuídos em vários centros de dados.

Segundo (FAHMY; ELGHANDOUR; NAGI, 2016), os dados espaciais normalmente são geo-distribuídos em vários centros de dados, devido ao fato dos mesmos serem gerados em diferentes locais espalhados pelo globo. Os dados gerados em cada local são enormes, portanto são transferidos para um centro de dados geograficamente próximo. Uma das alternativas para analisar esses dados é copiá-los e armazená-los de centro de dados remotos para centros de dados local e analisá-los juntos. No entanto, esta alternativa não é eficiente e tem um alto custo de latência de rede. A latência é causada pelas leituras remotas e não pode ser ignorada se o *cluster* for geo-distribuído e os nós estiverem espalhados por vários centros de dados. Desta forma, os blocos lidos de centro de dados remotos geram alto custo de tráfego de rede, aumentando o tempo de processamento das consultas espaciais. A solução proposta para o problema é alocar os dados espaciais que são acessados de forma conjunta pelas consultas espaciais. Diante deste cenário, é proposto o *CoS-HDFS*, uma extensão do *Hadoop* que aloca os blocos de dados espaciais geo-distribuídos com base em suas características espaciais.

Para realizar a alocação, o *CoS-HDFS* acrescenta consciência aos dados espaciais na camada de armazenamento do *HDFS*, o que permite alocar os blocos com base em suas características espaciais. A alocação do *CoS-HDFS* adota uma abordagem semelhante ao *CoHadoop* (ELTABAKH et al., 2011), ou seja, é criada uma tabela de metadados com informações sobre o local de armazenamento dos blocos de cada arquivo. Desta forma a política padrão de alocação de dados do *HDFS* é modificada através do uso de *MBRs* que serão utilizados para agrupar os vários blocos de arquivos com características semelhantes nos mesmos nós do *cluster*. O objetivo é que cada entrada da tabela abranja um *MBR* mais amplo para

poder incluir vários blocos. Para atingir este objetivo é construída uma tabela localizadora com informações sobre blocos de dados espaciais armazenados no *HDFS*, conforme Figura 4.3.

Localização do MBR	Blocos
L1: MBR (-170, -80, -40, 80)	A1, A2, B1
L2: MBR (-40, -80, 170, 80)	A3, B2

Figura 4.3 – Tabela localizadora. Reproduzida de (FAHMY; ELGHANDOUR; NAGI, 2016).

Cada entrada na tabela localizadora representa blocos de dados espaciais que são alocados nos mesmos nós do *HDFS*. Conforme Figura 4.3, o *MBR* L1 armazena os blocos A1, A2 e B1 e o L2 armazena os blocos A3 e B2. O conteúdo dos blocos atribuídos à mesma entrada da tabela localizadora precisa estar relacionado espacialmente.

A construção e utilização da tabela localizadora é feita com base nas características espaciais dos dados. A tabela localizadora é dividida em entradas múltiplas sendo que cada uma tem um *MBR* específico. A escolha do número de entradas da tabela localizadora pode afetar o desempenho do *CoS-HDFS*. Um número pequeno significa que existem muitos blocos alocados em um mesmo local. Um número grande de entradas faz com que os dados estejam espalhados, como se não existisse a alocação de dados. O ideal é um número de entradas próximo à quantidade de nós do cluster. Desta forma, decidido o número de entradas da tabela localizadora o mesmo é enviado para o módulo de criação de índice global do *SpatialHadoop* para construção da tabela localizadora. Para cada bloco de arquivo que será armazenado no *HDFS*, calcula-se o seu *MBR* e verifica se o mesmo intersecta com algum *MBR* da tabela localizadora para atribuir o bloco a essa entrada da tabela. Todas as réplicas de blocos de dados atribuídos à mesma entrada na tabela localizadora são alocados, o que significa que elas são atribuídas ao mesmo conjunto de nós no *cluster HDFS*.

A Figura 4.4 apresenta um conjunto de dados para ilustrar como é o processo de alocação na tabela localizadora.

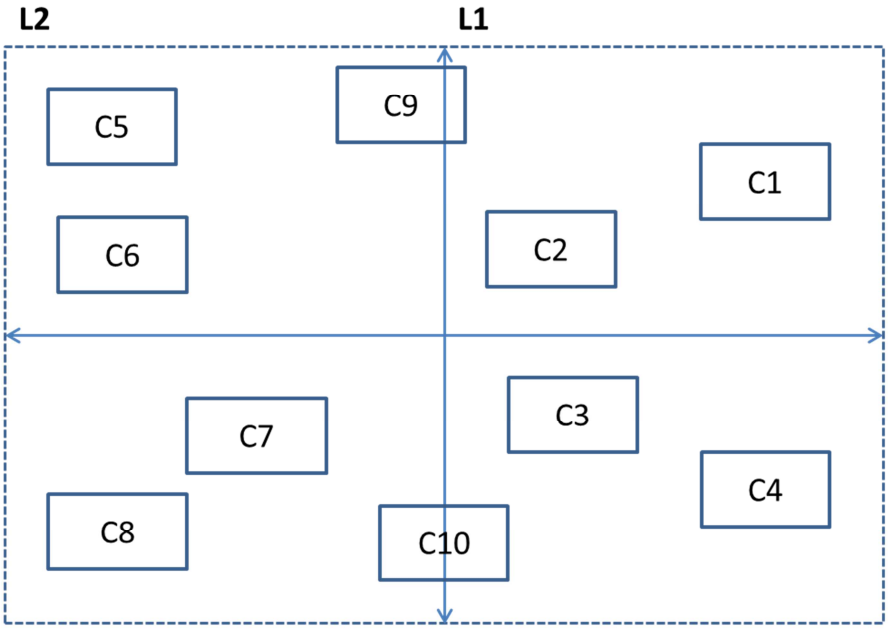


Figura 4.4 – Conjunto de dados. Reproduzida de (FAHMY; ELGHANDOUR; NAGI, 2016).

Os retângulos pontilhados L1 e L2 da Figura 4.4 representam os *MBRs* de duas entradas na tabela localizadora. Os retângulos sólidos C1 a C10 representam blocos de um arquivo de dados espacial. Na Figura 4.5 podem-se visualizar todos os blocos alocados na tabela localizadora.

Localização do MBR	Blocos
L1: MBR (coordenadas)	C1, C2, C3, C4 , C10
L2: MBR (coordenadas)	C5, C6, C7, C8, C9

Figura 4.5 – Tabela localizadora dos blocos. Reproduzida de (FAHMY; ELGHANDOUR; NAGI, 2016).

Os *MBRs* dos blocos C1, C2, C3 e C4 intersectam apenas com o *MBR* de L1, portanto os mesmos são atribuídos à entrada da tabela localizadora L1. Do mesmo modo, os *MBRs* dos blocos C5, C6, C7 e C8 intersectam apenas com o *MBR* de L2, desta forma são atribuídos à entrada da tabela localizadora L2. Os *MBRs* dos blocos C9 e C10 intersectam com ambas as entradas das tabelas L1 e L2. Neste caso, é necessário escolher apenas uma entrada da tabela localizadora para atribuir os blocos. O *MBR* do bloco C9 intersecta com o *MBR* da entrada da tabela localizadora L2 com mais de 80% e, portanto, atribui-se o mesmo a L2. Para o bloco C10 aplica-

se a condição especial que escolhe a entrada da tabela com o menor número de blocos atribuídos, no caso é o L1. Todas as réplicas dos blocos atribuídos à mesma entrada da tabela localizadora são alocados aos mesmos conjuntos de nós.

Com a alocação dos blocos realizada pelo *CoS-HDFS* é possível aumentar a probabilidade do escalonador do *Hadoop* agendar tarefas para os mesmos nós que estão os dados. Diante deste cenário, é possível reduzir o tráfego de rede incorrido durante a execução da consulta e, portanto, reduz o tempo de execução.

Foi demonstrado experimentalmente em testes de desempenho que a alocação dos blocos do *CoS-HDFS* melhora o tempo de execução das consultas de junção espacial em até 15% para arquivos com tamanho de 18 GB e a reduz o tráfego em até 47% para arquivos com tamanho de 150 GB. Observa-se que o *CoS-HDFS* alcança melhor desempenho quando lida com arquivos maiores.

4.5 Considerações finais

Neste capítulo, foram descritos os principais trabalhos encontrados na literatura que tratam de alocação de dados e escalonamento de tarefas em *clusters* de computadores que usam *Hadoop* com o objetivo de melhorar o desempenho do processamento das operações espaciais. Dentre os trabalhos que tratam de alocação de dados apenas o *CoS-HDFS* trata de dados espaciais e junção espacial, o restante dos trabalhos relacionados trata de alocação, porém de forma mais genérica. Com relação ao escalonamento de tarefas nenhum dos trabalhos encontrados enfoca no escalonamento de tarefas, para dados espaciais. Apesar deste problema ser reconhecido, ainda existe pouco entendimento sobre a interferência da localidade destes dados no desempenho do *Hadoop*.

No próximo capítulo será apresentado o algoritmo proposto *Cosphdp* que permite a alocação conjunta de dados espaciais relacionados e o escalonamento de tarefas para dados espaciais relacionados.

Capítulo 5

POLÍTICAS DE ALOCAÇÃO DE DADOS E DE ESCALONAMENTO DE TAREFAS PARA O *SPATIALHADOOP*

Este capítulo apresenta as principais contribuições desta pesquisa de doutorado. Na seção 5.1, são destacadas as características inovadoras das contribuições. Na seção 5.2, é apresentada a proposta de uma nova política de alocação conjunta de dados espaciais para o processamento de junção espacial, a qual é formalizada pelo algoritmo Cosphdp. A seção 5.3, descreve a proposta de uma nova política de escalonamento de tarefas para dados espaciais para o processamento de junção espacial para ser usada no SpatialHadoop. Na seção 5.4, é descrita a aplicação web desenvolvida para trabalhar com as novas implementações do SpatialHadoop que incorporam ambas as novas políticas de alocação de dados espaciais e de escalonamento de tarefas. Por fim, na seção 5.5, são destacadas as considerações finais deste capítulo.

5.1 Considerações iniciais

Este capítulo trata das principais contribuições em termos de originalidade desta tese de doutorado, tendo como principal objetivo:

Melhorar o desempenho do processamento de operações de junção espacial em *clusters* de computadores usando o *SpatialHadoop,* considerando grandes volumes de dados. Neste sentido, as soluções propostas exploram a alocação conjunta de dados espaciais relacionados e uma estratégia customizada de escalonamento de tarefas *MapReduce* para dados espaciais relacionados, alocados de forma conjunta.

Para alcançar o objetivo principal, foram traçados os seguintes objetivos específicos:

- Desenvolver um algoritmo para realizar a alocação conjunta de dados espaciais relacionados, visando o processamento eficiente de junção espacial;
- Desenvolver um algoritmo para realizar o escalonamento de tarefas *MapReduce* de acordo com a alocação conjunta de dados espaciais relacionados;
- Desenvolver uma aplicação *web* para a execução dos algoritmos das novas políticas de alocação conjunta de dados espaciais relacionados e do escalonamento de tarefas para dados espaciais relacionados no *SpatialHadoop*;

Na sequência, é detalhada a preparação do ambiente para execução da nova política de alocação conjunta de dados espaciais relacionados.

5.2 Preparação do ambiente

Antes de executar o algoritmo de alocação conjunta de dados espaciais relacionados é necessário configurar o ambiente do *HDFS*. O objetivo deste processo é criar uma pasta chamada /**cosphdp** que armazenará os dois arquivos de dados espaciais que serão utilizados pelo algoritmo de alocação de dados, conforme pode-se visualizar na Figura 5.1.

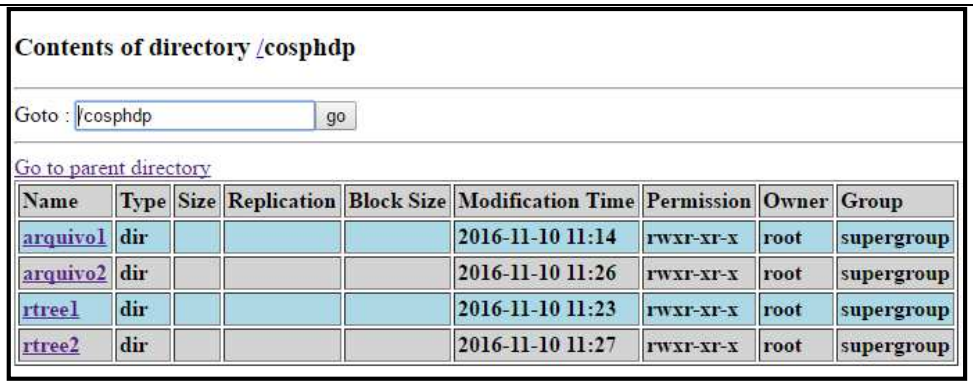


Figura 5.1 - Arquivos no *HDFS*.

Também será criada uma pasta **rtree2x** que irá receber o arquivo de parametrização chamado ***_cosphdp*** que contém os registros com os respectivos nomes dos arquivos que serão alocados. O arquivo ***_cosphdp*** será utilizado pela nova política de alocação de dados. Por meio da linha de comando ***printf "/cosphdp/rtree1\n/cosphdp/rtree2\n" | hadoop fs -put - /cosphdp/rtree2x/_cosphdp*** será criado o arquivo de parametrização, como pode-se visualizar na Figura 5.2.

Figura 5.2 - Arquivos de parametrização da política do *Cosphdp*.

Após a configuração do ambiente, pode-se proceder de duas formas para executar o algoritmo **Cosphdp**: (i) - movendo os dois arquivos de dados espaciais que se deseja usar no processamento da junção espacial para a pasta /**cosphdp** ou (ii) - criar os novos arquivos de dados espaciais na pasta /**cosphdp** do *HDFS,* conforme destacado na seção 3.7 deste trabalho.

A origem do nome ***Cosphdp*** é dada pela composição das palavras ***Co+sphdp.*** A palavra ***Co*** com o significado de coalocação de dados espaciais e ***sphdp*** sendo a abreviação para o *SpatialHadoop.* O algoritmo possui duas versões, a primeira versão realiza a alocação parcial dos dados com menor nível de replicação e menor custo de armazenamento e a segunda versão realização a alocação total dos dados com maior nível de replicação e maior custo de armazenamento.

5.3 Alocação parcial dos dados

O *Hadoop* trabalha com as funções *MapReduce,* com o objetivo de executar operações nos dados que estão armazenados no *HDFS*. A estratégia padrão adotada pelo *HDFS* é o armazenamento dos blocos e de suas respectivas réplicas em diferentes nós e *racks* de um *cluster* de computadores. Ele executa esta tarefa dividindo os grandes arquivos em blocos de tamanho fixo, por padrão de 64 MB, que serão distribuídos entre os nós do *cluster*. A alocação padrão dos blocos no *HDFS* está focada no balanceamento de carga e distribuição dos blocos de forma aleatória. Essa política aloca a primeira cópia do bloco no mesmo nó onde o dado foi criado; a segunda cópia é alocada em *rack* diferente da primeira cópia e a terceira cópia é alocada em um nó no mesmo *rack* da segunda cópia. Conforme ilustra Figura 5.3, pode-se visualizar o mapa de alocação padrão de blocos de dois arquivos **Rtree1** e **Rtree2**. Como exemplo podemos citar a alocação da partição **Part1** do arquivo **Rtree1,** destacado em amarelo também na Figura 5.3, cuja primeira réplica **R1** foi alocada no nó **BGD02**, **R2** no **BGD06** e **R3** no **BGD11**. Todas as réplicas estão alocadas conforme regras de alocação padrão do *HDFS*, ou seja, **R3** alocada no **BGD11** onde foi criada, **R1** e **R2** alocadas em um *rack* diferente de **R3**.

Observando-se a Figura 5.3, pode-se visualizar a lista de combinação que mostra quais processos *MapReduce* serão processados pela junção espacial (baseado na Figura 5.4 que mostra graficamente os dois arquivos de dados de entrada da junção espacial e a interseção entre as suas áreas) e a tabela de alocação que mostra para cada processo da lista de combinação a localização das três réplicas no *cluster* de computadores. Como exemplo, a primeira linha da tabela

de alocação mostra o processo *MapReduce* da **Part3** do arquivo **Rtree1** com interseção com a **Part1** do arquivo **Rtree2**. As réplicas da **Part3** do arquivo **Rtree1** estão alocadas nos nós **BGD01**, **BGD05**, **BGD09** e as réplicas da **Part01** do arquivo **Rtree2** estão alocadas nos nós **BGD02**, **BGD06** e **BGD12**. Nota-se na tabela de alocação que nenhum bloco está totalmente alocado. Isto se deve pelo fato da política padrão não focar a alocação conjunta dos blocos relacionados.

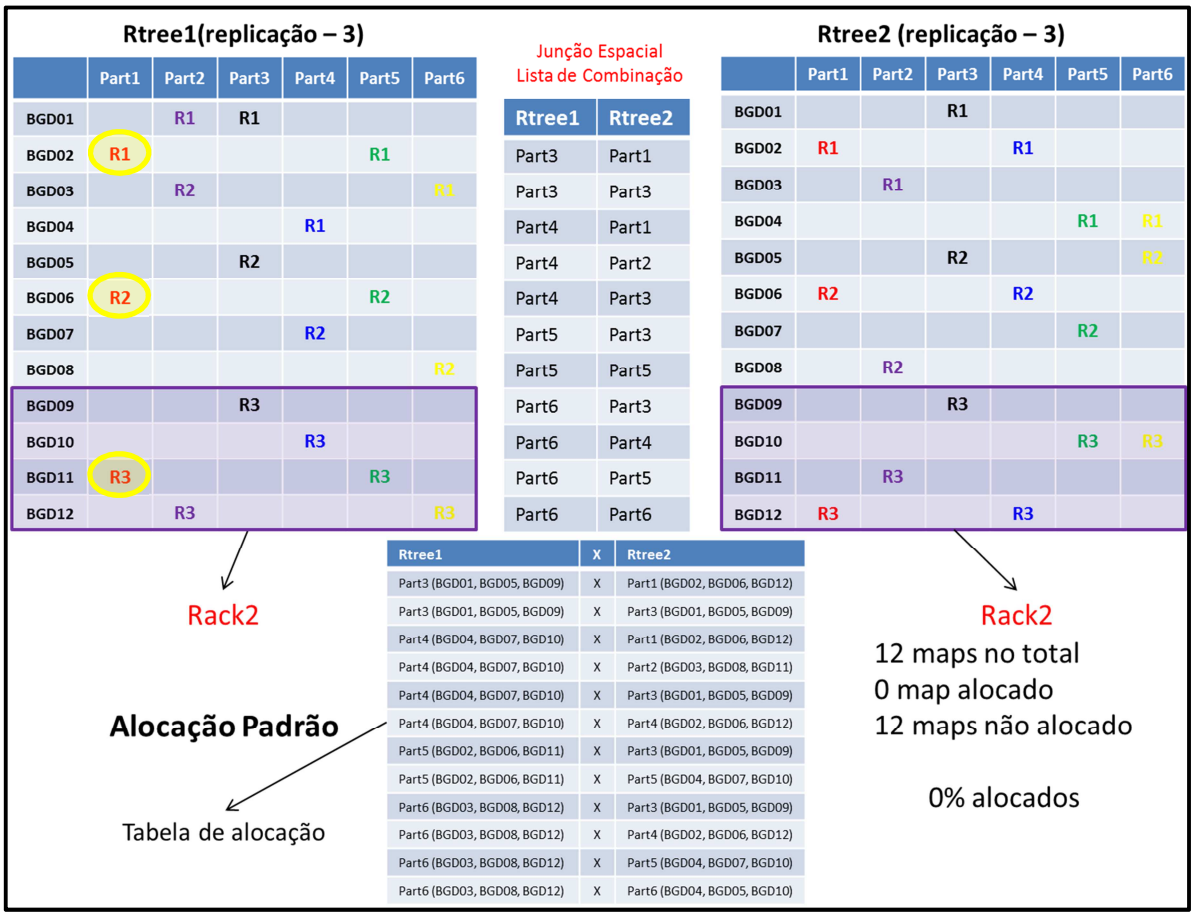


Figura 5.3 – Mapa de alocação padrão dos blocos do *HDFS*.

A política padrão do *HDFS* não considera qualquer tipo de relacionamento entre os dados de dois ou mais conjunto de dados durante o processo de alocação dos blocos nos nós dos *racks*. Ela é eficiente quando o *Hadoop* processa um único conjunto de dados, mas torna-se ineficiente quando o processamento envolve conjuntamente dois ou mais conjuntos de dados. Esta situação causa perda de desempenho no processamento da junção espacial que é o problema-alvo de investigação desta tese. Esse problema é gerado devido ao alto tráfego de rede e maior custo de E/S de disco que se faz necessário para movimentar blocos de dados

entre o nó de armazenamento e o nó de processamento. Tal situação acarretou a oportunidade de criação de uma nova política de alocação conjunta de dados espaciais relacionados para o processamento de junção espacial, sendo os blocos pertencentes aos dois arquivos de dados de entrada da junção espacial. A solução propõe uma mudança na política padrão de alocação de dados do *HDFS*. Desta forma, foi projetado o algoritmo *Cosphdp* para resolver a questão de alocação conjunta de dados espaciais relacionados no *SpatialHadoop*. Ele é uma extensão do *Hadoop*, ou seja, foi incluído no núcleo do *Hadoop* sem alterar as características básicas, desta forma é possível que as aplicações possam adotar o novo algoritmo facilmente sem prejudicar o funcionamento geral do *Hadoop*.

O algoritmo ***Cosphdp*** é executado pela política de alocação de dados do *SpatialHadoop* no momento que um bloco de arquivo é copiado no *HDFS*. Conforme a linha de comando ***hadoop dfs -cp /cosphdp/rtree2/* /cosphdp/rtree2x*** que irá informar ao *HDFS* que o arquivo **Rtree2** e todos os seus blocos devem ser copiados para a pasta ***/cosphdp/rtree2x*** e consequentemente alocados, levando-se em consideração a localização dos dados espaciais contidas no arquivo **Rtree1,** sendo o arquivo **Rtree1** definido na configuração do arquivo **_cosphdp**. Para cada bloco copiado do arquivo **Rtree2** o mesmo é adicionado à pasta ***/cosphdp/rtree2x,*** através da execução da nova política de alocação de dados espaciais que será executada pelo algoritmo ***Cosphdp***.

A ideia geral do algoritmo ***Cosphdp*** é verificar para cada bloco de **Rtree1** a sua interseção com blocos de **Rtree2**. Onde houver interseção, há a necessidade de alocação conjunta, pois existem dados espaciais relacionados. Caso mais de um bloco do arquivo **Rtree2** tenha interseção com o mesmo bloco do arquivo **Rtree1** é calculada a área de interseção entre o bloco **Rtree1** com cada bloco de **Rtree2**. Após encontrar as áreas de interseção entre o bloco **Rtree1** e os blocos **Rtree2**, escolhe-se a maior área de interseção e com isso obtêm-se os nós onde os blocos de **Rtree1** foram alocados para esta maior área. Com isso, alocam-se os blocos da **Rtree2** que intersectam a maior área dos blocos de **Rtree1** nos mesmos nós do *cluster*. A política altera apenas a alocação dos blocos de **Rtree2** baseado na alocação dos blocos **Rtree1**, sendo que os blocos **Rtree1** foram alocados de acordo com a política padrão do *HDFS*.

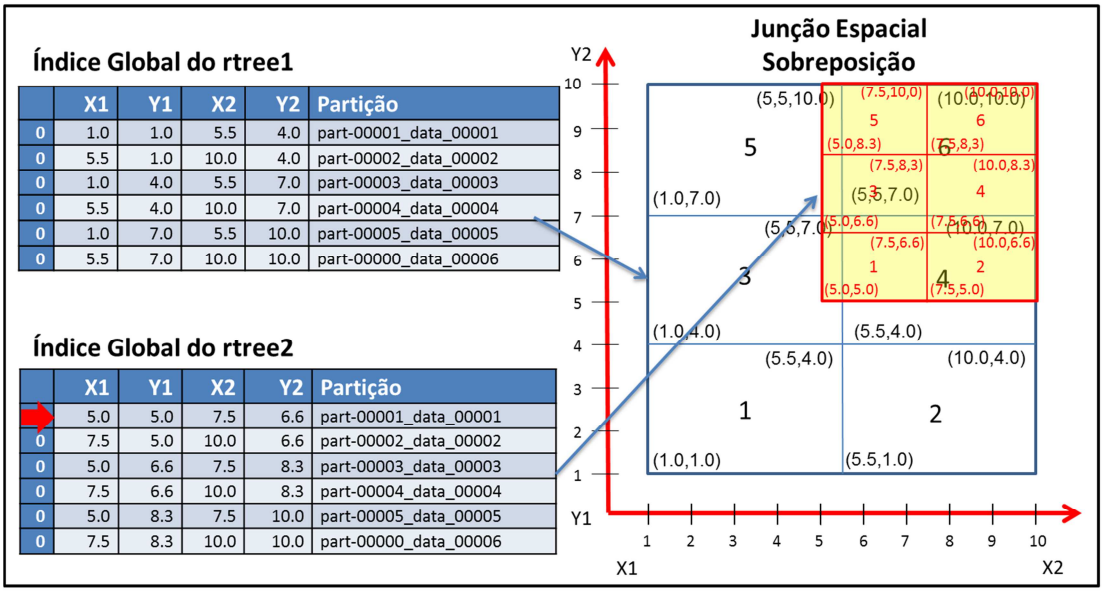
O pseudocódigo do algoritmo da nova política de alocação parcial de dados (Algoritmo 1) é apresentando na sequência, juntamente com o detalhamento do seu funcionamento.

Algoritmo 1: Cosphdp – Alocação parcial dos dados

```
01 Se existir um arquivo chamado _cosphdp Então
02 {
03      Leia a primeira linha do arquivo _cosphdp na variável file1Path
04      Leia a segunda linha do arquivo _cosphdp na variável file2Path
05      Atribui null a variável rectangle2
06      Para cada linha do arquivo file2Path/_master.rtree
07      {
08              Se o nome do arquivo for o mesmo arquivo criado Então
09              {
10                      Leia as coordenadas da linha na variável rectangle2
11              } Fim Então
12      } Fim Então
13      Atribui null a variável name1
14      Se rectangle2 for diferente de null Então
15      {
16              Atribui 0 a variável maxArea
17              Para cada linha do arquivo file1Path/_master.rtree
18              {
19                      Leia as coordenadas da linha na variável rectangle1
20                      Se rectangle1 intersecta com rectangle2 Então
21                      {
22                              Se a área de interseção for maior que maxArea Então
23                              {
24                                      Atribuí o nome do arquivo a variável name1
25                                      Atribuí a área de interseção a variável maxArea
26                              } Fim Então
27                      } Fim Então
28              } Fim para cada
29      }
30      Se name1 for diferente de null Então
31      {
32              Retorna os nós do file1Path/name1 para a gravação dos blocos e réplicas
33      } Fim Então
34 } Fim Então
35 Senão executa a lógica original de alocação de blocos Fim Senão
```

A política inicia a execução, verificando se o arquivo de parametrização **_cosphdp** existe (linha 1). Este arquivo foi criado na seção 5.2, e se o arquivo não for encontrado, a lógica é direcionada para a execução da política padrão de alocação de dados do *HDFS (linha 35)*. Se o arquivo de parametrização existir é realizada a leitura do arquivo **_cosphdp** para identificar os dois conjuntos que serão utilizados para alocação dos dados. Na linha 3, é realizada a leitura do primeiro registro do arquivo **_cosphdp** e o seu conteúdo /**cosphdp**/**rtree1** é armazenado na

variável **file1Path**. Na linha 4, é realizada a leitura da segunda linha do arquivo e o seu conteúdo /**cosphdp**/**rtree2** é armazenado na variável **file2Path**. O registro armazenado na variável **file2Path** será alocado pela política do **Cosphdp** baseado nas localizações dos blocos do registro da variável **file1Path**. Na linha 5, é atribuído nulo para a variável **rectangle2** que irá armazenar as coordenadas geográficas dos registros do arquivo **Rtree2**. A próxima etapa é realizar a leitura do índice global do arquivo **Rtree2** e para cada linha lida (linha 6) verifica se o nome do registro lido é o mesmo do registro que está sendo copiado (linha 8). Caso seja o mesmo registro, é realizada a leitura de suas coordenadas (5.0, 5.0, 7.5, 6.6) que são armazenadas na variável **rectangle2** (linha 10), conforme indicado pela seta vermelha da Figura 5.4.



	X1	Y1	X2	Y2	Partição
0	1.0	1.0	5.5	4.0	part-00001_data_00001
0	5.5	1.0	10.0	4.0	part-00002_data_00002
0	1.0	4.0	5.5	7.0	part-00003_data_00003
0	5.5	4.0	10.0	7.0	part-00004_data_00004
0	1.0	7.0	5.5	10.0	part-00005_data_00005
0	5.5	7.0	10.0	10.0	part-00000_data_00006

	X1	Y1	X2	Y2	Partição
	5.0	5.0	7.5	6.6	part-00001_data_00001
0	7.5	5.0	10.0	6.6	part-00002_data_00002
0	5.0	6.6	7.5	8.3	part-00003_data_00003
0	7.5	6.6	10.0	8.3	part-00004_data_00004
0	5.0	8.3	7.5	10.0	part-00005_data_00005
0	7.5	8.3	10.0	10.0	part-00000_data_00006

Figura 5.4 – Leitura do índice global do arquivo *Rtree2*.

Após encontrar o registro que está sendo copiado, a linha 13 atribui nulo a variável **name1** que irá receber o nome da partição do arquivo **Rtree1** que será utilizado para obter o conjunto de nós onde estão localizados seus blocos. Na linha 14, verifica se a variável **rectangle2** não é nula e caso não seja, é atribuído 0 para a variável **maxArea** (linha 16). Na linha 17, é realizada a leitura do índice global do arquivo /**cosphdp**/**rtree1**/**_master.rtree,** suas coordenadas são armazenadas na variável **rectangle1** (1.0, 1.0, 5.5, 4.0) (linha 19), conforme indicado pela seta amarela da Figura 5.5.

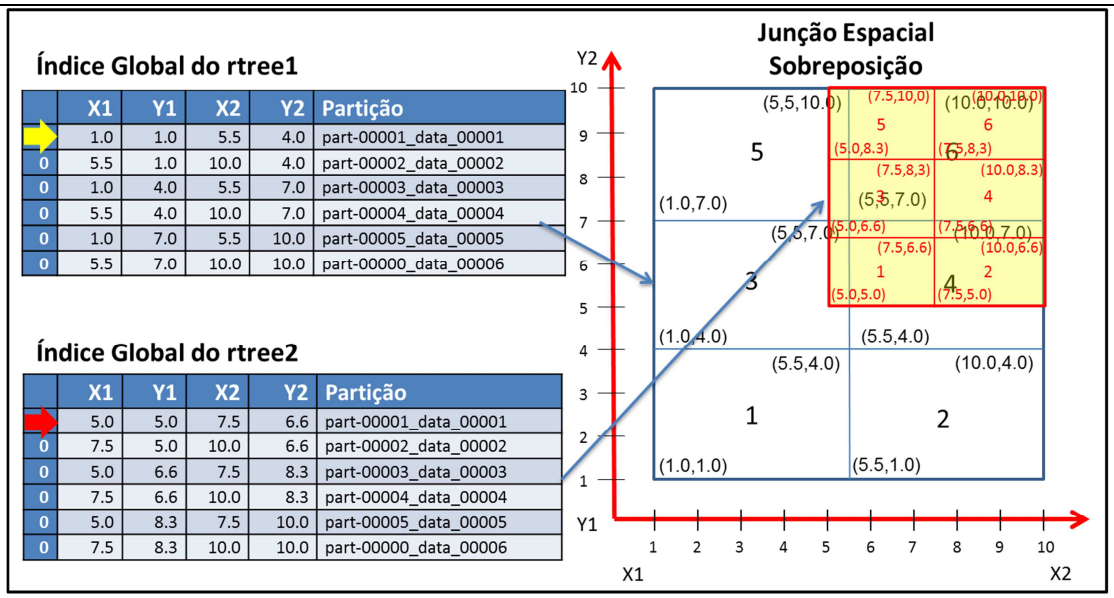


Figura 5.5 – Leitura do índice global do arquivo *Rtree1*.

Nesta etapa, para o exemplo corrente, o conteúdo da variável **rectangle1** é (5.0, 5.0, 7.5, 6.6) e o conteúdo da variável **rectangle2** é (1.0, 1.0, 5.5, 4.0), desta forma é analisado se as coordenadas da variável **rectangle1** intersectam às coordenadas da variável **rectangle2** (linha 20). Se ocorrer interseção das coordenadas é verificado se a área de interseção é maior do que o conteúdo da variável **maxArea** (linha 22). Se for verdadeiro, atribui o nome do registro para a variável **nome1** (linha 24), que irá receber o conteúdo **part-00001_data_00001** e atribui a área de interseção para a variável **maxArea** (linha 25). Na sequência, retorna o processamento para a linha 19 para a realização da leitura do próximo registro do arquivo **/cosphdp/rtree1/_master.rtree,** para identificar se existe mais registros que intersectam a variável **rectangle2**. Este processo se repete com o objetivo de selecionar a maior área de interseção com a variável **rectangle2.**

Ao final da leitura do arquivo **/cosphdp/rtree1/_master.rtree,** verifica-se o conteúdo da variável **name1** é diferente de nulo (linha 30). Neste caso, o conteúdo da variável é **part-00004_data_00004** que corresponde a maior área do arquivo **Rtree1** que intersecta a área **part-00001_data_00001** do arquivo **Rtree2.** Desta forma, o algoritmo seleciona o conjunto de nós que armazena o bloco original e as duas réplicas da partição **part-00004_data_00004** do arquivo **Rtree1** que corresponde a **(BGD04, BGD07, BGD10).** Esses nós serão utilizados no processo de gravação do novo bloco e das duas réplicas da partição **part-00001_data_00001**

do arquivo **Rtree2** (linha 32), conforme destaque em amarelo da Figura 5.6, onde visualiza-se a alocação padrão do *HDFS* que será atualizada conforme política *Cosphdp* na Figura 5.7.

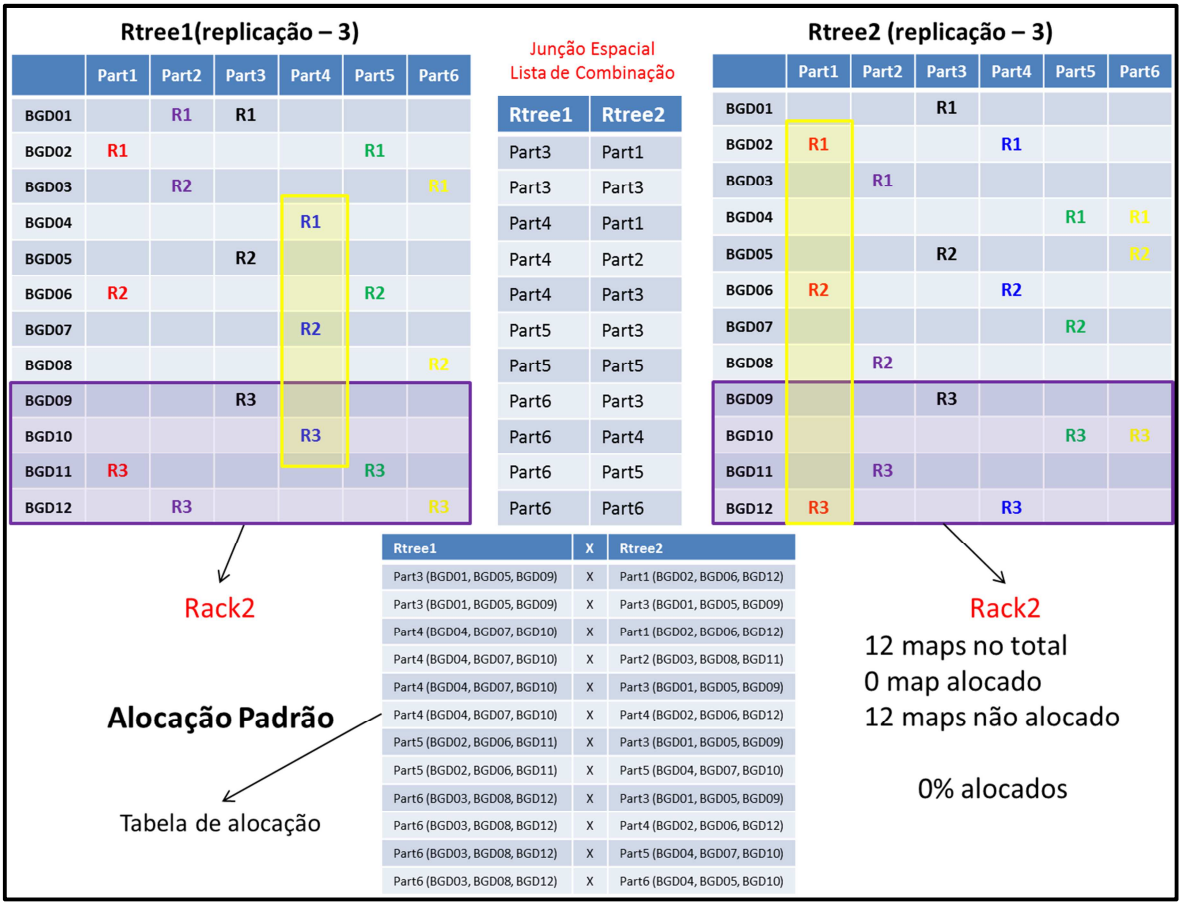


Figura 5.6 – Mapa de alocação dos blocos da partição part-00004_data_00004 do arquivo *Rtree1* e alocação dos blocos da partição part-00001_data_00001 do arquivo *Rtree2*.

Após a execução do algoritmo, os blocos e réplicas da **part-00001_data_00001** do arquivo **Rtree2** terão sido alocados com as mesmas localizações dos blocos e réplicas da partição **part-00004_data_00004** do arquivo **Rtree1,** conforme mostras a Figura 5.7.

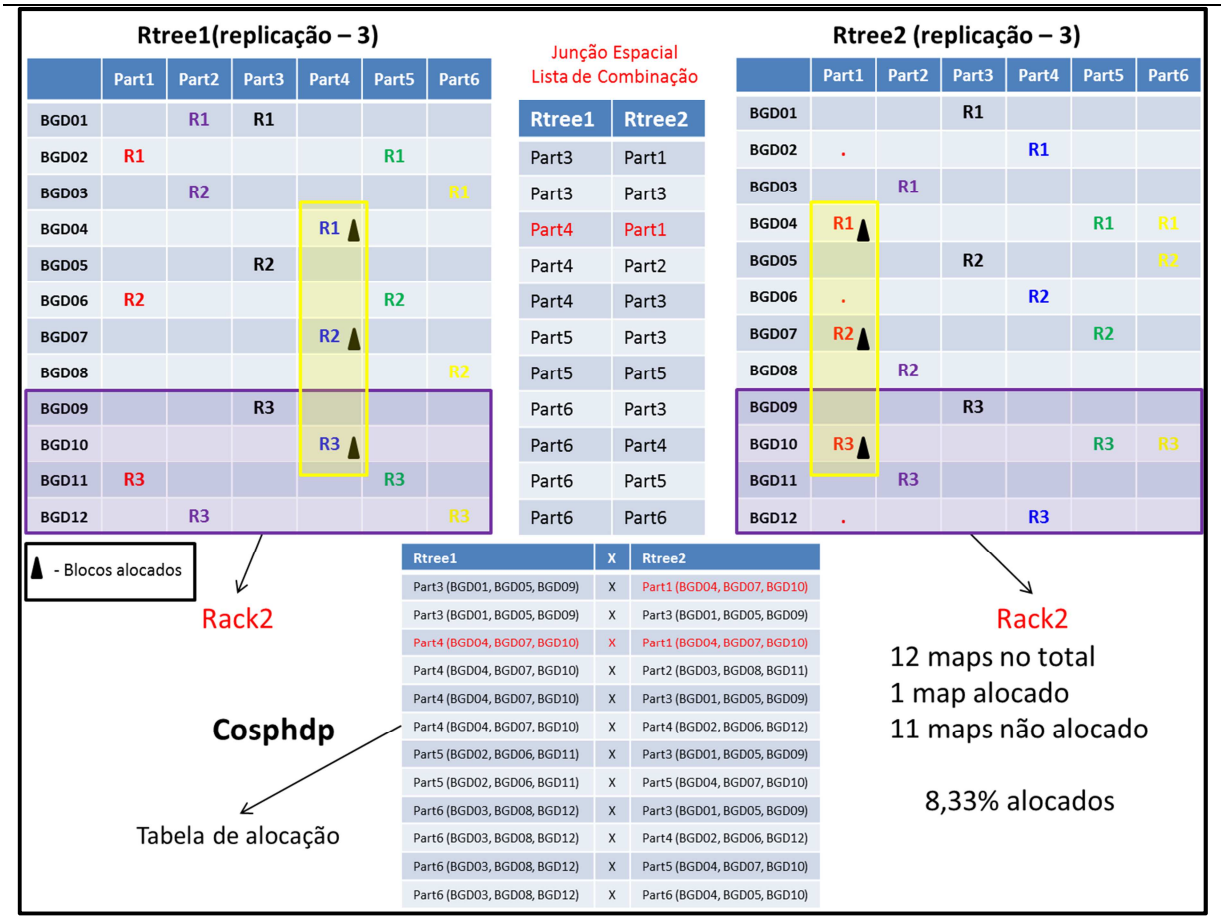


Figura 5.7 – Exemplo de alocação conjunta de dados espaciais relacionados pelo *Cosphdp*.

Observando-se a Figura 5.7, pode-se visualizar a lista de combinação de blocos dos dois arquivos de entrada da junção espacial e a tabela de alocação destes blocos, onde as linhas em vermelho indicam os blocos alocados de forma relacionada.

Esse mesmo procedimento se repete para cada bloco copiado do arquivo **Rtree2**. Ao final da execução, todos os blocos e réplicas do arquivo **Rtree2** que são relacionados pelo predicado espacial de interseção com os blocos e réplicas do arquivo **Rtree1** estarão armazenados nos mesmos nós do *cluster,* conforme pode-se visualizar no mapa de alocação dos blocos da Figura 5.8.

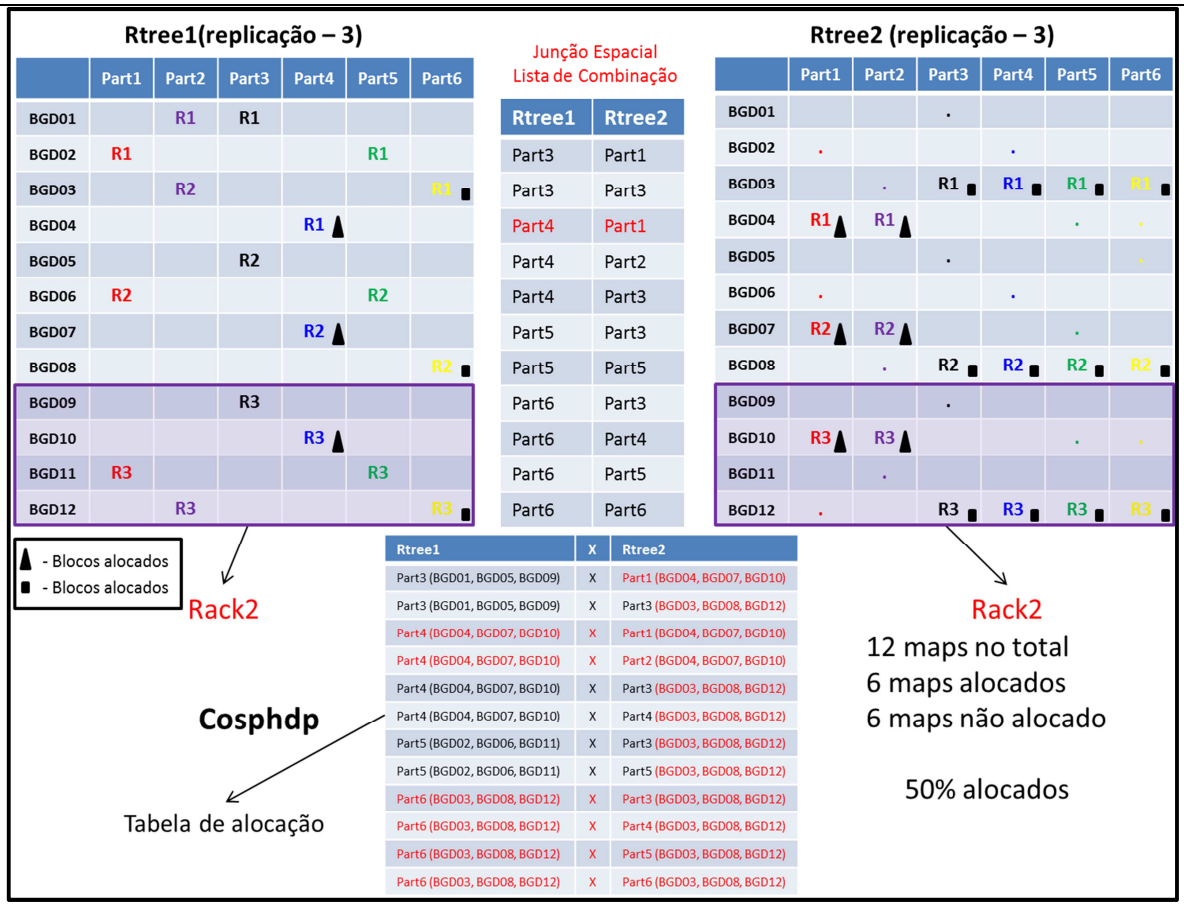


Figura 5.8 – Mapa de alocação parcial dos dados.

Observando-se a Figura 5.8, pode-se visualizar a tabela de alocação, onde as linhas em vermelho representam 6 blocos que estão alocados de forma relacionada.

Pode-se observar na Figura 5.8, que o processo de alocação dos dados deve ser utilizado com o foco na melhoria de desempenho das operações de junção espacial, pois questões relacionadas ao balanceamento de carga dos blocos são flexibilizadas e pode ocorrer um desbalanceamento de carga. Em geral, um excesso de blocos com características geográficas semelhantes pode levar à distorção na distribuição de dados. Se a política for utilizada de forma sensata, ela aumenta apenas ligeiramente a variação da carga. A solução para este problema é proposta como trabalhos futuros nesta tese de doutorado. Outro ponto, é quando muitos arquivos de um determinado conjunto, com características geográficas semelhantes, são armazenados em um mesmo nó, pode ocasionar uma falta de espaço. Neste caso, o *Cosphdp* sacrifica a alocação dos arquivos e adota a política padrão de alocação de dados. Esta situação pode ser evitada configurando os parâmetros do algoritmo *Cosphdp,* por meio da variável total de espaço alocado. Desta forma,

quando o armazenamento de dados no mesmo nó atingir o percentual da variável total de espaço alocado, a política padrão é executada.

5.3.1 Alocação total dos dados

O algoritmo de alocação total dos dados segue, em princípio, a mesma ideia geral do algoritmo de alocação parcial. Enquanto o algoritmo de alocação parcial seleciona somente a maior área de interseção para alocar os dados, o algoritmo de **alocação total** aloca os dados para todas as partições que se intersectam. Desta forma, é necessário alterar o parâmetro de replicação dos dados *dfs_replication*. A ideia do algoritmo é verificar para cada bloco de **Rtree1** a sua interseção com blocos de **Rtree2**. Onde houver interseção, há a necessidade de alocação conjunta, pois existem dados espaciais relacionados. Com isso, alocam-se os blocos de **Rtree2** que intersectam os blocos de **Rtree1** nos mesmos nós e caso necessário, altera o número de réplicas dos blocos do arquivo **Rtree2** para obter a alocação total dos dados.

O pseudocódigo do algoritmo da nova política de alocação total de dados (Algoritmo 2) é apresentando na sequência, juntamente com o detalhamento do seu funcionamento. As linhas destacadas em amarelo representam as mudanças necessárias em relação ao algoritmo de alocação parcial.

Algoritmo 2: Cosphdp – Alocação total dos dados

```
01 Se existir um arquivo chamado _cosphdp Então
02 {
03      Atribui o valor padrão de replicação para a variável dfs-replication do HDFS
04      Leia a primeira linha do arquivo _cosphdp na variável file1Path
05      Leia a segunda linha do arquivo _cosphdp na variável file2Path
06      Atribui null a variável rectangle2
07      Para cada linha do arquivo file2Path/_master.rtree
08      {
09          Se o nome do arquivo for o mesmo arquivo criado Então
10          {
11              Leia as coordenadas da linha na variável rectangle2
12          } Fim Então
13      } Fim Então
14      Atribui null a variável name1
15      Cria o vetor vazio nodes
16      Se rectangle2 for diferente de null Então
17      {
18          Para cada linha do arquivo file1Path/_master.rtree
19          {
20              Atribui ao nome do arquivo da linha atual a variável name1
21              Leia as coordenadas da linha na variável rectangle1
22              Se rectangle1 intersecta com rectangle2 Então
23              {
24                  Adiciona ao vetor nodes os nós que possuem réplicas do
                    arquivo name1
25              } Fim Então
26          } Fim para cada
27      }
28      Se  nodes não for um vetor vazio Então Então
29      {
30          Leia quantidade de registro do vetor nodes na variável qtdeVetor
31          Atualiza a variável dfs-replication conforme qtdeVetor
32          Retorna o vetor nodes eliminando os itens duplicados
33      } Fim Então
34 } Fim Então
35 Senão executa a lógica original de alocação de blocos Fim Senão
```

A linha 1 realiza a mesma ação descrita para o algoritmo de alocação parcial (Algoritmo 1). Se o arquivo de parametrização existir, é atualizado o parâmetro de replicação *dfs-replication* do *HDFS* para o valor padrão que é de 3 réplicas (linha 3). As linhas 4 a 14 realizam a mesma ação descrita no Algoritmo 1. Na linha 15 é criado o vetor **nodes** com valor nulo. O vetor irá armazenar todos os blocos do arquivo **Rtree1** que tenha interseção com os blocos do arquivo **Rtree2**. As linhas 16 a 23 realizam a mesma ação descrita no Algoritmo 1. Se ocorrer interseção das coordenadas, o vetor **nodes** é atualizado com os nós que possuem réplicas do arquivo da variável **name1** (linha 24). Na sequência, retorna o processamento para a

linha 20 para a realização da leitura do próximo registro do arquivo /**cosphdp**/**rtree1**/**_master.rtree** para identificar se existe mais registros que intersectam a variável **rectangle2**. Este processo se repete com o objetivo de selecionar todos os nós com interseção com a variável **rectangle2.** Ao final da leitura do arquivo /**cosphdp**/**rtree1**/**_master.rtree,** verifica se o conteúdo da variável **name1** é diferente de nulo (linha 28), caso seja, é lida do vetor **nodes** a quantidade de registros (linha 30) e baseado na quantidade de registros é feita a atualização do parâmetro de replicação *dfs-replication* (linha 31). Desta forma, o algoritmo retorna todos os nós do vetor **nodes** que serão utilizados para criar os novos arquivos alocados. Esse mesmo procedimento se repete para cada bloco copiado do arquivo **Rtree2**. Ao final da execução, todos os blocos e todas as réplicas do arquivo **Rtree2** que são relacionados pelo predicado espacial de interseção com os blocos e réplicas do arquivo **Rtree1** estarão armazenados nos mesmos nós do *cluster.* Na Figura 5.9, pode-se visualizar no mapa de alocação dos blocos assim como a lista de combinação e tabela de alocação, com todas as linhas em vermelho, demonstrando que todos os blocos que possuem interseção estão alocados de forma conjunta.

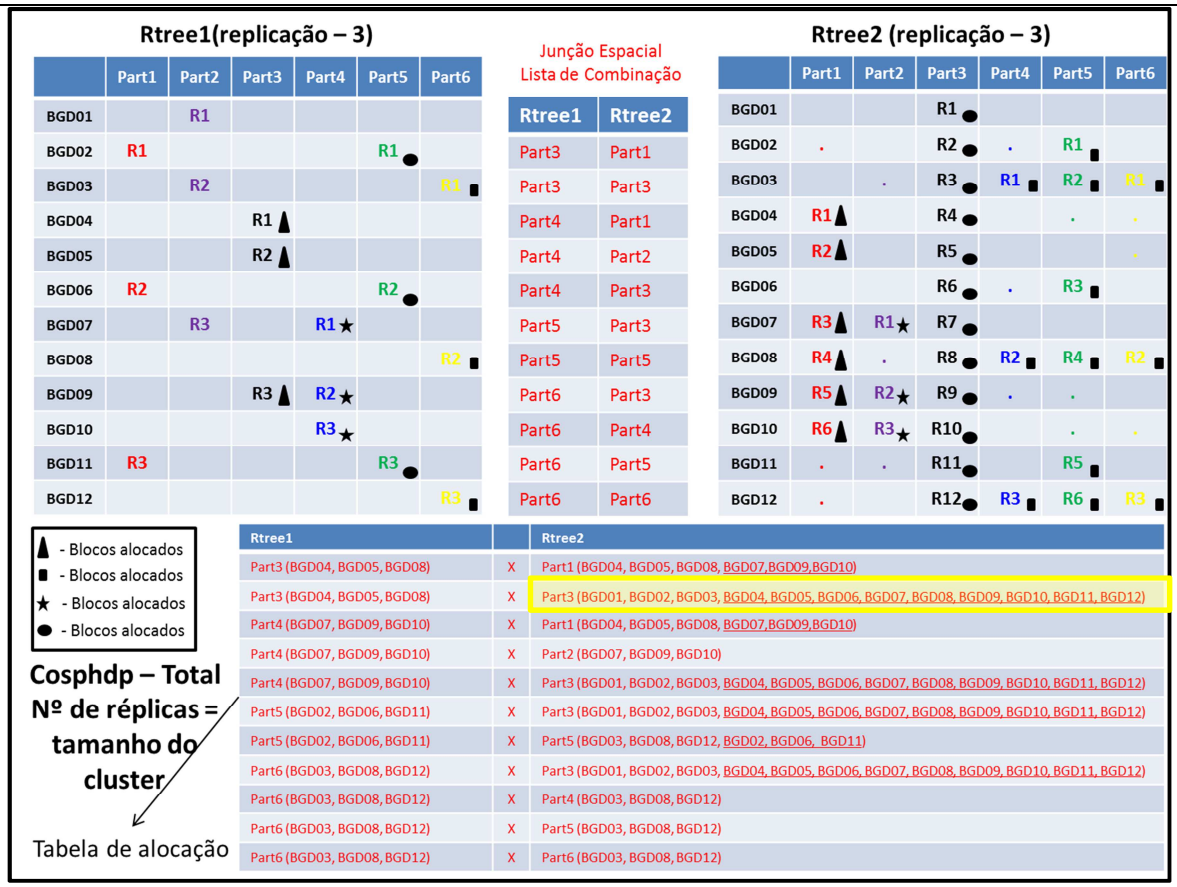


Figura 5.9 – Mapa de alocação total dos dados.

A política de alocação total dos dados aumenta o desbalanceamento do *cluster* por causa da alteração do parâmetro de replicação dos dados, podendo chegar ao máximo no mesmo número de nós do *cluster*, para este exemplo, doze. Conforme tabela, com destaque em amarelo, da Figura 5.9, é possível visualizar que a **Part3** do arquivo **Rtree2** foi alocada em todos os nós do *cluster*. Isto se deve pelo fato da partição possuir várias interseções. Diante deste contexto, pode-se afirmar que a política de alocação total dos dados deve ser utilizada quando há questões de desempenho crítico no processamento de operações de junção espacial sem restrições de custos de armazenamento.

5.4 Escalonador de tarefas

O objetivo da política é reduzir o tempo de processamento das tarefas *MapReduce* que serão executadas nos nós de um *cluster*. Para alcançar esse objetivo, foi alterada a política de escalonamento padrão do *Hadoop* que é usada pelo *SpatialHadoop*. O novo escalonador, busca alocar a execução de uma determinada tarefa *MapReduce* no mesmo nó onde estão os dados espaciais de entrada, alocados conjuntamente, conquistando os benefícios da redução do tráfego de rede e o custo de E/S de disco, que é o objeto de estudo desta tese.

O problema de escalonamento de tarefas constitui-se em um dos mais desafiadores em computação paralela e distribuída. A política padrão de alocação de dados do *Hadoop* resolve parcialmente a questão de tráfego de rede e o custo de E/S de disco, devido à possibilidade do escalonador de tarefas atribuir o processamento de uma função *MapReduce* para um determinado nó do *cluster,* que não contenha os dados que serão utilizados no processamento. Desta forma, a alocação conjunta de dados espaciais relacionados, criados pela nova política, pode não obter o melhor desempenho no processamento das operações de junção espacial.

O escalonador padrão do *Hadoop* é o *first-in-first-out* (*FIFO*) (WHITE, 2012), que agenda uma tarefa *MapReduce* para um nó disponível. O nó mestre é o responsável pelo escalonamento das tarefas no *Hadoop*. Este processo é realizado através do gerenciamento dos nós escravos que constantemente enviam pulsações sobre suas atividades para o nó mestre. O nó mestre recebe essas pulsações e atribui tarefas para os nós escravos, disponíveis em determinados períodos de tempo. Este processo pode ser visualizado na Figura 5.10, onde os nós escravos estão enviando pulsações para o nó mestre. Assim que recebe a pulsação dos nós escravos, o nó mestre pode atribuir tarefas aos mesmos.

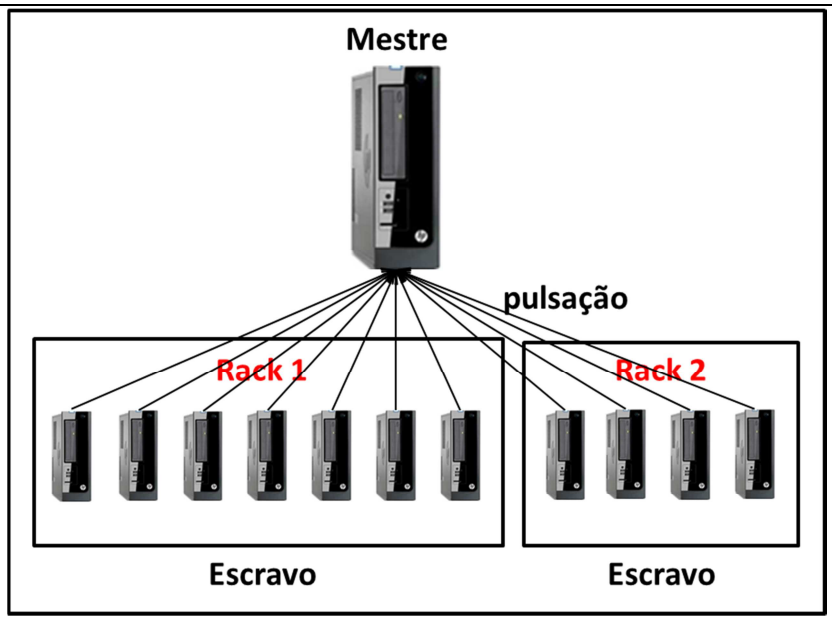


Figura 5.10 - Processo de pulsação.

Por padrão, o escalonador procura atribuir às tarefas para os mesmos nós onde estão localizados os dados. Porém, muitas vezes, as pulsações recebidas dos nós escravos, em um determinado momento, não coincidem com os nós onde estão armazenados os dados, ocasionando o agendamento da tarefa para um nó onde não estão localizados os dados; não aproveitando, desta forma, os benefícios da alocação conjunta de dados espaciais relacionados. Esta situação provoca a migração de dados do nó de armazenamento para o nó de processamento, aumentando de forma significativa o tráfego de rede e o custo de E/S de disco, consequentemente reduzindo o desempenho do processamento das operações de junção espacial, conforme pode-se visualizar na Figura 5.11.

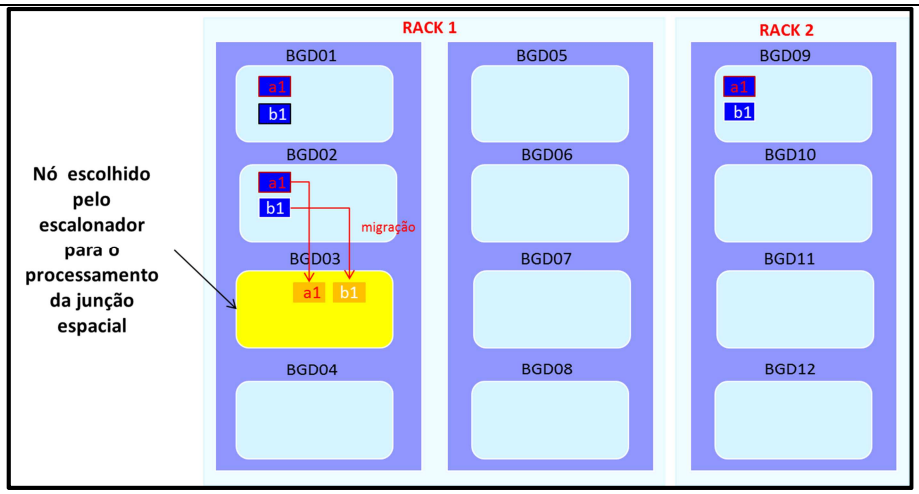


Figura 5.11 - Migração dos dados. Adptada de (GOLDMAN et al., 2012).

Neste exemplo, é solicitado pelo usuário o processamento da operação de junção espacial, utilizando o predicado espacial de interseção sobre os blocos de arquivos **a1** e **b1**, que estão localizados nos nós **BGD01**, **BGD02** e **BGD09** e são resultados da execução da nova política de alocação conjunta de dados espaciais relacionados ***Cosphdp***. Para atribuir o nó responsável pelo processamento da operação, a política padrão recebe as pulsações provenientes dos nós escravos. Porém, nenhum dos nós relacionados enviou pulsações no momento que o nó mestre foi atribuir às tarefas. Desta forma, é atribuído o processamento da tarefa ao primeiro nó que enviou a pulsação, no caso o nó **BGD03**. Observando a Figura 5.11, nota-se que o nó responsável pelo processamento não contém os blocos de dados que serão usados no processamento da junção espacial. Desta forma, é necessário mover todos esses blocos pela rede e armazená-los no nó **BGD03** para o seu processamento. Essa particularidade aumenta o tráfego de rede e o custo de E/S de disco, acarretando perda de desempenho no processamento de operações espaciais. Mesmo que os blocos relacionados estejam 100% alocados conjuntamente, se o escalonador de tarefas não conseguir escalonar tarefas de forma eficiente, a alocação não irá surtir o efeito desejado no desempenho final das operações de junção espacial.

A solução para resolver este problema é alterar a política padrão de escalonamento de tarefas do *Hadoop*, criando uma nova política que altera a forma como os nós são selecionados para o processamento. Deste modo, a nova política de escalonamento de tarefas para dados espaciais deverá atribuir as tarefas para os

mesmos nós que possuem os dados espaciais de entrada da junção espacial. Este objetivo é alcançado com a implementação do algoritmo de escalonamento de tarefas. O objetivo do processamento da política é decidir, em tempo de execução, qual o melhor nó do *cluster* para a execução de uma determinada tarefa sem prejudicar o balanceamento de carga, reduzindo o tráfego de rede e o custo de E/S de disco.

Para a realização do escalonamento de tarefas para dados espaciais relacionados foi necessário alterar a função original do *SpatialHadoop prioritizeLocations*. Para o processamento da lista de combinação da Figura 5.11, são executados os seguintes passos:

Primeiro – Recebe a lista de combinação proveniente da execução da junção espacial sobre o índice global dos dois conjuntos **Rtree1** e **Rtree2**. Para cada linha da lista de combinação, será executada uma função *map,* na sequência que a lista foi criada, visto que o escalonador do *Hadoop* agenda os processos para os nós do *cluster* por ordem de chegada, conforme lista de combinação da Figura 5.12.

Segundo – Considerando o exemplo descrito na Figura 5.12, o processamento pega o primeiro registro da lista de combinação, que será o primeiro *map* que será escalonado para o processamento que é a verificação da interseção das **Part3** do arquivo **Rtree1** com **Part1** do arquivo **Rtree2**. Diante deste cenário, as localizações das partições **Part3** (BGD1, BGD5, BGD9) e **Part1** (BGD04, BGD07, BGD10) são enviadas para o algoritmo *prioritizeLocations*.

Terceiro – O algoritmo *prioritizeLocations* recebe a lista de localizações (BGD1, BGD5, BGD9, BGD04, BGD07, BGD10) e ordena de forma decrescente pela frequência. Por exemplo, se as duas partições tem nós em comum o mesmo irá aparecer em primeiro lugar após o processamento do algoritmo. Como no exemplo, as partições não possuem nó em comum a lista de localizações de entrada será a mesma de saída, conforme vetores de entrada e saída da Figura 5.12.

Quarto – Com a lista de combinação ordenada, o primeiro elemento da lista é utilizado para o processamento da junção espacial, desde que o mesmo não esteja sendo utilizado. No exemplo da Figura 5.12, o processamento será atribuído para o nó BGD01.

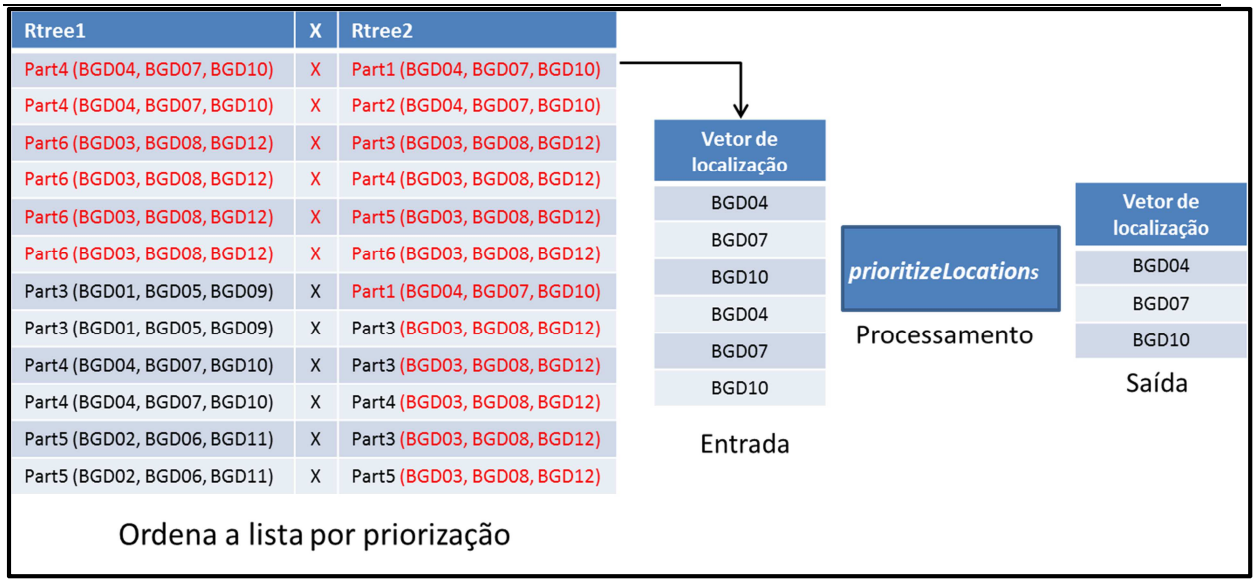


Figura 5.12 – Lista de combinação sem priorização com o vetor de localicação para o processamento *prioritizeLocations* original do *SpatialHadoop*.

Observando a Figura 5.12, nota-se, conforme lista de combinação, sem priorização, os próximos *maps* a serem escalonados não priorizam a alocação conjunta dos dados espaciais relacionados. Por exemplo, a primeira linha a ser processada que é a interseção das **Part3** com arquivo **Rtree1** com a **Part1** do Arquivo **Rtree2** que não possuem dados relacionados alocados de forma conjunta, entretanto ela é a primeira a ser escalonado para o processamento. Por outro lado, as linhas da Figura 5.12, em amarelo, que possuem dados relacionados alocados de forma conjunta (iniciando em **Part6** do arquivo **Rtree1** com interseção com **Part4** do arquivo **Rtree2**) serão as ultimas as serem escalonadas para o processamento. Diante deste contexto, os conjuntos alocados poderão ser escalonados para nós onde não existem os dados, conforme pode-se observar nas setas vermelhas da Figura 5.13. A consequência desta política é que o tempo de processamento para a alocação conjunta dos dados espaciais relacionados pode piorar devido ao custo quadrático de transferência dos dados do nó de armazenamento para o nó de processamento.

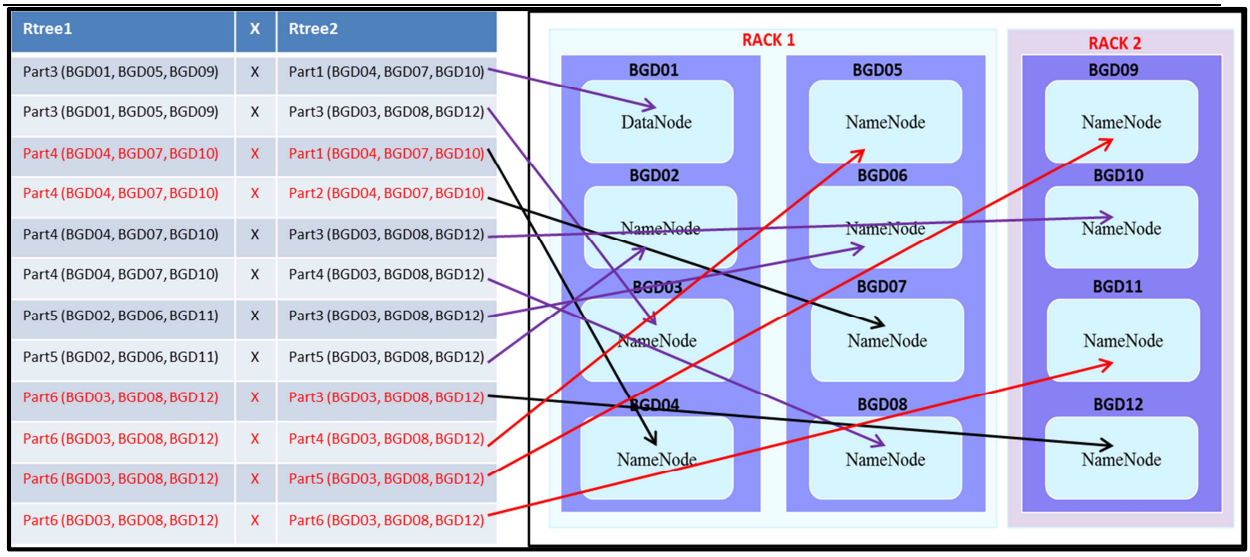


Figura 5.13 – Mapa de escalonamento da lista de combinação sem priorização.

A Figura 5.14 destaca a rotina onde as localizações dos blocos são ordenadas de forma decrescente pela frequência.

```java
public static String[] prioritizeLocations(Vector<String> vlocations) {
  Collections.sort(vlocations);
  @SuppressWarnings("unchecked")
  Vector<String>[] locations_by_count = new Vector[vlocations.size()+1];

  int unique_location_count = 0;
  int first_in_run = 0;
  int i = 1;
  while (i < vlocations.size()) {
    if (vlocations.get(first_in_run).equals(vlocations.get(i))) {
      i++;
    } else {
      // End of run
      unique_location_count++;
      int count = i - first_in_run;
      if (locations_by_count[count] == null) {
        locations_by_count[count] = new Vector<String>();
      }
      locations_by_count[count].add(vlocations.get(first_in_run));
      first_in_run = i;
    }
  }
  // add last run
  unique_location_count++;
  int count = i - first_in_run;
  if (locations_by_count[count] == null) {
    locations_by_count[count] = new Vector<String>();
  }
  locations_by_count[count].add(vlocations.get(first_in_run));

  String[] unique_locations = new String[unique_location_count];
  for (Vector<String> locations_with_same_count : locations_by_count) {
    if (locations_with_same_count == null)
      continue;
    for (String loc : locations_with_same_count) {
      unique_locations[--unique_location_count] = loc;
    }
  }
  if (unique_location_count != 0)
    throw new RuntimeException();
  return unique_locations;
}
```

Figura 5.14 – *FileSplitUtil.prioritizeLocations* original do *SpatialHadoop*.

Assim, o código original do algoritmo *prioritizeLocations* foi alterado para receber a lista de combinação de forma ordenada, priorizando a alocação conjunta de dados espaciais relacionados e o vetor de localizações proveniente da lista de combinação calcula a frequência de cada localização e retorna um vetor apenas com as localizações mais frequentes, conforme Figura 5.15.

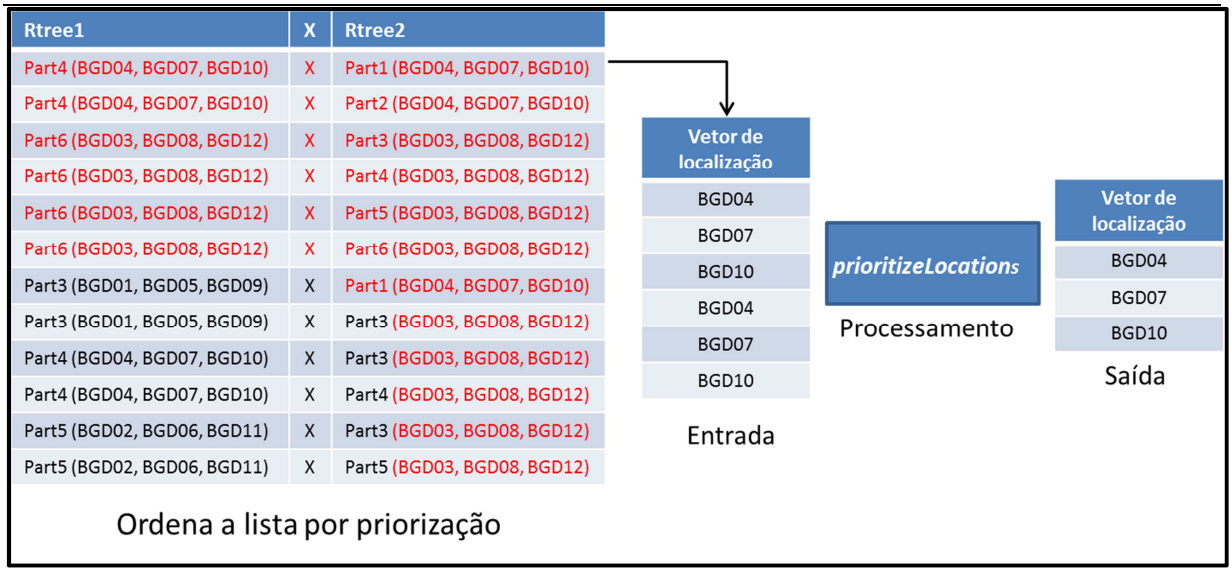


Figura 5.15 – Lista de combinação com priorização com o vetor de localicação para o processamento *prioritizeLocations* original do *SpatialHadoop*.

Observando a Figura 5.15, nota-se, que de acordo com a lista de combinação, com priorização, os próximos *maps* a serem escalonados são justamente aqueles que estão alocados de forma conjunta, os conjuntos alocados poderão ser escalonados para nós onde estão os dados, conforme pode-se observar nas setas pretas da Figura 5.16. Como resultado desta política, o tempo de processamento da junção espacial para a alocação conjunta dos dados espaciais relacionados é substancialmente melhorado.

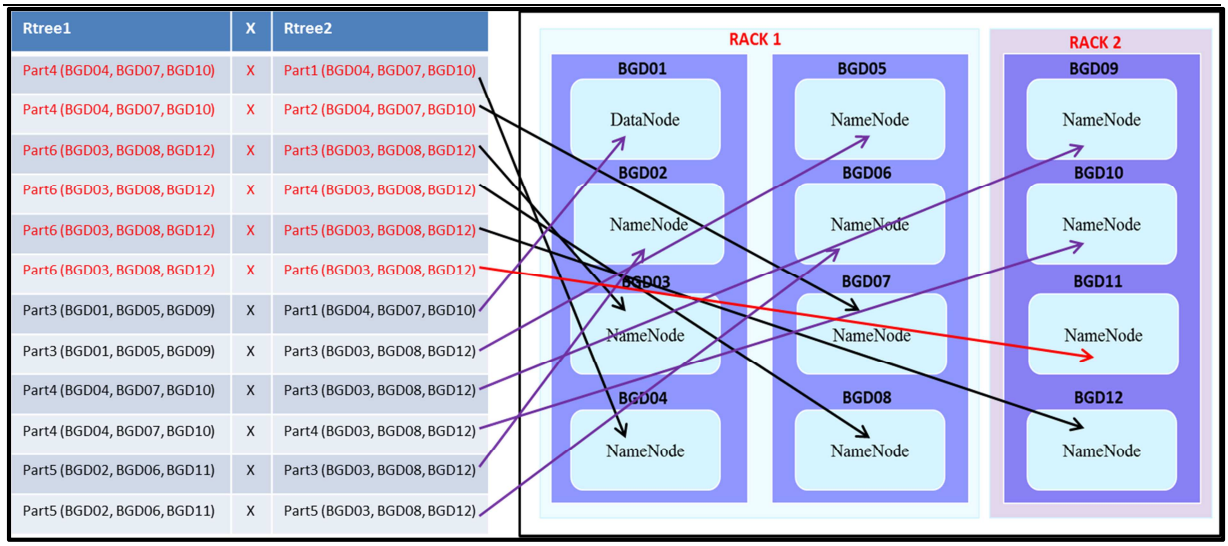


Figura 5.16 – Mapa de escalonamento da lista de combinação com priorização.

A Figura 5.17 destaca a rotina onde as localizações dos blocos são ordenados e retorna um vetor apenas com as localizações mais frequentes.

```java
public static String[] prioritizeLocations(Vector<String> vlocations) {
  Collections.sort(vlocations);
  @SuppressWarnings("unchecked")
  Vector<String>[] locations_by_count = new Vector[vlocations.size()+1];

  int unique_location_count = 0;
  int first_in_run = 0;
  int i = 1;
  while (i < vlocations.size()) {
    if (vlocations.get(first_in_run).equals(vlocations.get(i))) {
      i++;
    } else {
      // End of run
      unique_location_count++;
      int count = i - first_in_run;
      if (locations_by_count[count] == null) {
        locations_by_count[count] = new Vector<String>();
      }
      locations_by_count[count].add(vlocations.get(first_in_run));
      first_in_run = i;
    }
  }
  // add last run
  unique_location_count++;
  int count = i - first_in_run;
  if (locations_by_count[count] == null) {
    locations_by_count[count] = new Vector<String>();
  }
  locations_by_count[count].add(vlocations.get(first_in_run));

  for (int j = locations_by_count.length - 1; j >= 0; --j) {
    if (locations_by_count[j] != null)
      return locations_by_count[j].toArray(new String[0]);
  }

  throw new RuntimeException();
}
```

Figura 5.17 – *FileSplitUtil.prioritizeLocations* alterado.

Com a nova política de escalonamento de tarefas é possível alcançar um grau de utilização mais eficiente dos recursos do *cluster*, como consequência desta mudança, ocorre a redução do tráfego de rede e o custo de E/S de disco, foco de estudo deste trabalho.

5.5 Aplicação

Para auxiliar a execução dos testes de desempenho e avaliar as funcionalidades dos algoritmos de junção espacial, foi construído um protótipo de uma interface *web* para que os usuários finais do *SpatialHadoop* interajam com o

sistema. Com isso, os resultados das interações podem ser visualizados na *interface web*. Diversas operações espaciais e funções primitivas foram implementadas na interface *web*. Além das operações e funções primitivas do *SpatialHadoop,* novas funcionalidades foram implementadas para atender às exigências desta tese de doutorado, tais como, obtenção de estatísticas sobre a execução das operações de junção espacial e visualização dos dados espaciais. A Figura 5.18 ilustra a interface da aplicação web.

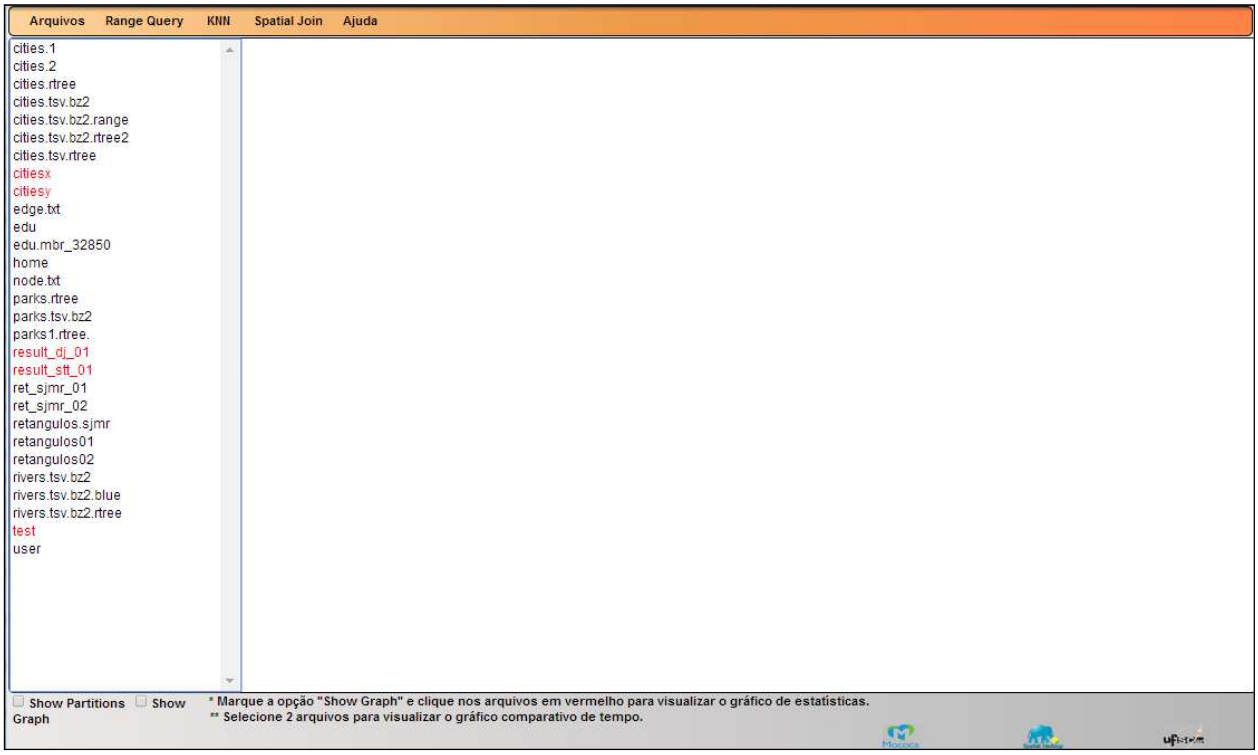


Figura 5.18 - Interface da aplicação *web*.

5.5.1 Visão geral da aplicação

A Figura 5.19 demonstra a tela principal da ferramenta de consulta e visualização, projetada para ajudar os usuários e administradores a interagir com *SpatialHadoop*.

Na parte superior, encontra-se o menu de opções, ao lado esquerdo está o bloco de dados, um painel de seleção que lista todos os arquivos do sistema. Usando as opções de adicionar ou remover, é possível carregar novos arquivos ou excluir arquivos existentes, respectivamente. Quando um arquivo é selecionado, o conteúdo deste é visualizado no bloco de resultados, como ilustra a Figura 5.19.

Quando vários arquivos são selecionados, estes são visualizados com cores diferentes, auxiliando a identificação dos mesmos. O usuário pode então usar o menu de opções para executar uma consulta por faixa (*Range Query)*, K-vizinhos mais próximos (*KNN)* e junção espacial (*Spatial Join)* com os arquivos selecionados. Após a seleção da operação, o progresso da consulta é exibido e, ao término, os resultados podem ser visualizados.

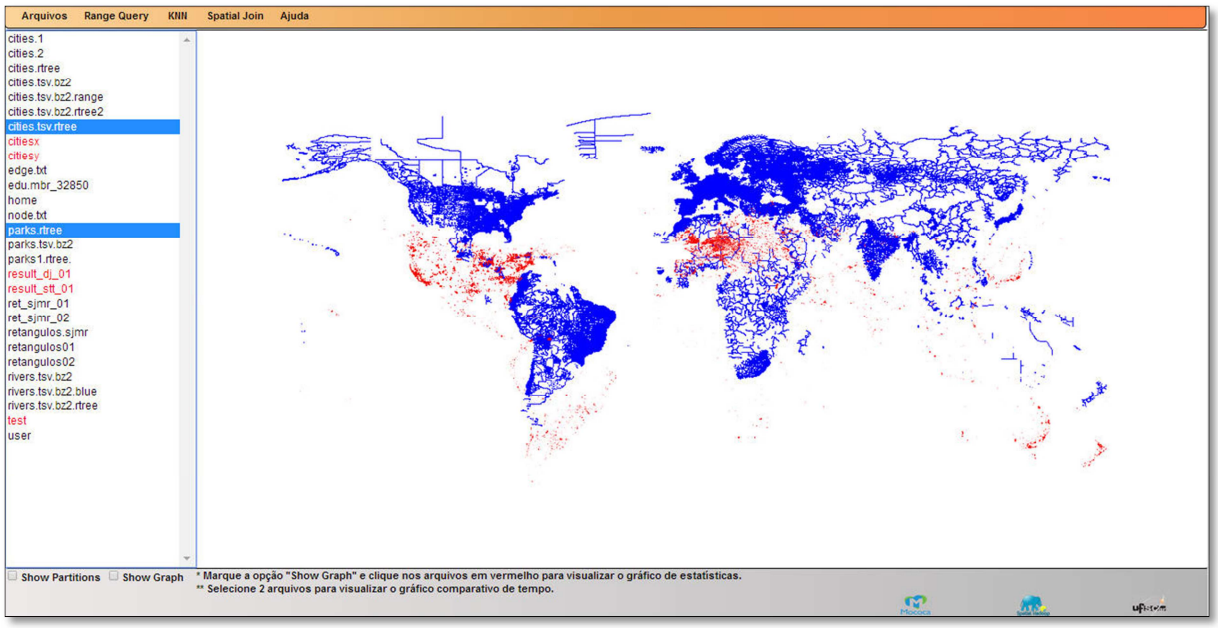


Figura 5.19 - Tela principal da aplicação.

5.5.2 Blocos de dados e resultados

Conforme ilustrado na Figura 5.20, a tela principal da aplicação é dividida em dois blocos principais chamados de: bloco de dados e bloco de resultados. No bloco de dados é possível visualizar os conjuntos de dados disponíveis para trabalho. Caso o usuário do sistema deseje inserir novos conjuntos de dados, foram implementadas funções específicas para essas finalidades. Já no bloco de resultados é possível visualizar os resultados das operações espaciais, disponíveis na aplicação. O usuário poderá selecionar duas formas de visualização dos resultados, *Show Partitions* e *Show Graph*, para visualização dos resultados em partições ou para visualizações dos gráficos de resultados das operações espaciais, respectivamente.

Na mesma tela, pode-se visualizar o menu de opções, no qual o usuário poderá selecionar uma função específica da aplicação, tais como: geração de arquivos de dados, indexação, visualização e as operações espaciais, consulta por faixa (*Range Query)*, K-vizinhos mais próximos (*KNN)* e junção espacial (*Spatial Join).*

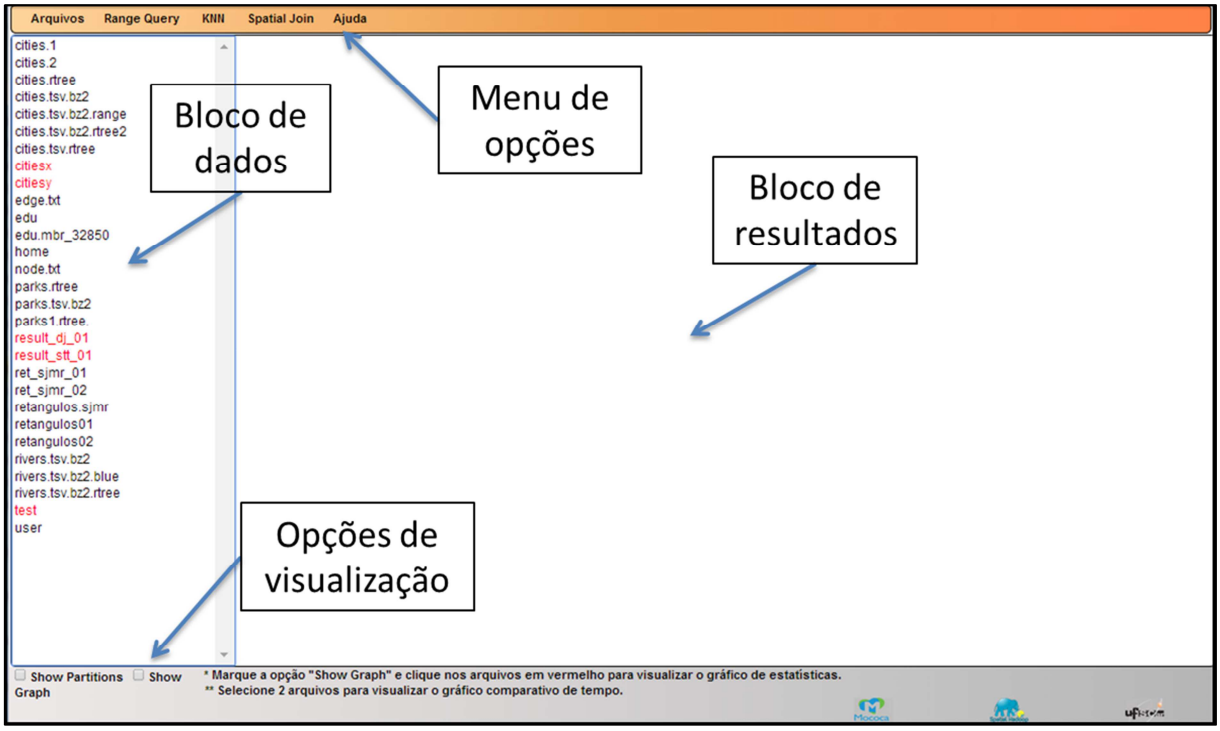


Figura 5.20 - Blocos e opções da aplicação.

5.5.3 Menu de opções

O menu de opções disponibiliza várias funções para o usuário da aplicação, tais como:

Adicionar: é possível adicionar arquivos de fontes diversas no *SpatialHadoop,* através da opção Adicionar do menu de opções. Uma janela será aberta para o usuário informar o local de origem e de destino do arquivo. Após serem adicionados, os arquivos serão disponibilizados no bloco de dados da aplicação para a execução das operações espaciais, conforme Figura 5.21.

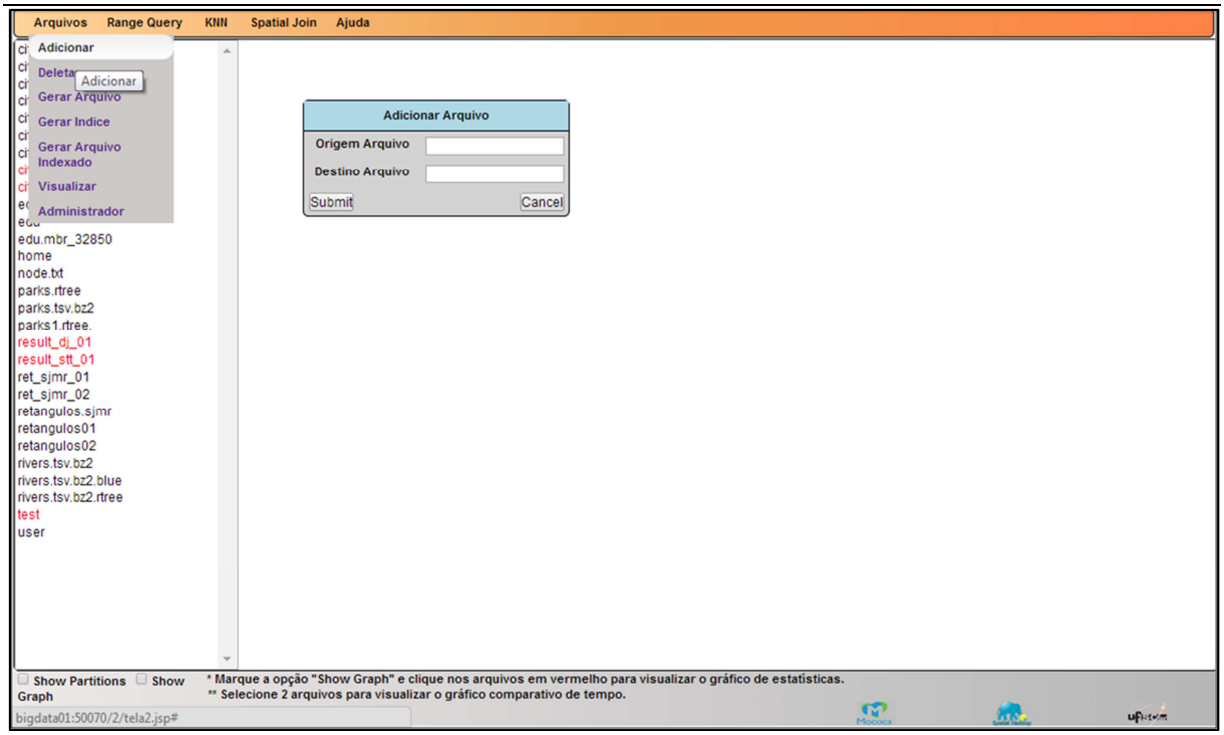


Figura 5.21 - Adicionar arquivos ao *SpatialHadoop*.

Remover: o usuário pode remover arquivos do *SpatialHadoop*. Para tanto, deve selecionar, no bloco de dados, o arquivo que deseja remover e no menu de opções, a função “Deletar”. Em seguida, o arquivo selecionado será excluído do bloco de dados.

Gerar Arquivo: arquivos de retângulos podem ser gerados no *SpatialHadoop*, selecionando a função Gerar Arquivo do menu de opções. Uma janela será aberta para o usuário fornecer os dados para a geração: nome, coordenadas do retângulo (xmin, ymin, xmax, ymax) e o tamanho, conforme visualizado na Figura 5.22. Sendo gerado o arquivo, este será disponibilizado no bloco de dados.

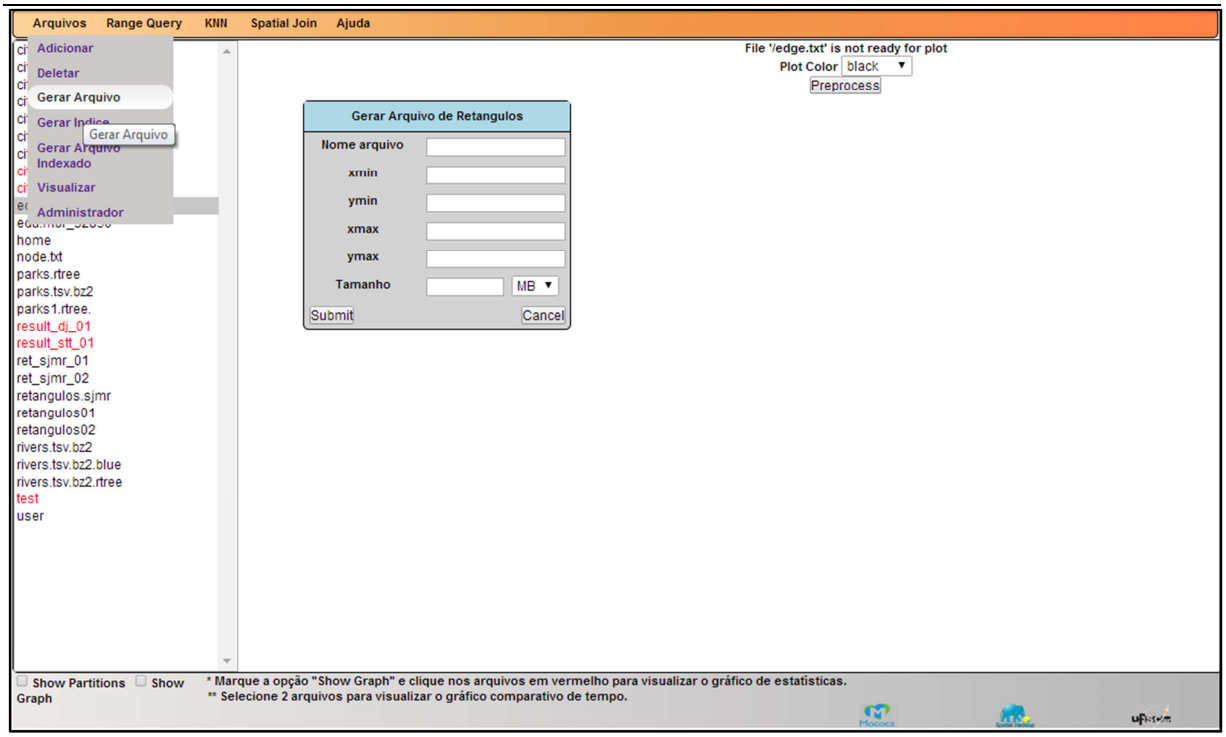


Figura 5.22 - Gerar arquivos de retângulos no *SpatialHadoop*.

Gerar índice: também é possível indexar os arquivos disponíveis no bloco de dados do *SpatialHadoop*, selecionando o arquivo no menu de dados e a função de Gerar Índice do menu de opções. A janela Indexar Arquivo será aberta para o usuário fornecer os dados de: destino, coordenadas do retângulo (xmin, ymin, xmax, ymax), *shape* e o tipo de índice (*Grid*, *R-Tree* e *R$^+$-Tree*), conforme visualizado na Figura 5.23. Após a execução da função, o arquivo indexado será disponibilizado no bloco de dados. Deve-se destacar que o algoritmo *Synchronized Tree Transversal* necessita que os dois conjuntos de dados que serão processados na junção espacial estejam indexados.

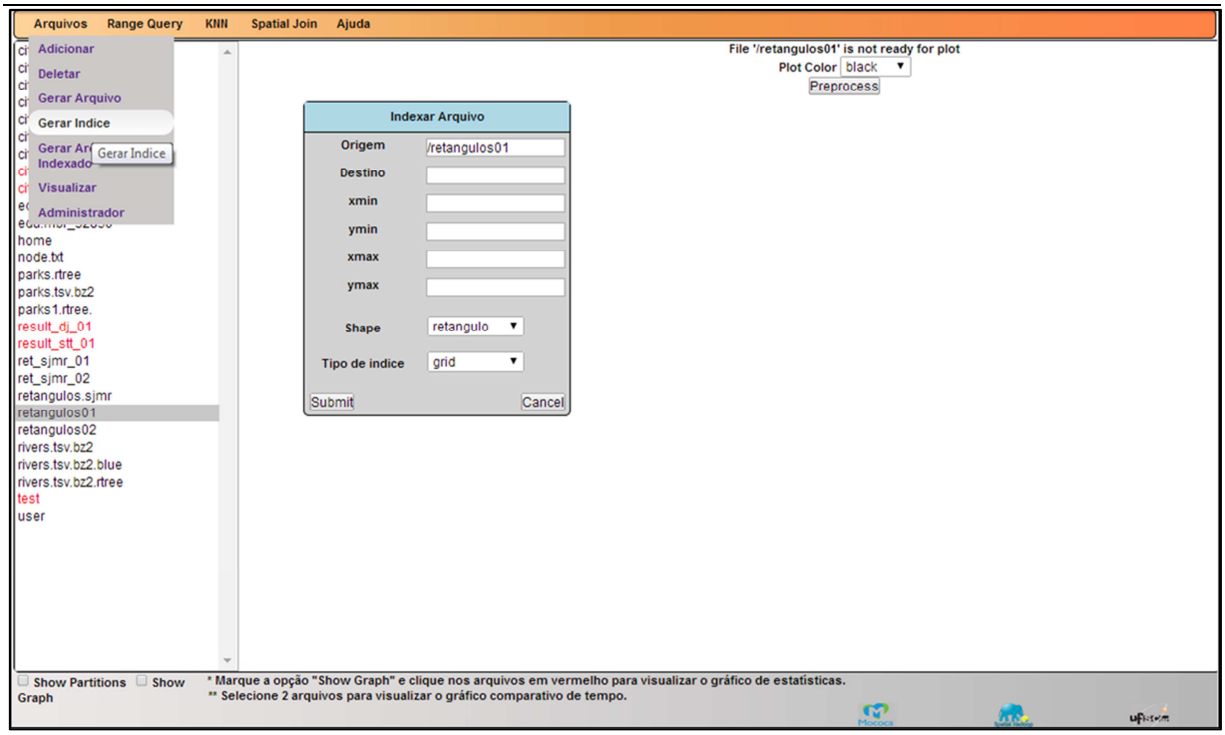


Figura 5.23 - Indexar arquivos no *SpatialHadoop*.

Gerar arquivo indexado: o usuário poderá gerar um arquivo indexado no bloco de dados do *SpatialHadoop*, selecionando a função de Gerar Arquivo Indexado no menu de opções. Uma janela será aberta para o usuário fornecer os dados de: nome do arquivo, coordenadas do retângulo (xmin, ymin, xmax, ymax) e o tamanho do arquivo. Após a execução, o arquivo indexado será disponibilizado no bloco de dados. As opções anteriores, gerar arquivo e gerar índice, são unidas em uma única opção, desta forma o arquivo é criado de forma indexada automaticamente, não sendo necessário fazer a indexação.

Visualizar: é possível visualizar os dados de um arquivo no bloco de dados do *SpatialHadoop*, selecionando o arquivo e a função de visualizar no menu de opções. Uma janela será aberta para o usuário confirmar e em seguida, os dados do arquivo serão visualizados no bloco de resultado.

Administrador: o usuário poderá acessar o administrador do sistema *Hadoop*, através da função Administrador do menu de opções. Uma nova janela será aberta no navegador para a administração do sistema, conforme Figura 5.24.

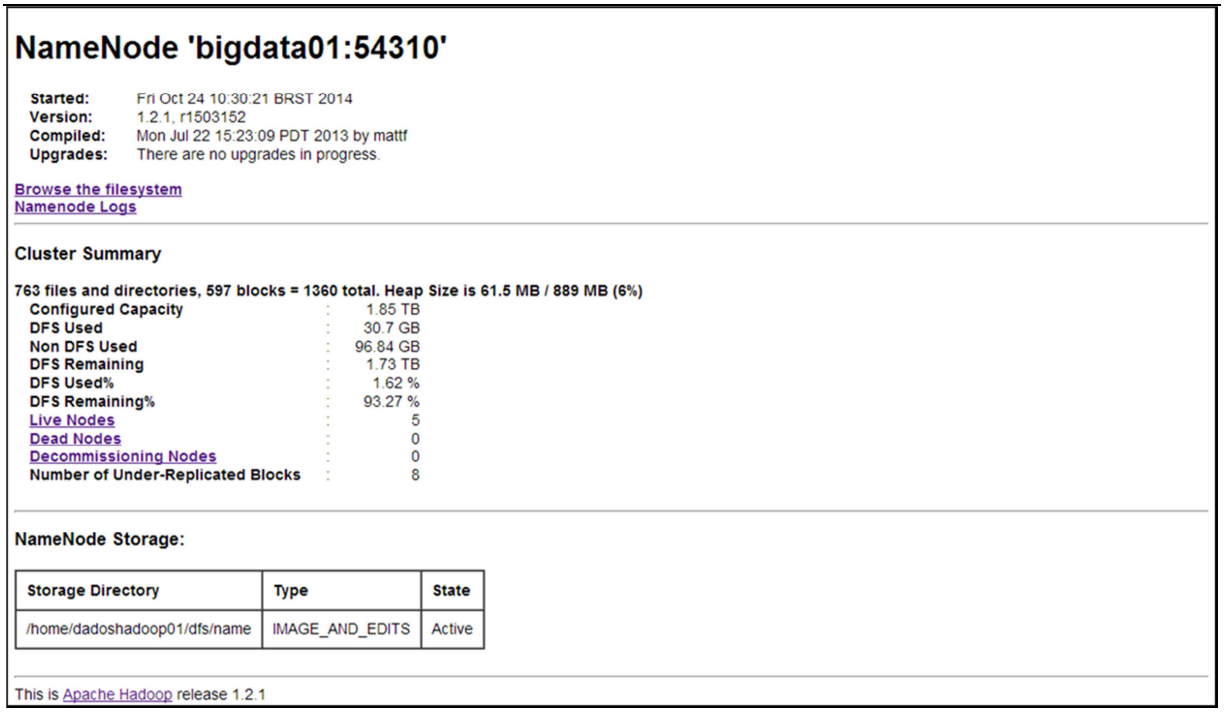


Figura 5.24 - Administrador do *Hadoop*.

Por meio do menu de opções é possível acessar as funções de operações espaciais como Consulta por faixa (*Range Query)*, K vizinhos mais próximos (*KNN)* e Junção Espacial (*Spatial Join)*, foco deste trabalho.

5.5.4 Operações espaciais

O menu de opções também disponibiliza o acesso às operações espaciais implementadas originalmente pelo *SpatialHadoop*, a saber: Consulta por faixa (*Range Query)*, K vizinhos mais próximos (*KNN)* e Junção Espacial (*Spatial Join)*, estas últimas implementadas com os seguintes algoritmos, *Synchronized Tree Transversal* (*STT*), *Spatial Join With MapReduce* (*SJMR*) e *Distributed Join* (*DJ*). Nesta seção, serão tratadas especificamente as operações de Junção Espacial, foco deste trabalho de doutorado.

***Synchronized Tree Transversal* (*STT*), *Spatial Join With MapReduce* (*SJMR*) e *Distributed Join* (*DJ*):** para executar esses algoritmos, o usuário deve selecionar dois conjuntos de dados que deseje processar, no bloco de dados, e depois, através do menu de opções, selecionar o algoritmo de Junção Espacial. Assim que a opção for acionada, uma tela será aberta para que o usuário informe os parâmetros necessários para o processamento do algoritmo, conforme visualizado

na Figura 5.25. Após o processamento, um novo arquivo com o resultado será apresentado no bloco de dados.

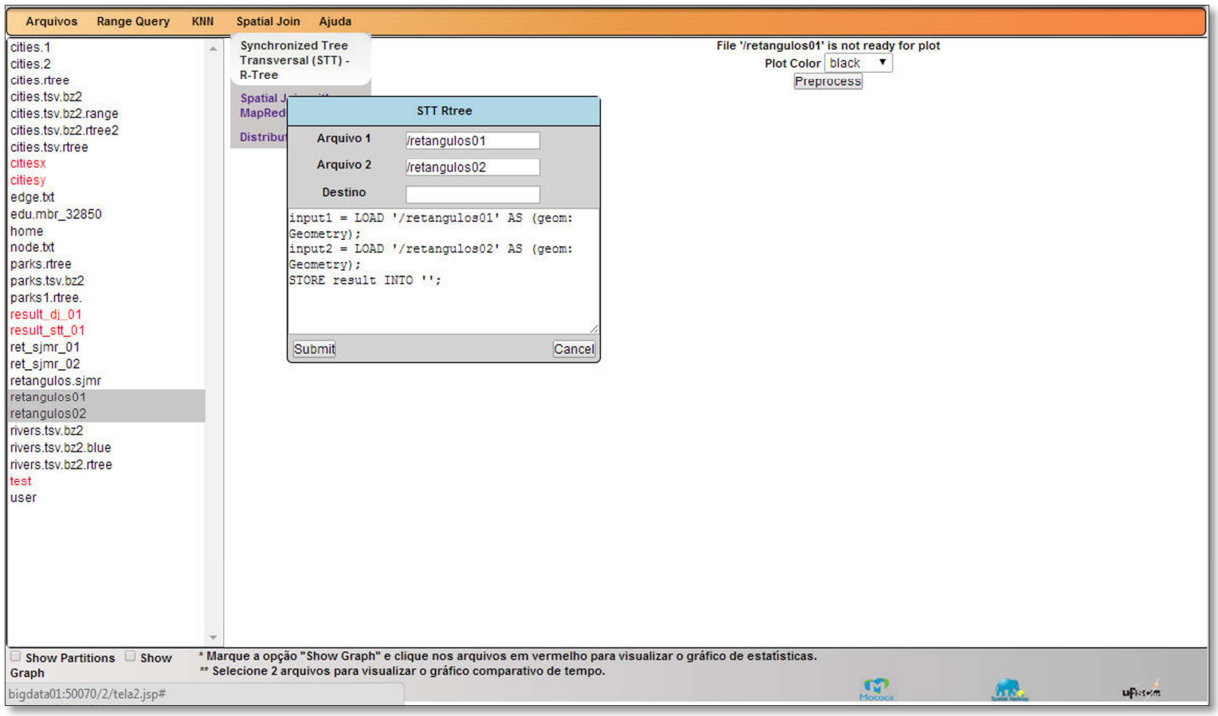


Figura 5.25 - Processamento do algoritmo *Synchronized Tree Transversal* (STT).

5.5.5 Visualização de resultados

Após o processamento de uma operação espacial, seus dados estatísticos são coletados e é possível visualizar graficamente esses dados, conforme é mostrado na Figura 5.26.

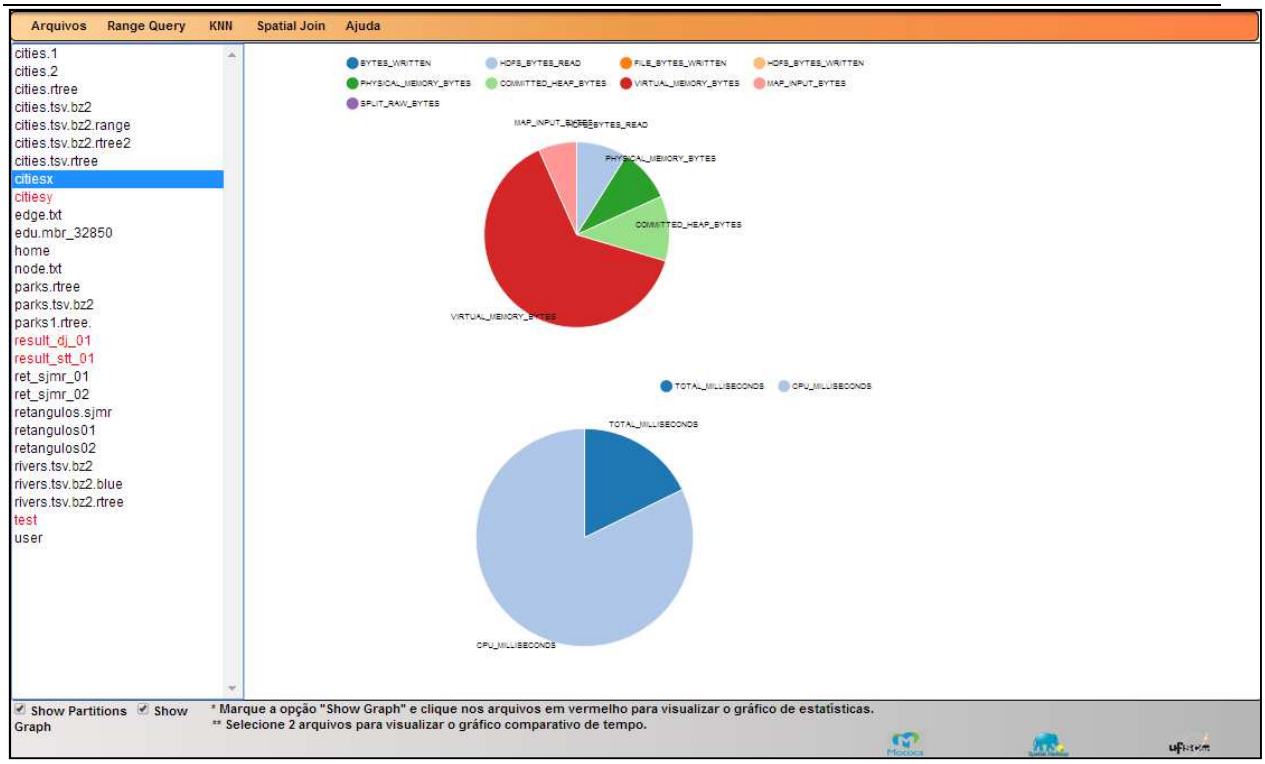


Figura 5.26 - Visualização dos dados estatísticos.

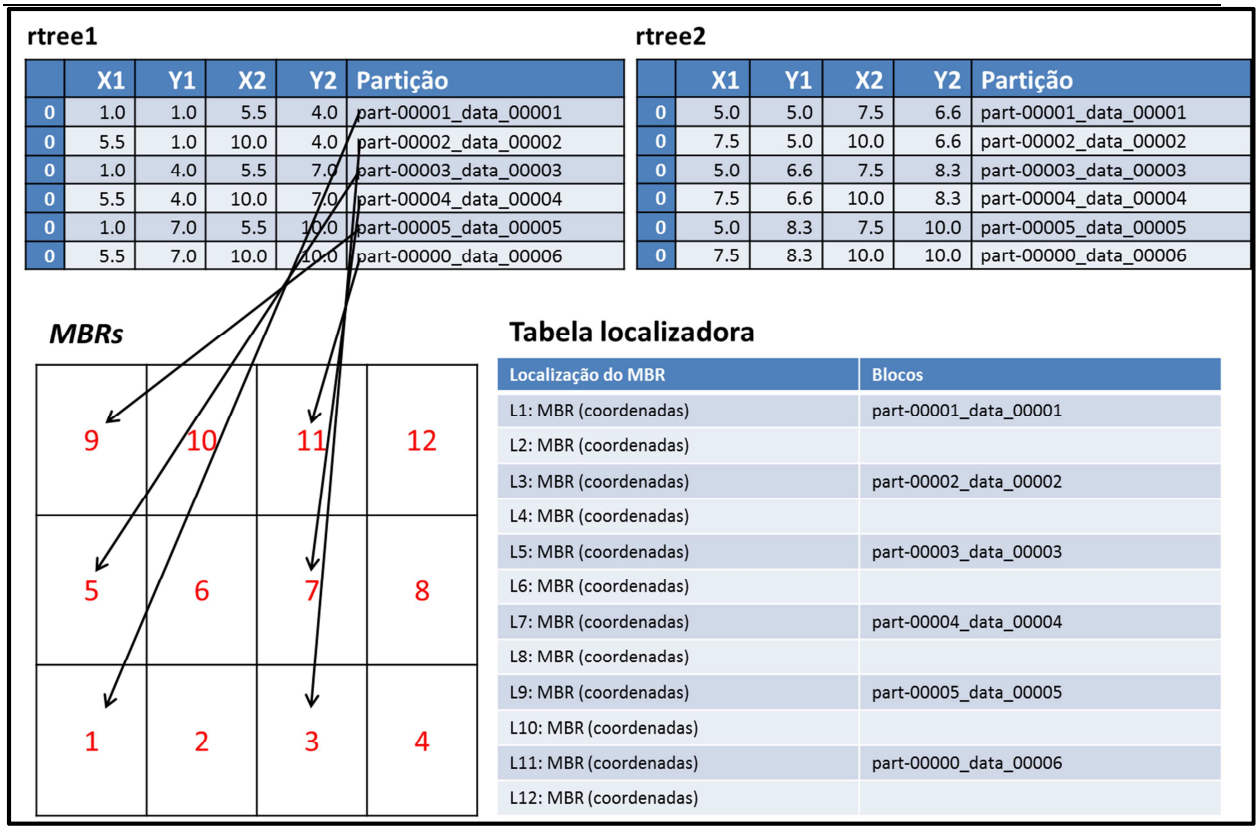
Também é possível comparar os resultados estatísticos de duas operações espaciais, conforme ilustrados da Figura 5.27.

Figura 5.27 - Comparação de resultados estatísticos.

5.6 Comparativo com o *CoS-HDFS*

Esta seção tem por objetivo comparar as principais diferenças entre as duas técnicas *Cosphdp* e *CoS-HDF*S. Para facilitar a comparação entre as técnicas foi utilizado o mesmo exemplo da Figura 5.4 para alocação e escalonamento de tarefas de dois arquivos de dados espaciais denominados **Rtree1** e **Rtree2**.

Para gerar o mapa de alocação de dados do *CoS-HDFS* em um *cluster* com 12 nós seguiu-se as regras estabelecidas na seção 4.4 desta tese, onde é definido que o melhor número de entradas da tabela localizadora é próximo ao número de nós do *cluster*. Desta forma, foi atribuído o número de nós do *cluster* para o número de entradas da tabela localizadora. Na Figura 5.28, pode-se visualizar a criação e alocação dos dados espaciais na tabela localizadora para o arquivo **Rtree1** com o número de entradas igual ao número de nós do *cluster*, ou seja, 12 entradas.



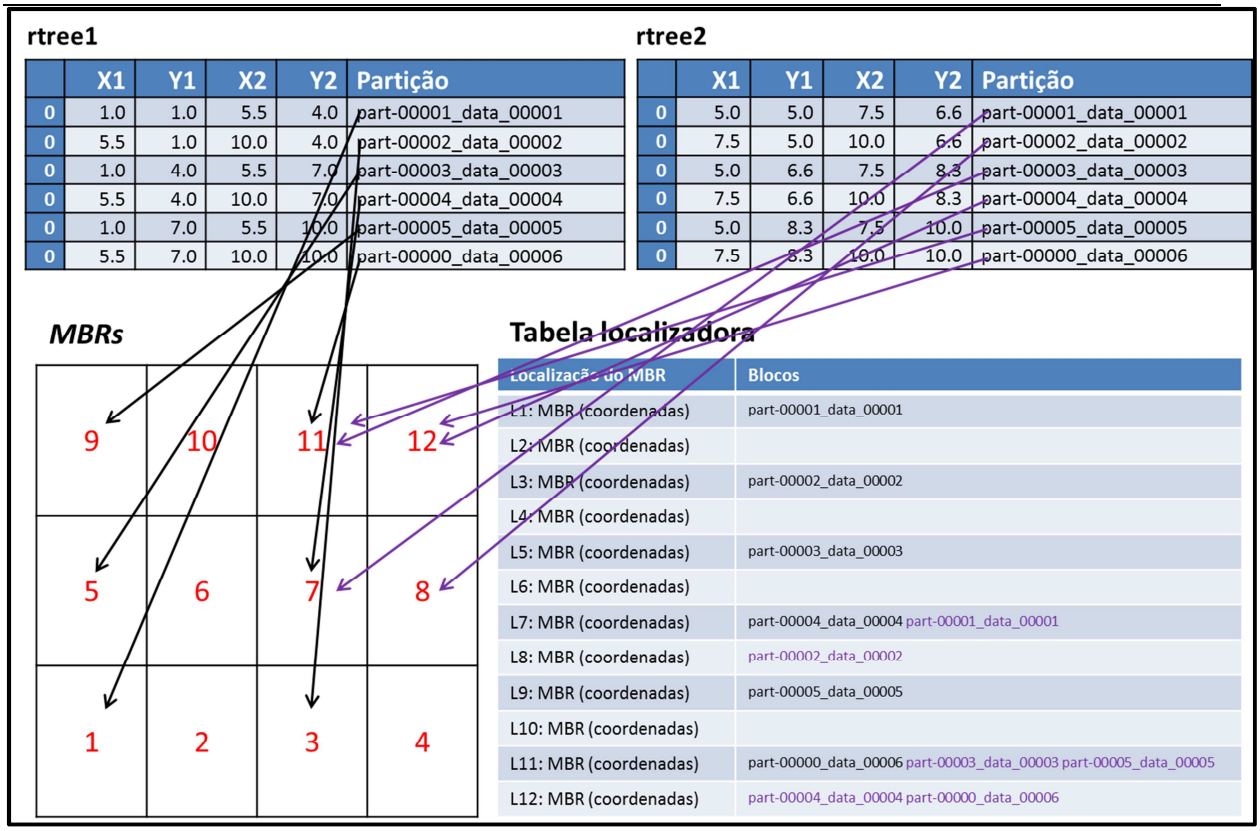
	X1	Y1	X2	Y2	Partição
0	1.0	1.0	5.5	4.0	part-00001_data_00001
0	5.5	1.0	10.0	4.0	part-00002_data_00002
0	1.0	4.0	5.5	7.0	part-00003_data_00003
0	5.5	4.0	10.0	7.0	part-00004_data_00004
0	1.0	7.0	5.5	10.0	part-00005_data_00005
0	5.5	7.0	10.0	10.0	part-00000_data_00006

	X1	Y1	X2	Y2	Partição
0	5.0	5.0	7.5	6.6	part-00001_data_00001
0	7.5	5.0	10.0	6.6	part-00002_data_00002
0	5.0	6.6	7.5	8.3	part-00003_data_00003
0	7.5	6.6	10.0	8.3	part-00004_data_00004
0	5.0	8.3	7.5	10.0	part-00005_data_00005
0	7.5	8.3	10.0	10.0	part-00000_data_00006

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001
L2: MBR (coordenadas)	
L3: MBR (coordenadas)	part-00002_data_00002
L4: MBR (coordenadas)	
L5: MBR (coordenadas)	part-00003_data_00003
L6: MBR (coordenadas)	
L7: MBR (coordenadas)	part-00004_data_00004
L8: MBR (coordenadas)	
L9: MBR (coordenadas)	part-00005_data_00005
L10: MBR (coordenadas)	
L11: MBR (coordenadas)	part-00000_data_00006
L12: MBR (coordenadas)	

Figura 5.28 – Alocação dos dados da *Rtree1* no *CoS-HDFS* com 12 entradas.

Na Figura 5.29, pode-se visualizar a alocação dos dados do arquivo **Rtree2** na tabela localizadora com 12 entradas.



	X1	Y1	X2	Y2	Partição
0	1.0	1.0	5.5	4.0	part-00001_data_00001
0	5.5	1.0	10.0	4.0	part-00002_data_00002
0	1.0	4.0	5.5	7.0	part-00003_data_00003
0	5.5	4.0	10.0	7.0	part-00004_data_00004
0	1.0	7.0	5.5	10.0	part-00005_data_00005
0	5.5	7.0	10.0	10.0	part-00000_data_00006

	X1	Y1	X2	Y2	Partição
0	5.0	5.0	7.5	6.6	part-00001_data_00001
0	7.5	5.0	10.0	6.6	part-00002_data_00002
0	5.0	6.6	7.5	8.3	part-00003_data_00003
0	7.5	6.6	10.0	8.3	part-00004_data_00004
0	5.0	8.3	7.5	10.0	part-00005_data_00005
0	7.5	8.3	10.0	10.0	part-00000_data_00006

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001
L2: MBR (coordenadas)	
L3: MBR (coordenadas)	part-00002_data_00002
L4: MBR (coordenadas)	
L5: MBR (coordenadas)	part-00003_data_00003
L6: MBR (coordenadas)	
L7: MBR (coordenadas)	part-00004_data_00004 part-00001_data_00001
L8: MBR (coordenadas)	part-00002_data_00002
L9: MBR (coordenadas)	part-00005_data_00005
L10: MBR (coordenadas)	
L11: MBR (coordenadas)	part-00000_data_00006 part-00003_data_00003 part-00005_data_00005
L12: MBR (coordenadas)	part-00004_data_00004 part-00000_data_00006

Figura 5.29 – Alocação dos dados da *Rtree2* no *CoS-HDFS* com 12 entradas.

Após a realização da alocação dos dados espaciais pelo *CoS-HDFS,* é possível visualizar na Figura 5.30, que apenas 3 conjuntos estão alocados nos mesmos nós em comparação com 6 conjuntos obtidos pela proposta do *Cosphdp*.

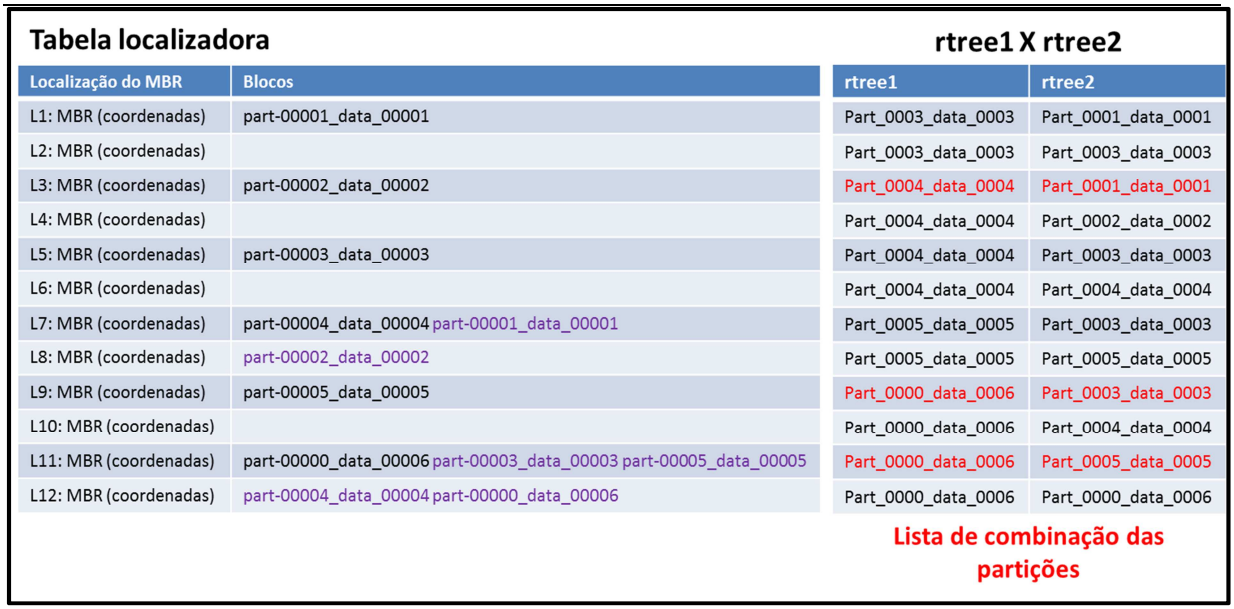


Tabela localizadora

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001
L2: MBR (coordenadas)	
L3: MBR (coordenadas)	part-00002_data_00002
L4: MBR (coordenadas)	
L5: MBR (coordenadas)	part-00003_data_00003
L6: MBR (coordenadas)	
L7: MBR (coordenadas)	part-00004_data_00004 part-00001_data_00001
L8: MBR (coordenadas)	part-00002_data_00002
L9: MBR (coordenadas)	part-00005_data_00005
L10: MBR (coordenadas)	
L11: MBR (coordenadas)	part-00000_data_00006 part-00003_data_00003 part-00005_data_00005
L12: MBR (coordenadas)	part-00004_data_00004 part-00000_data_00006

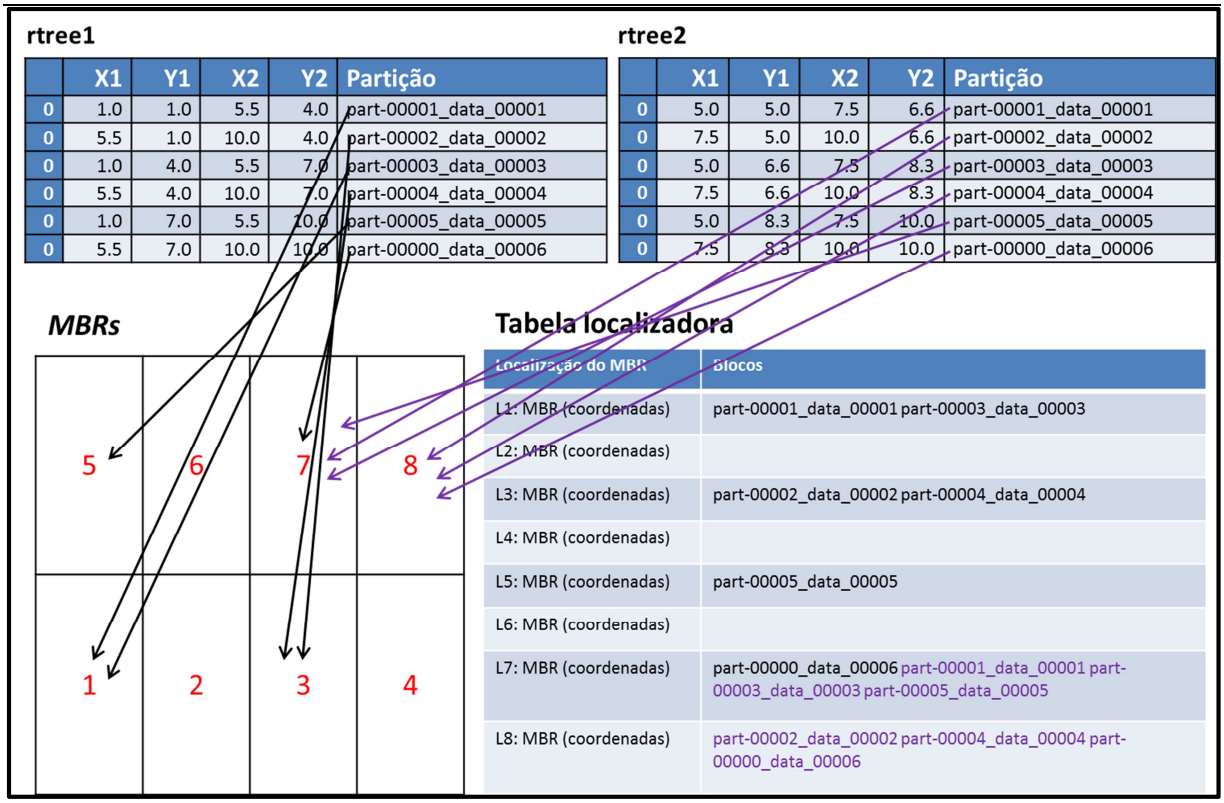
rtree1 X rtree2

rtree1	rtree2
Part_0003_data_0003	Part_0001_data_0001
Part_0003_data_0003	Part_0003_data_0003
Part_0004_data_0004	Part_0001_data_0001
Part_0004_data_0004	Part_0002_data_0002
Part_0004_data_0004	Part_0003_data_0003
Part_0004_data_0004	Part_0004_data_0004
Part_0005_data_0005	Part_0003_data_0003
Part_0005_data_0005	Part_0005_data_0005
Part_0000_data_0006	Part_0003_data_0003
Part_0000_data_0006	Part_0004_data_0004
Part_0000_data_0006	Part_0005_data_0005
Part_0000_data_0006	Part_0000_data_0006

Lista de combinação das partições

Figura 5.30 – Lista de alocação dos dados da *Rtree1* x *Rtree2* no *CoS-HDFS* com 12 entradas.

Com o objetivo de encontrar o melhor número de entradas para a tabela localizadora, foram realizados novos testes reduzindo o número de entradas. A Figura 5.31 demonstra a alocação dos dados dos arquivos **Rtree1** e **Rtree2** com a redução do número de entradas na tabela localizadora para 8.



	X1	Y1	X2	Y2	Partição
0	1.0	1.0	5.5	4.0	part-00001_data_00001
0	5.5	1.0	10.0	4.0	part-00002_data_00002
0	1.0	4.0	5.5	7.0	part-00003_data_00003
0	5.5	4.0	10.0	7.0	part-00004_data_00004
0	1.0	7.0	5.5	10.0	part-00005_data_00005
0	5.5	7.0	10.0	10.0	part-00000_data_00006

	X1	Y1	X2	Y2	Partição
0	5.0	5.0	7.5	6.6	part-00001_data_00001
0	7.5	5.0	10.0	6.6	part-00002_data_00002
0	5.0	6.6	7.5	8.3	part-00003_data_00003
0	7.5	6.6	10.0	8.3	part-00004_data_00004
0	5.0	8.3	7.5	10.0	part-00005_data_00005
0	7.5	8.3	10.0	10.0	part-00000_data_00006

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001 part-00003_data_00003
L2: MBR (coordenadas)	
L3: MBR (coordenadas)	part-00002_data_00002 part-00004_data_00004
L4: MBR (coordenadas)	
L5: MBR (coordenadas)	part-00005_data_00005
L6: MBR (coordenadas)	
L7: MBR (coordenadas)	part-00000_data_00006 part-00001_data_00001 part-00003_data_00003 part-00005_data_00005
L8: MBR (coordenadas)	part-00002_data_00002 part-00004_data_00004 part-00000_data_00006

Figura 5.31 – Alocação dos dados da *Rtree1* x *Rtree2* no *CoS-HDFS* com 8 entradas.

Após a realização da alocação dos dados espaciais pelo *CoS-HDFS* com 8 entradas, é possível visualizar na Figura 5.32, que apenas 2 conjuntos estão alocados no mesmo nó em comparação com 6 do *Cosphdp*.

Tabela localizadora

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001 part-00003_data_00003
L2: MBR (coordenadas)	
L3: MBR (coordenadas)	part-00002_data_00002 part-00004_data_00004
L4: MBR (coordenadas)	
L5: MBR (coordenadas)	part-00005_data_00005
L6: MBR (coordenadas)	
L7: MBR (coordenadas)	part-00000_data_00006 part-00001_data_00001 part-00003_data_00003 part-00005_data_00005
L8: MBR (coordenadas)	part-00002_data_00002 part-00004_data_00004 part-00000_data_00006

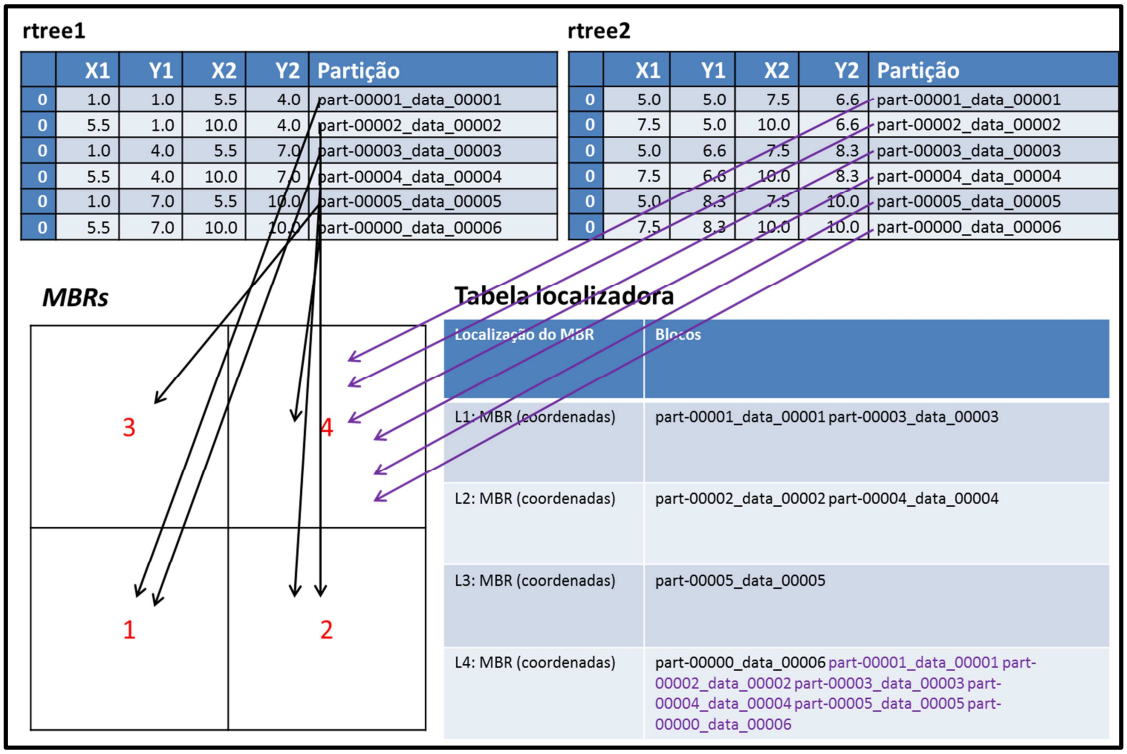
rtree1 X rtree2

rtree1	rtree2
Part_0003_data_0003	Part_0001_data_0001
Part_0003_data_0003	Part_0003_data_0003
Part_0004_data_0004	Part_0001_data_0001
Part_0004_data_0004	Part_0002_data_0002
Part_0004_data_0004	Part_0003_data_0003
Part_0004_data_0004	Part_0004_data_0004
Part_0005_data_0005	Part_0003_data_0003
Part_0005_data_0005	Part_0005_data_0005
Part_0000_data_0006	Part_0003_data_0003
Part_0000_data_0006	Part_0004_data_0004
Part_0000_data_0006	Part_0005_data_0005
Part_0000_data_0006	Part_0000_data_0006

Lista de combinação das partições

Figura 5.32 – Lista de alocação dos dados da *Rtree1* x *Rtree2* no *CoS-HDFS* com 8 entradas.

Novamente, na busca do melhor número de entradas da tabela localizadora foram realizados novos testes. A Figura 5.33 demonstra a alocação dos dados dos arquivos **Rtree1** e **Rtree2** com a redução do número de entradas na tabela localizadora para 4.



	X1	Y1	X2	Y2	Partição
0	1.0	1.0	5.5	4.0	part-00001_data_00001
0	5.5	1.0	10.0	4.0	part-00002_data_00002
0	1.0	4.0	5.5	7.0	part-00003_data_00003
0	5.5	4.0	10.0	7.0	part-00004_data_00004
0	1.0	7.0	5.5	10.0	part-00005_data_00005
0	5.5	7.0	10.0	10.0	part-00000_data_00006

	X1	Y1	X2	Y2	Partição
0	5.0	5.0	7.5	6.6	part-00001_data_00001
0	7.5	5.0	10.0	6.6	part-00002_data_00002
0	5.0	6.6	7.5	8.3	part-00003_data_00003
0	7.5	6.6	10.0	8.3	part-00004_data_00004
0	5.0	8.3	7.5	10.0	part-00005_data_00005
0	7.5	8.3	10.0	10.0	part-00000_data_00006

Figura 5.33 – Alocação dos dados da *Rtree1* e *Rtree2* no *CoS-HDFS* com 4 entradas.

Após a realização da alocação dos dados espaciais pelo *CoS-HDFS* com 4 entradas, é possível visualizar na Figura 5.34, que apenas 4 conjuntos estão alocados no mesmo nó em comparação com 6 do *Cosphdp*.

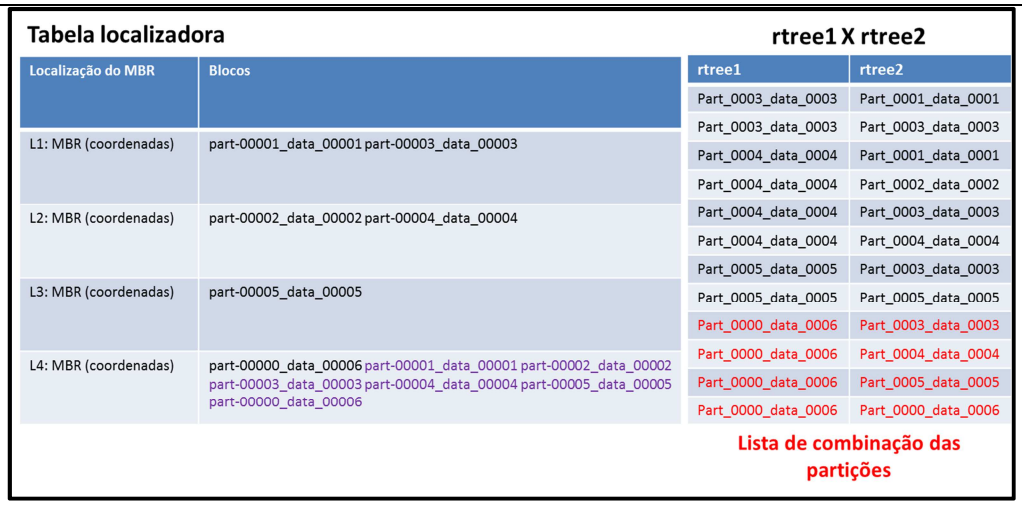


Tabela localizadora

Localização do MBR	Blocos
L1: MBR (coordenadas)	part-00001_data_00001 part-00003_data_00003
L2: MBR (coordenadas)	part-00002_data_00002 part-00004_data_00004
L3: MBR (coordenadas)	part-00005_data_00005
L4: MBR (coordenadas)	part-00000_data_00006 part-00001_data_00001 part-00002_data_00002 part-00003_data_00003 part-00004_data_00004 part-00005_data_00005 part-00000_data_00006

rtree1 X rtree2

rtree1	rtree2
Part_0003_data_0003	Part_0001_data_0001
Part_0003_data_0003	Part_0003_data_0003
Part_0004_data_0004	Part_0001_data_0001
Part_0004_data_0004	Part_0002_data_0002
Part_0004_data_0004	Part_0003_data_0003
Part_0004_data_0004	Part_0004_data_0004
Part_0005_data_0005	Part_0003_data_0003
Part_0005_data_0005	Part_0005_data_0005
Part_0000_data_0006	Part_0003_data_0003
Part_0000_data_0006	Part_0004_data_0004
Part_0000_data_0006	Part_0005_data_0005
Part_0000_data_0006	Part_0000_data_0006

Lista de combinação das partições

Figura 5.34 – Lista de alocação dos dados da *Rtree1* x *Rtree2* no *CoS-HDFS* com 4 entradas.

Duas entradas foi o melhor número encontrado durante os testes, neste caso o *CoS-HDFS* conseguiu alocar conjuntamente 8 conjuntos de blocos contra 6 do *Cosphdp*, com mostra a Figura 5.8. Mesmo que o número de entradas não seja o ideal, conforme regras definido na seção 4.4, este exemplo foi usado em comparações com o *Cosphdp*.

Comparando o mapa de alocação de dados do *Cosphdp* (Figura 5.35), em relação ao mapa do *CoS-HDFS* (Figura 5.36), nota-se que as políticas do *CoS-HDFS* conseguem maior percentual de blocos alocados conjuntamente, 8 blocos que corresponde a 66,66% dos blocos alocados contra 6 blocos que corresponde a 50% do *Cosphdp*. Entretanto, a forma de alocação do *CoS-HDFS* não permite a utilização eficiente dos recursos do *cluster.* Isto se deve pelo fato de todos os blocos do arquivo **Rtree2** estarem alocados nos mesmos nós (BGD03, BGD04 e BGD12) sobrecarregando o *cluster*, conforme mostra a Figura 5.36. No processamento da junção espacial, são gerados 12 *maps,* conforme lista de combinação da tabela 5.35 e 5.36 e os mesmos deverão ser escalonados para o maior número de nós disponíveis, a fim de alcançar o maior nível utilização dos recursos do *cluster*. Com a alocação do *CoS-HDFS,* vários nós serão agendados para a execução das tarefas sem possuir qualquer dos conjuntos de dados. Diante deste contexto, o escalonador de tarefas perde em desempenho para as novas políticas já que o *Cosphdp* permite uma melhor utilização dos recursos do *cluster*. Este problema pode ser visualizado analisando os dois mapas de escalonamento de tarefas do *Cosphdp* na Figura 5.37 e o mapa do *CoS-HDFS* na Figura 5.38.

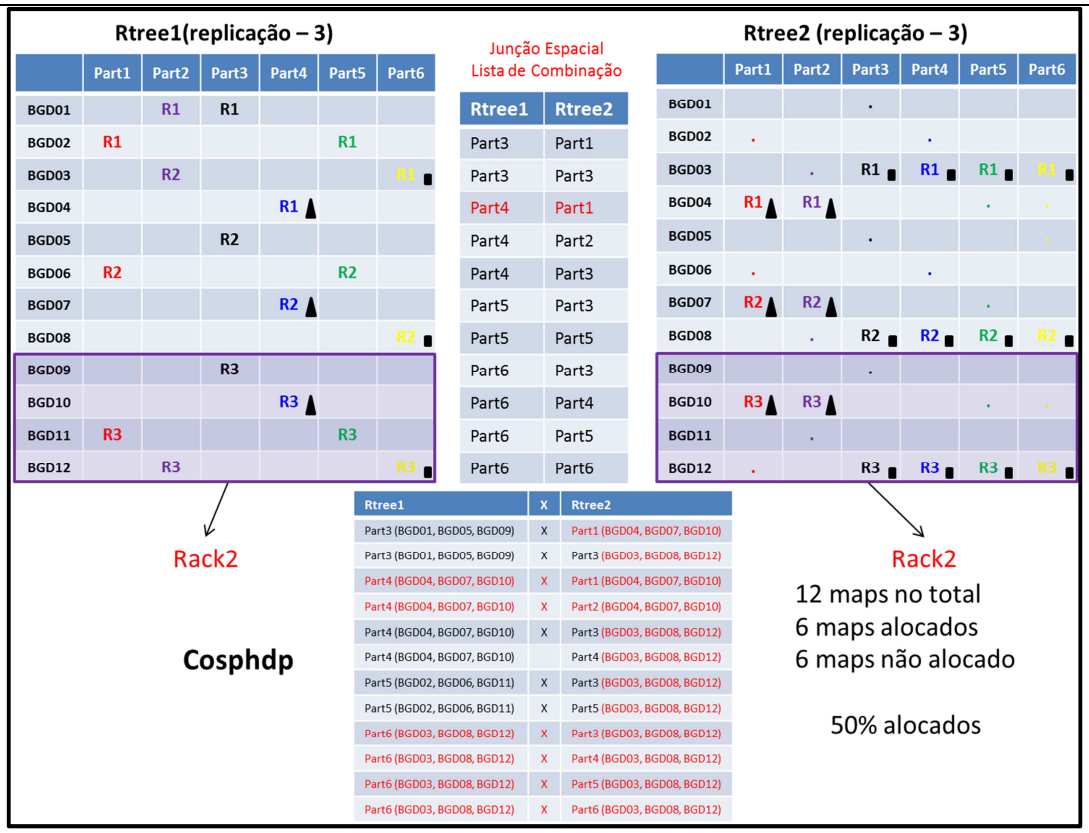


Figura 5.35 – Mapa de alocação de dados do *Cosphdp*.

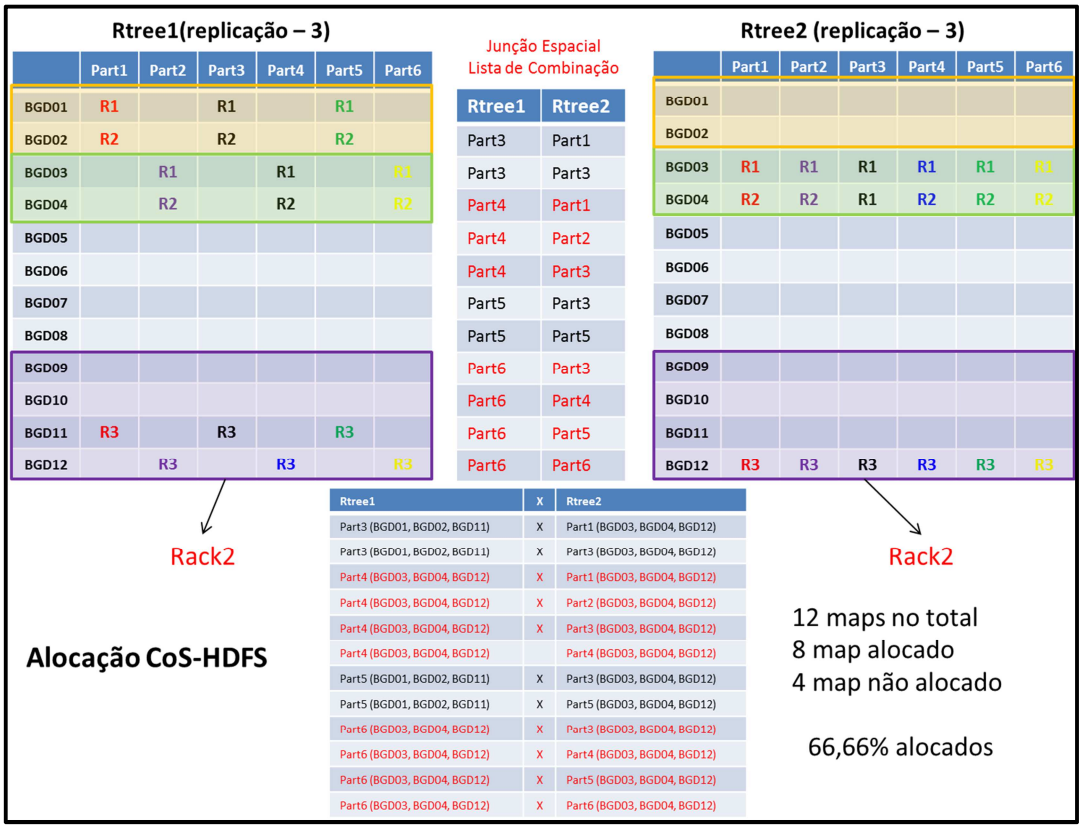


Figura 5.36 – Mapa de alocação de dados *do CoS-HDFS*.

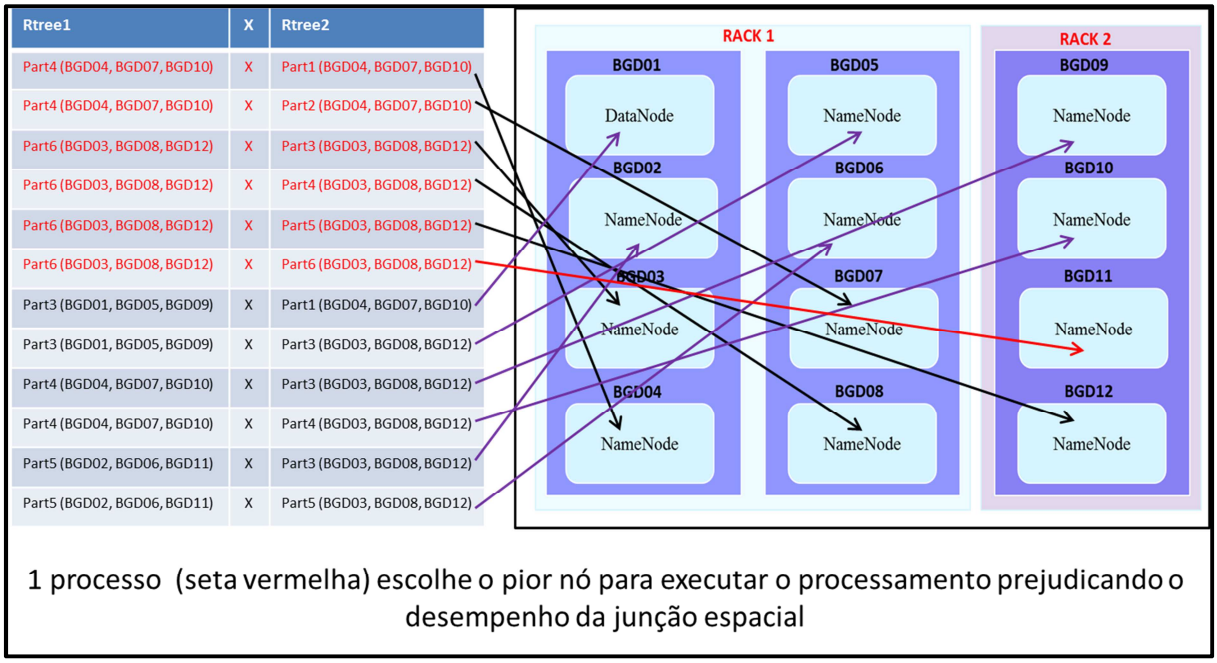


Figura 5.37 – Mapa de escalonamento de tarefas do *Cosphdp*.

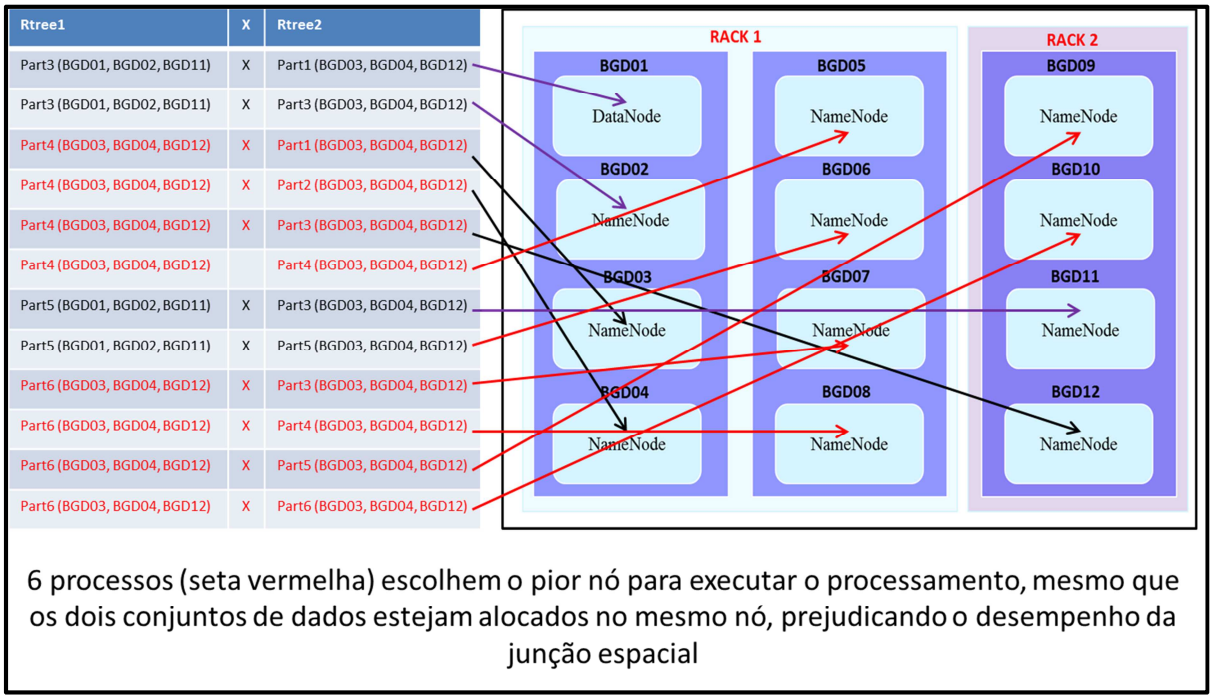


Figura 5.38 – Mapa de escalonamento de tarefas do *CoS-HDFS*.

Apesar dos objetivos serem os mesmos, que é a alocação conjunta de dados espaciais relacionados, as técnicas utilizadas são diferentes. Após avaliar as duas técnicas é possível constatar as seguintes diferenças:

Primeira: o foco do *CoS-HDFS* é em dados espaciais alocados em *cluster* geo-distribuídos em vários locais do mundo. O custo de obter os dados em *clusters* remotos é mais alto. O *Cosphdp* foca em dados alocados em *cluster* local, que representam mais a realidade de processamento paralelo e distribuído para dados espaciais. Acredita-se que o *Cosphdp* tenha um desempenho ainda melhor quando processado em *cluster* geo-distribuídos.

Segundo: para alocar os dados espaciais relacionados o *Cosphdp* primeiro faz a interseção dos blocos, executando o algoritmo de junção espacial sobre índice global dos dois conjuntos, para induzir onde os blocos relacionados deverão ser alocados, garantido que os blocos alocados de forma conjunta possua interseção. O *CoS-HDFS* processa de forma diferente, primeiro particiona as regiões utilizando os *MBRs* amplos para criar entradas na tabela localizadora e depois aloca os *MBRs* dos blocos que intersecta com os *MBRs* da tabela localizadora, desta forma não é possível garantir que os blocos alocados nos mesmos nós possuem interseção.

Terceiro: o *CoS-HDFS* não utiliza o índice global do *SpatialHadoop,* considera não ser adequado para alocar os blocos devido ao índice ser de um único nível, onde cada nó folha corresponde a um bloco de dados, gerando várias entradas na tabela localizadora. O foco é que cada entrada da tabela abranja um *MBR* mais amplo para poder armazenar vários blocos. O *Cosphdp* usa o particionamento do índice global do *SpatialHadoop* para realizar a alocação dos dados, não alterando o padrão do *SpatialHadoop.*

Quarta: as técnicas utilizadas pelo *CoS-HDFS* não atacam a política de alocação de tarefas, desta forma não há garantias de que os processos possam ser alocados nos mesmos nós que armazenam os dados.

Quinto: no intuito de manter o balanceamento de carga, a política de alocação do *CoS-HDFS,* às vezes, precisa ser desconsiderada para manter o balanceamento dos dados no *cluster*. A técnica proposta por esta tese permite desbalanceamento de carga em prol de um melhor desempenho do processamento de operações de junção espacial.

Sexto: a alocação de dados do *CoS-HDFS* sobrecarrega os nós do *cluster* devido à entrada de vários blocos nos mesmos locais. A técnica de alocação de dados do *Cosphdp* evita sobrecarregar os nós, alocando os dados de forma distribuída, com isso utiliza de forma mais eficiente os recursos do *cluster*.

O *Cosphdp* é uma solução baseada em um princípio simples e efetivo, muito fácil de ser implementado e que não altera as demais funcionalidades e funcionamento padrão do *SpatialHadoop*.

5.7 Considerações finais

Nesta seção, foram apresentadas as implementações das soluções propostas nesta tese de doutorado. Inicialmente, foi desenvolvida a nova política de alocação conjunta de dados espaciais, denominada *Cosphdp*, que permite que dados espaciais relacionados no processamento de junção espacial sejam armazenados conjuntamente nos mesmos nós de um *cluster*. Também foi apresentada a implementação do novo algoritmo de escalonamento de tarefas que permite aproveitar de todos os benefícios proporcionados pela nova política de alocação de dados espaciais, possibilitando que as tarefas sejam escalonadas para o mesmo nó onde foram armazenados dados espaciais relacionados. Finalmente, foi apresentada a aplicação *web* que irá auxiliar os usuários finais no trabalho com o *SpatialHadoop*. Também foi realizado uma seção comparativa entre as técnicas do *CoS-HDFS* e o *Cosphdp* com o objetivo de destacar as principais diferenças. Espera-se que, concluídas as implementações, esta tese possibilite que as operações de junção espacial sejam executadas no *SpatialHadoop* de forma mais eficiente do que as atuais operações.

No próximo capítulo serão apresentados resultados e testes de desempenho experimentais para avaliar os resultados gerados pelas novas políticas de alocação e escalonamento de tarefas para dados espaciais relacionados.

Capítulo 6

TESTES DE DESEMPENHO

Após apresentar as novas propostas de políticas de alocação de dados espaciais e de escalonamento de tarefas, o próximo passo desta tese é tratar da análise e interpretação dos resultados de desempenho obtidos nos testes experimentais, visando validar a eficiência das políticas propostas e concluir sobre as hipóteses a serem provadas ou refutadas.

6.1 Considerações iniciais

As novas políticas de alocação conjunta de dados espaciais relacionados e a estratégia de escalonamento de tarefas *MapReduce* para dados espaciais relacionados alocados de forma conjunta foram validadas por meio de testes de desempenho experimentais realizados com dados reais e dados sintéticos. Os testes de desempenho tiveram como foco principal comparar o tempo de processamento das novas políticas em relação à política implementada nativamente pelo *SpatialHadoop* e com relação ao trabalho correlato *CoS-HDFS* que constitui atualmente o estado da arte. Ao final deste capítulo, após análise dos testes de desempenho, as hipóteses definidas nesta tese de doutorado descritas na Seção 1.4 serão confirmadas ou refutadas.

6.2 Ambiente de teste

Para a realização dos testes de desempenho foram utilizadas três arquiteturas de *cluster* de computadores. As principais diferenças em cada arquitetura são com relação ao número de nós presentes no *cluster* e a quantidade de *swtichs* de rede que conecta os nós ao *cluster* para a formação de 1 *rack* ou 2 *racks*. Todas as arquiteturas são compostas por nós homogêneos modelo HP SFF 6200, com processador Intel Core i5 3.10 GHZ, memória RAM de 8 GB e armazenamento secundário de 500GB. Os nós foram conectados por meio de redes *Ethernet* 1.0 Gbps. O sistema operacional utilizado é o *Linux Fedora* versão 20, onde foi instalado o *Hadoop* versão 1.2.1, também foi utilizado o Java versão 1.6.0. Todas as configurações padrão do *Hadoop* foram mantidas. O número de réplicas para blocos de dados foi definido como três e o tamanho do bloco padrão foi alterado durante os testes. Na sequência, serão detalhadas as configurações das três arquiteturas.

A primeira arquitetura, denominada AC1, é constituída por seis nós homogêneos conectadas em uma *swtich* de rede. Esta arquitetura visa investigar o efeito local das políticas propostas em um ambiente local de *cluster* de computadores. Na Figura 6.1, pode-se visualizar o esquema da arquitetura AC1.

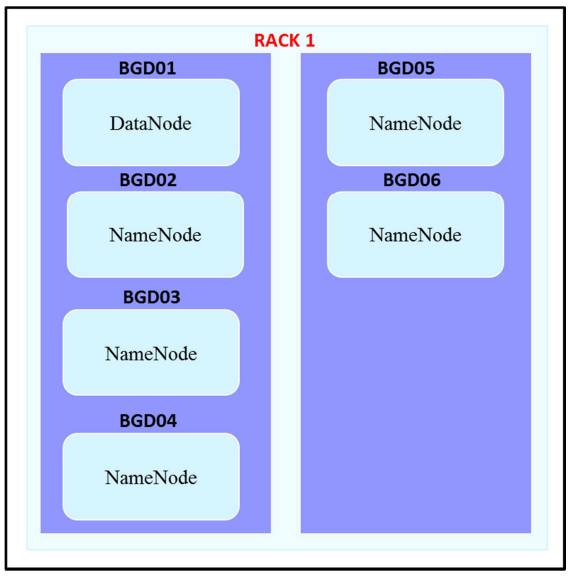


Figura 6.1 - Arquitetura AC1 com seis nós homogêneos conectados a uma *swtich* de rede.

A segunda arquitetura, denominada AC2, estende a AC1 aumentando os números de nós para avaliar o impacto do aumento do número de nós na alocação conjunta dos dados espaciais relacionados. Na Figura 6.2, pode-se visualizar o esquema da arquitetura AC2.

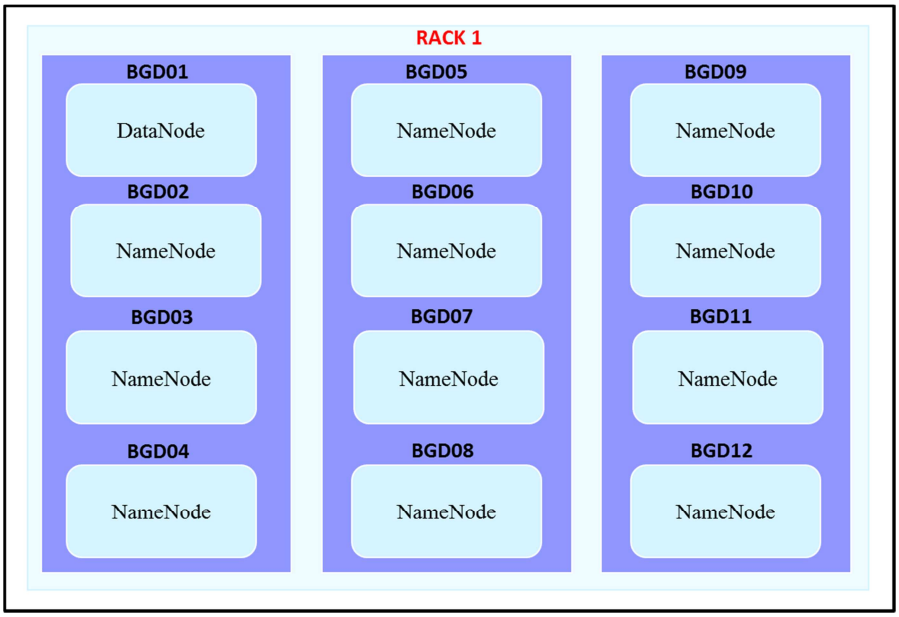


Figura 6.2 - Arquitetura AC2 com doze nós homogêneos conectados a uma *swtich* de rede.

A terceira arquitetura, denominada AC3, visa reproduzir um cenário mais real formado por vários *clusters* que se comunicam. A finalidade de alterar a arquitetura de rede é para medir a influência que o segundo *rack*, com custo quadrático, pode produzir nas novas políticas de alocação de dados relacionados em relação à política padrão do *SpatialHadoop* e do *CoS-HDFS*. As discussões sobre o custo de obter os dados para o processamento dos dados em diferentes nós e *racks* estão descritos na seção 4.2. Na Figura 6.3, pode-se visualizar o esquema da arquitetura AC3.

.

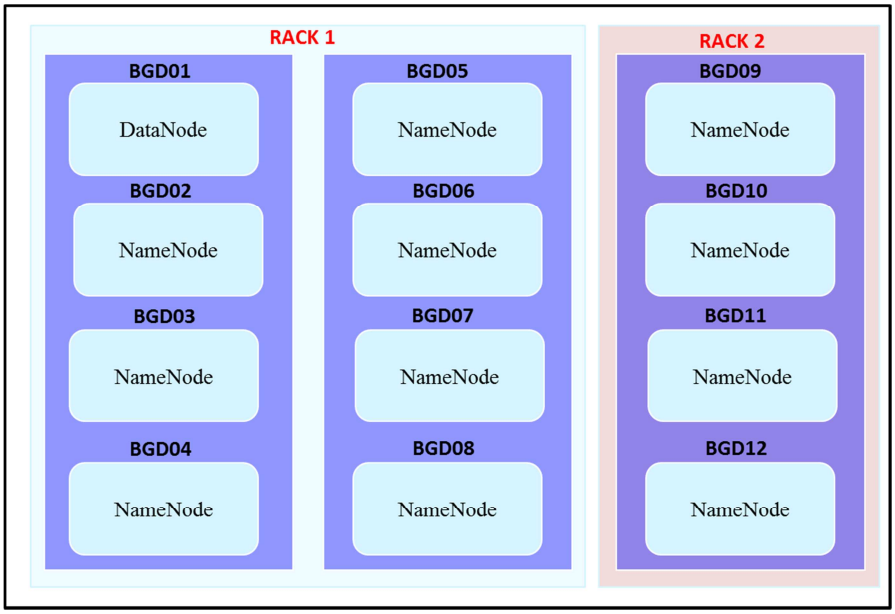


Figura 6.3 - Arquitetura AC3 com doze nós homogêneos conectados a duas *swtich* de rede.

6.2.1 Medidas de desempenho

As medidas de desempenho coletadas e posteriormente avaliadas a fim de quantificar os resultados da execução das operações de junção espacial, foram as seguintes:

Tempo de processamento: o tempo total, em milissegundos, para o processamento da junção espacial. Obtidos através da média aritmética da execução dos *scripts* das políticas. Cada *script* foi concebido para executar cem vezes cada política.

Percentual de alocação dos dados: o percentual de dados alocados. Essa medida é obtida através da divisão da quantidade de *maps* alocados pelo total de *maps,* conforme demonstrado na Figura 6.4.

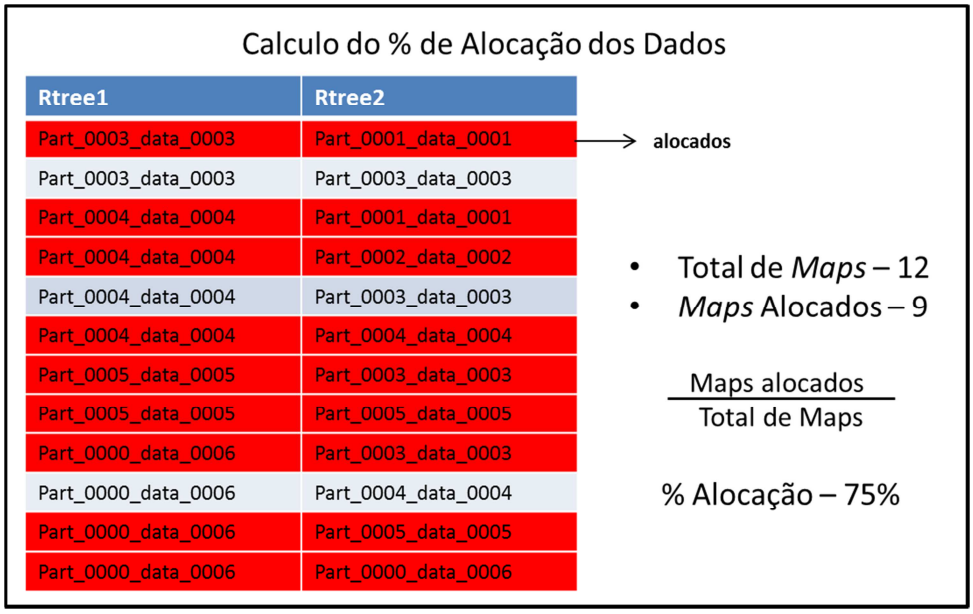


Figura 6.4 - Calculo do % de alocação dos dados.

Com base no total geral de registros gerados pela lista de combinação, no caso, 12 registros, os quais foram denominados como total de *maps.* Os registros que possuem os dois arquivos alocados no mesmo nó são denominados de alocados conjuntamente e são representados pelas linhas vermelhas da Figura 6.4. O percentual de alocação é obtido dividindo-os, no caso 9, pelo total de *maps,* que são 12, chegando ao resultado de 75% de alocados.

Desvio padrão: a média aritmética obtida do desvio padrão dos testes executados para as 100 execuções.

6.2.2 Características adicionais do ambiente de teste

Durante os testes de desempenho foram estudados os melhoramentos alcançados com a proposta da nova política *Cosphdp* e da nova política de escalonamento por meio da alteração de variáveis que afetam a eficiência das operações de junção espacial. As variáveis modificadas nos testes de desempenho foram as seguintes:

Tipos de dados: foram usados dados sintéticos e dados reais. Os dados sintéticos são polígonos com distribuição uniforme, criados pelo gerador de dados do *SpatialHadoop* com diferentes volumes de dados, variando da seguinte forma: 200

MB, 400 MB, 800 MB, 1.600 MB, 3.200 MB e 6.400 MB. Na Figura 6.5, pode-se visualizar um exemplo dos polígonos usados para os testes com dados sintéticos.

Cell #0 Rectangle: (-0.26752020000000004,49.058542499999994)-(5.9347001,62.6677811)
Cell #0 Rectangle: (21.5381913,55.5624771)-(179.9336535,74.9837666)
Cell #0 Rectangle: (-176.043354,-84.41596309999997)-(-65.24779420000002,34.411125000000006)
Cell #0 Rectangle: (-178.44359300000002,-55.04331)-(-65.039,34.57998760000001)
Cell #0 Rectangle: (5.182719,-13.534279799999998)-(13.300513,47.4322459)
Cell #0 Rectangle: (5.7455750000000005,-2.9107437)-(12.7963605,47.404393899999995)
Cell #0 Rectangle: (-124.41512519999999,34.0202574)-(-73.886652,40.80648)
Cell #0 Rectangle: (-124.134889,34.189167000000005)-(-73.88506,41.1842534)
Cell #0 Rectangle: (5.737884799999999,47.321473)-(12.568252600000001,50.1416966)
Cell #0 Rectangle: (5.8018404,47.37654479999999)-(12.4992094,50.124537200000006)
Cell #0 Rectangle: (-178.95728100000002,40.00247)-(-64.5174759,74.12826959999998)
Cell #0 Rectangle: (-179.231086,40.392422)-(-65.7934321,79.1299817)
Cell #0 Rectangle: (5.5140601,50.0482148)-(12.8364528,67.7852025)
Cell #0 Rectangle: (-67.5599855,-46.000000899999996)-(0.38387099999999996,38.452909)
Cell #0 Rectangle: (11.3747358,-34.767249)-(23.056669000000003,48.21986009999999)
Cell #0 Rectangle: (-66.00742819999999,38.270847200000006)-(-0.1413684,48.1974198)

Figura 6.5 – Dados sintéticos gerados pelo *SpatialHadoop*.

Os dados reais foram obtidos do *Tiger* (BUREAU, 2007) e do *OpenStreetMap* (CURRAN; CRUMLISH; FISHER, 2012), são polígonos, sendo 945 MB de dados de rios e 20.6 GB de rodovias, conforme Figura 6.6.

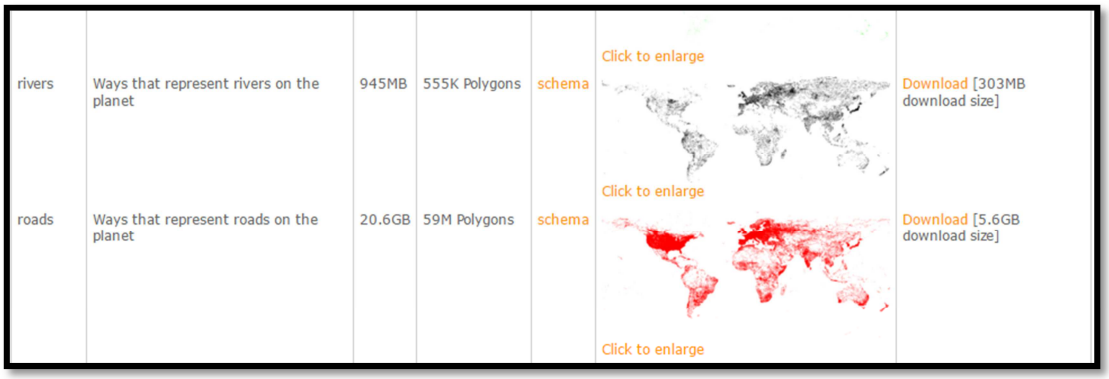


Figura 6.6 - Dados reais rios e rodovias. Reproduzida de (ELDAWY; MOKBEL, 2013).

Tamanho dos blocos: adotaram-se durante os testes os seguintes tamanhos de blocos: 32 MB, 64 MB e 128 MB. O objetivo de alterar o tamanho dos blocos foi avaliar o impacto do tráfego de rede para blocos com tamanhos diferentes e consequentemente avaliar o desempenho das operações de junção espacial. O tamanho de bloco padrão do *Hadoop* e *SpatialHadoop* é 64 MB.

Algoritmo de junção espacial: O algoritmo de junção espacial, utilizado durante os testes, foi o *Spatial Join with MapReduce (*SJMR) (ZHANG et al., 2009), implementado na versão original do *SpatialHadoop.*

6.3 Experimentos com dois *racks*

Inicialmente, os experimentos foram realizados usando dados sintéticos gerados pelo *SpatialHadoop* e alocados de forma parcial. Para obter os resultados dos experimentos, foi executada a operação de junção espacial no *SpatialHadoop,* utilizando como entrada um conjunto de dados constituídos por dois arquivos indexados por ***R-Tree****,* nomeados de: **Rtree1** e **Rtree2**. As coordenadas geográficas usadas durante os testes foram padronizadas, sendo: para **Rtree1** as coordenadas **x1=0, y1=0, x2=1.000.000 e y2=1.000.000** e para **Rtree2,** as coordenadas **x1=500.000, y1=500.000, x2=1.000.000 e y2=1.000.000**. O propósito da adoção destes limites foi criar uma área de interseção entre os dois arquivos.

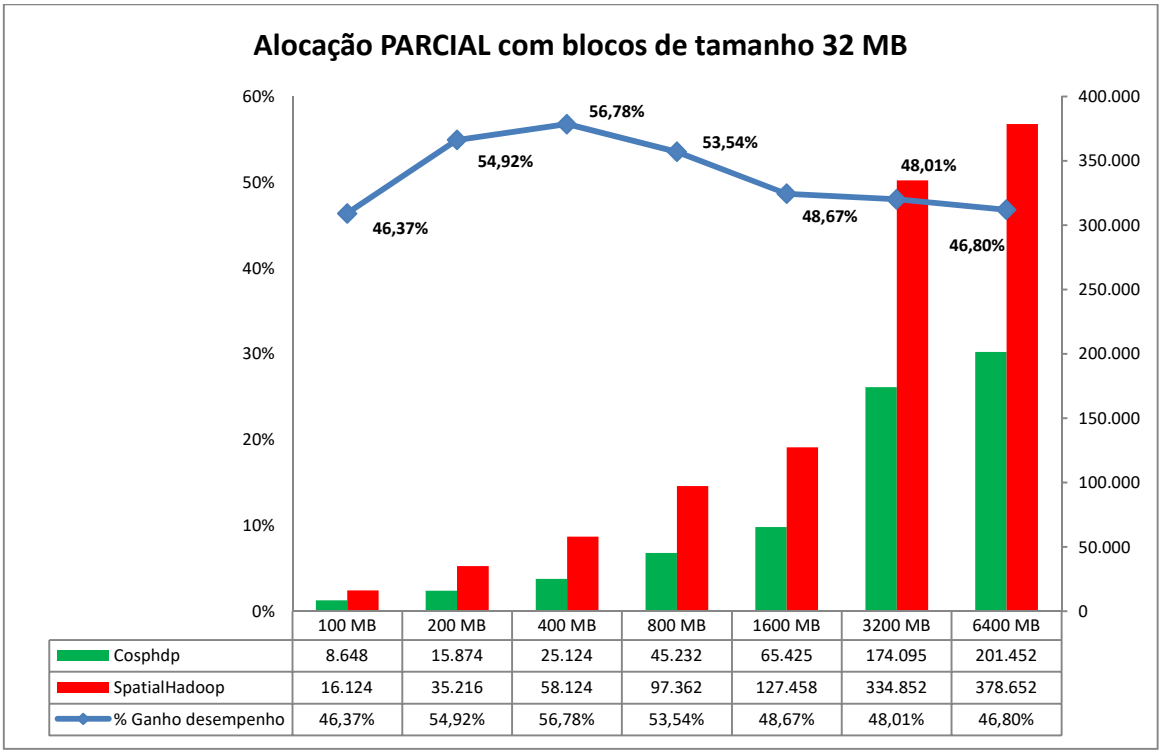
				Coordenadas				
Teste	Arquivos	Replicação	Tamanho	X1	Y1	X2	Y2	Quantidade
Teste100	Arquivo01	3	100	0	0	1.000.000	1.000.000	100
	Arquivo02	3	100	500.000	500.000	1.000.000	1.000.000	100
Teste200	Arquivo01	3	200	0	0	1.000.000	1.000.000	100
	Arquivo02	3	200	500.000	500.000	1.000.000	1.000.000	100
Teste400	Arquivo01	3	400	0	0	1.000.000	1.000.000	100
	Arquivo02	3	400	500.000	500.000	1.000.000	1.000.000	100
Teste800	Arquivo01	3	800	0	0	1.000.000	1.000.000	100
	Arquivo02	3	800	500.000	500.000	1.000.000	1.000.000	100
Teste1600	Arquivo01	3	1.600	0	0	1.000.000	1.000.000	100
	Arquivo02	3	1.600	500.000	500.000	1.000.000	1.000.000	100
Teste3200	Arquivo01	3	3.200	0	0	1.000.000	1.000.000	100
	Arquivo02	3	3.200	500.000	500.000	1.000.000	1.000.000	100
Teste6400	Arquivo01	3	6.400	0	0	1.000.000	1.000.000	100
	Arquivo02	3	6.400	500.000	500.000	1.000.000	1.000.000	100

Figura 6.7 - Parametros de testes de desempenho.

Na Figura 6.7, pode-se visualizar os diferentes parâmetros utilizados para os testes com dados sintéticos. Os parâmetros dos testes de desempenho para os testes Teste100, Teste200, Teste400, Teste800, Teste1600, Teste3200 e Teste6400, considerando a arquitetura de clusters com 2 racks AC3, onde a coluna arquivos identifica os arquivos de entrada da junção espacial, a coluna replicação indica o número de réplicas de cada bloco de dados, a coluna tamanho indica o volume de dados de cada arquivo de dados de entrada, as colunas x1, y1, x2, y2 definem o *extent* dos dados espaciais e a coluna quantidade indica o número de vezes que os testes foram executados para ser aferido a média de desempenho usada nas comparações. O tamanho do bloco foi alterado durante os testes para os seguintes valores: 32 MB, 64 MB e 128 MB. Ao final, foi calculada a média aritmética dos resultados obtidos, possibilitando uma comparação de desempenho entre as duas políticas.

6.3.1 Experimentos com blocos de 32 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com dois *racks*, utilizando dados sintéticos e blocos com tamanho de 32 MB e alocação parcial, é apresentado na Figura 6.8. De acordo com os resultados, os tempos de execução das novas políticas foram mais eficientes do que a atual política do *SpatialHadoop,* em média 50,73%, com ganho de desempenho variando de 46,37% a 56,78%. Observou-se em alguns casos, que o tempo de processamento da atual política do *SpatialHadoop* foi semelhante ao das novas políticas, devido à eventualidade do processamento ocorrer no mesmo nó onde estão localizados os dados. Este fato se deve a algumas situações esporádicas, onde a alocação aleatória do *SpatialHadoop* aloca os dois conjuntos de dados relacionados no mesmo nó do processamento, produzindo o mesmo resultado proposto pelas novas políticas. Também foi constatado que, a partir do volume de dados de 400 MB, conforme o tamanho dos arquivos aumenta o percentual de diferença do desempenho entre as duas políticas é reduzido gradativamente. Acredita-se que esta situação, se deve ao fato de que os arquivos maiores produzem uma quantidade maior de blocos, aumentando a probabilidade de serem alocados de forma relacionada no mesmo nó pela política padrão, semelhante às novas políticas.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	8.648	15.874	25.124	45.232	65.425	174.095	201.452
SpatialHadoop	16.124	35.216	58.124	97.362	127.458	334.852	378.652
% Ganho desempenho	46,37%	54,92%	56,78%	53,54%	48,67%	48,01%	46,80%

Figura 6.8 - Alocação parical para blocos de 32 MB.

A Figura 6.9 ilustra a média aritmética do desvio padrão para diferentes tamanhos de arquivos e o percentual de alocação de dados entre as duas políticas. Nota-se que, o percentual de alocação do *SpatialHadoop* aumenta conforme cresce o tamanho do arquivo. É possível concluir que, pelo fato do *cluster* de 12 nós ser considerado pequeno, a probabilidade de um arquivo de 6.400 MB, com 240 blocos para um tamanho de bloco de 32 MB, estar alocado de forma relacionada pela política padrão é muito maior e consequentemente, melhora o tempo de processamento da junção espacial para o *SpatialHadoop* com as suas políticas-padrão. Neste contexto, a política atual e as novas políticas podem obter tempo de processamento similar, reduzindo o percentual de diferença entre as políticas, conforme pode-se constatar na Figura 6.8. Analisando o intervalo gerado pelo tempo médio com desvio padrão (i.e. tempo médio – desvio padrão a tempo médio + desvio padrão), obtido pelas novas políticas para diferentes tamanhos arquivos, verifica-se que não houve interseção com o intervalo gerado pelo tempo médio com desvio padrão obtido pelo *SpatialHadoop*. Desta forma, pode-se concluir que os resultados comparativos que mostram a superioridade das novas políticas propostas com

relação ao *SpatialHadoop* não foram afetados pela variação dos resultados coletados nas 100 execuções.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	1743	2074	1542	1021	784	2541	1945
DP SpatialHadoop	945	742	1975	2346	2719	2837	1843
%AL Cosphdp	60%	68%	71%	68%	65%	74%	71%
%AL SpatialHadoop	14%	15%	18%	26%	38%	42%	47%

Figura 6.9 - Devio padrão e percentual de alocação parcial para blocos de 32 MB.

É possível visualizar, na figura 6.10, o resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop,* em uma arquitetura de *cluster* com dois *racks*, utilizando dados sintéticos, blocos com tamanho de 32 MB e **alocação total**. O tempo de execução é obtido quando ocorre a **alocação total** dos dados. De acordo com os resultados, o tempo de execução da **alocação total** é ligeiramente mais eficiente que o tempo de alocação parcial, em média 51,71% para a **alocação total**, contra 50,73% da alocação parcial. O pequeno ganho em desempenho pode não compensar devido ao aumento desproporcional em relação à necessidade de armazenamento dos dados, mais de 30% em comparação com a alocação parcial, provocado pela alteração do fator de replicação dos blocos, podendo estes passarem de três réplicas para até 12 réplicas, alocadas para cada bloco do arquivo. A quantidade de réplicas pode atingir um tamanho máximo de 12, que corresponde ao número máximo de nós do *cluster*. Quanto ocorre este fato, pode-se afirmar que o bloco está totalmente alocado no *cluster*.

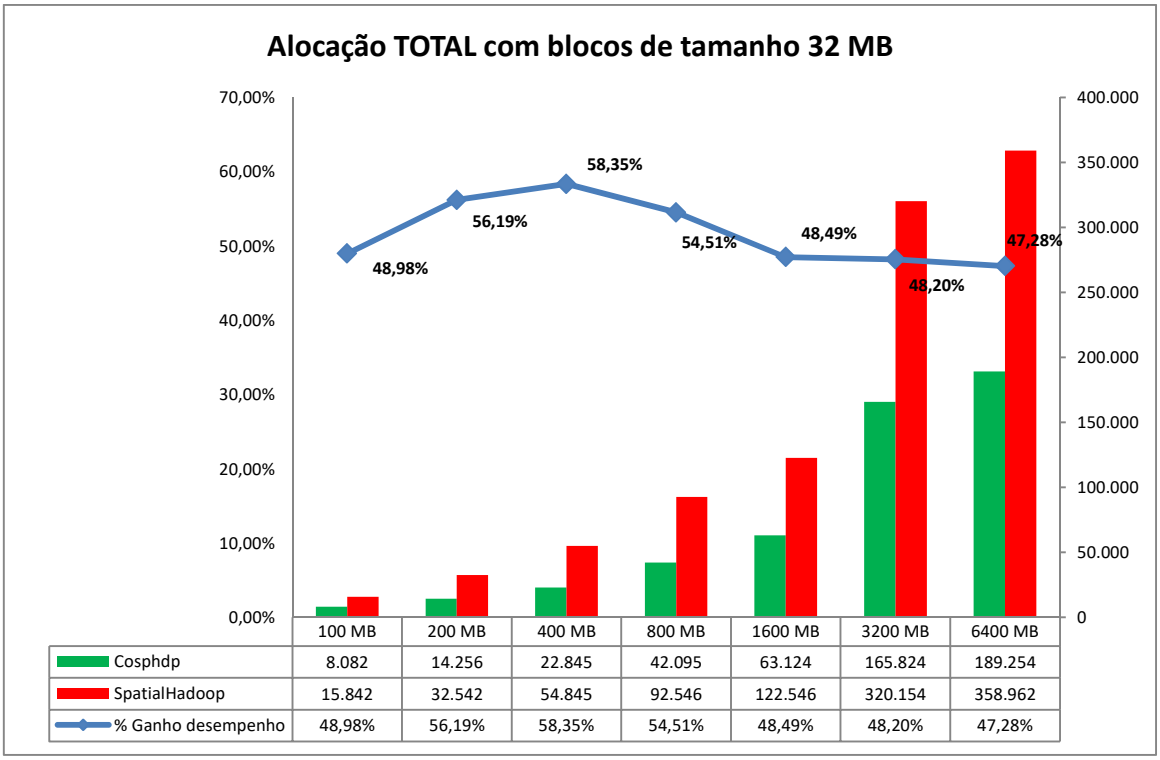


Figura 6.10 - Alocação total para blocos de 32 MB.

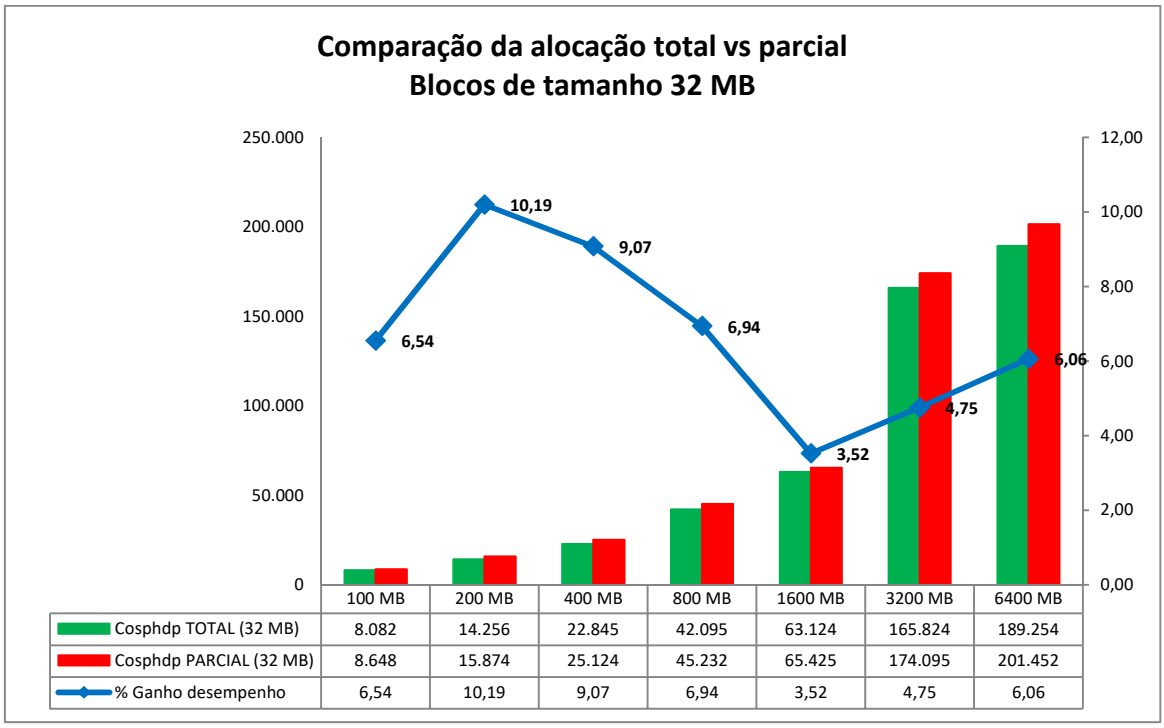
A Figura 6.11 exibe o desvio padrão médio das diferentes políticas e o percentual de alocação de dados. Neste contexto, as novas políticas estão com 100% de alocação dos dados; já a política do *SpatialHadoop* alcança um maior percentual de alocação quando o conjunto de dados cresce.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	1025	1232	1075	785	1824	1874	1748
DP SpatialHadoop	1145	1425	1024	978	1244	1425	1925
%AL Cosphdp	100%	100%	100%	100%	100%	100%	100%
%AL SpatialHadoop	15%	14%	16%	12%	21%	27%	28%

Figura 6.11 - Devio padrão e percentual de alocação total para blocos de 32 MB.

A comparação entre as duas diferentes formas de alocação: parcial ou total pode ser visualizada na Figura 6.12. Observa-se que a **alocação total** é ligeiramente mais eficiente em termos de tempo de processamento do que a alocação parcial. Todavia, esta situação fica comprometida quando se analisa a quantidade de espaço utilizado para realizar a **alocação total** dos dados em relação à alocação parcial. O espaço da **alocação total** é em média 30% superior ao espaço

utilizado na alocação parcial. Além do aumento substancial no espaço utilizado para o armazenamento, o fator de replicação também é alterado devido à necessidade do algoritmo obter a **alocação total** dos dados.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp TOTAL (32 MB)	8.082	14.256	22.845	42.095	63.124	165.824	189.254
Cosphdp PARCIAL (32 MB)	8.648	15.874	25.124	45.232	65.425	174.095	201.452
% Ganho desempenho	6,54	10,19	9,07	6,94	3,52	4,75	6,06

Figura 6.12 - Comparação da alocação total x parcial para blocos de 32 MB.

6.3.2 Experimentos com blocos de 64 MB

Os próximos experimentos foram realizados para avaliar o desempenho da junção espacial em blocos de tamanho de 64 MB que é o tamanho padrão do *Hadoop* e do *SpatialHadoop*. Na Figura 6.13, é possível visualizar o resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com dois *racks*, utilizando dados sintéticos, blocos com tamanho de 64 MB e alocação parcial. Conforme análise dos resultados, o tempo de execução das novas políticas é em média 54,22%, com o ganho de desempenho variando de 52,35% a 56,55%, sendo mais eficiente que a atual política do *SpatialHadoop*. Outro ponto identificado é que, com o aumento do tamanho do bloco, o desempenho das novas políticas melhorou em relação à política do *SpatialHadoop*. Com os blocos com tamanho de 32 MB, obteve-se um tempo médio

de 50,73%; entretanto, para os blocos com tamanho de 64 MB, houve uma ligeira melhora, alcançando o tempo médio de 54,22%.

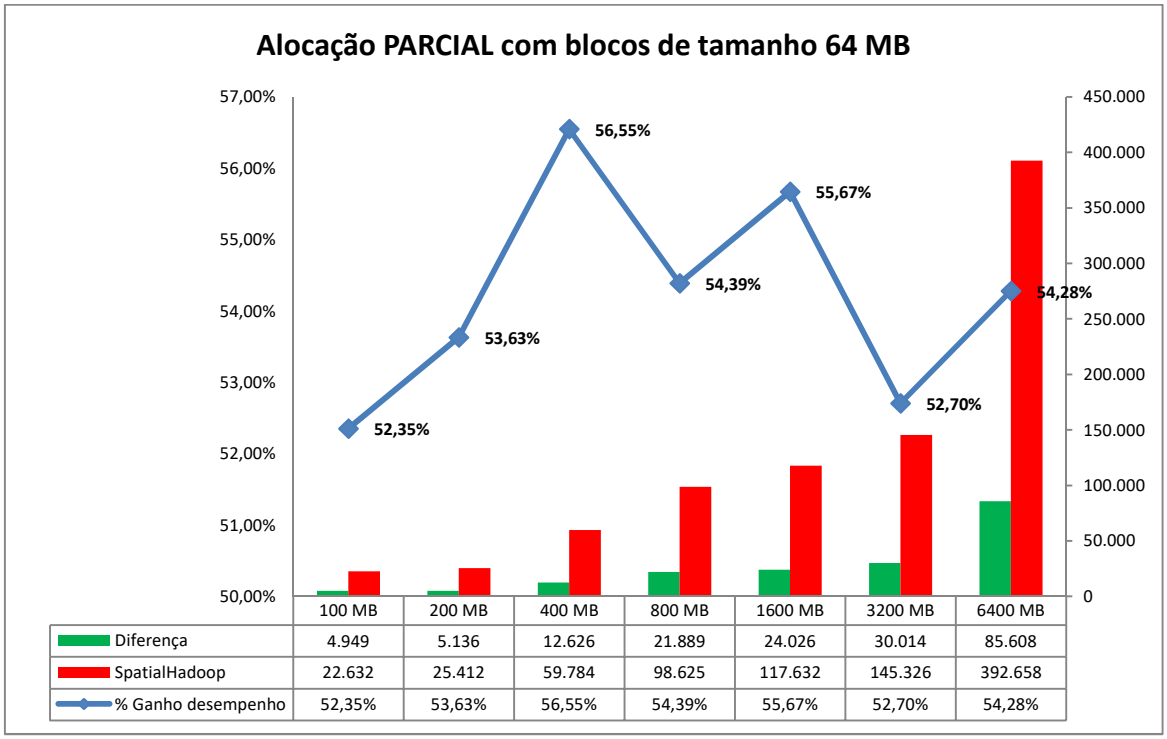


Figura 6.13 - Alocação parical para blocos de 64 MB.

A Figura 6.14 ilustra a média aritmética do desvio padrão para diferentes tamanhos de arquivo e o percentual de alocação de dados entre as duas políticas para a alocação parcial. Assim o percentual de alocação da atual política do *SpatialHadoop* se amplia, devido ao aumento no tamanho do arquivo. Este comportamento já foi identificado em análises anteriores e se repete. Da mesma forma como ocorreu na seção 6.3.1, analisando o intervalo gerado pelo tempo médio com desvio padrão (i.e. tempo médio – desvio padrão a tempo médio + desvio padrão) obtido pelas novas políticas para diferentes tamanhos arquivos, verifica-se que não houve interseção com o intervalo gerado pelo tempo médio com desvio padrão obtido pelo *SpatialHadoop*.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	1254	2054	2845	2452	1569	2041	1002
DP SpatialHadoop	2054	2013	1524	3102	2985	2042	3685
%AL Cosphdp	67%	78%	76%	71%	68%	74%	64%
%AL SpatialHadoop	15%	17%	16%	19%	24%	28%	38%

Figura 6.14 - Desvio padrão e percentual de alocação parcial para blocos de 64 MB.

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com dois *racks*, utilizando dados sintéticos, blocos com tamanho de 32 MB e **alocação total,** é demonstrado na Figura 6.15. Analisando a Figura 6.15, é possível notar uma melhora de 56,49% de desempenho médio com a execução da política de **alocação total** dos dados em relação à atual política do *SpatialHadoop*.

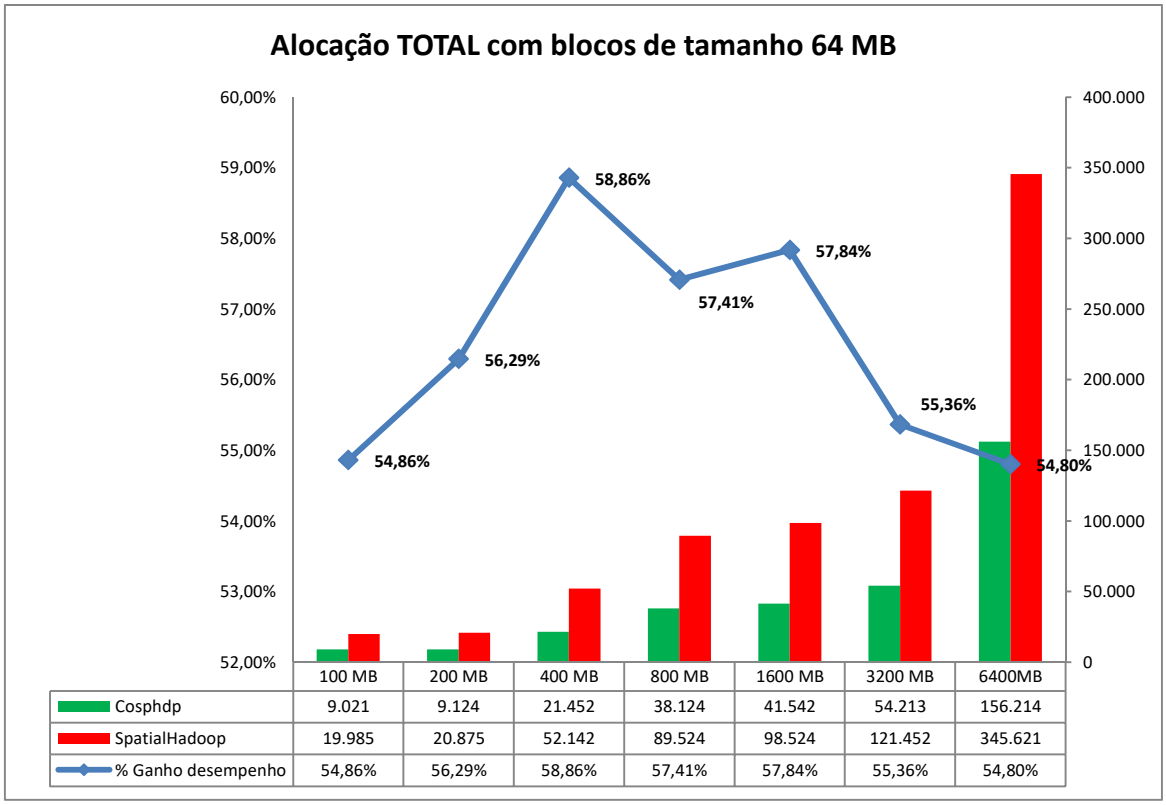


Figura 6.15 - Alocação total para blocos de 64 MB.

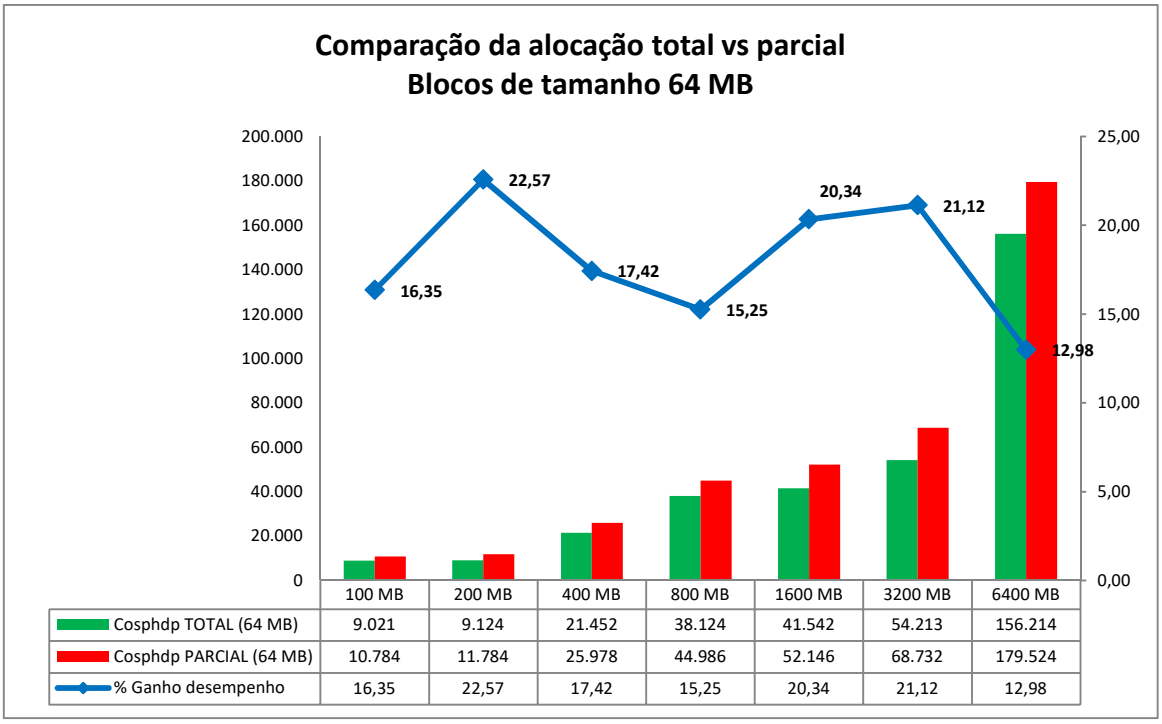
O desvio padrão e o percentual de alocação obtido pelas duas políticas para arquivos com blocos de tamanho de 64 MB são expostos na Figura 6.16. O mesmo comportamento de alocação de dados é observado na política do *SpatialHadoop*, ou

seja, quanto maior o arquivo, maior a probabilidade dos dados estarem alocados de forma relacionada.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	1847	1925	1654	1045	2074	2141	2415
DP SpatialHadoop	1425	1652	852	1624	1748	1572	1745
%AL Cosphdp	100%	100%	100%	100%	100%	100%	100%
%AL SpatialHadoop	22%	17%	25%	21%	24%	27%	29%

Figura 6.16 - Devio padrão e percentual de alocação total para blocos de 64 MB.

A Figura 6.17 demonstra os tempos de comparação entre as novas políticas de **alocação total** e parcial para blocos de 64 MB. Em geral, nota-se que, a **alocação total** é ligeiramente mais eficiente em termos de tempo de processamento do que a alocação parcial. Da mesma forma, como ocorreu na análise anterior com blocos de 32 MB, o espaço utilizado pela política de **alocação total** é superior ao espaço da política de alocação parcial, em média 26%, devido à necessidade de aumentar o número de réplicas dos blocos para atingir o objetivo da **alocação total** dos dados relacionados. Este percentual de 26% de aumento do espaço é inferior ao percentual de aumento para blocos de 32 MB, que corresponde a 30%, devido ao tamanho do bloco.

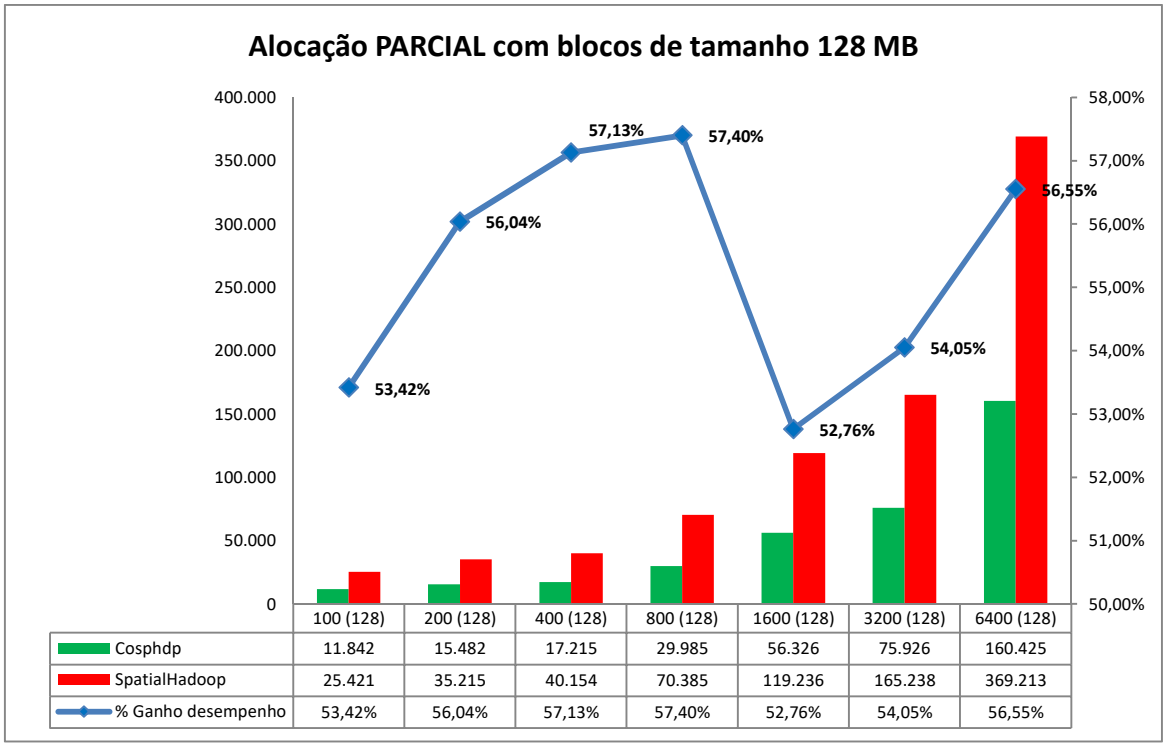


	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp TOTAL (64 MB)	9.021	9.124	21.452	38.124	41.542	54.213	156.214
Cosphdp PARCIAL (64 MB)	10.784	11.784	25.978	44.986	52.146	68.732	179.524
% Ganho desempenho	16,35	22,57	17,42	15,25	20,34	21,12	12,98

Figura 6.17 - Comparação da alocação total x parcial para blocos de 64 MB.

6.3.3 Experimentos com blocos de 128 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com dois *racks*, utilizando dados sintéticos, blocos com tamanho de 128 MB e alocação parcial podem ser visualizados na Figura 6.18. As novas políticas de alocação parcial dos dados é, em média, 55,33%, com ganho de desempenho variando de 52,76% a 57,40%, mais eficiente do que a política padrão do *SpatialHadoop*. Em todos os testes, as novas políticas obtiveram um desempenho melhor do que a atual. Diante disto, pode-se comprovar a hipótese de que a organização dos dados é fator crítico de sucesso para se obter um melhor desempenho das operações de junção espacial no *SpatialHadoop*.



	100 (128)	200 (128)	400 (128)	800 (128)	1600 (128)	3200 (128)	6400 (128)
Cosphdp	11.842	15.482	17.215	29.985	56.326	75.926	160.425
SpatialHadoop	25.421	35.215	40.154	70.385	119.236	165.238	369.213
% Ganho desempenho	53,42%	56,04%	57,13%	57,40%	52,76%	54,05%	56,55%

Figura 6.18 - Alocação parical para blocos de 128 MB.

A Figura 6.19 demonstra os resultados obtidos do desvio padrão médio das duas políticas, juntamente com o percentual de alocação dos dados de cada uma. Nota-se que, os blocos maiores podem proporcionar um ligeiro aumento no percentual de alocação, como já foi relatado nos testes anteriores. O mesmo comportamento dos experimentos anteriores se repete quando, analisando o intervalo gerado pelo tempo médio com desvio padrão (i.e. tempo médio – desvio padrão a tempo médio + desvio padrão), obtido pelas novas políticas para diferentes tamanhos arquivos, verifica-se que não houve interseção com o intervalo gerado pelo tempo médio com desvio padrão obtido pelo *SpatialHadoop*.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	1524	1692	3021	2513	2983	785	1021
DP SpatialHadoop	3201	1524	2983	3201	1423	892	1245
%AL Cosphdp	77%	75%	70%	64%	60%	74%	68%
%AL SpatialHadoop	19%	22%	29%	27%	33%	38%	34%

Figura 6.19 – Desvio padrão e percentual de alocação parcial para blocos de 128 MB.

Na Figura 6.20, é possível visualizar o resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster*

com dois *racks*, utilizando dados sintéticos, blocos com tamanho de 128 MB e **alocação total**. Uma comparação é feita entre as novas políticas **alocação total** dos dados e a alocação padrão *SpatialHadoop* para blocos de tamanho de 128 MB. Os resultados obtidos com a **alocação total** em relação ao *SpatialHadoop* são, em média, 57,17% mais rápidos. Porém, se comparados à alocação parcial média de 55,33%, o ganho de desempenho é pequeno.

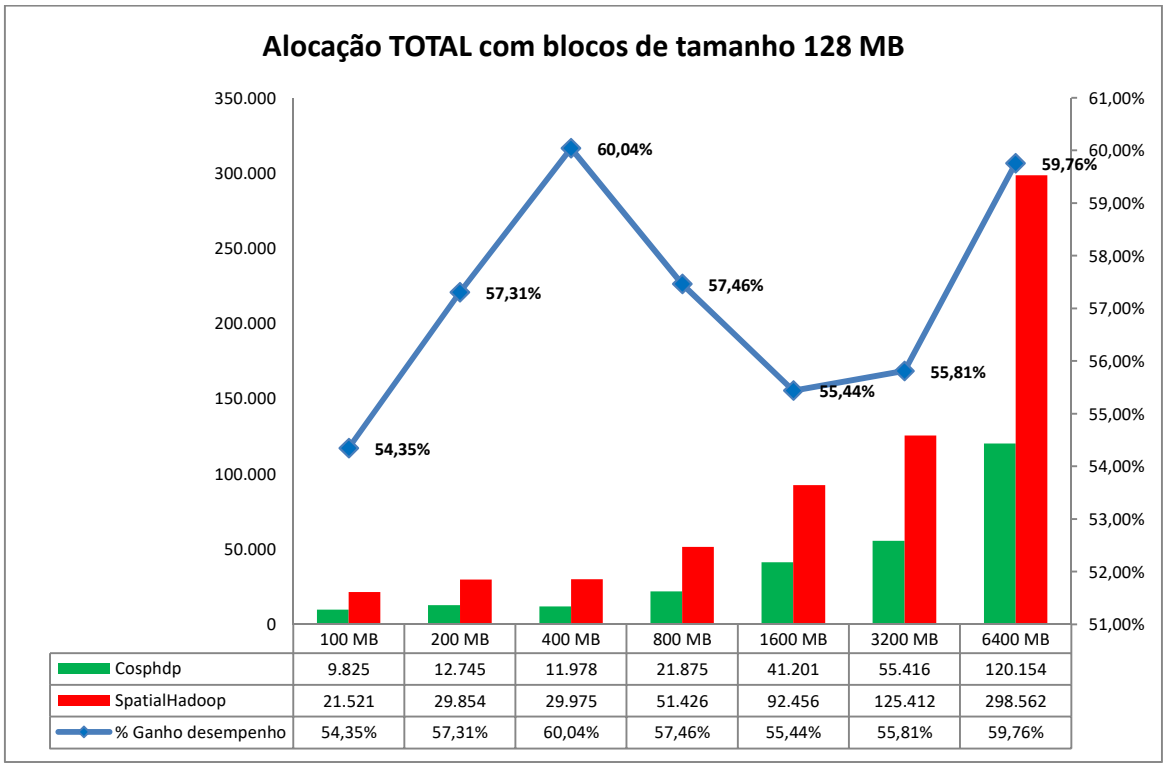


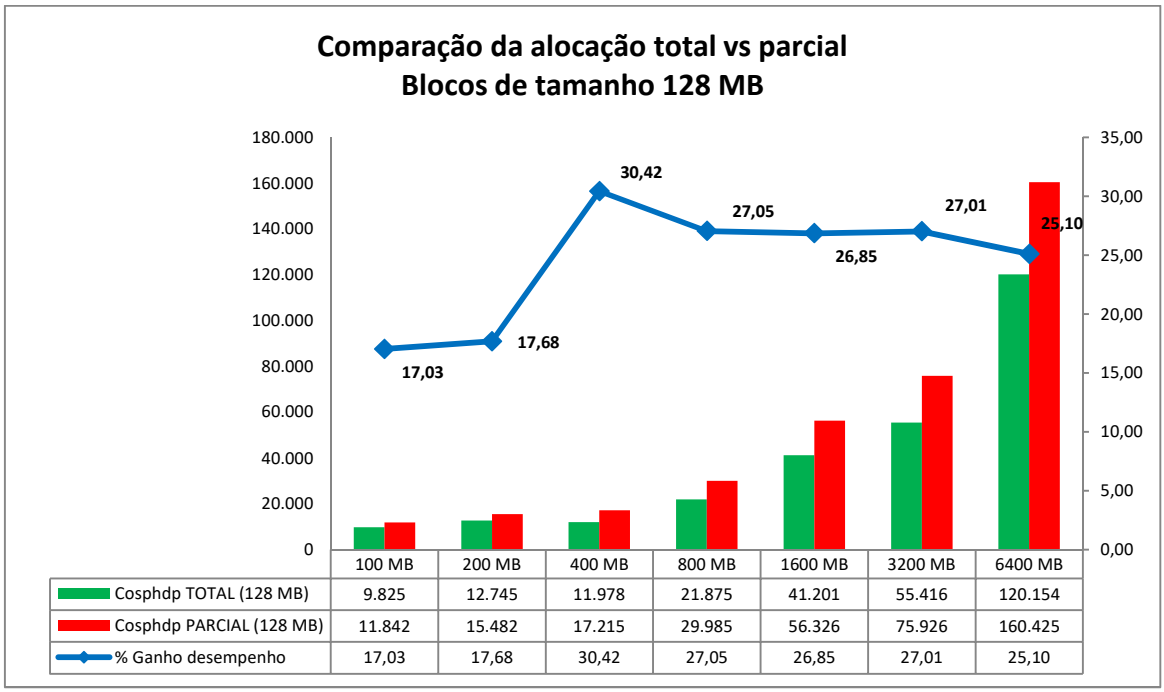
Figura 6.20 - Alocação total para blocos de 128 MB.

O desvio padrão médio e o percentual de **alocação total** das novas políticas e *SpatialHadoop* são apresentados na Figura 6.21.

	Desvio 100 mb	Desvio 200 mb	Desvio 400 mb	Desvio 800 mb	Desvio 1600 mb	Desvio 3200 mb	Desvio 6400 mb
DP Cosphdp	2154	1425	2415	2487	1201	2012	1984
DP SpatialHadoop	1478	1985	2875	1625	1354	2875	1793
%AL Cosphdp	100%	100%	100%	100%	100%	100%	100%
%AL SpatialHadoop	17%	14%	22%	20%	28%	31%	36%

Figura 6.21 - Devio padrão e percentual de alocação total para blocos de 128 MB.

Finalizando as análises dos testes para uma arquitetura de dois *racks*, a Figura 6.22 exibe a comparação do tempo de processamento das duas novas formas de alocar os dados, parcial ou total para blocos de 128 MB.

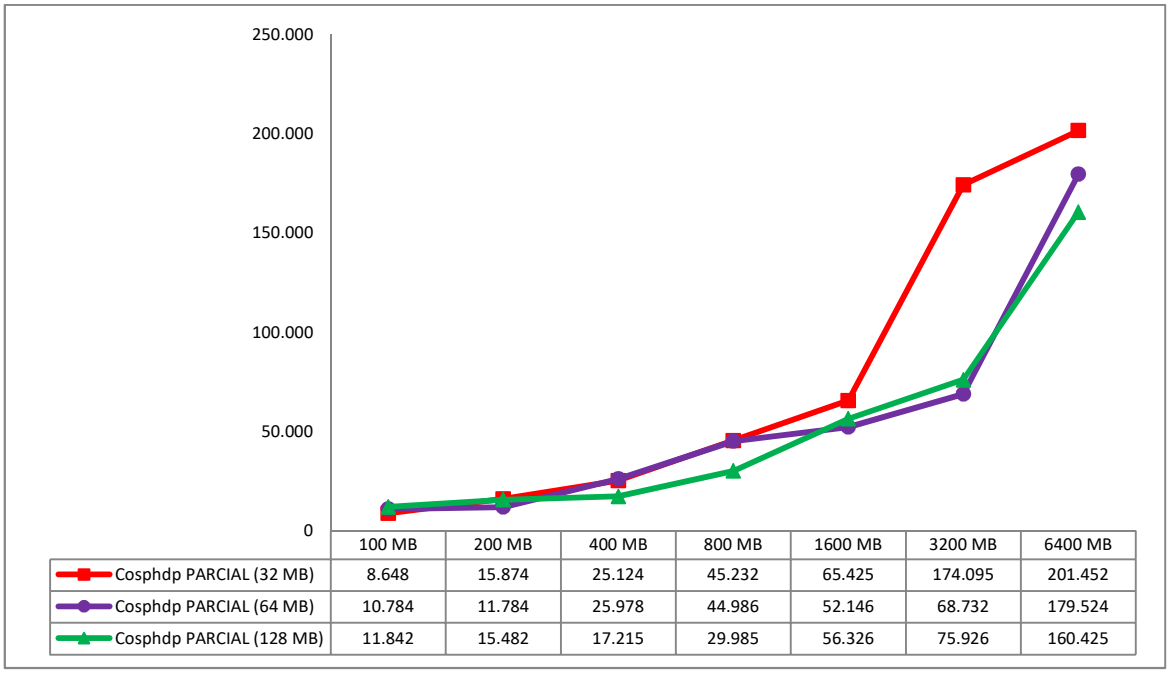


	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp TOTAL (128 MB)	9.825	12.745	11.978	21.875	41.201	55.416	120.154
Cosphdp PARCIAL (128 MB)	11.842	15.482	17.215	29.985	56.326	75.926	160.425
% Ganho desempenho	17,03	17,68	30,42	27,05	26,85	27,01	25,10

Figura 6.22 - Comparação da alocação total x parcial para blocos de 128 MB.

6.3.4 Comparações dos experimentos

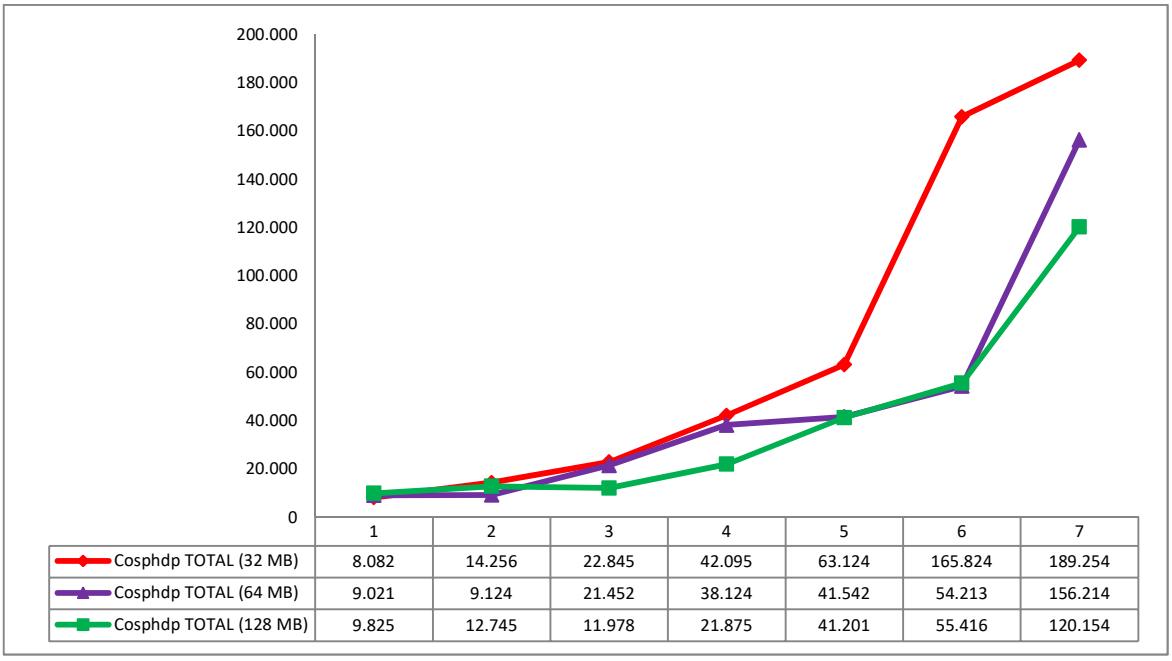
A Figura 6.23 apresenta uma comparação das novas políticas de alocação parcial dos dados para os diferentes tamanhos de blocos. Uma pequena diferença de desempenho pode ser notada quando se usou blocos de 128 MB, em relação aos blocos 64 MB. Com o uso de blocos de 32 MB ocorreu perda de desempenho para as políticas propostas conforme o tamanho do arquivo aumenta.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp PARCIAL (32 MB)	8.648	15.874	25.124	45.232	65.425	174.095	201.452
Cosphdp PARCIAL (64 MB)	10.784	11.784	25.978	44.986	52.146	68.732	179.524
Cosphdp PARCIAL (128 MB)	11.842	15.482	17.215	29.985	56.326	75.926	160.425

Figura 6.23 - Comparação dos experimentos com alocação parcial.

A comparação, das novas políticas de **alocação total** dos dados para os diferentes tamanhos de blocos, é apresentada na Figura 6.24. Neste caso, o uso de blocos de 128 MB proporciona um melhor desempenho em relação ao uso de blocos de 32 MB e de 64 MB. Assim como ocorreu na alocação parcial, com o uso de blocos de 32 MB ocorreu uma perda de desempenho para as políticas propostas conforme o tamanho do arquivo aumenta.



	1	2	3	4	5	6	7
Cosphdp TOTAL (32 MB)	8.082	14.256	22.845	42.095	63.124	165.824	189.254
Cosphdp TOTAL (64 MB)	9.021	9.124	21.452	38.124	41.542	54.213	156.214
Cosphdp TOTAL (128 MB)	9.825	12.745	11.978	21.875	41.201	55.416	120.154

Figura 6.24 - Comparação dos experimentos com alocação total.

6.4 Experimentos com um *rack*

Os dados utilizados neste segundo teste são os mesmos utilizados na seção 6.3, porém foi alterada a arquitetura do *cluster,* reduzindo o número de racks para apenas 1 *rack*.

6.4.1 Experimentos com blocos de 32 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com um *rack*, utilizando dados sintéticos*,* blocos com tamanho de 32 MB e alocação parcial é demonstrado na Figura 6.25. O tempo médio de processamento das novas políticas em apenas um *rack* é em média 48,14%, com ganho de desempenho variando de 39,26% a 62,36%, mais ágil que a atual política do *SpatialHadoop.* Se comparado com os testes executados com dois *racks,* o tempo médio de processamento é de 50,73% mais ágil que o tempo de 48,14%, quando se utiliza apenas um *rack*. Em geral, os testes nos arquivos com apenas um *rack* foram mais eficientes do que os arquivos com dois *racks*, perdendo apenas para o arquivo de 200 MB. Como a diferença foi

grande, afetou o cálculo da média. Analisando os dados sob esta ótica, pode-se concluir que os testes executados em apenas um *rack* é mais ágil do que os testes em dois *racks*. Isso está de acordo com os custos previstos na Seção 4.2.

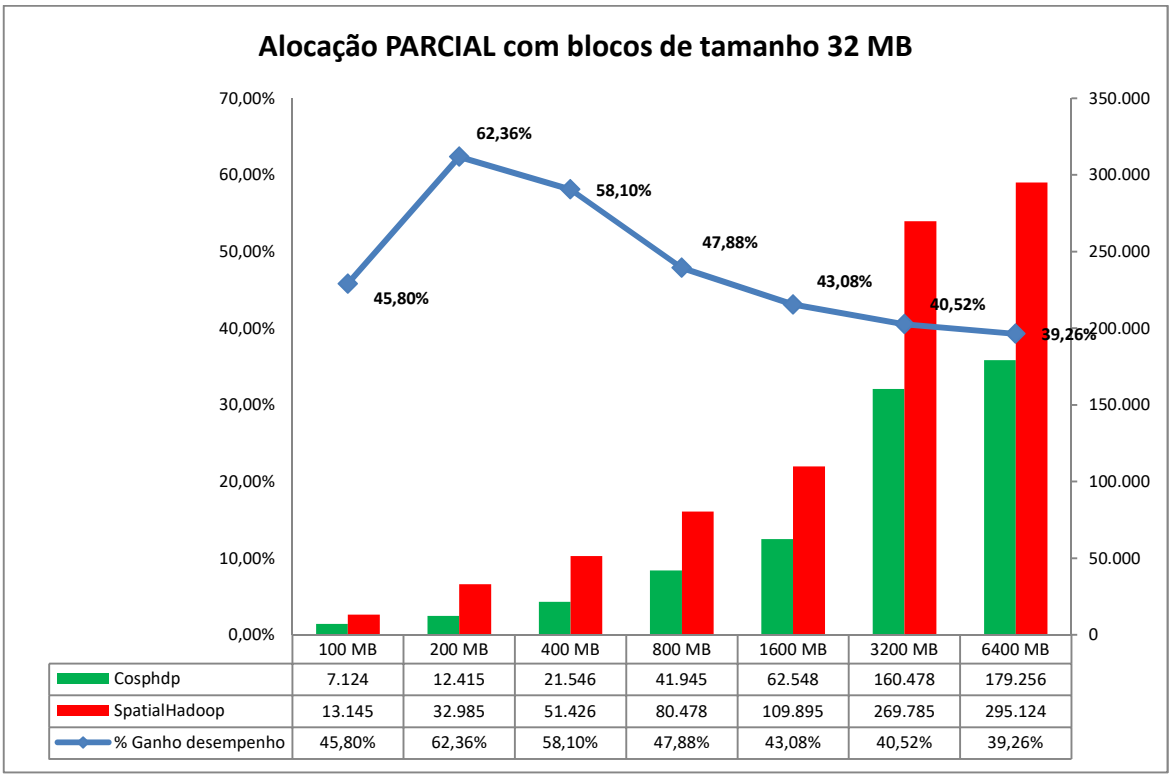


Figura 6.25 - Alocação parical para blocos de 32 MB.

6.4.2 Experimentos com blocos de 64 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com um *rack*, utilizando dados sintéticos, blocos com tamanho de 64 MB e alocação parcial pode ser visualizado na Figura 6.26. O tempo médio de processamento das novas políticas em apenas um *rack* é em média 51,70%, com ganho de desempenho variando de 45,76% a 65,06%, mais ágil que a atual política do *SpatialHadoop.* Se comparado com os testes executados com dois *racks,* o tempo médio de processamento é de 54,22% mais ágil que o tempo de 48,14%, quando se utiliza apenas um *rack*.

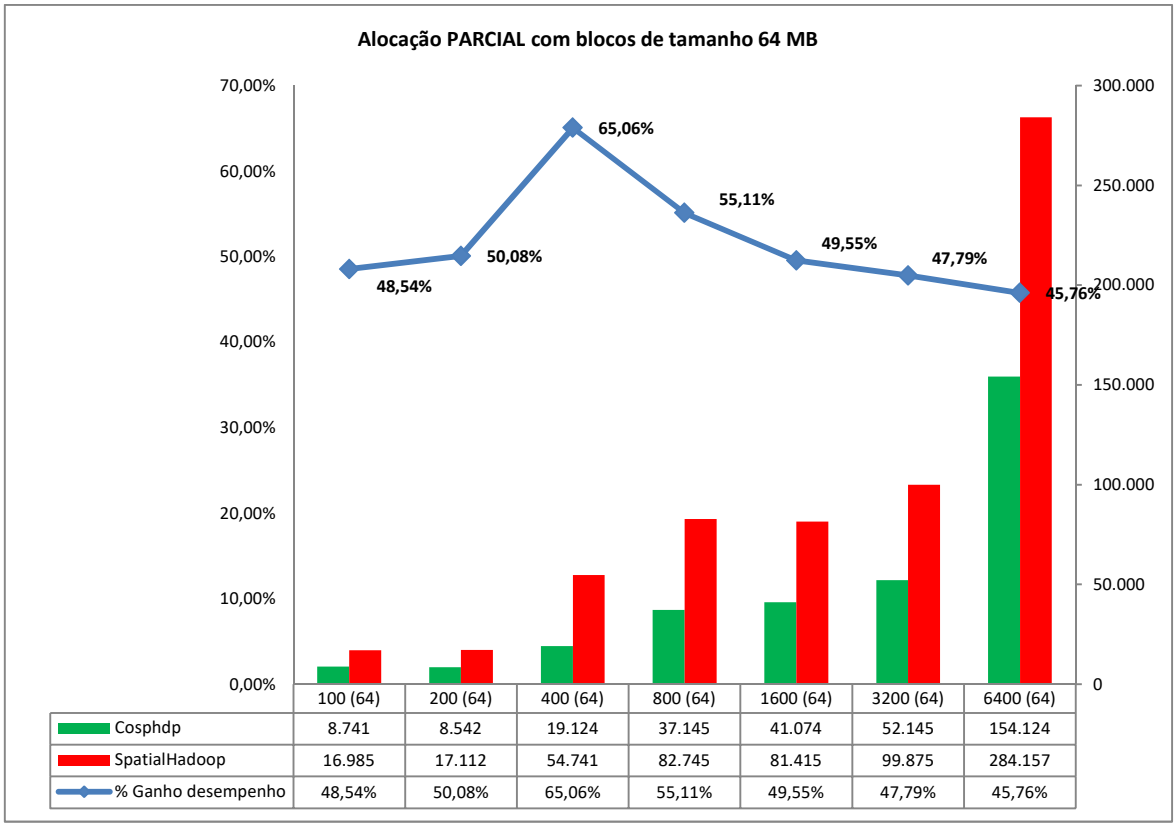
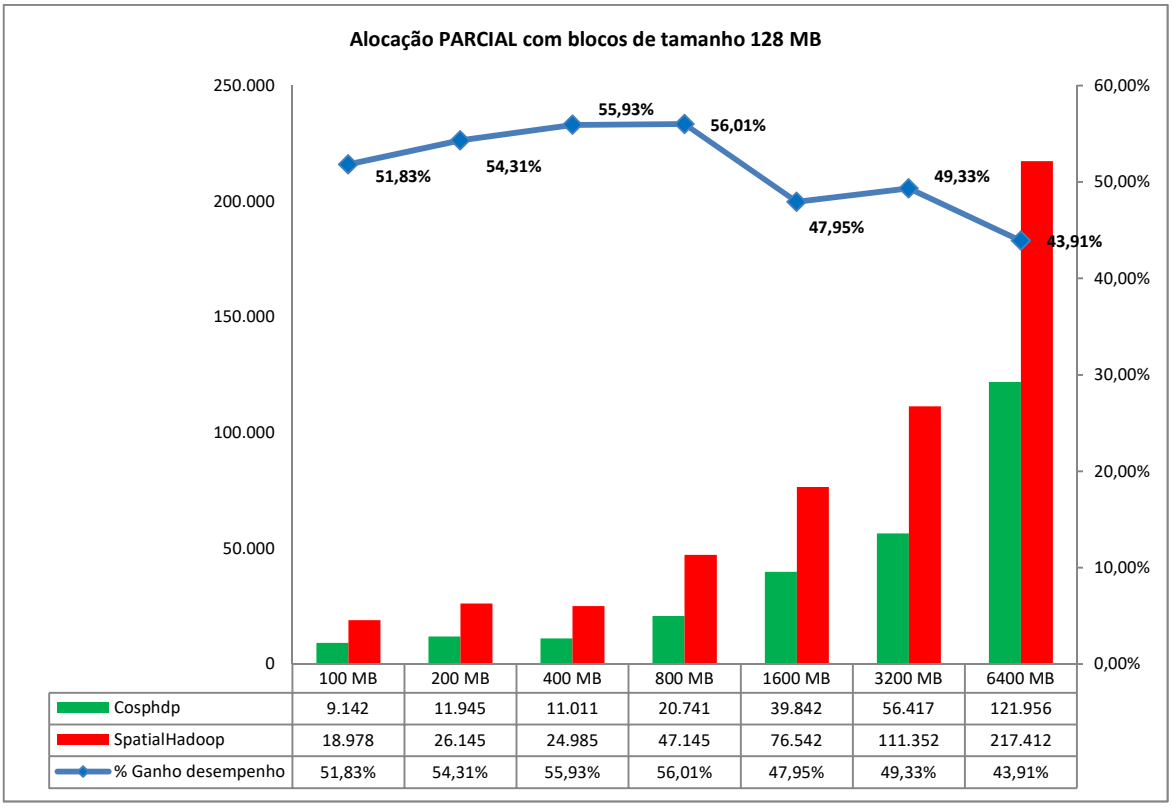


Figura 6.26 - Alocação parical para blocos de 64 MB.

6.4.3 Experimentos com blocos de 128 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* com um *rack*, utilizando dados sintéticos, blocos com tamanho de 128 MB e alocação parcial é exibido na Figura 6.27. O tempo médio de processamento das novas políticas em apenas um *rack* é em média 51,32%, com ganho de desempenho variando de 43,91% a 56,01%, mais ágil que a atual política do *SpatialHadoop.* Se comparado com os testes executados com dois *racks,* o tempo médio de processamento é de 54,22% mais ágil que o tempo de 48,14%, quando se utiliza apenas um *rack*.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	9.142	11.945	11.011	20.741	39.842	56.417	121.956
SpatialHadoop	18.978	26.145	24.985	47.145	76.542	111.352	217.412
% Ganho desempenho	51,83%	54,31%	55,93%	56,01%	47,95%	49,33%	43,91%

Figura 6.27 - Alocação parical para blocos de 128 MB.

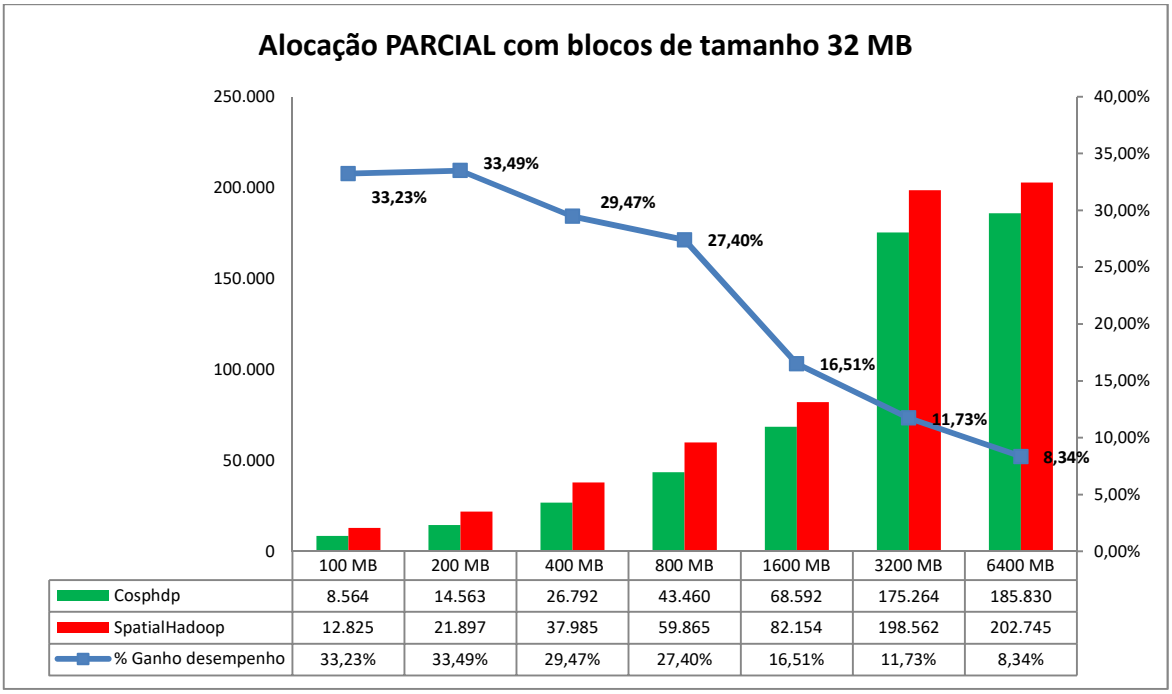
6.5 Experimentos com redução do *cluster*

Os experimentos realizados na arquitetura reduzida AC1 utilizaram os mesmos dados das arquiteturas anteriores AC2 e AC3. Na sequência, serão apresentados os resultados obtidos com os testes de desempenho em uma arquitetura reduzida.

6.5.1 Experimentos com blocos de 32 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* reduzida, utilizando dados sintéticos, blocos com tamanho de 32 MB e alocação parcial, é demonstrado na Figura 6.28. Constata-se que a junção espacial executada com as novas políticas é 22,88%, mais eficiente com ganho de desempenho variando de 8,34% a 33,49%, em média mais rápida do que a atual política de *SpatialHadoop*. No entanto, a redução do *cluster* gerou um comportamento atípico, ou seja, à medida que o

tamanho dos arquivos aumenta, reduz o percentual de desempenho entre as duas políticas. Isto se deve ao fato do *cluster* ser menor e, consequentemente, existe uma alta probabilidade de alocação conjunta de dados espaciais relacionados em um mesmo nó pela política padrão, tornando o processamento das duas políticas semelhantes.

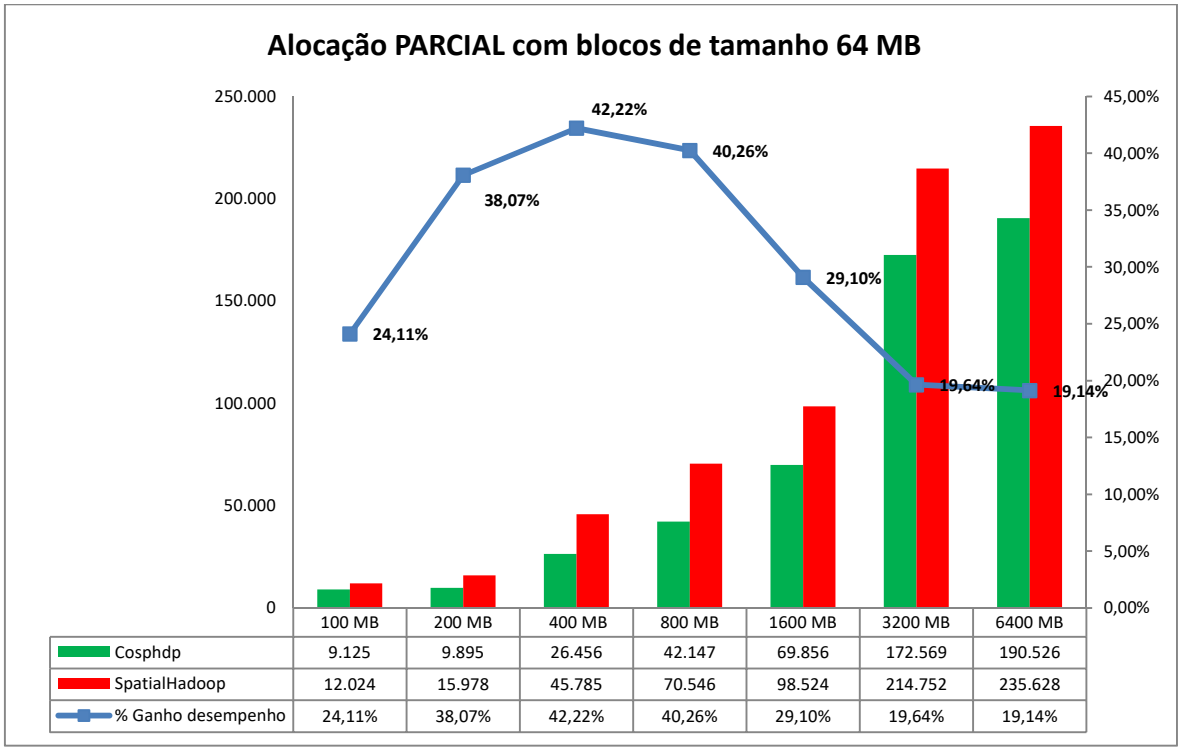


	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	8.564	14.563	26.792	43.460	68.592	175.264	185.830
SpatialHadoop	12.825	21.897	37.985	59.865	82.154	198.562	202.745
% Ganho desempenho	33,23%	33,49%	29,47%	27,40%	16,51%	11,73%	8,34%

Figura 6.28 - Alocação parical para blocos de 32 MB.

6.5.2 Experimentos com blocos de 64 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* reduzida, utilizando dados sintéticos, blocos com tamanho de 64 MB e alocação parcial é apresentado na Figura 6.29. As novas políticas são 30,36%, com ganho de desempenho variando de 19,14% a 42,22%, em média, mais rápida do que a política de *SpatialHadoop*. Em geral, o comportamento dos dados se assemelha ao comportamento para blocos de 32 MB, porém com o aumento do tamanho do bloco reduziu o percentual de blocos alocados de forma relacionados pela política padrão, isto justifica o aumento no ganho médio de 22,88% para blocos de 32 MB para 30,36% em média para blocos de 64 MB.

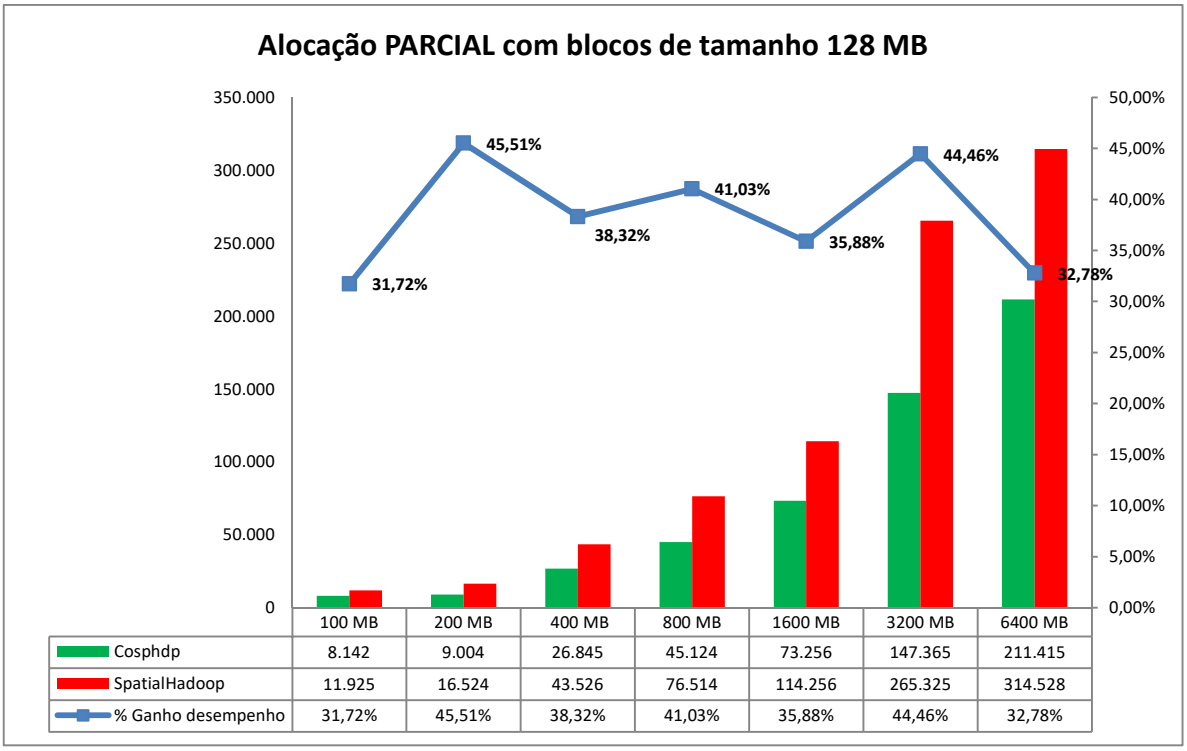


	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	9.125	9.895	26.456	42.147	69.856	172.569	190.526
SpatialHadoop	12.024	15.978	45.785	70.546	98.524	214.752	235.628
% Ganho desempenho	24,11%	38,07%	42,22%	40,26%	29,10%	19,64%	19,14%

Figura 6.29 - Alocação parical para blocos de 64 MB.

6.5.3 Experimentos com blocos de 128 MB

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* reduzida, utilizando dados sintéticos, blocos com tamanho de 128 MB e alocação parcial pode ser visualizado na Figura 6.30. As novas políticas são 38,53%, mais eficientes com ganho de desempenho variando de 31,72% a 45,51%, em média, mais rápida do que a política do *SpatialHadoop*. Ao contrário do que ocorreu nas análises de blocos de tamanho de 32 MB e 64 MB, existe uma tendência do percentual de desempenho permanecer alto, provocada pelo tamanho do bloco. Nota-se que, o aumento do tamanho do bloco em uma arquitetura de *cluster* reduzido torna menos frequente que blocos relacionados sejam alocados no mesmo nó pela política de alocação padrão, o que aumenta o percentual de ganho entra as duas políticas propostas e a política padrão.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	8.142	9.004	26.845	45.124	73.256	147.365	211.415
SpatialHadoop	11.925	16.524	43.526	76.514	114.256	265.325	314.528
% Ganho desempenho	31,72%	45,51%	38,32%	41,03%	35,88%	44,46%	32,78%

Figura 6.30 - Alocação parical para blocos de 128 MB.

6.6 Experimentos com dados reais

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* de dois *racks*, utilizando dados reais de rios e estradas com blocos de tamanho de 32 MB, 64 MB e 128 MB e alocação parcial, pode ser visualizado na Figura 6.31. As novas políticas são 56,94%, com ganho de desempenho variando de 54,90% a 60,32%, em média, mais rápida do que o *SpatialHadoop*. Em todos os diferentes tamanhos de blocos, as novas políticas apresentaram um desempenho melhor em relação à política padrão do *SpatialHadoop,* com destaque para os blocos de tamanho de 64 MB, onde a diferença entre as duas políticas foi de 60,32%. Assim como ocorreu com os testes em dados sintéticos, os dados reais demonstraram que a alocação conjunta de dados espaciais relacionados e o escalonamento de tarefas para dados espaciais relacionados é mais vantajoso do que as atuais políticas adotados pelo *SpatialHadoop*.

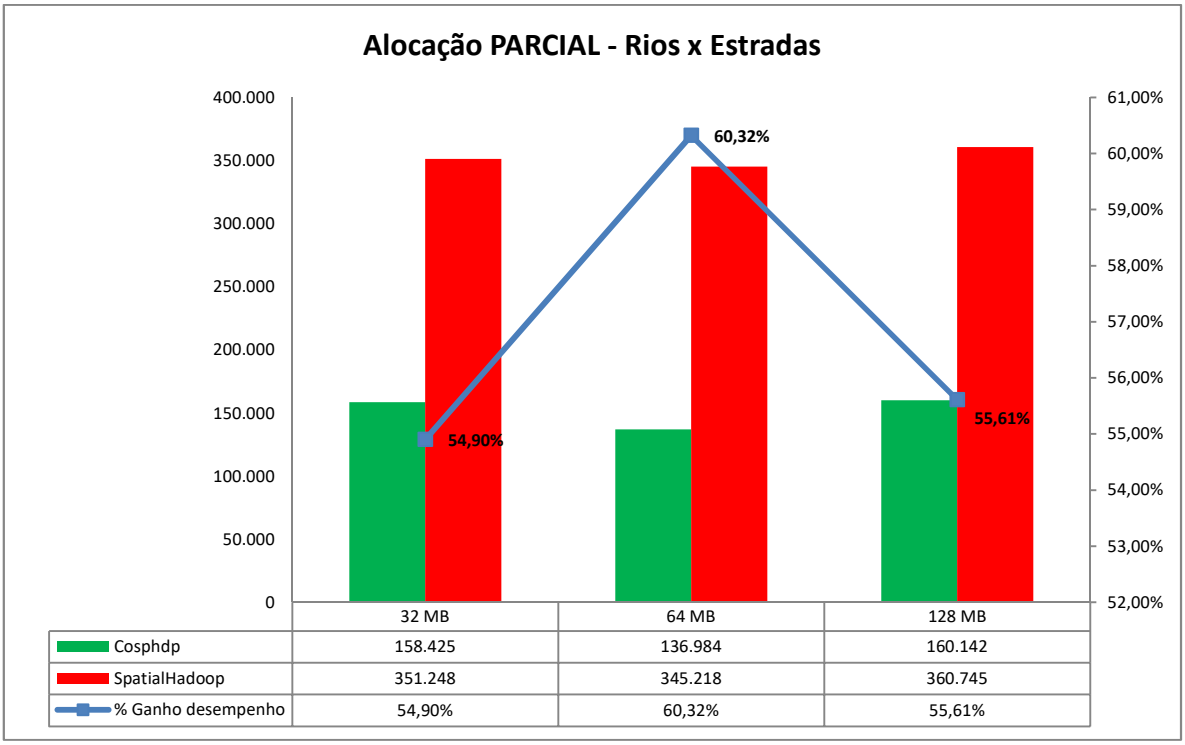


Figura 6.31 - Desempenho da alocação parcial de rios e estradas.

O resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* de dois *racks*, utilizando dados reais, blocos com tamanho de 32 MB, 64 MB e 128 MB e **alocação total,** pode ser visualizado na Figura 6.32. As novas políticas são 58,04%, com ganho de desempenho variando de 55,47% a 61,11%, em média, mais rápida do que a política do *SpatialHadoop* e também melhor do que a política de alocação parcial 56,94% em média, apresentada anteriormente. Outro ponto de destaque foi o desbalanceamento do *cluster* devido à alocação dos dados relacionados. Os blocos com tamanho de 64 MB foi o que proporcionou a melhor diferença de desempenho entre as duas políticas com ganho de desempenho de 61,11% das políticas propostas com relação ao SpatialHadoop. Apesar de a **alocação total** proporcionar um melhor desempenho em relação à alocação parcial o custo de armazenamento dos dados que chega a ultrapassar 33% torna o uso da política não apropriado para muitas situações, onde o percentual de dados relacionado é muito alto.

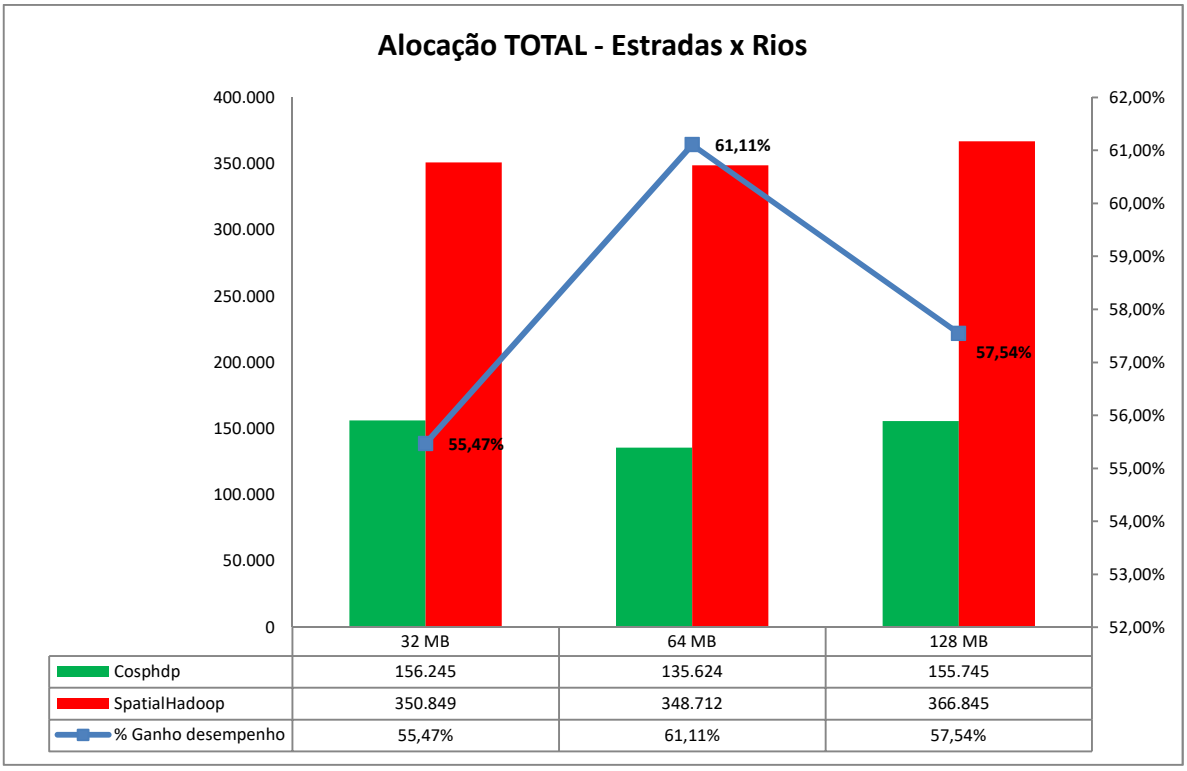


Figura 6.32 - Desempenho da alocação total de rios e estradas.

O comparativo das políticas de **alocação total** e parcial pode ser visualizado na Figura 6.33, onde é apresentado o resultado do processamento dos testes de desempenho da junção espacial no *SpatialHadoop* em uma arquitetura de *cluster* de dois *racks*, utilizando dados reais, blocos com tamanho de 32 MB, 64 MB e 128 MB. A **alocação total** demonstrou ser mais eficiente do que a alocação parcial. Porém o custo de alocar todos os dados relacionados torna o uso da política inadequado, levando-se em consideração o percentual de ganho com a **alocação total** que ficou abaixo de 3%.

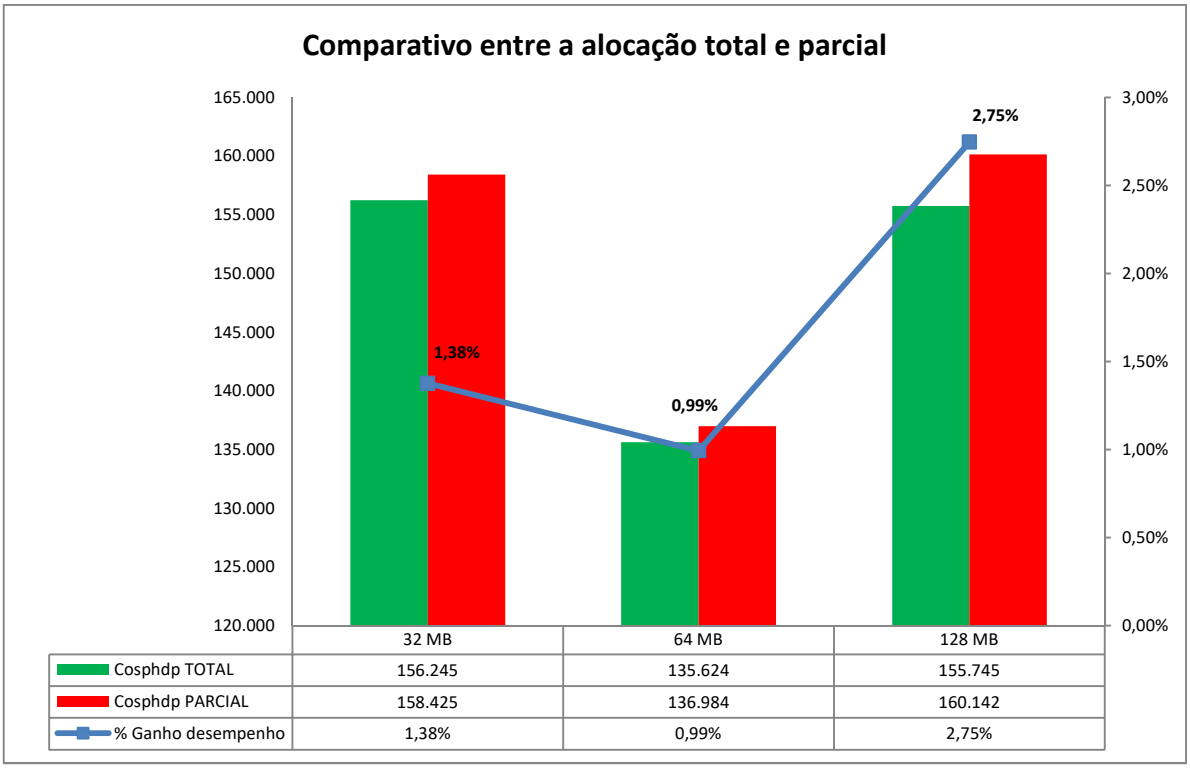
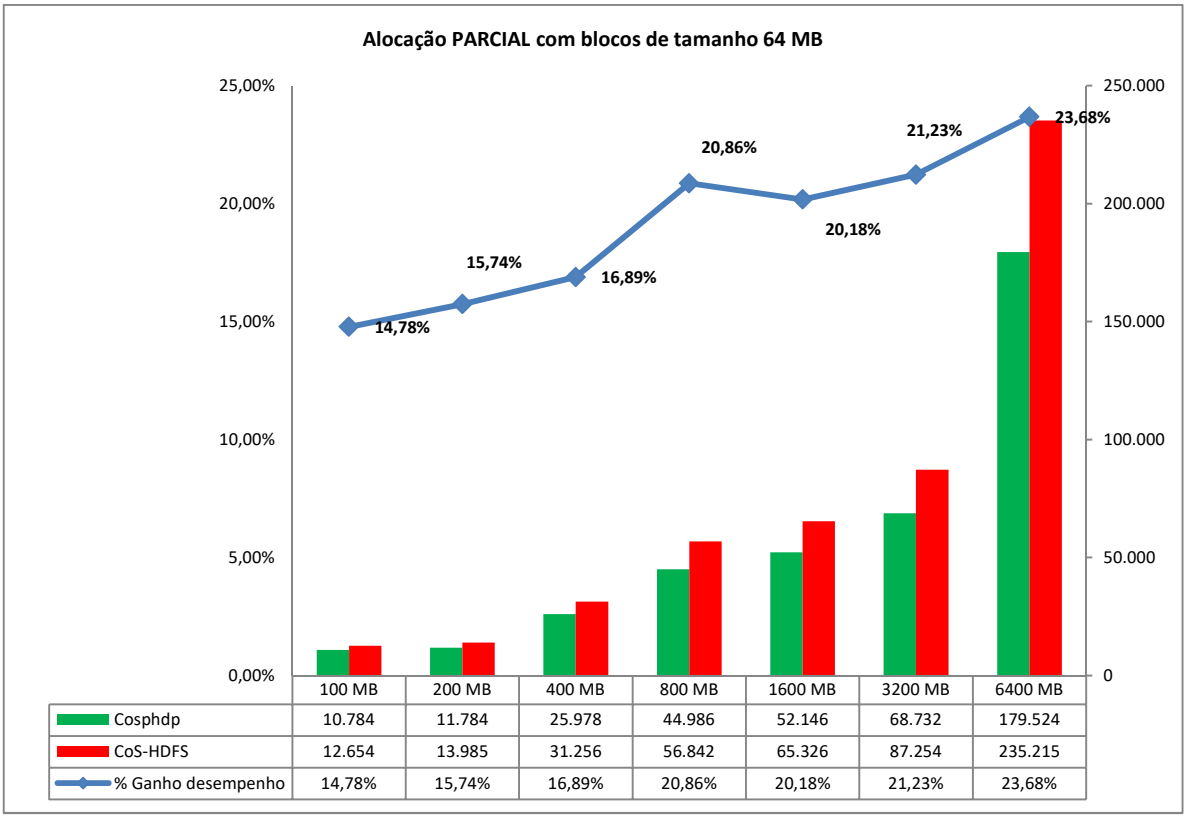


Figura 6.33 - Comparativo entre a alocação total e parcial.

6.7 *CoS-HDFS* versus *Cosphdp*

O resultado do processamento dos testes de desempenho da junção espacial no *CoS-HDFS* na arquitetura AC3, utilizando dados sintéticos e blocos com tamanho de 64 MB e alocação parcial, é apresentado na Figura 6.34. As novas políticas são 19,05%, com ganho de desempenho variando de 14,78% a 23,68%, em média, mais rápida do que o *CoS-HDFS*. Em todos os diferentes tamanhos de blocos, as novas políticas apresentaram um desempenho melhor em relação à política do CoS-HDFS.



	100 MB	200 MB	400 MB	800 MB	1600 MB	3200 MB	6400 MB
Cosphdp	10.784	11.784	25.978	44.986	52.146	68.732	179.524
CoS-HDFS	12.654	13.985	31.256	56.842	65.326	87.254	235.215
% Ganho desempenho	14,78%	15,74%	16,89%	20,86%	20,18%	21,23%	23,68%

Figura 6.34 - Comparativo entre Cosphdp e Cos-HDFS.

O *CoS-HDFS* apresenta um melhor desempenho com arquivos maiores e a mesma observação aplica-se ao *Cosphdp*, ou seja, quanto maior o arquivo melhor é o desempenho. Analisando os dados da Figura 6.34, conclui-se que o *Cosphdp* obteve o maior percentual de desempenho com relação ao *CoS-HDFS* em arquivos de 6.400 MB. Acredita-se que a diferença de desempenho obtida a favor das novas políticas esteja relacionada ao escalonamento de tarefas. O *Cosphdp* utiliza-se de uma técnica mais eficiente para utilização dos recursos disponíveis do *cluster* ao contrário do *CoS-HDFS*, conforme destacado na seção 5.6 desta tese. Acredita-se que a diferença de desempenho entre as duas técnicas pode ser maior se o processamento do *Cosphdp* ocorrer em um *cluster* geo-distribuídos, o mesmo utilizado nos testes de desempenho do *CoS-HDFS*.

6.8 Considerações finais

Este capítulo apresentou a análise dos dados e a interpretação dos resultados referentes aos testes de desempenho experimentais. Foram realizados amplos experimentos para avaliar o desempenho das novas políticas em comparação com a política do *SpatialHadoop* e *CoS-HDFS.* Os experimentos foram realizados com diferentes tipos de dados, em três arquiteturas de *cluster*, blocos de tamanhos distintos e formas de alocação alteradas. Constatou-se que, no geral, as novas políticas tiveram um desempenho superior em relação à atual política adotada pelo *SpatialHadoop* e à política do *CoS-HDFS*. Sendo assim, é possível afirmar que a proposta desta tese está corroborada, pois não se conseguiu refutar as hipóteses propostas na seção 1.4 para esta tese de doutorado.

Capítulo 7

CONCLUSÃO

Este capítulo apresenta as conclusões obtidas após a análise dos resultados de desempenho experimentais que foram usados para validar as duas novas políticas de alocação conjunta de dados espaciais relacionados no processamento de junção espacial e de escalonamento de tarefas. Também serão destacadas as principais contribuições. Por fim, são listados trabalhos futuros para uma possível continuidade desta pesquisa.

7.1 Comprovação das hipóteses

Esta tese de doutorado investigou o problema de alocação conjuntas de dados espaciais relacionados no processamento de junção espacial e o escalonamento de tarefas para dados espaciais relacionados para obter um melhor desempenho na execução de operações de junção espacial em *clusters* de computadores, usando o *framework SpatialHadoop*. Após análise dos resultados obtidos, pode-se inferir com relação às hipóteses:

Hipótese 1: "O processamento das operações de junção espacial, quando ocorre no mesmo nó onde estão armazenados os dois conjuntos de dados espaciais, é mais eficiente do que o processamento dos dados espalhados em nós de armazenamento distintos".

Hipótese 2: "A alocação conjunta dos dados espaciais relacionados e o escalonamento de tarefas para dados espaciais relacionados reduzem o tráfego de rede e o custo de E/S de disco e consequentemente melhora o desempenho no processamento de junção espacial em *SpatialHadoop".*

Para a comprovação das hipóteses foram realizados diversos testes de desempenho em diferentes arquiteturas de *clusters* de computadores para avaliar o efeito da alocação conjunta dos dados espaciais relacionados no que tange a redução do tráfego de rede e o custo de E/S de disco proporcionando, desta forma, um melhor desempenho no processamento das operações de junção espacial. Com a implementação das políticas de alocação e escalonamento, o tempo de processamento da junção espacial é substancialmente mais eficiente do que o processamento da alocação aleatória. As alterações realizadas para obter o melhor desempenho foram com relação aos dados espaciais relacionados alocados de forma conjunta e o escalonamento de tarefas para dados espaciais. Desta forma, foi possível quantificar a redução do tráfego da rede e o custo de E/S de disco, visto que com as novas políticas não há necessidade de movimentação dos dados entre o nó de armazenamento e o nó de processamento do *cluster*. Com os resultados obtidos nas tabelas seguintes é possível comprovar que o processamento da junção espacial, quando forem realizados no mesmo nó onde se encontra os dois conjuntos de dados, é mais eficiente.

As tabelas 7.1 e 7.2 demonstram o percentual de redução do tempo de processamento da junção espacial no *SpatialHadoop* com relação ao *Cosphdp*, utilizando o algoritmo original de junção espacial *SJMR*. Nota-se que, o tempo médio de redução foi acima de 50% em uma arquitetura realista composta de 2 *racks*, com dados sintéticos e utilizando a alocação parcial e total dos dados espaciais.

Tabela 7.1 - Resultados dos testes em na arquitetura AC3 com dados sintéticos e alocação parcial.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Arquivo 100 MB	46,37%	52,35%	53,42%
Arquivo 200 MB	54,92%	53,63%	56,04%
Arquivo 400 MB	56,78%	56,55%	57,13%
Arquivo 800 MB	53,54%	54,39%	57,40%
Arquivo 1600 MB	48,67%	55,67%	52,76%
Arquivo 3200 MB	48,01%	52,70%	54,05%
Arquivo 6400 MB	46,80%	54,28%	56,55%
Média	50,73%	54,22%	55,34%

Tabela 7.2 - Resultados dos testes em na arquitetura AC3 com dados sintéticos e alocação total.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Arquivo 100 MB	48,98%	54,86%	54,35%
Arquivo 200 MB	56,19%	56,29%	57,31%
Arquivo 400 MB	58,35%	58,86%	60,04%
Arquivo 800 MB	54,51%	57,41%	57,46%
Arquivo 1600 MB	48,49%	57,84%	55,44%
Arquivo 3200 MB	48,20%	55,36%	55,81%
Arquivo 6400 MB	47,28%	54,80%	59,76%
Média	51,71%	56,49%	57,17%

A arquitetura AC2 avalia o impacto do número de nós na alocação conjunta dos dados espaciais relacionados. A tabela 7.3 demonstra o percentual de redução do tempo de processamento da junção espacial no *SpatialHadoop* com relação ao *Cosphdp*, utilizando o algoritmo original de junção espacial *SJRM*.

Tabela 7.3 - Resultados dos testes em na arquitetura AC2 com dados sintéticos e alocação parcial.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Arquivo 100 MB	45,80%	48,54%	51,83%
Arquivo 200 MB	62,36%	50,08%	54,31%
Arquivo 400 MB	58,10%	65,06%	55,93%
Arquivo 800 MB	47,88%	55,11%	56,01%
Arquivo 1600 MB	43,08%	49,55%	47,95%
Arquivo 3200 MB	40,52%	47,79%	49,33%
Arquivo 6400 MB	39,26%	45,76%	43,91%
Média	48,14%	51,70%	51,32%

A arquitetura AC1 visa investigar o efeito local das políticas propostas em um ambiente local de *cluster* de computadores. A tabela 7.4 demonstra o percentual de redução do tempo de processamento da junção espacial no *SpatialHadoop* com relação ao *Cosphdp*, utilizando o algoritmo original de junção espacial *SJRM*. Nota-se que, o tempo médio de redução caiu em relação à arquitetura AC2 e AC3. Com a

quantidade reduzida de nós do *cluster,* a probabilidade de ocorrer alocação conjunta de dados espaciais pelas técnicas aleatórias do *SpatialHadoop* é maior, demonstrando que o tráfego de rede e o custo de E/S de disco menor, nesta arquitetura, proporcionam um melhor desempenho da junção espacial.

Tabela 7.4 - Resultados dos testes em na arquitetura AC1 com dados sintéticos e alocação parcial.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Arquivo 100 MB	33,23%	24,11%	31,72%
Arquivo 200 MB	33,49%	38,07%	45,51%
Arquivo 400 MB	29,47%	42,22%	38,32%
Arquivo 800 MB	27,40%	40,26%	41,03%
Arquivo 1600 MB	16,51%	29,10%	35,88%
Arquivo 3200 MB	11,73%	19,64%	44,46%
Arquivo 6400 MB	8,34%	19,14%	32,78%
Média	22,88%	30,36%	38,53%

As tabelas 7.5 e 7.6 mostram o percentual de redução do tempo de processamento da junção espacial no *SpatialHadoop* com relação ao *Cosphdp*, utilizando o algoritmo original de junção espacial *SJRM* com dados reais. A principal diferença entre as duas técnicas é a alocação conjunta de dados espaciais relacionados e a estratégia de escalonamento de tarefas *MapReduce* para dados espaciais relacionados, alocados de forma conjunta. Conclui-se que, as mudanças realizadas proporcionaram redução do tráfego de rede e o custo de E/S de disco e consequentemente reduziu o tempo de execução das operações de junção espacial no *SpatialHadoop,* devido ao processamento ocorrer no mesmo nó dos dados.

Tabela 7.5 - Resultados dos testes em na arquitetura AC3 com dados reais e alocação parcial.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Rios X Rodovias	54,90%	60,32%	55,61%

Tabela 7.6 - Resultados dos testes em na arquitetura AC3 com dados reais e alocação total.

	Blocos 32 MB	Blocos 64 MB	Blocos 128 MB
Rios X Rodovias	55,47%	61,11%	57,54%

A tabela 7.7 o demonstra os resultados comparativos entre a alocação do *CoS-HDFS* e o Cosphdp. Nota-se que, a alocação proporcionada pelo *Cosphdp* é mais eficiente no que tange a redução do tráfego de rede e o custo de E/S de disco. Mesmo que o algoritmo do *CoS-HDFS* produza mais conjuntos alocados à utilização dos recursos de rede não é eficiente e desfavorece o resultado final.

Tabela 7.7 - Resultados dos testes em na arquitetura AC3 com dados sintéticos e alocação parcial.

	Blocos 64 MB
Arquivo 100 MB	14,78%
Arquivo 200 MB	15,74%
Arquivo 400 MB	16,89%
Arquivo 800 MB	20,86%
Arquivo 1600 MB	20,18%
Arquivo 3200 MB	21,23%
Arquivo 6400 MB	23,68%
Média	19,05%

Diante deste contexto, pode-se afirmar que a primeira e segunda hipóteses são verdadeiras.

7.2 Contribuições

As principais contribuições que este trabalho de doutorado proporciona são as seguintes:

- Criação do algoritmo *Cosphdp* para realizar a alocação conjunta de dados espaciais relacionados, visando o processamento eficiente de junção espacial;

- Adaptação do algoritmo *prioritizeLocations* do *SpatialHadoop* para realizar o escalonamento de tarefas *MapReduce* de acordo com a alocação conjunta de dados espaciais relacionados;
- Criação de uma aplicação *web* para a execução dos algoritmos das novas políticas de alocação conjunta de dados espaciais relacionados e do escalonamento de tarefas para dados espaciais relacionados no *SpatialHadoop*.

As contribuições citadas permitiram um processamento mais eficientes da junção espacial em *clusters* de computadores, usando o *framework SpatialHadoop*.

7.3 Trabalhos futuros

Embora diversas questões tenham sido exploradas, existem possibilidades de melhoria relacionadas a este trabalho. Durante a sua elaboração, foram destacados vários pontos a serem melhorados que são listados como possíveis trabalhos futuros:

- Suporte a outros relacionamentos topológicos entre dois polígonos (tais como, disjunto, toca, contém, cobre), visto que a tese investigou apenas o relacionamento topológico de interseção no processamento de junção espacial;
- Investigar o desbalanceamento de carga devido à alta taxa de interseção em coordenadas geográficas semelhante dos dados espaciais. Em geral, um excesso de arquivos com características geográficas semelhantes leva à distorção na distribuição de dados;
- Investigar o tratamento de dados intermediários que são gerados entre as fases *Map* e *Reduce,* visando otimizar ainda mais o processamento de junção espacial;
- Investigar as novas políticas em ambiente de teste heterogêneo, visto que os testes foram realizados em *clusters* homogêneos;
- Implementar as novas políticas, utilizando o *Yarn* para melhorar o escalonamento de tarefas com o *SpatialHadoop.*

REFERÊNCIAS

ABOUZEID, A. et al. HadoopDB. **Proceedings of the VLDB Endowment**, v. 2, n. 1, p. 922–933, 1 ago. 2009.

AJI, A. et al. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. **Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases**, v. 6, n. 11, p. 1009–1020, ago. 2013.

AL-NAAMI, K. M.; SEKER, S. E.; KHAN, L. GISQAF: MapReduce guided spatial query processing and analytics system. **Software: Practice and Experience**, v. 46, n. 10, p. 1329–1349, out. 2016.

ALARABI, L. et al. **TAREEG**. Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14. **Anais**...New York, New York, USA: ACM Press, 2014Disponível em: <http://dl.acm.org/citation.cfm?doid=2588555.2594528>. Acesso em: 3 out. 2016

ANANTHANARAYANAN, G. et al. Disk-Locality in Datacenter Computing Considered Irrelevant. **HotOS**, p. 1–5, 2011a.

ANANTHANARAYANAN, G. et al. **Scarlett**. Proceedings of the sixth conference on Computer systems - EuroSys '11. **Anais**...New York, New York, USA: ACM Press, 2011bDisponível em: <http://portal.acm.org/citation.cfm?doid=1966445.1966472>. Acesso em: 25 ago. 2016

ANTÔNIO, J.; MACÊDO, F. DE; MACHADO, J. C. Análise em Big Data e um Estudo de Caso utilizando Ambientes de Computação em Nuvem. **Simpósio Brasileiro de Banco de Dados - SBBD 2013**, p. 1–26, 2013.

APACHE. **HBase**. Disponível em: <http://hbase.apache.org/>.

APACHE. **Storm**. Disponível em: <http://storm-project.net/>.

APACHE. **Tez**. Disponível em: <http://tez.apache.org/>.

APACHE. **Giraph**. Disponível em: <http://giraph.apache.org/>.

APACHE. **The Apache Software Foundation**. Disponível em: <http://hadoop.apache.org/>.

AREF, W. G. **Pipelined spatial join processing for quadtree-based indexes**. Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems - GIS '07. **Anais**...: GIS '07.New York, New York, USA: ACM Press, 2007Disponível em: <http://portal.acm.org/citation.cfm?doid=1341012.1341073>. Acesso em: 14 jan. 2015

ARGE, L. et al. **Scalable Sweeping-Based Spatial Join**. Proceedings of the 24rd International Conference on Very Large Data Bases. **Anais**...San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998

ARGE, L. et al. A Unified Approach for Indexed and Non-indexed Spatial Joins. In: **Advances in Database Technology — EDBT 2000**. [s.l.] Springer Berlin Heidelberg, 2000. p. 413–429.

AWAD, A.; KETTERING, B.; SOLIHIN, Y. **Non-volatile memory host controller interface performance analysis in high-performance I/O systems**. 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). **Anais**...IEEE, mar. 2015Disponível em: <http://ieeexplore.ieee.org/document/7095793/>. Acesso em: 20 dez. 2016

BACK, H. et al. Efficient processing of spatial joins with DOT-based indexing. **Information Sciences**, v. 180, n. 8, p. 1292–1312, abr. 2010.

BAE, W. D.; ALKOBAISI, S.; LEUTENEGGER, S. T. An Incremental Refining Spatial Join Algorithm for Estimating Query Results in GIS. In: **Database and Expert Systems Applications**. [s.l.] Springer, 2006. p. 935–944.

BAE, W. D.; ALKOBAISI, S.; LEUTENEGGER, S. T. IRSJ: Incremental refining spatial joins for interactive queries in GIS. **GeoInformatica**, v. 14, n. 4, p. 507–543, 2010.

BARROSO, L. A.; CLIDARAS, J.; HÖLZLE, U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition. **Synthesis Lectures on Computer Architecture**, v. 8, n. 3, p. 1–154, 31 jul. 2013.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, v. 18, n. 9, p. 509–517, 1 set. 1975.

BORGES, K. A. V. Open Geospatial Consortium. **Measurement**, v. 2007, n. 5, p. 1–17, 2011.

BORTHAKUR, D. The hadoop distributed file system: Architecture and design. **Hadoop Project Website**, p. 1–14, 2007.

BRINKHOFF, T. et al. **Multi-step processing of spatial joins**. Proceedings of the 1994 ACM SIGMOD international conference on Management of data - SIGMOD '94. **Anais**...: SIGMOD '94.New York, New York, USA: ACM Press, 1994Disponível em: <http://portal.acm.org/citation.cfm?doid=191839.191880>. Acesso em: 14 jan. 2015

BRINKHOFF, T.; KRIEGEL, H.-P.; SEEGER, B. **Efficient processing of spatial joins using R-trees**. Proceedings of the 1993 ACM SIGMOD international

conference on Management of data - SIGMOD '93. **Anais**...New York, New York, USA: ACM Press, 1993Disponível em: <http://portal.acm.org/citation.cfm?doid=170035.170075>. Acesso em: 26 nov. 2014

BRITO, J. J. **Processamento de consultas SOLAP drill-across e com junção espacial em data warehouses geográficos**. (Dissertação de Mestrado) São Carlos: Universidade de São Paulo, 28 nov. 2012.

BUREAU, U. C. **TIGER Products**. Disponível em: <http://www.census.gov/geo/maps-data/data/tiger.html>.

CÂMARA, G. **Anatomia de sistemas de inforamcao geografica**. 10. ed. Campinas: Escola de Computação, 1996.

CASTILLO, C.; SPREITZER, M.; STEINDER, M. **Towards efficient resource management for data-analytic platforms**. 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops. **Anais**...IEEE, maio 2011Disponível em: <http://ieeexplore.ieee.org/document/5990676/>. Acesso em: 14 abr. 2016

CAULFIELD, A. M. et al. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. **ACM SIGPLAN Notices**, v. 44, n. 3, p. 217, 28 fev. 2009.

CAULFIELD, A. M.; SWANSON, S. **QuickSAN: a storage area network for fast, distributed, solid state disks**. Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2485922.2485962>. Acesso em: 20 dez. 2016

CHAIKEN, R. et al. SCOPE: easy and efficient parallel processing of massive data sets. **Proceedings of the VLDB Endowment**, v. 1, n. 2, p. 1265–1276, 1 ago. 2008.

CHANG, Y.-I.; LIAO, C.-H.; CHEN, H.-L. NA-Trees: A Dynamic Index for Spatial Data *. **JOURNAL OF INFORMATION SCIENCE AND ENGINEERING**, v. 19, p. 103–139, 2003.

CHEN, H.-L.; CHANG, Y.-I. Spatial joins based on NA-trees. **Information Processing Letters**, v. 109, n. 13, p. 713–718, jun. 2009.

CHEN, J.; WANG, H.; LU, H. Research for GIS Raster Data Storage and Display under the FrameWork of Private Cloud. **2012 International Conference on Computer Science and Electronics Engineering**, p. 449–452, mar. 2012.

CHENT, R.; SHAO, Z.; LI, T. **Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach**. 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). **Anais**...IEEE, out. 2016Disponível em: <http://ieeexplore.ieee.org/document/7783712/>. Acesso em: 20 dez. 2016

CHUN, B.-G. et al. REEF: retainable evaluator execution framework. **Proceedings of the VLDB Endowment**, v. 6, n. 12, p. 1370–1373, 28 ago. 2013.

CIFERRI, R. R. **Análise da Influência do Fator Distribuição Espacial dos Dados no Desempenho de Métodos de Acesso Multidimensionais**. (Tese de Doutorado) Recife: Universidade Federal de Pernambuco, 2002.

CLEMENTINI, E.; FELICE, P.; OOSTEROM, P. A small set of formal topological relationships suitable for end-user interaction. In: [s.l.] Springer, Berlin, Heidelberg, 1993. p. 277–295.

CURRAN, K.; CRUMLISH, J.; FISHER, G. OpenStreetMap. **International Journal of Interactive Communication Systems and Technologies**, v. 2, n. 1, p. 69–78, 2012.

DE BERG, M. et al. **Computational Geometry: Algorithms and Applications**. [s.l.] Springer-Verlag, 2008. v. 17

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107, 1 jan. 2008.

DEBNATH, B. K. Spatial join. **Department of Electrical and Computer Engineering - University of Minnesota**, 2005.

DEMCHENKO, Y. et al. **Addressing big data issues in Scientific Data Infrastructure**. 2013 International Conference on Collaboration Technologies and Systems (CTS). **Anais**...IEEE, maio 2013Disponível em: <http://ieeexplore.ieee.org/document/6567203/>. Acesso em: 20 abr. 2016

DESPAIN, A. M.; PATTERSON, D. A. **X-Tree: A tree structured multi-processor computer architecture**. Proceedings of the 5th annual symposium on Computer architecture - ISCA '78. **Anais**...New York, New York, USA: ACM Press, 1978Disponível em: <http://portal.acm.org/citation.cfm?doid=800094.803041>. Acesso em: 29 dez. 2016

DITTRICH, J. et al. Hadoop++. **Proceedings of the VLDB Endowment**, v. 3, n. 1–2, p. 515–529, 1 set. 2010.

DITTRICH, J. et al. Only aggressive elephants are fast elephants. **Proceedings of the VLDB Endowment**, v. 5, n. 11, p. 1591–1602, 1 jul. 2012.

DOULKERIDIS, C.; NØRVÅG, K. A survey of large-scale analytical query processing in MapReduce. **The VLDB Journal**, v. 23, n. 3, p. 355–380, 8 jun. 2014.

EGENHOFER, M. A model for detailed binary topological relationships. **Geomatica**, v. 47, n. 3–4, p. 261–273, 1993.

EGENHOFER, M.; HERRING, J. **Categorizing binary topological relations between regions, lines, and points in geographic databasesThe**. [s.l: s.n.]. Disponível em: <http://didattica.univaq.it/moodle/file.php/1508/9intReport_Egenhofer_1992.pdf>.

EGENHOFER, M. J.; CLEMENTINI, E.; DI FELICE, P. Topological relations between regions with holes. **International Journal of Geographical Information Science**, v. 8, n. 2, p. 129–142, 1994.

ELDAWY, A. et al. **CG_Hadoop: Computational geometry in MapReduce**. Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL'13. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2525314.2525349>

ELDAWY, A. **SpatialHadoop: towards flexible and scalable spatial processing using mapreduce**. Proceedings of the 2014 SIGMOD PhD symposium on - SIGMOD'14 PhD Symposium. **Anais**...New York, New York, USA: ACM Press, 2014Disponível em: <http://dl.acm.org/citation.cfm?doid=2602622.2602625>. Acesso em: 24 out. 2014

ELDAWY, A. et al. **SHAHED: A MapReduce-based system for querying and visualizing spatio-temporal satellite data**. 2015 IEEE 31st International Conference on Data Engineering. **Anais**...IEEE, abr. 2015aDisponível em: <http://ieeexplore.ieee.org/document/7113427/>. Acesso em: 3 out. 2016

ELDAWY, A. et al. **A demonstration of Shahed: A MapReduce-based system for querying and visualizing satellite data**. 2015 IEEE 31st International Conference on Data Engineering. **Anais**...IEEE, abr. 2015bDisponível em: <http://ieeexplore.ieee.org/document/7113397/>. Acesso em: 3 out. 2016

ELDAWY, A.; ALARABI, L.; MOKBEL, M. F. Spatial partitioning techniques in SpatialHadoop. **Proceedings of the VLDB Endowment**, v. 8, n. 12, p. 1602–1605, 1

ago. 2015.

ELDAWY, A.; MOKBEL, M. F. A demonstration of SpatialHadoop. **Proceedings of the VLDB Endowment**, v. 6, n. 12, p. 1230–1233, 28 ago. 2013.

ELDAWY, A.; MOKBEL, M. F. **Pigeon: A spatial MapReduce language**. 2014 IEEE 30th International Conference on Data Engineering. **Anais**...IEEE, mar. 2014Disponível em: <http://ieeexplore.ieee.org/document/6816751/>. Acesso em: 3 out. 2016

ELDAWY, A.; MOKBEL, M. F. **SpatialHadoop: A MapReduce framework for spatial data**. 2015 IEEE 31st International Conference on Data Engineering. **Anais**...IEEE, abr. 2015aDisponível em: <http://ieeexplore.ieee.org/document/7113382/>. Acesso em: 3 out. 2016

ELDAWY, A.; MOKBEL, M. F. The ecosystem of SpatialHadoop. **SIGSPATIAL Special**, v. 6, n. 3, p. 3–10, 22 abr. 2015b.

ELTABAKH, M. Y. et al. CoHadoop. **Proceedings of the VLDB Endowment**, v. 4, n. 9, p. 575–585, 1 jun. 2011.

ESRI. **ArcGIS Desktop: Release 10.2Redlands CA**, 2013.

FAHMY, M. M.; ELGHANDOUR, I.; NAGI, M. **CoS-HDFS: co-locating geo-distributed spatial data in hadoop distributed file system**. Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies - BDCAT '16. **Anais**...New York, New York, USA: ACM Press, 2016Disponível em: <http://dl.acm.org/citation.cfm?doid=3006299.3006314>. Acesso em: 27 dez. 2016

FEDOTOVSKY, P. et al. **To sort or not to sort: The evaluation of R-Tree and B+-tree in transactional environment with ordered result set requirement**.

CEUR Workshop Proceedings. **Anais**...2013

FORNARI, M. **Análise e desenvolvimento de um novo algoritmo de junção espacial para SGBD geográficos**. (Tese de Doutorado) Rio Grande do Sul: Universidade Federal do Rio Grande do Sul., 2006.

GAEDE, V.; GÜNTHER, O. Multidimensional access methods. **ACM Computing Surveys**, v. 30, n. 2, p. 170–231, 1 jun. 1998.

GARDELS, K. **The Open GIS Approach to Distributed Geodata and Geoprocessing**. Third International Conference/Workshop on Integrating GIS and Environmental Modeling. **Anais**...Santa Fe, NM, USA: 1996

GARGANTINI, I. An effective way to represent quadtrees. **Communications of the ACM**, v. 25, n. 12, p. 905–910, 1 dez. 1982.

GATTI, S. D. **Fatores que Afetam o Desempenho de Métodos de Junções Espaciais: um Estudo Baseado em Dados Reais**. (Dissertação de Mestrado) Campinas: Universidade Estadual de Campinas, 2000.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. **ACM SIGOPS Operating Systems Review**, v. 37, n. 5, p. 29, 1 dez. 2003.

GOLDMAN, A. et al. Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. **VII Jornadas de Atualização em Informática**, p. 86–136, 2012.

GRAY, P. M. D. et al. Parallel Database Management. In: **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 2026–2029.

GUNTHER, O. **Efficient computation of spatial joins**. Proceedings of IEEE 9th International Conference on Data Engineering. **Anais**...IEEE Comput. Soc.

Press, 1993Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=344078>. Acesso em: 14 jan. 2015

GUNTHER, O. et al. **Benchmarking spatial joins a la carte**. Proceedings. Tenth International Conference on Scientific and Statistical Database Management (Cat. No.98TB100243). **Anais**...IEEE Comput. Soc, 1998Disponível em: <http://ieeexplore.ieee.org/document/688109/>

GUO, Z.; FOX, G.; ZHOU, M. **Investigation of Data Locality in MapReduce**. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). **Anais**...IEEE, maio 2012Disponível em: <http://ieeexplore.ieee.org/document/6217449/>. Acesso em: 18 abr. 2016

GURRET, C.; RIGAUX, P. **The sort/sweep algorithm: a new method for R-tree based spatial joins**. Proceedings. 12th International Conference on Scientific and Statistica Database Management. **Anais**...IEEE Comput. Soc, 2000Disponível em: <http://ieeexplore.ieee.org/document/869785/>

GUTTMAN, A. R-trees. **ACM SIGMOD Record**, v. 14, n. 2, p. 47, 1 jun. 1984.

HASHEM, I. A. T. et al. MapReduce: Review and open challenges. **Scientometrics**, v. 109, n. 1, p. 389–422, 15 out. 2016.

HOTT, B.; ROCHA, R.; GUEDES, D. Escalonamento de processos sensíveì a localidade de dados em sistemas de arquivos distribuídos. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**, 2016.

IBRAHIM, S. et al. **Maestro: Replica-Aware Map Scheduling for MapReduce**. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). **Anais**...IEEE, maio 2012Disponível em: <http://ieeexplore.ieee.org/document/6217451/>. Acesso em: 26 set. 2016

INTERFACE, W. **SHAHED**.

ISARD, M. et al. **Dryad: distributed data-parallel programs from sequential building blocks**. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 - EuroSys '07. **Anais**...New York, New York, USA: ACM Press, 2007Disponível em: <http://dl.acm.org/citation.cfm?id=1273005>. Acesso em: 28 nov. 2014

ISARD, M. et al. **Quincy**. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09. **Anais**...New York, New York, USA: ACM Press, 2009Disponível em: <http://portal.acm.org/citation.cfm?doid=1629575.1629601>. Acesso em: 26 out. 2016

JACOX, E. H.; SAMET, H. **Iterative spatial joinACM Transactions on Database Systems**, 2003.

JACOX, E. H.; SAMET, H. Spatial join techniques. **ACM Transactions on Database Systems**, v. 32, n. 1, p. 7–es, 1 mar. 2007.

JEON, H.; EL MAGHRAOUI, K.; KANDIRAJU, G. B. **Investigating hybrid SSD FTL schemes for Hadoop workloads**. Proceedings of the ACM International Conference on Computing Frontiers - CF '13. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2482767.2482793>. Acesso em: 20 dez. 2016

JIA, Z. et al. **Characterizing and subsetting big data workloads**. IISWC 2014 - IEEE International Symposium on Workload Characterization. **Anais**...IEEE, out. 2014Disponível em: <http://ieeexplore.ieee.org/document/6983058/>. Acesso em: 20 dez. 2016

JIANG, D. et al. The performance of MapReduce. **Proceedings of the VLDB Endowment**, v. 3, n. 1–2, p. 472–483, 1 set. 2010.

JIANG, D.; TUNG, A. K. H.; CHEN, G. MAP-JOIN-REDUCE: Toward Scalable and Efficient Data Analysis on Large Clusters. **IEEE Transactions on Knowledge and Data Engineering**, v. 23, n. 9, p. 1299–1311, set. 2011.

JIONG XIE et al. **Improving MapReduce performance through data placement in heterogeneous Hadoop clusters**. 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). **Anais**...IEEE, abr. 2010Disponível em: <http://ieeexplore.ieee.org/document/5470880/>. Acesso em: 12 abr. 2016

KAMEL, I.; FALOUTSOS, C. **Hilbert R-tree: An Improved R-tree using Fractals**. International Conference on Very Large Databases (VLDB). **Anais**...1994

KOCBERBER, O. et al. **Meet the walkers: Accelerating index traversals for inmemory databases**. Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2540708.2540748>. Acesso em: 20 dez. 2016

KORTH, H.; SILBERSCHATZ, A. Sistemas de bancos de dados. **São Paulo. Ed. Makron Books**, 1999.

KOUDAS, N.; SEVCIK, K. C. Size separation spatial join. **ACM SIGMOD Record**, v. 26, n. 2, p. 324–335, 1 jun. 1997.

KUMAR, K. et al. Measurement of Viscoelastic Properties of Polyacrylamide-Based Tissue-Mimicking Phantoms for Ultrasound Elastography Applications. **IEEE Transactions on Instrumentation and Measurement**, v. 59, n. 5, p. 1224–1232, maio 2010.

LI, Z. et al. RPK-table based efficient algorithm for join-aggregate query on MapReduce. **CAAI Transactions on Intelligence Technology**, v. 1, n. 1, p. 79–89, jan. 2016.

LO, M.-L.; RAVISHANKAR, C. V. Spatial joins using seeded trees. **ACM SIGMOD Record**, v. 23, n. 2, p. 209–220, 1 jun. 1994.

LO, M.-L.; RAVISHANKAR, C. V. Spatial hash-joins. **ACM SIGMOD Record**, v. 25, n. 2, p. 247–258, 1 jun. 1996.

LU, J.; GUTING, R. H. **Parallel Secondo: Boosting Database Engines with Hadoop**. 2012 IEEE 18th International Conference on Parallel and Distributed Systems. **Anais**...IEEE, dez. 2012Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6413613>. Acesso em: 29 out. 2014

LU, W. et al. Efficient Processing of k Nearest Neighbor Joins using MapReduce. **Proceedings of the VLDB Endowment**, v. 5, n. 10, p. 1016–1027, 30 jun. 2012.

MA, Q. et al. **Query processing of massive trajectory data based on mapreduce**. Proceeding of the first international workshop on Cloud data management - CloudDB '09. **Anais**...New York, New York, USA: ACM Press, 2009Disponível em: <http://portal.acm.org/citation.cfm?doid=1651263.1651266>

MANOLOPOULOS, Y. et al. **R-Trees: Theory and Applications**. London: Springer London, 2006.

MANOULIS, N. Spatial join. In: **Encyclopedia of Database Systems**. [s.l: s.n.].

MANOULIS, N.; THEODORIDIS, Y.; PAPADIAS, D. Spatial joins: Algorithms, cost models and optimization techniques. In: **Spatial Databases: Technologies, Techniques and Trends, Harpenden, Hertfordshire**. Spatial Da ed. [s.l: s.n.]. p. 155–184.

MANYIKA, M. BYJAMES et al. **Big data: The next frontier for innovation, competition, and productivity**. [s.l: s.n.].

MCKENNEY, M.; SCHNEIDER, M. **Topological relationships between map geometries**. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). **Anais**...2008

MOKBEL, M. F. et al. MNTG: An Extensible Web-Based Traffic Generator. In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**. [s.l.] Springer Berlin Heidelberg, 2013. v. 8098 LNCSp. 38–55.

MOKBEL, M. F. et al. **A demonstration of MNTG - A web-based road network traffic generator**. 2014 IEEE 30th International Conference on Data Engineering. **Anais**...IEEE, mar. 2014Disponível em: <http://ieeexplore.ieee.org/document/6816752/>. Acesso em: 24 out. 2014

NAAMI, K. M. AL; SEKER, S.; KHAN, L. **GISQF: An Efficient Spatial Query Processing System**. 2014 IEEE 7th International Conference on Cloud Computing. **Anais**...IEEE, jun. 2014Disponível em: <http://ieeexplore.ieee.org/document/6973802/>. Acesso em: 11 out. 2016

NIEVERGELT, J.; HINTERBERGER, H.; SEVCIK, K. C. The Grid File: An Adaptable, Symmetric Multikey File Structure. **ACM Transactions on Database Systems**, v. 9, n. 1, p. 38–71, 23 jan. 1984.

NISHANTH S et al. **CoHadoop++: A load balanced data co-location in Hadoop Distributed File System**. 2013 Fifth International Conference on Advanced Computing (ICoAC). **Anais**...IEEE, dez. 2013Disponível em: <http://ieeexplore.ieee.org/document/6921934/>. Acesso em: 1 ago. 2016

NISHIMURA, S. et al. MD-HBase: Design and Implementation of an Elastic Data Infrastructure for Cloud-scale Location Services. **Distributed and Parallel …**,

p. 289–319, 2013.

OGC. **Open Geospatial Consortium**. Disponível em: <http://www.opengeospatial.org/>.

OLSTON, C. et al. **Pig Latin: A Not-So-Foreign Language for Data Processing**. Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08. **Anais**...New York, New York, USA: ACM Press, 2008Disponível em: <http://portal.acm.org/citation.cfm?doid=1376616.1376726>

ORENSTEIN, J. A. Spatial query processing in an object-oriented database system. **ACM SIGMOD Record**, v. 15, n. 2, p. 326–336, 15 jun. 1986.

OUSTERHOUT, J. et al. The case for RAMClouds. **ACM SIGOPS Operating Systems Review**, v. 43, n. 4, p. 92, 27 jan. 2010.

PAIVA, J. Topological equivalence and similarity in multi-representation geographic databases. 1998.

PALANISAMY, B. et al. **Purlieus**. Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11. **Anais**...New York, New York, USA: ACM Press, 2011Disponível em: <http://dl.acm.org/citation.cfm?doid=2063384.2063462>. Acesso em: 13 abr. 2016

PAPADOPOULOS, A.; RIGAUX, P.; SCHOLL, M. A performance evaluation of spatial join processing strategies. **Advances in Spatial Databases**, 1999.

PATEL, J. M.; DEWITT, D. J. **Partition based spatial-merge joinACM SIGMOD Record**, 1996.

PAVLO, A. et al. **A comparison of approaches to large-scale data analysis**. Proceedings of the 35th SIGMOD international conference on

Management of data - SIGMOD '09. **Anais**...New York, New York, USA: ACM Press, 2009Disponível em: <http://portal.acm.org/citation.cfm?doid=1559845.1559865>. Acesso em: 14 abr. 2016

PHILIP CHEN, C. L.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. **Information Sciences**, v. 275, p. 314–347, ago. 2014.

PREPARATA, F. P.; SHAMOS, M. I. **Computational geometry: an introduction**. [s.l.] Springer-Verlag, 1985. v. 47

RASOOLI, A.; DOWN, D. G. COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems. **Future Generation Computer Systems**, v. 36, p. 1–15, jul. 2014.

RICHTER, S. et al. Towards Zero-Overhead Adaptive Indexing in Hadoop. 14 dez. 2012.

ROBINSON, J. T. The KDB-tree: a search structure for large multidimensional dynamic indexes. **Proceedings of the 1981 ACM SIGMOD international conference on Management of data - SIGMOD '81**, p. 10–18, 1981.

ROCHA, S. R. **Análise de Desempenho de Junções Espaciais: Uma Comparação entre os Métodos Analítico e Experimental**. (Dissertação de Mestrado) Campinas: Unicamp, 2003.

SAMET, H. The Quadtree and Related Hierarchical Data Structures. **ACM Computing Surveys**, v. 16, n. 2, p. 187–260, 1 jun. 1984.

SELLIS, T. K.; ROUSSOPOULOS, N.; FALOUTSOS, C. **The R+-tree: A Dynamic Index for Multi-dimensional Objects**. (P. M. Stocker, W. Kent, P. Hammersley, Eds.)International Conference on Very Large Databases (VLDB).

Anais...Morgan Kaufmann, 1987

SENTHILKUMAR, M.; ILANGO, P. A Survey on Job Scheduling in Big Data. **Cybernetics and Information Technologies**, v. 16, n. 3, p. 41–49, 1 jan. 2016.

SHOWCASE, R. et al. DOT: A Spatial Access Method Using Fractals. **Int. Conference on Data Engineering**, p. 152–159, 1991.

SHVACHKO, K. et al. **The Hadoop Distributed File System**. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). **Anais**...IEEE, maio 2010Disponível em: <http://cloudcomputing.googlecode.com/svn/trunk/??/Hadoop_0.18_doc/hdfs_design.pdf>

SILVA, Y. N.; XIONG, X.; AREF, W. G. The RUM-tree: Supporting frequent updates in R-trees using memos. **VLDB Journal**, v. 18, n. 3, p. 719–738, 20 jun. 2009.

SOUSA, F.; MOREIRA, L.; MACHADO, J. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **Ceará: Universidade Federal do …**, 2009.

SUN, C.; AGRAWAL, D.; EL ABBADI, A. Hardware acceleration for spatial selections and joins. **Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03**, v. 2, p. 455, 2003.

THUSOO, A. et al. Hive - A Warehousing Solution Over a Map-Reduce Framework. **Proceedings of the VLDB Endowment**, v. 2, n. 2, p. 1626–1629, 1 ago. 2009.

VAVILAPALLI, V. K. et al. **Apache Hadoop YARN**. Proceedings of the 4th annual Symposium on Cloud Computing - SOCC '13. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em:

<http://dl.acm.org/citation.cfm?doid=2523616.2523633>

WEI, Q. et al. **CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster**. 2010 IEEE International Conference on Cluster Computing. **Anais**...IEEE, set. 2010Disponível em: <http://ieeexplore.ieee.org/document/5600309/>. Acesso em: 12 abr. 2016

WHITE, T. Hadoop: The definitive guide. **Online**, v. 54, p. 258, 2012.

YAMAGUCHI, K. et al. Octree-Related Data Structures and Algorithms. **IEEE Computer Graphics and Applications**, v. 4, n. 1, p. 53–59, 1984.

YAO, Y. et al. **HaSTE: Hadoop YARN Scheduling Based on Task-Dependency and Resource-Demand**. 2014 IEEE 7th International Conference on Cloud Computing. **Anais**...IEEE, jun. 2014Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6973740>. Acesso em: 27 out. 2016

ZAHARIA, M. et al. **Spark : Cluster Computing with Working Sets**. HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. **Anais**...2010a

ZAHARIA, M. et al. **Delay scheduling**. Proceedings of the 5th European conference on Computer systems - EuroSys '10. **Anais**...New York, New York, USA: ACM Press, 2010bDisponível em: <http://portal.acm.org/citation.cfm?doid=1755913.1755940>. Acesso em: 18 abr. 2016

ZHANG, C.; LI, F.; JESTES, J. **Efficient parallel kNN joins for large data in MapReduce**. Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12. **Anais**...New York, New York, USA: ACM Press, 2012Disponível em: <http://dl.acm.org/citation.cfm?id=2247596.2247602>. Acesso em: 28 nov. 2014

ZHANG, S. et al. **SJMR: Parallelizing spatial join with MapReduce on clusters**. 2009 IEEE International Conference on Cluster Computing and Workshops. **Anais**...IEEE, 2009Disponível em: <http://ieeexplore.ieee.org/document/5289178/>

ZHAO, H. et al. **K%-Fair scheduling: A flexible task scheduling strategy for balancing fairness and efficiency in MapReduce systems**. Proceedings of 2012 2nd International Conference on Computer Science and Network Technology. **Anais**...IEEE, dez. 2012Disponível em: <http://ieeexplore.ieee.org/document/6526015/>. Acesso em: 14 abr. 2016

ZHOU, J. et al. SCOPE: parallel databases meet MapReduce. **The VLDB Journal**, v. 21, n. 5, p. 611–636, 28 out. 2012.

ZHOU, J. et al. **SUORA: A Scalable and Uniform Data Distribution Algorithm for Heterogeneous Storage Systems**. 2016 IEEE International Conference on Networking, Architecture and Storage (NAS). **Anais**...IEEE, ago. 2016Disponível em: <http://ieeexplore.ieee.org/document/7549423/>. Acesso em: 30 ago. 2016

ZHOU, M.; XU, C. Optimized Data Placement for Column-Oriented Data Store in the Distributed Environment. In: **Proceedings of the 16th international conference on Database systems for advanced applications**. [s.l.] Springer-Verlag, 2011. p. 440–452.