UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FELIPE BIANCHI DA SILVA

TÉCNICAS OTIMIZADAS DE MINERAÇÃO DE DADOS COM HADOOP

PATO BRANCO

2023

FELIPE BIANCHI DA SILVA

TÉCNICAS OTIMIZADAS DE MINERAÇÃO DE DADOS COM HADOOP

Optimized data mining techniques with Hadoop

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Ives Rene Venturini Pola

PATO BRANCO

2023

FELIPE BIANCHI DA SILVA

TÉCNICAS OTIMIZADAS DE MINERAÇÃO DE DADOS COM HADOOP

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 22/junho/2023

Prof. Dr. Ives Rene Venturini Pola
Universidade Tecnológica Federal do Paraná

Prof. Dr. Luis Cassiano Goularte Rista
Universidade Tecnológica Federal do Paraná

Prof. Dr. Fabio Favarim
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2023

RESUMO

Na atualidade é indiscutível que os dados estão aumentando de forma exponencial e são provenientes das mais diversas fontes, esta expansão acarreta a criação de uma massa de dados complexos. Essa massa de dados é denominada *Big Data* e atrelado a esse conceito existe a necessidade da criação de uma arquitetura de dados que suporte a extração de informações em um grande e variável volume de dados e ainda que faça a transformação desses dados em informações relevantes para os mais variáveis seguimentos, de forma rápida. Essa nova realidade exige novos meios para acompanhar a complexidade dos dados, dentre estas está a computação paralela e distribuída que utiliza um aglomerado de computadores. Entretanto esta forma de computação exige do usuário conhecimento especifico e neste contexto surgiu o *Apache Hadoop* com a finalidade de resolver os problemas da computação distribuída, através da aplicação de um único arcabouço de código aberto buscando isolar o programador que trabalha com grandes quantidades de dados da necessidade de tratar os problemas tradicionais da computação distribuída. O objetivo deste trabalho é aplicar técnicas de mineração de dados em *cluster* em conjunto com técnicas de particionamento no arcabouço *Hadoop* e verificar a acurácia dos dados e tempo de execução de obtidos com esta técnica em comparação a utilização de um único computador, na busca de demonstrar a eficácia do *Hadoop* em armazenar e tratar esses dados, que estão na proporção de petabytes diariamente, e a sua capacidade de diminuir custos de sistemas de armazenamento e crescimento da capacidade de processamento.

Palavras-chave: *hadoop*; mineração; *big data*; particionamento.

ABSTRACT

Nowadays it is indisputable that the data is increasing exponentially and comes from the most diverse sources, this expansion leads to the creation of a mass of complex data. This mass of data is called Big Data and, linked to this concept, there is a need to create a data architecture that supports the extraction of information in a large and variable volume of data and even that makes the transformation of this data into information relevant to the more variable segments, quickly. This new reality requires computing new ways to keep up with the complexity of the data, among which is parallel and distributed computing, which uses a cluster of computers. However, this form of computation requires a specific knowledge from the user and in this context the Apache Hadoop appeared in order to solve the problems of distributed computing, through the application of a single open source framework seeking to isolate the programmer who works with large amounts of data from the need to address the traditional problems of distributed computing. The objective of this work is to apply data mining techniques in cluster together with partitioning techniques in the Hadoop framework and to verify the accuracy of the data and execution time obtained with this technique in comparison to the use of a single computer, in an attempt to demonstrate the effectiveness of Hadoop in storing and processing this data, which is in the proportion of petabytes daily, and its ability to reduce storage system costs and increase processing capacity.

Keywords: hadoop; mining; big data; partitioning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A expansão dos dados no mundo é inegável e provenientes de diversas fontes, segundo Machado (2018) podem ser de aplicações corporativas, Internet, redes sociais, equipamentos RFID, câmeras de segurança, entre outros, gerando uma massa de dados complexos, podendo ou não serem estruturados. Segundo White (2012), os meios citados anteriormente chegam a produzir volume de dados na ordem de *petabytes* diariamente.

Esse crescimento e a grande quantidade de dados é o que denomina-se *Big Data* e evidencia a necessidade de uma nova arquitetura de dados que busque extrair informações em uma grande variedade e volume de dados, e com velocidade de processamento, afim de transformá-las em informações relevantes para os mais diversos segmentos (MACHADO, 2018).

Nos últimos anos a redução do custo dos sistemas de armazenamento de dados e o crescimento da capacidade de processamento permitiram o alastramento da informação digital e o seu processamento (MACHADO, 2018).

Entretanto, o crescimento dos dados e a sua complexidade não acompanharam os avanços tecnológicos, e a utilização da computação paralela e distribuída surge como uma alternativa tanto para a alocação quando para tratamento destes, na busca de amenizar os problemas encontrados nas arquiteturas convencionais. Essa computação geralmente se utiliza de um aglomerado de computadores, conhecido como *cluster*, sendo cada unidade de computador chamado de nó, tendo assim um poder de processamento e armazenamento maior, porém com um custo relativamente menor (GOLDMAN *et al.*, 2012).

Apesar de todas as vantagens de utilização, a computação paralela e distribuída necessita que o novo usuário tenha um grande volume de conhecimento, a divisão de uma tarefa em subtarefas, no tamanho correto para não sobrecarregar um dos computadores do aglomerado, coordenar o acesso a uma determinada região de memória e coordenar o fim de cada uma das subtarefas (GOLDMAN *et al.*, 2012).

Neste contexto, desenvolveu-se o *Apache Hadoop*, que tem por finalidade resolver os problemas da computação distribuída em um único arcabouço (*framework*) de código aberto, isolando o programador que busca trabalhar com o tratamento de grandes quantidades de dados da necessidade de tratar os problemas de integridade dos dados, disponibilidade dos nós, escalabilidade da aplicação e recuperação de falhas (GOLDMAN *et al.*, 2012).

De acordo com Goldman *et al.* (2012) seu desenvolvimento tem como base dois trabalhos produzidos pela empresa *Google*, *The Google Files System* e *MapReduce: Simplified Data Processing on Large Clusters*. Goldschmidt, Passos e Bezerra (2015) elucida que tal arcabouço foi inspirado “(...) no modelo de processamento da *Google*, tal ambiente funciona sobre um sistema de arquivos organizados em *clusters* distribuídos e se baseia no modelo *MapReduce* para implementar as tarefas desejadas”.

Os principais componentes que implementam os conceitos básicos de armazenamento e processamento distribuído são: o HDFS (*Hadoop Distributed File System*), responsável por gerenciar os dados armazenados em discos no *cluster*, e o YARN (*Yet Another Resource Negotiator*), que atua como gerenciador dos recursos, ou seja a disponibilidade de processamento e memória dos nós que realizam as tarefas (BENGFORT; KIM, 2019).

Já *MapReduce*, foi um modelo de programação proposto em 2004 por Jeffrey Dean and Sanjay Ghemawat e publicado no artigo *MapReduce: Simplified Data Processing on Large Clusters*, que tem o propósito de processar grandes volumes de dados de forma paralela (GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

Um exemplo de funcionamento do *MapReduce* é o censo de cidadãos romanos: a população foi dividida e cada um dos responsáveis por fazer a apuração recebeu uma parcela aonde deveria realizar a contagem. Desta forma o levantamento ocorreu paralelamente e ao final, todos informaram os valores que haviam obtido e como resultado, o coordenador geral pode calcular o valor total (GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

O *MapReduce* foi projetado para trabalhar com a aplicação em duas funções distintas: mapeamento (*Map*) e redução (*Reduce*) (MACHADO, 2018). Durante a etapa de *Map* é possível aplicar técnicas de mineração de dados que serão apresentados no resultado do estágio *Reduce*.

Em Tan *et al.* (2009) “A modelagem de previsão se refere à tarefa de construir um modelo para a variável alvo como uma função das variáveis explicativas”. Subdividindo-as em: classificação, neste caso a variável alvo é discreta; e regressão para variáveis contínuas, ambas com o enfoque encontrar um determinado modelo em que o valor real e o previsto tenham o erro minimizado. Para exemplificar a diferença entre as subdivisões, em uma loja virtual saber se o cliente finalizará um pedido é uma tarefa de classificação, já prever o valor que ele gastará é uma tarefa de regressão.

Dentre os vários algoritmos de classificação Goldschmidt, Passos e Bezerra (2015) apresenta alguns dos diversos existentes, como C4.5, K-Nearest Neighbor e Classificadores Bayesianos, complementado a essa informação destaca o teorema NFL (*No Free Lunch Theorem*) ressaltando que não há um algoritmo que seja superior aos demais levando em conta todos os problemas de Classificação existentes.

Um classificador de vizinhos mais próximos (KNN - *K-Nearest Neighbor*) representa cada instância da base de dados em um espaço d-dimensional, onde d é o número de atributos. Dado um exemplo teste, calcula-se a distância, utilizando alguma forma de medida (Euclidiana, Mahalanobis, entre outras), deste a todos os dados e o k números de vizinhos mais próximos da instância definirá a classe que esta faz parte (TAN *et al.*, 2009).

Este trabalho visa elaborar um classificador KNN utilizando-se um *cluster Hadoop* com estratégias de particionamento dos nós. Além disso, visa também elaborar consultas por similaridade num *cluster Hadoop*. Serão utilizadas como *datasets* imagens representadas por

vetores de características geradas pelo descritor SIFT (*Scale Invariant Feature Transform*) e outras bases relevantes que possam existir *clusters* na sua distribuição.

Dentro do HDFS, o trabalho visará encontrar uma forma de particionar os dados para que os nós que não contenham os dados de interesse não sejam utilizados, economizando processamento desnecessário.

1.1 Objetivo Geral

Este trabalho busca aplicar técnicas otimizadas de mineração de dados em *cluster* com o arcabouço *Hadoop* e verificar os tempos de consulta e processamentos das tarefas de mineração em decorrência do aumento de nós de dados e particionamento do banco de dados.

1.2 Objetivos Específicos

- Elaborar técnicas de particionamento de dados para suprir as entradas dos *mappers*.
- Armazenar o *dataset* em um Sistema de Gestão de Base de Dados que suporte o *Hadoop* (*Hive*).
- Elaborar técnicas otimizadas de recuperação de informação (vizinhos mais próximos, consulta por abrangência) e tarefas de mineração de dados dentro do arcabouço *Hadoop*.

2 REFERENCIAL TEÓRICO

2.1 *Big Data*

Para Jain (2017) “’*Big Data*"descreve conjuntos de dados tão grandes e complexos, que eles são impossíveis de gerenciar com as ferramentas de software tradicionais“. E afirma que com as tecnologias atuais é possível obter interpretações dos dados de um *Big Data*, como o *Apache Hadoop*. Para ilustrar a transformação de dados em valores, o autor usa do exemplo de vendedores que conseguem informações dos possíveis clientes na rede com o intuito de extrair as tendências comportamentais e delas moldar seus produtos e campanhas para atender o público alvo. Um *Big Data* não necessariamente precisar ser grande, mas sua estrutura deve ser complexa o suficiente para que bancos de dados relacionais não consigam armazenar.

Porém o conceito do que é um *Big Data* não é algo único a todos, diversos artigos e materiais tem as suas perspectivas deste conjunto de processos e tecnologias(MACHADO, 2018). Com isto alguns autores associam o *Big Data* e as suas técnicas em três dimensões básicas.(GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

2.1.1 Os três V’s

Goldschmidt, Passos e Bezerra (2015) explana que para compreender melhor *Big Data* e as suas técnicas, separa-se em três partes básicas, chamada de três V’s:

- Volume: O volume de dados é grande e pode estar em constante crescimento, com isto o tratamento e análise a este grande conjunto se faz necessário.
- Variedade: Como os dados apresentam grande heterogenia, tanto os formatos quanto as fontes, as técnicas de armazenamento devem estar aptas para reunir e organiza-los de forma eficiente.
- Velocidade: Pelo fato de diversas fontes incrementarem o conjunto de dados se torna muito complicado estipular a velocidade em que cada dado chegará, então as técnicas de análise devem ser rápidas tanto quanto a coleta de informações.

Em Machado (2018) é acrescentado dois V’s para conseguir explicar melhor o que é um *Big Data*, que são: Veracidade onde a importância em se verificar se o determinado dado é verdadeiro e autêntico, em relação sua fonte; E o Valor, segundo Goldschmidt, Passos e Bezerra (2015) é onde se busca transformar os dados dentro de um conjunto para formar um conhecimento para a aplicação.

2.1.2 Os tipos de dados em um *Big Data*

Jain (2017) considera três tipos de *Big Data*: dados estruturados; dados semi-estruturados; e dados não estruturados.

2.1.2.1 Dados Estruturados

Os dados estruturados são encontradas principalmente em bancos de dados relacionais, sendo utilizado tabelas para organizar esses dados. Frequentemente gerenciados utilizando a linguagem SQL (*Structered Query Language*[1]), uma linguagem de programação, desenvolvida nos primeiros anos de 1970 por a empresa IBM, para fazer consultas e gerenciar um SGBD(Sistema de Gestão de Banco de Dados) (JAIN, 2017).

Estes, antes de serem alocados, necessariamente precisam de um modelo que informe quais serão cada um de seus campos, os tipos de dados armazenados e as restrições que eles podem ter, a exemplo: o campo nome, seu tipo é uma cadeia de caracteres e o número de caracteres máximo, apresentam respectivamente os itens comentados. A sua principal vantagem perante aos outros tipos de dados é a facilidade em que podem ser inseridos, armazenados, consultados e analisados. Já que as primeiras tecnologias tinham o custo de armazenamento muito alto e não existia um poder alto de processamento, os bancos de dados relacionais e os dados estruturados eram a melhor forma de gerenciar e analisar estas informações (JAIN, 2017).

2.1.2.2 Dados Semi Estruturados

Este tipo de dados são aqueles que não são frequentemente encontrados em banco de dados relacionais, porém tem uma forma de organização que se torna mais fácil de analisar e guardar seu conteúdo. Exemplos são: CSV (*Comma-separated values*)[2], XML (*Extensible Markup Language*)[3] e JSON (*JavaScript Object Notation*)[4]. Há alguns dados semi estruturados que podem ser armazenados em um SGBD, porém com algumas dificuldades (JAIN, 2017).

2.1.2.3 Dados Não estruturados

Os dados não estruturados são dados que não tem um modelo de dados pré definido ou não organizados de uma maneira pré definida. São exemplos: livros, jornais, documentos, metadados, imagens, vídeos, áudios, arquivos, dados de redes sociais, páginas da Internet, entre outros. O que gera problemas de irregularidades e ambiguidade nas tecnologias tradicio-

[1] Em tradução livre: Linguagem de Consulta Estruturada.
[2] Em tradução livre: Valores separados por virgulas.
[3] Em tradução livre: Linguagem de marcação extensível.
[4] Em tradução livre: Notação de objeto JavaScript.

nais, sendo importante uso de novas tecnologias, como a mineração de dados, para descobrir padrões ou interpreta-los (JAIN, 2017).

2.2 *Apache Hadoop*

O *Apache Hadoop* foi desenvolvido por Mike Cafarella e Doug Cutting enquanto trabalhavam em um projeto chamado Nutch, que buscava criar um motor de busca em páginas na rede. Este por sua vez fazia parte de um projeto ainda maior o Lucene, um programa para fazer buscas e uma API (*Application Programming Interface*)[5] para indexação de documentos. A empresa Yahoo! os contratou para investir no projeto e assim ganhou o nome de *Hadoop*, nome do elefante de pelúcia filho de Cutting que se tornou a logomarca do arcabouço, e se desvinculou dos projetos supracitados. Os principais artigos em que o *Hadoop* se baseou para o seu desenvolvimento foram "*The Google File System*"e "*MapReduce: Simplified Data Processing on Large Clusters*"(WHITE, 2012).

Segundo Goldman *et al.* (2012) "*Hadoop* é um arcabouço de código aberto, implementado em Java e utilizado para o processamento e armazenamento em larga escala, para alta demanda de dados, utilizando máquinas comuns". E um dos seus principais pontos é a garantia de conceitos de um sistema distribuído dentre eles a tolerância a falhas, capacidade de recuperação, consistência e escalabilidade, ao nível de programação do arcabouço, ou seja, transparente ao usuário que deve ter o foco na organização e na analítica dos dados com a ferramenta.

Sendo os elementos principais para seu funcionamento o *MapReduce*, o sistema de arquivos HDFS e o gerenciador de recursos e carga de trabalho do *cluster* YARN, que surgiu com uma melhoria na versão dois do arcabouço (GOLDMAN *et al*., 2012) e (BENGFORT; KIM, 2019).

2.2.1 Conceitos e nomes de um *cluster Hadoop*

Bengfort e Kim (2019) elucida que "Um conjunto de máquinas executando o HDFS e YARN é conhecido como um *cluster*, e as máquinas individuais são chamadas nós". Com a possibilidade de crescimento do *cluster* adicionando um novo nó, aumentando a capacidade e desempenho de forma linear. Os dois componentes anteriormente citados são implementados por diversos serviços que executam em segundo plano e não precisam de entradas do usuário (*daemon*). Os processos do Hadoop são serviços, e semelhante a um servidor HTTP(*Hypertext Transfer Protocol*)[6], cada nó opera com os seus serviços o tempo todo e por meio da rede enviam saídas ou recebem entradas. Cada um dos *daemons*, é processado em uma JVM(*Java*

[5] Em tradução livre: Interface de Programação de Aplicativos

[6] Em tradução livre: Protocolo de Transferência de Hipertexto

Virtual Machine)[7] e os recursos pra cada um destes são alocados e gerenciados por o próprio sistema operacional(BENGFORT; KIM, 2019).

O conceito de trabalho no arcabouço se refere a execução completa de um programa *MapReduce* e cada parte até a conclusão é conhecida como tarefa.(GOLDMAN *et al.*, 2012)

Cada nó do *cluster* tem sua denominação de acordo com os processos que são executados:

- Nós mestres (*masters*):
  São os nós em que normalmente o usuário utiliza para comandar as máquinas que processaram as tarefas. Sem eles não há coordenação e o armazenamento e processamento distribuído não seria possível (BENGFORT; KIM, 2019).

- Nós trabalhadores (*workers*):
  Estes são os nós que executam as tarefas que os mestres determinam, seja de armazenamento ou de operação de dados. Normalmente representam a maioria dos computadores dentro de um *cluster* (BENGFORT; KIM, 2019).

Tanto o HDFS quanto o YARN tem serviços que são direcionados aos mestres e aos trabalhadores. Para o HDFS há o seguintes serviços:

- *NameNode* (Mestre)
  É responsável por gerenciar os arquivos que estão no HDFS, dentre as tarefas realizadas destaca-se: realização da divisão dos arquivos em blocos, armazenando a árvore do diretório do sistema de arquivos, os metadados e a localização de cada réplica. Ao fato de muitos acessos a ele, suas informações são armazenadas em memória(BENGFORT; KIM, 2019) e (GOLDMAN *et al.*, 2012).

- *NameNode* Secundário (Mestre)
  Realiza tarefas de manutenção e de pontos de verificação a encargo do *NameNode*, apesar do nome sugerir esta ideia, não é um *NameNode* cópia de segurança (BENGFORT; KIM, 2019).

- *DataNode* (Trabalhador)
  Os *DataNodes* realizam o armazenamento dos dados e os administra localmente nos computadores que fazem parte desde *cluster*. O mesmo *DataNode* pode ser responsável por multiplos blocos ou até mesmo de arquivos diferentes e constantemente informando ao *NameNode* a condições dos arquivos (BENGFORT; KIM, 2019) e (GOLDMAN *et al.*, 2012).

Quando acessado um determinado arquivo no HDFS, primeiramente a aplicação cliente envia um requerimento ao *NameNode* para localizar os dados dentro do *cluster*, então o *Name-*

[7] Em tradução livre: Máquina Virtual Java

Node lista todos os *DataNodes* referentes ao arquivo de requisição e o cliente será responsável por requisitar diretamente os blocos de dados dos *DataNodes* (BENGFORT; KIM, 2019).

Para o YARN existem os seguintes serviços:

- *ResourceManager* (Mestre)
  É incumbido de controlar o escalonamento dos trabalhos, alocar e gerenciar os recursos do *cluster* para as aplicações(BENGFORT; KIM, 2019).

- *ApplicationMaster* (Mestre)
  É o serviço que gerencia a aplicação que o *ResourceManager* escalonou para ser executada no *cluster*(BENGFORT; KIM, 2019).

- *NodeManager* (Trabalhador)
  Além de executar as tarefas de processamento, as administra e em informa o status destas durante a execução, para um nó único(BENGFORT; KIM, 2019).

De forma semelhante ao HDFS, quando um cliente deseja executar um serviço este deve fazer uma requisição ao *ResourceManager* que atribuirá um *ApplicationMaster* para este serviço. O *ApplicationMaster* é responsável por monitorar o andamento do serviço, o *ResourceManager* o estado de cada nó e o *NodeManager* criará contêineres e executará as tarefas dentro deles (BENGFORT; KIM, 2019).

Apesar de processos mestres terem grande importância para coordenar e distribuir as tarefas e normalmente são executados em nós separados pra não competir por recursos, em *cluster* pequenos *daemons* mestre podem ser executados em um mesmo nó (BENGFORT; KIM, 2019).

2.2.2 HDFS

Em um sistema operacional os arquivos são manipulados por um sistema gerenciador de arquivos que implementa diversas funcionalidades como: armazenar, organizar, nomear, compartilhar, proteger e permissões para acesso, execução e alteração. Porém tais ações devem ser o mais transparente possível ao usuário quanto a sua complexidade de organização (GOLDMAN *et al.*, 2012).

Um sistema de arquivos distribuídos contém as mesmas características de um sistema de arquivos local, com o adicional de garantir todas as funcionalidade a todas as máquinas que estão conectas a rede e ao nível do usuário a manipulação de um arquivo ser parecida com um gerenciamento local, além de prover a escalabilidade, garantir que o arquivo não perca parte de suas informações durante a transferência, tolerância a falhas de acesso a um arquivo em um determinado nó, manter a integridade e consistência do arquivo em caso de alterações e acessos de usuários não autorizados e o desempenho deve ser alto apesar de todas as camadas de controle que devem ser implementadas (GOLDMAN *et al.*, 2012).

O *Hadoop Distributed File System* (HDFS) foi baseado no artigo de 2003 escrito por Ghemawat, Gobioff e Leung, "*The Google File System*", que tem como arquitetura mestre escravo como base dividindo os arquivos em tamanhos menores e os alocando em diversos nós escravos. Diferentemente do GFS, que fora implementado em C++, o HDFS foi escrito na linguagem Java e é um código aberto (GOLDMAN *et al.*, 2012).

Em sua proposta, o HDFS é uma camada de *software* que atua diretamente no sistema de arquivos nativo da máquina e, da forma em que fora projetado, consegue armazenar arquivos muito grandes e o seu acesso é em *streaming* e apresenta alguns pontos que devem ser destacados(BENGFORT; KIM, 2019):

- O HDFS tem desempenho superior quando seus dados são maiores e em grandes quantidades do que arquivos menores e em quantidades muito maiores (BENGFORT; KIM, 2019).
- Implementa o padrão WORM (Write Once, Read Many)[8], com isso escritas aleatórias em arquivos não são permitidas. O HDFS não é otimizado para leituras aleatórias, mas sim para a leitura em streaming (BENGFORT; KIM, 2019).

Com estes itens supracitados, o HDFS se torna mais viável como uma forma de armazenamento de dados brutos para o processamento, execução e resultados de um *job*, não sendo este a melhor opção para o um *backend* de análise de dados para uma determinada aplicação em tempo real. Normalmente o HDFS é utilizado para armazenar uma grande quantidade de arquivos heterogêneos, que as vezes são chamados de *data lake*[9], devido a suas funcionalidades a esses tipos de dados (BENGFORT; KIM, 2019).

2.2.2.1 Blocos

Para armazenar os arquivos no HDFS estes são divididos em blocos que podem variar o seu tamanho dado que o valor é configurável. O tamanho do bloco é quantidade miníma de informação que o sistema lerá ou escreverá por vez, semelhante a um sistema de arquivos local em disco, com a diferença de que caso um bloco não ocupe todo o espaço reservado a ele não terá essa parcela sem informação. Deste modo o HDFS opta por arquivos grandes, com tamanho de blocos menores para ter o menor espaço não preenchido (BENGFORT; KIM, 2019).

Os blocos do sistemas HDFS são replicados, por padrão porém esse valor é configurável, em três *DataNodes*, ou seja, três máquinas diferentes, em um ambiente distribuído se torna mais efetivo. Caso dois falhem ainda existirá um que contém o bloco a ser acessado. Em decorrência de redundância de blocos, em um *cluster*, o espaço disponível para armazenamento é de apenas um terço do valor original (BENGFORT; KIM, 2019).

[8] Em tradução livre: Escreva um Vez, Leia Várias
[9] Em tradução livre: Lago de dados

2.2.2.2 Gerenciamento de dados

Para o acesso a um determinado arquivo, o cliente faz a requisição ao *NameNode* que por sua vez busca nos metadados do arquivo em quais *DataNodes* encontrão-se os blocos que compõem o arquivo que deseja ser acessado. Estes metadados ficam no *NameNode* mestre para melhor desempenho da busca. Desta forma o *NameNode* trabalha como uma tabela de buscas e não cria um gargalo de leituras simultâneas. Porém o maior problema deste sistema é que quando o *NameNode* para de funcionar todo o *cluster* ficará incessável. (BENGFORT; KIM, 2019).

Como explicado no item 2.2.1 os *NameNode* Secundário não são um *backup* do *NameNode* mestre, mas dentre as suas funções também incluem tarefas de manutenção do *NameNode* e periodicamente gerar imagens instantânea dos dados e um registro das edições geradas. Este log é utilizado para manter a consistência dos dados, caso aconteça alguma falha no *NameNode*, a combinação destes registros podem ser utilizadas para recuperar o estado dos *DataNodes* (BENGFORT; KIM, 2019) .

Alguns itens que foram mencionados remete a problemas que o HDFS tem em sua construção e modo de operação. A partir da versão 2 do arcabouço *Hadoop* adicionou uma nova camada de software que auxilia em algumas das limitações que o HDFS tinha (BENGFORT; KIM, 2019).

2.2.3 YARN

Na primeira versão do *Hadoop* o principal limitador era o fato do *MapReduce* ser executado no HDFS e as funções de gerenciamento de trabalhos eram extremamente atreladas as funções de gerenciamento dos recursos do *cluster*, dessa forma não havia outras formas de utilizar de todas a infraestrutura do aglomerado para outras cargas de trabalho (BENGFORT; KIM, 2019).

Em cargas de trabalho executadas em lotes o *MapReduce* tem um ótimo desempenho porém utiliza muito dos recursos de entrada e saída do *cluster* o que em trabalhos de análise interativa, processamento de grafos, aprendizado de máquina e outros algoritmos que exigem intensivamente da memória sofram grandes limitações. Para solucionar o problema, houve o desmembramento do gerenciamento de carga de trabalho do gerenciamento de recursos, que o YARN é o responsável por gerenciar todos os recursos que há no *cluster* (BENGFORT; KIM, 2019) .

2.2.4 *MapReduce*

O *MapReduce* é um modelo de programação que foi baseado no paradigma funcional de programação, sendo este um estilo de programação aonde os dados devem ser processados sem avaliar seus estados, ou seja, dependem somente das suas entradas, são fechadas e não compartilham estados. Esta característica é extremamente interessante para a sua utilização em ambientes distribuídos e paralelos, no caso do primeiro aponta-se o fato de que como cada máquina em um ambiente de *Big Data* pode conter uma grande quantidade de arquivos fragmentados e cada uma das máquinas consegue executar esse serviço e retornar o resultado a um ponto central e o seu resultado não terá influência no processamento dos outros fragmentos (BENGFORT; KIM, 2019) e (GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

O artigo que serviu de base para a implementação do *MapReduce* no *Hadoop* foi escrito por Jeffrey Dean e Sanjay Ghemawat com o titulo: "*MapReduce: simplified data processing on large clusters*"em 2004. O *MapReduce* foi projetado para trabalhar com a aplicação em duas funções distintas: mapeamento (Map) e redução (Reduce) (MACHADO, 2018).

De acordo com Goldman *et al.* (2012), os dados que são utilizados na etapa de mapeamento, em geral, estão armazenados no HDFS e divididas em um número de blocos que o próprio sistema de arquivos aloca. Cada um destes blocos será destinado a uma tarefa *Map*, sendo que um escalonador irá determinar quais máquinas executarão a tarefa e qual cada uma realizará. A função deverá selecionar, com base nos dados de entrada, quais serão chaves e quais serão valores. De maneira genérica, o resultado de uma tarefa *map* para as instâncias (k1, v1), sendo k1 uma chave aplicada a um valor v1, após seu processamento gera um outro par chave/valor (k2, v2), ao término de todas as tarefas os conjuntos de mesmas chaves podem ser agrupados em uma lista, esta etapa é opcional, contudo auxilia na eficiência da fase intermediária. Com isto é possível, de forma ampla, que:

$$map(k1, v1) \rightarrow list(k2, v2) \tag{1}$$

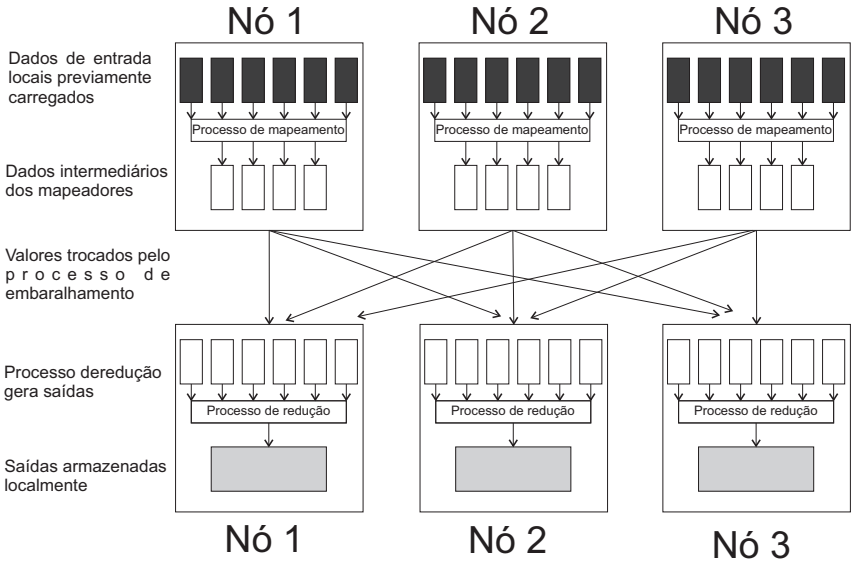
A fase intermediária é denominada *Shuffle*, nesta etapa os dados são agrupados por suas chaves e assim gerarão o conjunto de tuplas (k2, list(v2)), para cada chave k2 agrupar-se-ão todos os valores que estão associados a está mesma chave. Com a conclusão deste estágio, então, o arcabouço ficará responsável por dividir e replicar os dados gerados para os nós que realizarão a etapa de redução.

A função *reduce* utiliza-se dos dados provenientes da etapa anterior como entrada para realizar uma ação que o usuário definiu no momento que gerou a tarefa. De forma genérica a ação pode ser expressa:

$$reduce(k2, list(v2)) \rightarrow (k3, v3) \tag{2}$$

Na Figura 1 há um exemplo de execução de um trabalho em um pequeno *cluster* com três nós trabalhadores e um nó mestre que não está na imagem porém gerencia todo o aglomerado.

Figura 1 – Figura exemplificando como uma tarefa *MapReduce* é executada em um pequeno *cluster*



Fonte: Adaptado de: Bengfort e Kim (2019).

O nó mestre, que não está na imagem, é responsável por coordenar qual dos nós ficará responsável por um determinado fragmento do arquivo. Assim como o *NameNode* o *JobTracker* tem função de gerenciamento sob os trabalhos do tipo *MapReduce*, então este monitora e designa os diferentes nós que receberam as partes do arquivo analisado. Em caso de falha de algum nó o *JobTracker* reiniciará a tarefa e executará no mesmo nó ou até em um diferente (GOLDMAN *et al.*, 2012). Após designado as partes o que cada *DataNode*, que nas tarefas de *MapReduce* tem o nome de *TaskTracker*, executará a primeira etapa que *map*.

O resultado parcial dessa ação passa por uma etapa intermediaria chamada *Shuffle* para fazer o agrupamento de resultados gerados por aquele nó. Na Figura 1 nota-se que há alguns resultados dessa etapa que são acessados indiretamente e outros que serão acessados localmente pela máquina para servir de entrada para a próxima etapa, a redução que apresenta os resultados de forma local e o nó mestre então faz a leitura remota desses valores (BENGFORT; KIM, 2019).

Uma aplicação no arcabouço *Hadoop* precisa de duas ferramentas a priori: uma para armazenamento e uma de processamento, apresentadas anteriormente HDFS e *MapReduce* respectivamente. Porém o ecossistema do arcabouço é muito mais amplo e apresenta diversas ferramentas que atuam juntos porém com um nível de abstração maior. No próximo subitem será apresentado *Hive* e o *Pig*, ferramentas que atuam na camada de processamento agregadas ao *MapReduce*.

2.2.5 *Hive*

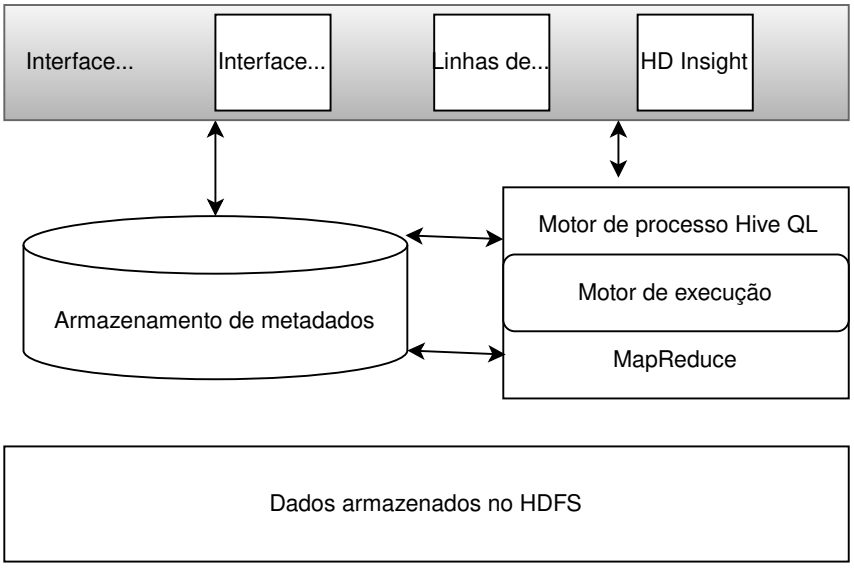
O Hive é um subprojeto do *Hadoop* que foi inicialmente desenvolvido por funcionários do *Facebook* e que em 2008 se tornou um projeto de código aberto. Sua principal funcionalidade é, a partir da sua linguagem *Hive QL* que é bem próxima da linguagem SQL e os demais conceitos de um banco de dados relacional, ser aplicado a grandes dados não relacionais que estão em um *cluster Hadoop* (GOLDMAN *et al*., 2012).

Para ilustrar a arquitetura do *Hive* a Figura 2 apresenta seus principais componentes. Na interface de usuário consegue-se gerar a interação do usuário com o HDFS, e há três formas: por linhas de comando, pela interface *Web* e, para *Windows Server* o *HD Insight* (JAIN, 2017).

No *Meta Store* o *Hive* armazena informações como esquemas e metadados de tabelas, colunas, tipos dos dados e o mapeamento do HDFS de uma determinada base de dados que está sendo acessada (JAIN, 2017).

O *Execution Engine* serve de intermediador entre a consulta que está localizada no *HiveQL Process Engine* e o *MapReduce*, transformando a consulta em um *job* para ser executado no *MapReduce* que busca no HDFS os dados que satisfazem as condições da consulta e retorna esses resultados ao *HiveQL Process Engine* (JAIN, 2017).

Figura 2 – Ilustração da arquitetura do arcabouço *Hive*

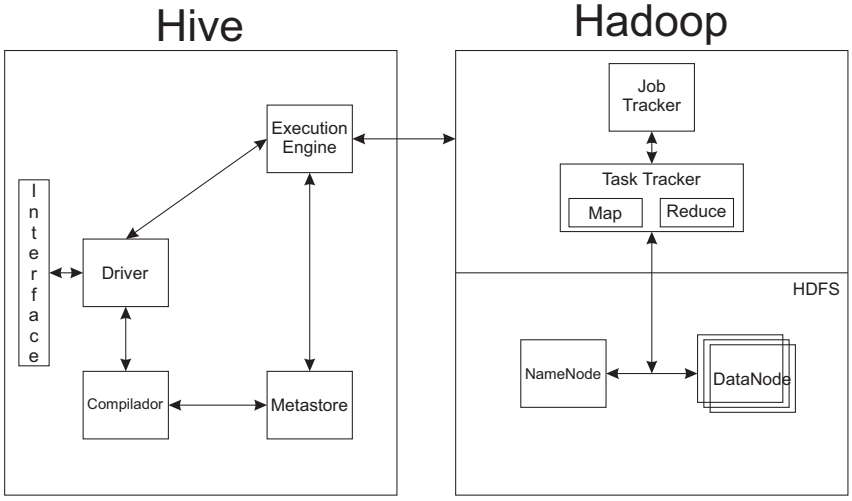


Fonte: Adaptado de: Jain (2017).

Para explicar o fluxo de uma consulta do *Hive* ao *Hadoop* a Figura 3 mostra a interação entre os componentes de cada um dos arcabouços.

No primeiro momento a consulta vinda da interface é enviada a um *driver* de conexão a uma base de dados para ser executado, então envia ao compilador de consultas que verifica sintaxe, o plano e as requisições da consulta. No próximo passo o compilador envia as requisições de metadados ao *Meta Store* que após as verificações envia ao compilador a resposta. Ao

Figura 3 – Processo de execução de uma consulta no arcabouço *Hive*



Fonte: Adaptado de: Jain (2017).

enviar para o *driver* a consulta já está compilada e será enviada ao *Execution Engine* que terá de transformar a consulta em um *job* (JAIN, 2017).

Esse *job* é atribuído a um *Job Tracker* que tem a função de a cada nó trabalhador enviar uma parcela da tarefa do bloco a qual cada um terá de executar as funções. Ao mesmo tempo da execução da tarefa de *MapReduce* o *Execution Engine* realiza as operações em cima dos metadados no *Meta Store*. Quando finalizado os procedimentos o *Execution Engine* é responsável por montar todos os resultados vindos dos *DataNodes* e enviar ao *driver* que depois será apresentado ao usuário pela interface do *Hive* (JAIN, 2017).

2.2.6 *Pig*

O *Pig* é uma ferramenta do projeto *Hadoop* desenvolvida pela empresa *Yahoo!* que buscava facilitar a escrita dos códigos de mineração de dados e poder representá-los como um fluxo de dados. Seu nome vem da sua filosofia onde porcos comem tudo: todos os tipos de dados podem ser processados; porcos vivem em qualquer lugar: o *Pig* pode ser executado em qualquer lugar, inicialmente ele foi desenvolvido no *Hadoop* porém tem a possibilidade de ser executado fora; porcos são animais domésticos: corresponde a facilidade de utilizar a linguagem *Pig Latin* para desenvolver os fluxos de dados; e porcos voam: devido a velocidade de processamento que o *Pig* tem e seu desempenho é sempre uma meta para atingir (GATES, 2011).

O *Pig* contém dois principais componentes, a linguagem *Pig Latin* e o ambiente de execução, local ou distribuído, que inclui a interface de linhas de comando *Grunt*. Um *script* em *Pig Latin* quando compilada se torna um *job MapReduce* (BENGFORT; KIM, 2019).

A linguagem *Pig Latin* se diferencia do *HiveQL* principalmente por ser procedural e não declarativa como a segunda, o que facilita no desenvolvimento de fluxos de dados e *pipeline* de

uma rotina. O *HiveQL* em consultas pode ser muito viável, porém em transformações de dados complexas em sequencia podem ser mais difíceis de serem escritas (BENGFORT; KIM, 2019).

Outro ponto a se destacar sobre a *Pig Latin* é a capacidade de utilizar códigos de outras linguagens como *Java*, *Python* ou *JavaScript* utilizando Funções Definidas pelo Usuário (em inglês UDFs) facilitando a utilização da ferramenta (BENGFORT; KIM, 2019).

2.3 Métodos de Acesso e Métodos de Acessos Métricos

Em bancos de dados relacionais, os Métodos de Acesso utilizam de buscas entre as relações de igualdade e do tipo dos dados fundamentais, como cadeia de caracteres e números. Na utilização desses métodos em dados mais complexos, como imagens e sons, não se obtém resultados satisfatórios. O grau de similaridade se torna um fator importante para ser avaliado nesses dados mais complexos (VIEIRA, 2004).

Para se comparar o quão similar dois dados são, compara-se as características que definem o dado e utiliza da métrica de cálculo de uma distância $d()$. O resultado desse cálculo gera um número sempre maior ou igual a zero, sendo valores próximos ou iguais a zeros o maior grau de similaridade e quanto mais próximo de infinito menor a similaridade (VIEIRA, 2004).

Essas consultas são muito custosas, pois avalia todo o conjunto de dados para verificar quais são os dados mais similares. Para minimizar o custo, utiliza-se de um algoritmo de indexação que é um procedimento em que na inserção dos dados ao conjunto, cria-se um índice planejado para minimizar os cálculos de distâncias e acessos a discos em consultas de similaridade (VIEIRA, 2004).

Os dados mais complexos, como supracitado, são avaliados por similaridade. Esses dados podem ser abstraídos por um espaço métrico, e recebem o nome de Dados em Domínios Métricos e são indexados através dos Métodos de Acesso Métricos (VIEIRA, 2004).

Os Métodos de Acesso Métricos tem como ideia geral partindo de um ou mais objetos pertencentes aquele conjunto de dados como representantes. Quando um novo dado é adicionado ao conjunto, a distância entre o novo dado e cada um dos representantes é um valor adicionado junto ao novo dado. As principais diferenças entre cada um dos Métodos de Acesso Métricos são: como o representativo é escolhido; como são alocados os objetos de acordo com o representativo; se podem sofrer alterações sem que a sua estrutura seja comprometida (VIEIRA, 2004).

2.3.1 Hiperplano Generalizado e GH-Tree

Em Uhlmann (1991) o Hiperplano Generalizado é definido por dois pontos p_1 e p_2 que não são iguais. O conjunto de pontos q são todos aqueles que cumprem a igualdade $d(q, p_1) =$

$d(q, p_2)$. Um ponto x pode ser dito que está do lado de p_1 se $d(q, p_1) < d(q, p_2)$. A estrutura que se é utilizada para gerar um Hiperplano Generalizado é uma árvore binária.

O GH-Tree é um algoritmo que segue os seguintes passos para gerar a separação dos objetos. Inicialmente escolhe-se dois objetos distantes um dos outros para serem a referência, os objetos que estiverem mais próximos do p_1 do que p_2 ficarão do lado da árvore, no caso contrário, ficarão do lado p_2 da árvore. O algoritmo é recursivo com a condição de parada quando todos os objetos estiverem inseridos árvore. Caso a escolha dos representantes seja apropriada, a árvore tende a ser balanceada (VIEIRA, 2004).

2.4 Mineração de dados

Em Tan *et al.* (2009) define "A mineração de dados é um processo de descoberta automática de informações úteis em grandes depósitos de dados". As técnicas empregadas na mineração podem encontrar desde padrões nos dados como prever os possíveis resultados. Porém nem todas as tarefas de descoberta de dados são minerações, a exemplo a recuperação de dados que consiste em uma busca de um determinado dado em um banco de dados ou a busca por uma determinada página na Internet, pois essas tarefas utilizam de técnicas tradicionais da ciência da computação e em recursos de indexação (TAN *et al.*, 2009).

2.4.1 Mineração de Dados e Descoberta de Conhecimento

O *Knowledge Discovery in Databases*[10] (KDD) é dividido em três etapas principais: o pré-processamento, a mineração de dados e o pós-processamento (TAN *et al.*, 2009).

Os dados que servirão de entradas para o processo podem estar armazenados desde arquivos de textos até banco de dados relacionais e armazenados de forma local ou distribuída. Cabe ao pré-processamento diversas atividades para fornecer os dados de uma forma aprimorada para a próxima etapa, que é a mineração. Dentre as funções que o pré processamento pode ter incluem: limpeza de ruídos em dados, completar dados faltantes, unir os dados das diversas base de dados as quais serão usadas nas análises, entre outras que possam ser importantes para a próxima etapa(TAN *et al.*, 2009).

Na etapa de pós-processamento há a verificação da validade dos dados que foram gerados na etapa de mineração para que somente os resultados válidos e úteis sejam utilizados por aplicações que visam apresentar a um usuário final o que foi obtido. Medições estatísticas ou hipóteses podem ser aplicadas para a validação dos resultados e eliminar resultado não legítimos (TAN *et al.*, 2009).

Dentre as tarefas de mineração de dados existe a modelagem de previsão, que com base nos atributos do dados, busca-se criar um modelo que determine uma variável alvo. Estas

[10] Em tradução livre: Descoberta de Conhecimento em Bancos de dados

são ainda divididas em dois tipos: a classificação e a regressão. Sendo a maior diferença o domínio da variável alvo, discreta e contínua, respectivamente (TAN *et al.*, 2009).

Em Tan *et al.* (2009) afirma "Uma técnica de classificação (ou classificadora) é uma abordagem sistemática para construção de modelos de classificação a partir de um conjunto de dados de entrada", cada técnica aplica um algoritmo de aprendizagem que busca encontrar um modelo que mais se aproxime do relacionamento das variáveis descritivas com a variável alvo, que determina a classe a qual determinado dado pertence.

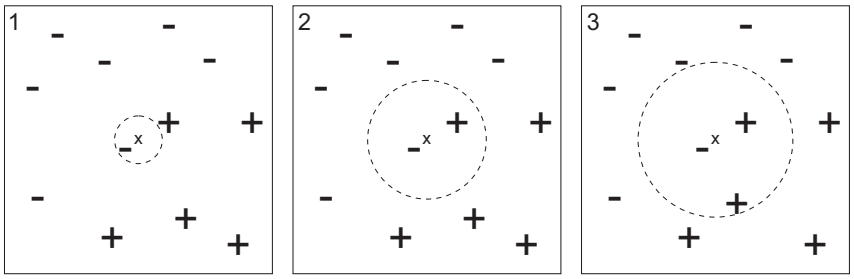
Nesse trabalho será apresentado o algoritmo de classificação por vizinhos próximos e o algoritmo de analise de grupos *K-means*, que por suas métricas de avaliação dos dados serem muito parecidos com o GH-Tree apresentado anteriormente.

2.4.2 Classificador de Vizinho Mais Próximo

Para Tan *et al.* (2009) "Um classificador de vizinho mais próximo representa em exemplo como um ponto de cada dado em um espaço d-dimensional, em que d é o número de atributos". Utilizando uma base de dados e um ponto de teste, calcula-se a distância entre o o exemplo e todos os dados do conjunto, pode ser utilizado diversos métodos de cálculos de distância como Euclidiana, Mahalanobis entre outras, os vizinhos mais próximos k do exemplo z são os k pontos que estão mais próximos de z (TAN *et al.*, 2009).

Na Figura 4 há três situações de exemplos para demonstrar como o classificador atua. Existem duas classes, os que pertencem a classe $+$ e os que pertencem a classe $-$. Quando escolhido $k = 1$ calcula-se a distancia a todos os pontos e o mais próximo e o um vizinho mais próximo é o que determina qual classe o ponto x pertence. No terceiro quadro o k tem valor três, portanto a classe dos três vizinhos mais próximos são analisadas, no caso de de mais de uma classe ser candidata a representar a classe de x escolhe-se a classe majoritária, que no caso é a classe $+$. Quando o número de pontos tiver quantidades iguais de classes, escolhe-se aleatoriamente a classe que x irá receber, esse caso acontece no segundo quadro da Figura 4 (TAN *et al.*, 2009).

Figura 4 – Exemplos de 1, 2 e 3 vizinhos próximos de um determinado ponto x



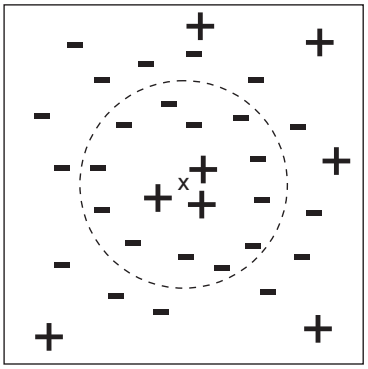
Fonte: Adaptado de: Tan *et al.* (2009).

O ajuste do valor de k é extremamente importante para decisão, pois se o valor de k for muito pequeno o classificador pode sofrer o *overfitting*, que é quando o modelo comente muitos

erros de classificação por se adaptar muito aos dados de treinamento e pouco aos possíveis dados futuros, em decorrência dos ruídos que a base de dados de treinamento possa ter, isso ocorre no quadro um da Figura 4 (TAN *et al.*, 2009).

Quando valor de k é muito alto, os dados avaliados para k vizinhos próximos contém uma grande parte dos pontos e eles podem estar muito longe dos ponto x que é o desejado a se classificar, esse caso pode ser visualizado na Figura 5 (TAN *et al.*, 2009).

Figura 5 – Uma classificação com k muito muito grande



Fonte: Adaptado de: Tan *et al.* (2009).

2.4.2.1 Algoritmo

Para cada exemplo teste $z = (x',y')$ é calculado a sua distância a todos os dados de treinamento $(x,y) \in D$. Com isso gera-se uma lista de vizinhos próximos D_z, essa etapa pode ser muito custosa caso existam muitos dados para avaliar, para melhorar o desempenho técnicas de indexação podem ser utilizadas para reduzir a quantidade de cálculos executados (TAN *et al.*, 2009).

Com a lista de vizinhos próximos avalia-se a classe majoritária da vizinhança do ponto z com a Equação 3:

$$y' = argmax \sum_{(x_i,y_i)\in D_z} I(v = y_i) \tag{3}$$

em que v é um rótulo da classe, y_i é o rótulo da classe para um dos vizinhos mais próximos, e $I(.)$ é uma função de verificação do argumento, caso verdadeiro retorna um, caso falso zero(TAN *et al.*, 2009).

Uma alternativa para que a distância entre z e cada item lista de vizinhos próximos sejam levados em consideração na classificação, a Equação 4 adiciona o parâmetro $w_i = 1/d(x',x_i)^2$ para adicionar esse novo critério na escolha da classe(TAN *et al.*, 2009).

$$y' = argmax \sum_{(x_i,y_i)\in D_z} w_i \times I(v = y_i) \tag{4}$$

Um dos principais pontos em que o classificador de vizinhos próximos comete erros na sua predição acontece quando a escala de um determinado atributo não é levada em consideração, exemplo de altura e peso de uma determinada população, as alturas variam entre 1,50 e 2 metros, enquanto peso podem variar de 40 a 100 quilos. Esse problema deve ser levado em consideração na etapa de pré-processamento (TAN *et al.*, 2009).

2.4.3 K-means

O *K-means* é um método de agrupamento que considera cada atributo uma dimensão no espaço R^n utiliza o parâmetro k para a quantidade de grupos a serem identificados(GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

O algoritmo, inicialmente, utiliza de pontos k pontos de dentro do conjunto, são chamados de sementes, estes iniciaram o processo de cálculo para cada termo do conjunto. Os pontos são atribuídos a um e apenas um centroide que obteve a menor distância entre todos os k centroides que foram inicializados. Ao final de uma iteração, para cada grupo que gerou-se na etapa de cálculo de distâncias, é atualizado o valor do centroide com a média dos pontos que formam o grupo. A condição de parada se da por duas possibilidades: quando os centroides não mudam ou por uma quantidade de iterações máxima (GOLDSCHMIDT; PASSOS; BEZERRA, 2015).

3 MATERIAIS E MÉTODOS

O desenvolvimento deste trabalho seguiu algumas etapas para que cada uma fosse bem executada, esse capítulo apresentará os materiais utilizados e em ordem cada uma das etapas para desenvolver o fluxo para os ensaios.

3.1 Materiais

Os ensaios utilizaram um *cluster* de computadores com a mesma configuração de equipamentos sendo a seguinte:

- Processador Intel i7 10700 @2.9GHz
- Memória RAM 16 Gb 3200MHz em dual channel
- HD WDC WD10SPSX-75A 1TB

Para interligar-los formando a rede utilizou-se um *switch* da marca Cisco modelo SG500-28P com a velocidade de comunicação 1000 GigaBits.

A ferramenta utilizada para implementar o arcabouço foi o *Docker* que é uma ferramenta de virtualização que utiliza do próprio sistema operacional onde está instalado para compor novas imagens adicionando apenas os componentes faltantes. Sua escolha vem de dois pontos principais: acelerar o processo de desenvolvimento, dado que as imagens com o arcabouço já estavam previamente configuradas e para escalar de dois nós para quatro, poucas linhas de um arquivo seriam alteradas; e caso exista alguém problema em determinada máquina do *cluster* a troca dos computadores era feita em apenas um arquivo de informações sobre a organização desse aglomerado.

Os *Datanodes* armazenavam os blocos dos conjunto de dados no disco rígido por conter maior espaço disponível que no SSD, evitando possíveis problemas de o *cluster* entrar em modo de segurança durante a execução dos testes e afetar nos tempos.

Em todos os ensaios a topologia utilizada foi sempre a mesma: no computador onde se encontrava o *NameNode* estava também o servidor do *Hive*, já os computadores que serviriam para armazenar as informações teriam apenas essa função. Para realizar a tarefa de acessar o *cluster*, fazer a consulta e o treinamento, e realizar a predição foi um computador que não tinha nenhuma imagem, apenas estava conectado na mesma rede.

O programa que criou essas tarefas e avaliou o tempo de cada uma foi desenvolvido em *Python*, com a utilização da biblioteca *Scikit Learn*, Pedregosa *et al.* (2011), que em sua documentação o algoritmo kNN força bruta tem sua complexidade $O[DN^2]$ em que D é a dimensionalidade e N é o número de elementos, como a dimensionalidade será sempre a mesma para todos os elementos, o número de cálculos se dará pela quantidade de elementos que deverão ser analisados.

3.2 Métodos

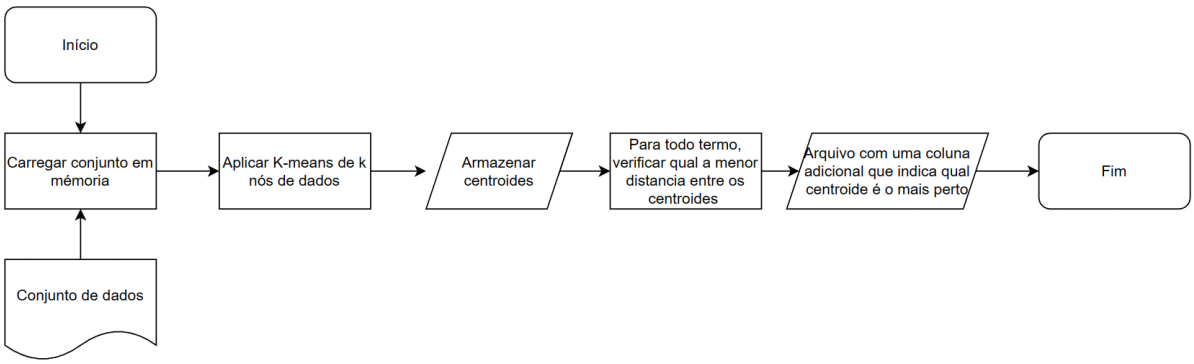
3.2.1 Pré ensaios

Realizou-se uma análise dos dados que serviram de base para o desenvolvimento do trabalho. Este *dataset* está localizado no site UCI[1] com um total de 11164866 imagens no formato PNG 41x41 *pixels* e cada uma das instâncias tem 128 atributos que os valores variam de 0 a 255. Em uma forma mais simples de organizar o conjunto de dados, há uma tabela em que cada linha representa uma imagem e as colunas os atributos. Utilizou-se desta para as primeiras etapas de particionamento e indexação dos dados.

Antes da análise desse conjunto de dados, acreditava-se que existiria uma separação por classes entre cada uma das instâncias, porém apenas existia suas características retiradas do algoritmo de SIFT inviabilizando a utilização de algoritmos de classificação. Com isso utilizou-se do algoritmo de kNN não supervisionado, onde apenas a busca por k vizinhos com a menor distância do dado analisado. Com a analise sobre os dados, a próxima etapa seria utilizar o método de acesso métrico no conjunto.

A estratégia de indexação dada pela *GH-Tree* serviu como base de distribuição dos dados. Em Uhlmann (1991) não é claro qual seria o algoritmo para definir os pontos iniciais que geram a *GH-Tree* desta forma optou-se por utilizar como pontos de interesse os centroides do algoritmo *KMeans* para a quantidade de divisões que existiram de nós de dados. Para o cálculo de distância entre os centroides e cada elemento do conjunto utilizou-se da distância euclidiana, adicionando uma nova coluna que informava a qual centroide aquele elemento estaria mais próximo. Com isso foram criados dos tipos de arquivos CSV: um para cada centroide e outro arquivo com todas os elementos, para verificar se haveria diferença entre as consultas para cada tipo de arquivo.

Figura 6 – Fluxograma de desenvolvimento do arquivo que foi implantado no *cluster*



Fonte: Autoria própria.

[1] https://archive.ics.uci.edu/ml/datasets/SIFT10M

O fluxo para o desenvolvimento dos arquivos que irão ser armazenados nos *Datanodes* é apresentado na Figura 6.

Uma das propostas iniciais do trabalho seria alocar um aquivo (dos arquivos separados) em cada nó e diversas estratégias foram utilizadas para que isso foi implantando, porém como vai de encontro com as políticas do arcabouço e como ele foi implementado esses dados sempre voltavam a ser espalhados para todo o *cluster*, com isso essa abordagem foi descartada.

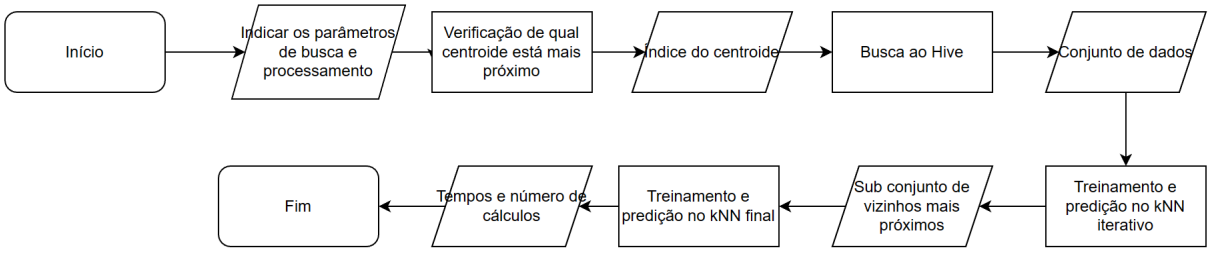
3.2.2 Ensaios

Inicialmente implantou-se o *cluster*, começando de um nó de dados e aumentando de acordo com a exponenciação de 2 até chegar em 8 computadores. Dentro de cada cenário utilizou-se de quatro métricas de distâncias para ser avaliada (L1, L2, *Canberra* e *Mahalanobis*), escolheu-se três elementos aleatoriamente (porém sempre estes foram utilizados em todo o ensaio) e para cada um destes consultou-se nas duas formas em que os arquivos estavam organizados (separados e agrupados).

Durante os ensaios deparou-se com um problema de estouro de memória, no primeiro momento encontrou-se esse problema durante a etapa de buscar os vizinhos próximos quando os dados estavam divididos em dois nós. Este foi corrigido mudando a forma em que o algoritmo de busca a vizinhos funciona, ao invés de utilizar todo o conjunto de dados para selecionar os vizinhos, dividiu-se em três e resultou em um subconjunto que seria novamente executado o algoritmo para selecionar os três mais próximos.

Outro momento que houve o mesmo problema foi quando havia apenas um nó de dados durante a etapa de consulta ao banco de dados apresentava, e a única solução seria utilizar um computador com mais memória, como no laboratório não havia este computador os ensaios começam de dois, quatro e oito nós.

Figura 7 – Fluxograma da tarefa de avaliação dos tempos e números de cálculos



Fonte: Autoria própria.

A Figura 7 representa o fluxo da tarefa de avaliação, que consistia em indicar os parâmetros (tipo de organização de arquivos, métrica de distancia utilizada e qual amostra teste) com isso aplicava-se um cálculo de distancia euclidiana para verificar qual o centroide era o mais próximo daquele elemento. Com esse dado uma busca no *cluster* utilizando esse termo

como condição de retorno, com este conjunto de dados aplicou-se o treinamento e predição do algoritmo de forma iterativa, gerando assim um novo subconjunto de vizinhos mais próximos que viriam alimentar um novo treinamento e predição. O retorno da tarefa seriam os tempos e o número de cálculos.

3.2.3 Métricas

Para cada uma das experiências foi analisado os seguintes itens:

- Tempo de cada etapa (consulta, treinamento e predição)
- Número de cálculos

Além disso também avaliou-se o fato de aceleração e eficiência das da aplicação de acordo com o aumento de nós de dados e por consequência o particionamento do conjunto de dados. Segundo Dias *et al.* (2010) "O Fator de Aceleração (ou speedup, do inglês), representa a razão entre o tempo gasto na solução de um problema utilizando um único processador, Ts, e o tempo gasto por um programa que utiliza p processadores, de forma paralela, Tp"no contexto desse trabalho, ao invés de ser avaliado por número de processadores processadores será utilizado a quantidade de nós de armazenamento de dados. E sobre o conceito de eficiência Dias *et al.* (2010) expõe "mede a fração de tempo no qual cada unidade de processamento é efetivamente utilizada", e apresenta que sua representação matemática é dada pela razão entre o fator de aceleração e o número de processadores, que será trocado pelo número de nós (DIAS *et al.*, 2010).

4 RESULTADOS

Após a realização de diversos ensaios para verificar se as hipóteses levantadas no decorrer deste trabalho se concretizaram, obtiveram-se os resultados dispostos em tabelas e posteriormente apresentado gráficos para melhor comparação de valores. Cada divisão de nós será apresentado em formato de tabela para a mesma amostra em diferentes situações e uma discussão de análise de todos os ensaios. As tabelas serão separadas por amostras e por tipo de arquivo que consultado pelo *Hive*, sendo agrupados em um único arquivo e separados cada arquivo contém apenas os elementos que estão mais próximos dos centroides, isto é, se há dois nós existem dois arquivos e em um deles há apenas exemplos próximos do centroide um e no outro os próximos do centroide dois.

Tabela 1 – Resultados para dois nós com arquivo agrupado para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	598,5979502	3,452435017	5154541
L2	598,4827523	2,760221481	5154541
Canberra	609,0038836	3,468696356	5154541
Mahalanobis	597,7264676	28,53707719	5154541

Fonte: Autoria Própria.

Tabela 2 – Resultados para dois nós com arquivo separado para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	596,7070506	2,915260315	5154541
L2	608,6484639	3,160989046	5154541
Canberra	609,2310596	3,645336390	5154541
Mahalanobis	606,6134667	29,01830602	5154541

Fonte: Autoria Própria.

Tabela 3 – Resultados para quatro nós com arquivo agrupado para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	367,2635555	1,567731380	3151023
L2	366,3048751	1,582962990	3151023
Canberra	363,8158650	1,958441496	3151023
Mahalanobis	364,7236373	17,54704833	3151023

Fonte: Autoria Própria.

Observando-se os valores de tempo de consulta e tempo de processamento, nota-se que conforme a quantidade de nós aumenta o tempo decresce. Isso se deve a seletividade dos valores que se tem conforme criamos mais planos no método de indexação.

Já analisando os valores de tempo de consulta entre os tipos do arquivos com a mesma quantidade de nós é possível perceber que quase não há muita discrepância dos valores. Durante execuções diretamente no banco de dados foi possível verificar que em todos os casos,

Tabela 4 – Resultados para quatro nós com arquivo separados para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	365,7553895	1,536364317	3151023
L2	368,1738718	1,699705124	3151023
Canberra	363,5518413	2,094491482	3151023
Mahalanobis	362,7240548	17,59870601	3151023

Fonte: Autoria Própria.

Tabela 5 – Resultados para oito nós com arquivo agrupado para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	236,2188163	1,012871265	1907497
L2	236,2686629	0,973326683	1907497
Canberra	233,1104853	1,272953510	1907497
Mahalanobis	235,7440376	10,52424216	1907497

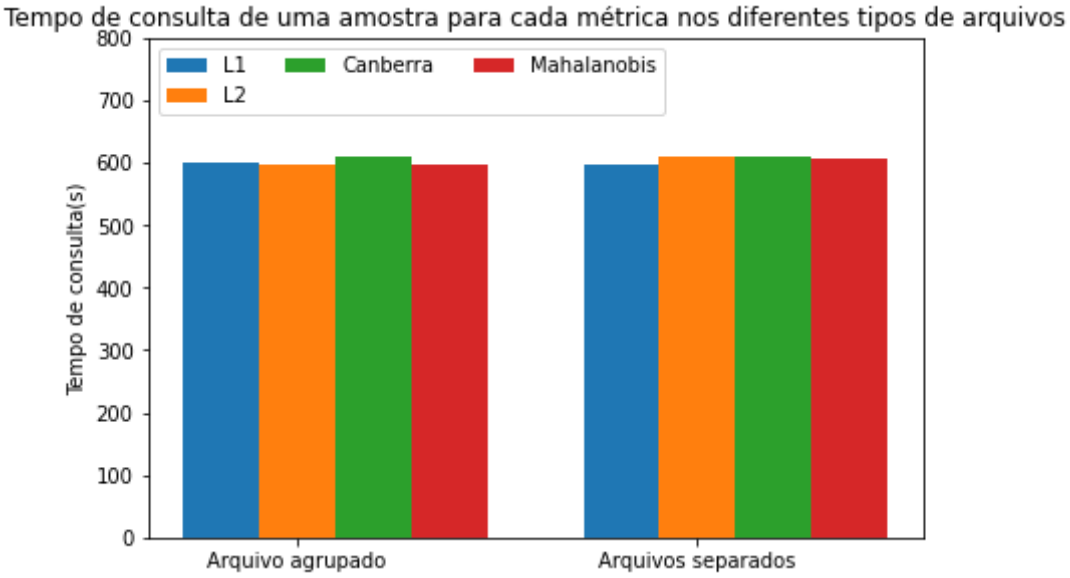
Fonte: Autoria Própria.

Tabela 6 – Resultados para oito nós com arquivo separados para uma amostra

Métrica de distância	Tempo de consulta(s)	Tempo de processamento(s)	Número de cálculos
L1	234,7344334	1,002742767	1907497
L2	233,0906272	0,966208935	1907497
Canberra	233,0480118	1,188298941	1907497
Mahalanobis	236,6652377	10,59309459	1907497

Fonte: Autoria Própria.

Figura 8 – Gráfico de tempo de consulta de uma amostra para cada métrica nos diferentes tipos de arquivos



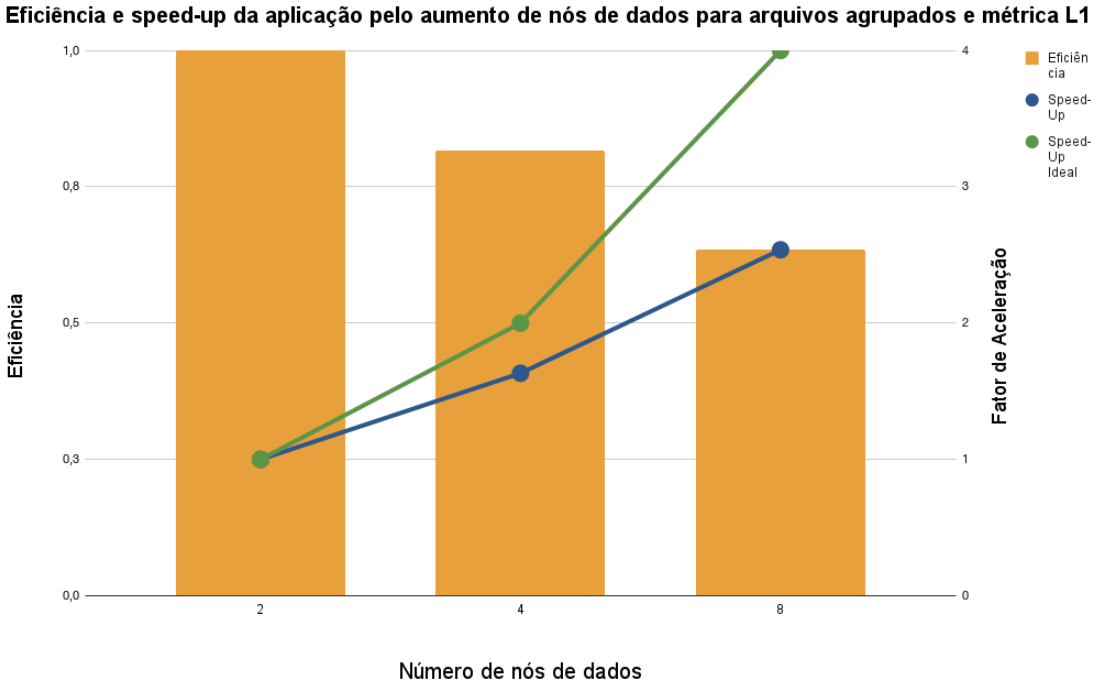
Fonte: Autoria própria.

tanto agrupado quanto separados, os arquivos eram todos unidos e depois aplicado o processo de *MapReduce* explicando o possível fator gerou valores tão parecidos. A Figura 8 complementa

essa analise utilizando os valores das Tabela 1, Tabela 2, Tabela 3, Tabela 4, Tabela 5, Tabela 6 na coluna de Tempo de Consulta.

Em relação a velocidade de processamento, pode-se notar que os tempos de consulta diminuíram de forma significativa bem como os valores de tempo de processamento para a métrica de distância mais custosa. Os gráficos a seguir mostram essa relação do esperado para o medido. É possível verificar que as curvas ficaram com valores parecidos tanto para a métrica L1 quanto para Mahalanobis e o mesmo se aplica para a distribuição de arquivos de dados no *cluster*.

Figura 9 – Gráfico de eficiência e *speed-up* da aplicação pelo aumento de nós de dados para arquivos agrupados e métrica L1



Fonte: Autoria própria.

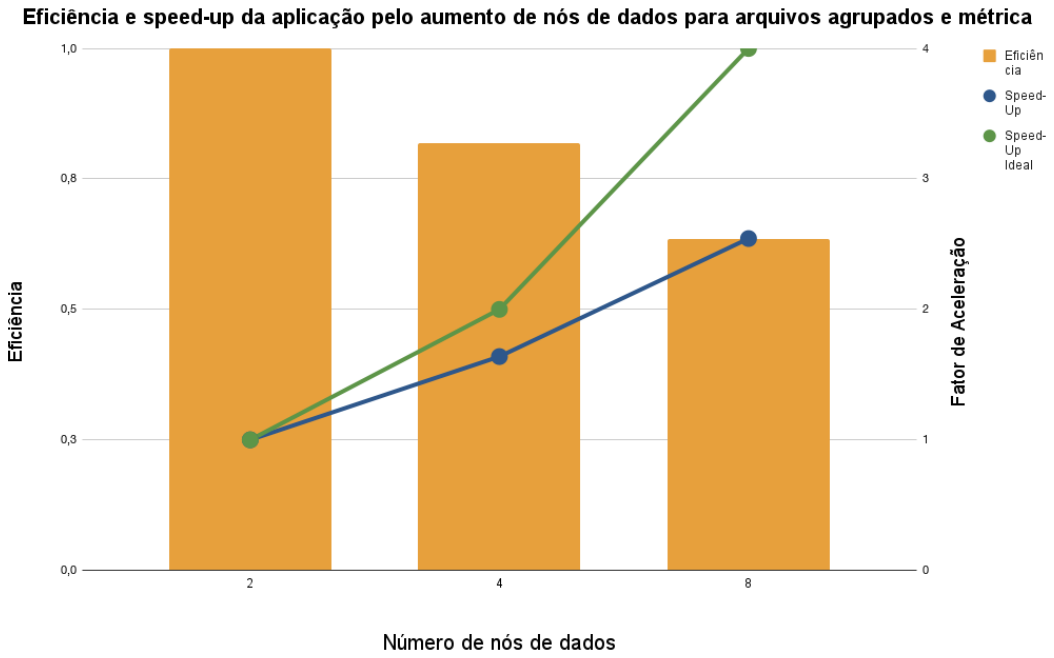
Tabela 7 – Resultados de *speed-up* e eficiência para arquivos agrupados e métrica L1

Nós	Tempo de execução médio(s)	*Speed-up*	*Speed-up* Ideal	Eficiência
2	602,0503852	1,0	1	1,0
4	368,8312869	1,6	2	0,8
8	237,2316875	2,5	4	0,6

Fonte: Autoria Própria.

E nota-se também que houve a queda na número de cálculos de acordo com divisão de nós de dados, a consulta se torna mais seletiva quando se há um número maior de partições resultando em uma consulta mais rápida e um processamento menor. A Figura 13 demonstra esse acontecimento.

Figura 10 – Gráfico de eficiência e *speed-up* da aplicação pelo aumento de nós de dados para arquivos agrupados e métrica Mahalanobis



Fonte: Autoria própria.

Tabela 8 – Resultados de *speed-up* e eficiência para arquivos agrupados e métrica Mahalanobis

Nós	Tempo de execução médio(s)	*Speed-up*	*Speed-up* Ideal	Eficiência
2	599,6223109	1,0	1	1,0
4	367,2917538	1,6	2	0,8
8	235,7371762	2,5	4	0,6

Fonte: Autoria Própria.

Tabela 9 – Resultados de *speed-up* e eficiência para arquivos separados e métrica L1

Nós	Tempo de execução médio(s)	*Speed-up*	*Speed-up* Ideal	Eficiência
2	599,6223109	1,0	1	1,0
4	367,2917538	1,6	2	0,8
8	235,7371762	2,5	4	0,6

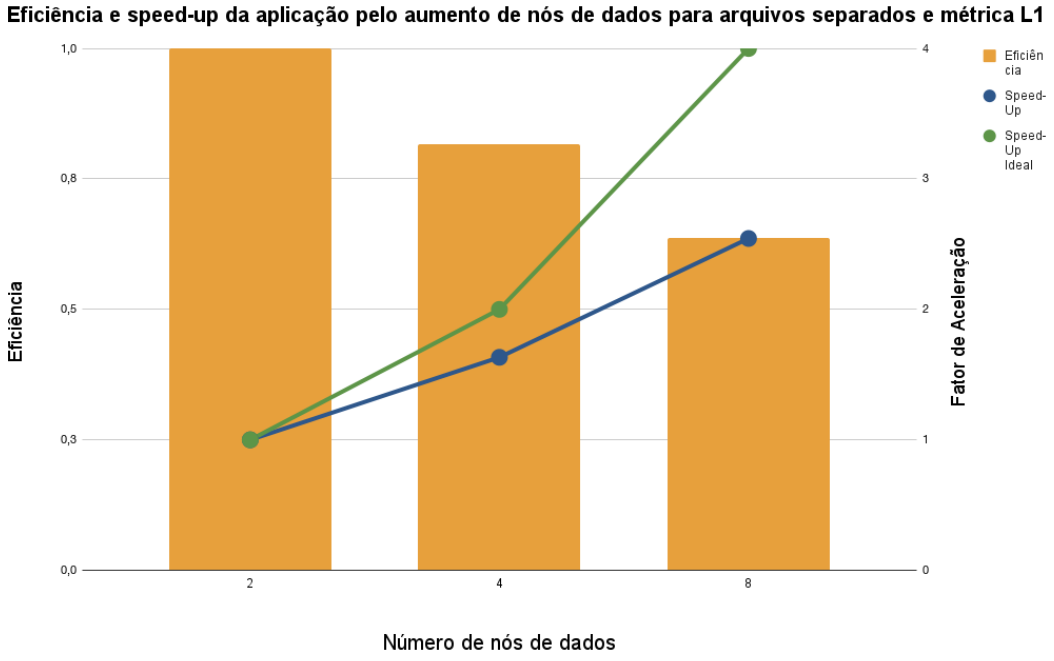
Fonte: Autoria Própria.

Tabela 10 – Resultados de *speed-up* e eficiência para arquivos separados e métrica Mahalanobis

Nós	Tempo de execução médio(s)	*Speed-up*	*Speed-up* Ideal	Eficiência
2	635,6317728	1,0	1	1,0
4	380,3227608	1,7	2	0,8
8	247,2583323	2,6	4	0,6

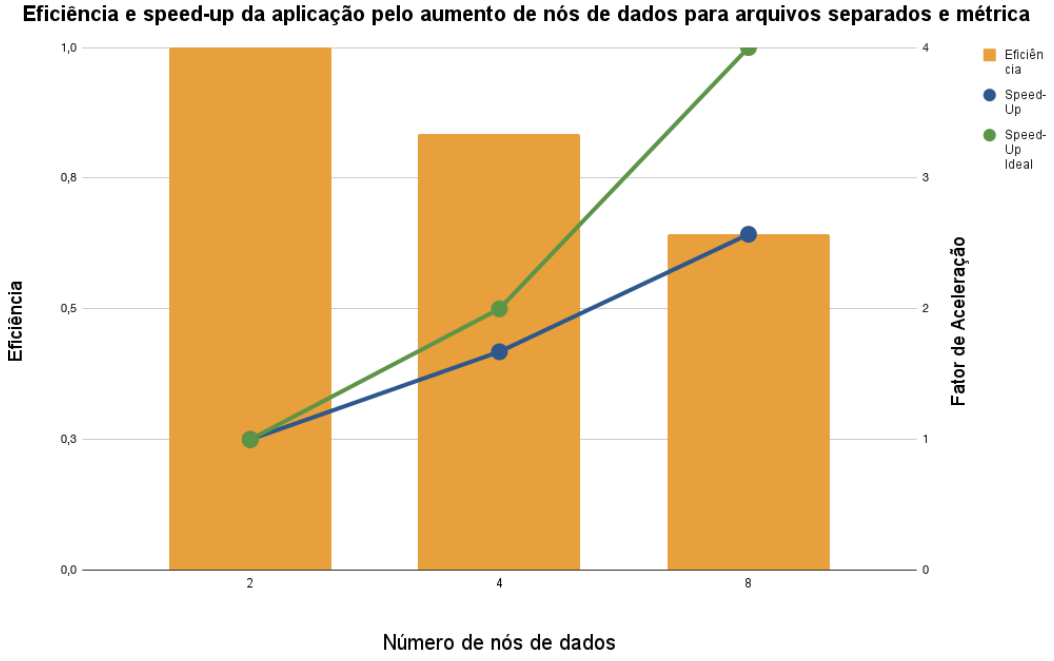
Fonte: Autoria Própria.

Figura 11 – Gráfico de eficiência e *speed-up* da aplicação pelo aumento de nós de dados para arquivos separados e métrica L1



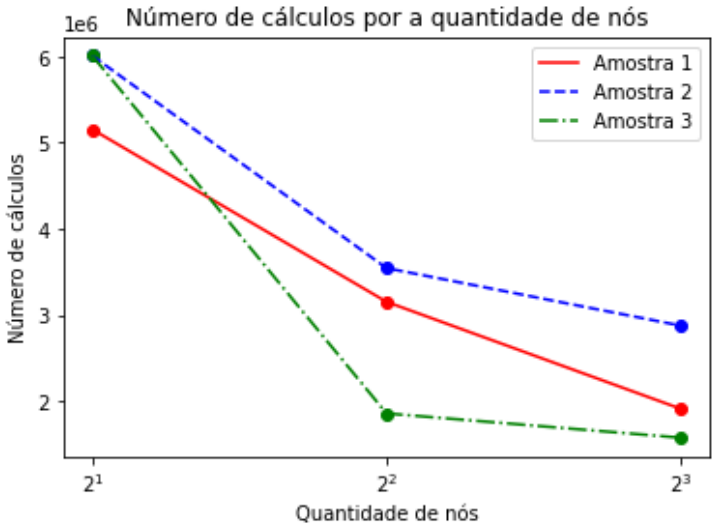
Fonte: Autoria própria.

Figura 12 – Gráfico de eficiência e *speed-up* da aplicação pelo aumento de nós de dados para arquivos separados e métrica Mahalanobis



Fonte: Autoria própria.

Figura 13 – Gráfico de número de cálculos por a quantidade de nós para todas as amostras



Fonte: Autoria própria.

Tabela 11 – Resultados de número de cálculos por a quantidade de nós de dados

Amostra	Dois nós	Quatro nós	Oito nós
1	5154541	3151023	1907497
2	6010342	3542648	2873933
3	6010342	1854141	1571840

Fonte: Autoria Própria.

5 CONCLUSÃO

Em algumas hipóteses que o trabalho propôs obtiveram-se resultados satisfatórios, como por exemplo a redução no número de cálculos e no tempo de consulta conforme o número de nós de dados foi aumentando, porém houveram problemas em alguns aspectos, como por exemplo o estouro de memória para um nó de dados. Tal problema foi contornado com métodos iterativos, entretanto o problema seria solucionado se tivesse acesso a um computador com mais memória RAM e neste caso não haveria modificações nas tarefas.

Em outras hipóteses não foi obtido melhoria, como no caso da separação de arquivo, posto que o arcabouço faz uso da ferramenta *MapReduce*, sendo assim, o arquivo, mesmo que separado, era verificado por completo antes do retorno. Outro ponto foi a falta de controle quanto ao armazenamento do arquivo, que impossibilitou que houvesse apenas um arquivo por nó nos ensaios. Mesmo desativando o as maquinas de dados brevemente e deixando apenas a máquina de interesse, no momento que as demais eram inseridas no *cluster* o arcabouço enviava os blocos de arquivos que antes estavam concentrados a outros nós.

A métrica de distância que obteve maior redução no tempo foi a Mahalanobis, de aproximadamente 29 segundos para 2 nós e chegando em aproximadamente 10 segundos para 8 nós. Analisando a complexidade do algoritmo utilizado ($O[DN^2]$) dado que D é constante para todos os elementos o que vária é o número de elementos que foram buscados na consulta, porém este varia em uma função quadrática e é possível ver que o tempo de processamento na Tabela 1 para a Tabela 3 diminui bem mais que em comparação da Tabela 3 para Tabela 5 para as métricas L1, L2 e Canberra.

A técnica de particionamento utilizada, *GH-Tree*, se tornou útil para a redução do espaço de busca no algoritmo KNN, mas em decorrência do conjunto de dados não ter classes não foi possível concluir se esse particionamento gerou problemas na predição dos dados. Para a continuação desse trabalho a utilização de um conjunto de dados com classes conseguiria avaliar se houve influência na predição e também a utilização de outras técnicas para avaliar qual teve menor menor impacto na etapa de predição e ainda sim conseguiu reduzir o espaço amostral.

REFERÊNCIAS

BENGFORT, B.; KIM, J. **Analítica de dados com Hadoop: Uma introdução para cientistas de dados**. Novatec Editora, 2019. ISBN 9788575227626. Disponível em: https://books.google.com.br/books?id=IrSZDwAAQBAJ.

DIAS, B. H. *et al.* Programação dinâmica estocástica aplicada ao planejamento da operação do sistema elétrico brasileiro através do uso de processamento paralelo. **XLII Simpósio Brasileiro de Pesquisa Operacional, Bento Gonçalves, Setembro de**, 2010.

GATES, A. **Programming Pig: Dataflow Scripting with Hadoop**. O'Reilly Media, 2011. ISBN 9781449317683. Disponível em: https://books.google.com.br/books?id=0zimI7NoObIC.

GOLDMAN, A. *et al.* Apache hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. **XXXI Jornadas de atualizaçoes em informatica**, p. 88–136, 2012.

GOLDSCHMIDT, R.; PASSOS, E.; BEZERRA, E. **Data Mining: Conceitos, técnicas, algoritmos, orientações e aplicações**. [*S.l.*: *s.n.*], 2015. ISBN 978-85-352-7822-4.

JAIN, V. **Big Data and Hadoop**. KHANNA Publishers, 2017. ISBN 9789382609131. Disponível em: https://books.google.com.br/books?id=i6NODQAAQBAJ.

MACHADO, F. **Big Data O Futuro dos Dados e Aplicações**. Editora Saraiva, 2018. ISBN 9788536527611. Disponível em: https://books.google.com.br/books?id=2LdiDwAAQBAJ.

PEDREGOSA, F. *et al.* Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

TAN, P. *et al.* **Introdução ao datamining: mineração de dados**. Ciencia Moderna, 2009. ISBN 9788573937619. Disponível em: https://books.google.com.br/books?id=69d6PgAACAAJ.

UHLMANN, J. K. Satisfying general proximity / similarity queries with metric trees. **Information Processing Letters**, v. 40, n. 4, p. 175 – 179, 1991. ISSN 0020-0190. Disponível em: http://www.sciencedirect.com/science/article/pii/002001909190074R.

VIEIRA, M. R. **DBM-tree: método de acesso métrico sensível à densidade local**. 2004. Tese (Doutorado) — Universidade de São Paulo, 2004.

WHITE, T. **Hadoop: The Definitive Guide**. O'Reilly, 2012. (Oreilly and Associate Series). ISBN 9781449311520. Disponível em: https://books.google.com.br/books?id=drbI_aro20oC.