

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS

DANIEL ORLANDO KERNECHI DENESIUK

HADOOP:
ANÁLISE DA FERRAMENTA E ESTUDO DE CASO

TRABALHO DE CONCLUSÃO DE CURSO

MEDIANEIRA 2016

DANIEL ORLANDO KERNECHI DENESIUK

**HADOOP:
ANÁLISE DA FERRAMENTA E ESTUDO DE CASO**

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Evando Pessini.

AGRADECIMENTOS

Desejo expressar o meu agradecimento, sem nenhuma ordem em particular, às seguintes pessoas que –de alguma ou outra forma– fizeram parte importante durante este ciclo que se aproxima da sua culminação:

Agradeço e reverencio o Professor Dr. Evando Pessini pela orientação e pela dedicação durante o desenvolvimento deste trabalho.

Da mesma maneira, a minha sincera gratidão aos professores: Professor Dr. Neylor Michel, Professor Dr. Cláudio Bazzi e o Professor M.Sc. Nelson Betzek pelos momentos de aprendizado e pela inestimável orientação e constante apoio recebidos durante todo o tempo transcorrido nas atividades relacionadas com o âmbito acadêmico.

Devo também minha gratidão ao Coordenador do Curso, o Professor Dr. Paulo Lopes de Menezes, pela oportunidade e apoio incondicional recebido durante a fase final do curso.

Agradeço também aos pesquisadores e professores da banca examinadora pela atenção e contribuição dedicadas a este estudo.

Em continuidade, desejo externar meu reconhecimento e satisfação aos colegas acadêmicos, Luiz Fernando Buss e Douglas Taube, pela importância dos momentos compartilhados e pela concretização de importantes objetivos durante o ciclo acadêmico, e, em especial ao colega André Menegasso pelo grande apoio e amizade durante todo este tempo na Instituição.

Igualmente, gostaria de deixar registrado o meu grande agradecimento ao Professor Mauro Dietrich, atual Diretor da Escola Estadual Indígena “Araju Porã” (Itamarã) do Município de Diamante d’Oeste. Sem a sua grande ajuda, Mauro, o meu caminho inicial houvesse tido obstáculos muito maiores.

Finalmente, o reconhecimento merecido à minha esposa Marizete, pois devo a ela todo o trajeto percorrido para alcançar esta instância.

RESUMO

Kernechi Denesiuk, Daniel Orlando. Hadoop: análise de ferramenta e estudo de caso. 2016. 64 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas), da Universidade Tecnológica Federal do Paraná – UTFPR.

Este trabalho apresenta um estudo sobre Big Data, relacionando os conceitos mais importantes que fazem parte da área, incluindo o modelo de programação MapReduce junto a plataforma de software Hadoop para computação distribuída, voltado para o trabalho com grandes volumes de dados.

Complementa-se com um estudo de caso desenvolvido através de linguagem de programação Java, utilizando dados reais do Instituto Nacional de Meteorologia (INMET).

O trabalho mostra como resultado a importância em grande escala que a plataforma Hadoop tem adquirido num período de tempo relativamente recente, em função das características e vantagens oferecidas pelo uso desta tecnologia.

Palavras-chave: Big Data, MapReduce, Hadoop, computação distribuída.

ABSTRACT

Kernechi Denesiuk, Daniel Orlando. Hadoop: tool analysis and case study. 2016, 64 s. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas), da Universidade Tecnológica Federal do Paraná – UTFPR.

This work presents an approach in Big Data area, relating the most important concepts including MapReduce programming model with Hadoop software platform in distributed computing systems, designed to work with large volumes of data.

It also is presented a case study developed in Java programming language, using real data from the Instituto Nacional de Meteorologia (INMET).

This research shows as result the importance that the large-scale Hadoop platform has achieved in a relatively short time, exhibiting the features and benefits offered by the use of this technology.

Keywords: Big Data, MapReduce, Hadoop, distributed computational environments.

LISTA DE FIGURAS

Figura 1 - Arquitetura típica de um <i>cluster</i>	16
Figura 2 - Gráfico comparativo da média de interesse entre Big Data.	18
Figura 3 - Disciplinas chave para a Ciência de Dados	19
Figura 4 - Arquitetura Big Data em camadas.	22
Figura 5 - Fluxo de Dados num processo típico MapReduce	26
Figura 6 - Exemplo de um processo MapReduce, para contagem de palavras.	28
Figura 7 - Algoritmo da função <i>Map</i> e da função <i>Reduce</i> representado através de pseudocódigo <i>WordCount</i>	29
Figura 8 - Fluxo lógico num processo de execução do MapReduce.	30
Figura 9 - Arquitetura HDFS.	36
Figura 10 - Etapas do MapReduce.	37
Figura 11 – Comparação no fluxo de bytes.	39
Figura 12 - Tela do Console exibindo as versões instaladas indicadas nos retângulos em vermelho.	42
Figura 13 - Console mostrando os processos Hadoop em ativo.	43
Figura 14 - Interface Web do cluster local.	44
Figura 15 - Interface Web do <i>NameNode</i>	44
Figura 16 - Interface Web para o acesso aos dados do BDMET.	46
Figura 17 - Interface Web da INMET mostrando as estações meteorológicas do Paraná.	47
Figura 18 - Interface Web da INMET para série histórica da estação Curitiba – PR.	47
Figura 19 - Estrutura do diretório input (<i>dataset</i>) pela interface Web do Hadoop.	48
Figura 20 – Código do <i>Mapper</i>	52
Figura 21 – Trecho de código do <i>reducer</i>	53
Figura 22 - <i>job</i> para o <i>MapReduce</i>	55

LISTA DE TABELAS

Tabela 1 - Máximos diários de temperatura por ano para Curitiba para o período 1966 – 2015	57
Tabela 2 - Mínimos diários de temperatura por ano para Curitiba para o período 1966 – 2015	57
Tabela 3 – Máximo de precipitação diária por ano para Paraná (INMET) no período 1966 – 2015	59

LISTA DE SIGLAS

BD	Big Data
BDMEP	Banco de Dados Meteorológicos para Ensino e Pesquisa
BI	Business Intelligence
CPU	Central Processing Unit
DB	Data Base
DM	Data Mining
DS	Data Scientist
GB	Giga Byte
GPS	Global Position System
HDFS	Hadoop File System
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines
INMET	Instituto Nacional de Meteorologia
JDK	Java Development Kit
JVM	Java Virtual Machine
LAN	Local Area Network
MB	Mega Byte
PB	Peta Byte
RAM	Random Access Memory
SGBD	Sistema de Gerenciamento de Banco de Dados
SO	Sistema Operacional
SQL	Structured Query Language
SSH	Secure Shell
TB	Tera Byte
TI	Tecnologia da Informação
YARN	Yet Another Resource Negotiator

SUMÁRIO

1 INTRODUÇÃO	11
1.1 OBJETIVO GERAL.....	12
1.2 OBJETIVOS ESPECÍFICOS.....	12
1.3 MOTIVAÇÃO	13
1.4 ORGANIZAÇÃO DO TEXTO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 PROCESSAMENTO DISTRIBUÍDO / PARALELO	15
2.1.1 CLUSTERS E NÓS.....	15
2.2 INTRODUÇÃO AO BIG DATA.....	17
2.2.1 OS 5 V.....	20
2.3 ARQUITETURA BD	21
2.3.1 COLETA DE DADOS.....	22
2.3.2 ARMAZENAMENTO.....	22
2.3.3 BASES DE DADOS	23
2.3.4 PROCESSAMENTO E ANÁLISE	24
2.3.5 VISUALIZAÇÃO.....	24
2.4 ARQUITETURA DO FRAMEWORK MAPREDUCE	24
2.4.1 FUNÇÃO <i>MAP</i> E FUNÇÃO <i>REDUCE</i>	25
2.4.2 ALGORITMOS.....	29
2.4.3 LIMITAÇÕES DO MAPREDUCE	30
3 APACHE HADOOP	32
3.1 O QUÉ É HADOOP	32
3.2 CARACTERÍSTICAS	33
3.3 ARQUITETURA HDFS	34
3.3.1 CARACTERÍSTICAS DA ARQUITETURA HDFS	34

3.3.2	SERVIÇOS OU <i>DAEMONS</i> HDFS	35
3.4	ETAPAS DE UMA EXECUÇÃO MAPREDUCE HADOOP	37
3.4.1	SERIALIZAÇÃO DE DADOS	38
3.4.2	COMPARAÇÃO NO FLUXO DE DADOS	39
3.4.3	CONSIDERAÇÕES FINAIS DO CAPÍTULO	40
4	CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO	41
4.1	HARDWARE E SOFTWARE UTILIZADO	41
4.2	CONFIGURAÇÃO HDFS	42
4.3	OBTENÇÃO DOS DADOS (<i>DATASET</i>)	44
4.4	ANÁLISE E PREPARAÇÃO DO <i>DATASET</i>	48
4.5	DESENVOLVIMENTO DO PROJETO	49
5	DESENVOLVIMENTO DO ESTUDO DE CASO	50
5.1	IMPLEMENTAÇÃO JAVA	50
5.2	A CLASSE DE GERENCIAMENTO	51
5.2.1	MAPPER	51
5.2.2	REDUCER	53
5.2.3	DRIVER	53
5.3	EXECUÇÃO DO PROJETO	55
5.4	APRESENTAÇÃO DOS RESULTADOS	56
5.4.1	TEMPERATURA MÁXIMA E MÍNIMA PARA CURITIBA	56
5.4.2	MÁXIMO DE PRECIPITAÇÃO POR ANO PARA O PARANÁ	58
5.5	CONSIDERAÇÕES FINAIS	60
6	CONCLUSÃO	61
6.1	TRABALHOS FUTUROS	61
7	REFERÊNCIAS	63

1 INTRODUÇÃO

Atualmente a ciência e a tecnologia têm alcançado um nível de complexidade e de informação superior aos dos anos anteriores. Assim, o desenvolvimento de uma atividade, seja comercial, educacional, de pesquisa ou artística, deve levar em consideração a grande quantidade de dados existentes.

Efetivamente, existem dados que podem em uma primeira instância não guardar relação entre si, tais como o consumo de combustível de um veículo, o desempenho acadêmico de um aluno, a intenção de voto para um determinado candidato, a previsão do tempo para a lavoura, etc., porém, estes dados em conjunto, exercem grande influência no processo de tomada de decisões.

Igualmente, nota-se um crescente aumento no compartilhamento de dados, principalmente por meio das diversas redes de relacionamentos entre as pessoas, ao disponibilizar seus próprios dados para os demais, o que definitivamente contribui para o aumento do nível de complexidade nas diversas atividades praticadas na sociedade moderna.

De acordo com White (2012, p. 27), o aumento no fluxo de dados vem de diversas fontes. Considere os seguintes exemplos:

- O mercado de Nova Iorque *New York Stock Exchange* gera aproximadamente de 4 a 5 terabytes (TB) de dados por dia;
- Facebook hospeda mais de 240 bilhões de fotos, aumentando uns 7 petabytes (PB) ao mês;
- Ancestry.com, o sítio web genealógico armazena aproximadamente 10 PB de dados;
- A organização “*Internet Archive*” mantém arquivos da web alcançando 18,5 PB;
- O acelerador de partículas “Grande Colisionador de Hádrons” na Genebra, Suíça, produz aproximadamente 30 PB de dados ao ano.

Em continuidade com os dados demonstrativos, segundo Zikopoulos (2013, p.19), em 2005 nos Estados Unidos existiam 1,3 milhões de dispositivos RFID em circulação, e para o final de 2011 o número tinha aumentado até 30 milhões. Considerando que para 2015 o preço de um destes dispositivos deve cair para

aproximadamente um centavo de dólar, as possibilidades de medição de dados manterão a tendência de aumento.

Em continuidade com Zikopoulos (2013, p.19), um motor de jato comercial moderno gera em torno a 10 TB de dados a cada meia hora. Assim, um voo de Londres a Nova Iorque gera cerca de 650 TB de dados.

Segundo Hilbert e López (2011), todo este volume de dados é denominado “*Big Data*” (BD), isto é “*megadados*”, grandes volumes de dados, e o profissional especializado nesta prática recebe o nome de “Data Scientist” (DS), isto é o Cientista de Dados.

De maneira específica, segundo Provost e Fawcet (2013), o BD é caracterizado essencialmente como um grande conjunto de dados (*data sets*) excessivamente massivo para ser tratado utilizando os sistemas tradicionais de processamento de dados. Entretanto, assim como ocorre com estes sistemas, a tecnologia do BD é utilizada para várias aplicações, incluindo a engenharia de dados e a implementação de técnicas de Data Mining (DM).

Finalmente, de acordo com Hilbert e López (2011), o BD é visto como uma ferramenta eficaz contra os problemas socioeconômicos nos últimos anos, o que significa que poderá desempenhar no futuro um papel de maior proeminência.

1.1 OBJETIVO GERAL

O presente trabalho tem como objetivo geral realizar um estudo teórico sobre as técnicas de mineração de dados em Big Data e desenvolver um estudo de caso com o framework Hadoop.

1.2 OBJETIVOS ESPECÍFICOS

Como etapas para alcançar o objetivo geral, coloca-se como proposta os seguintes objetivos específicos:

- Adquirir referencial teórico sobre BD, motivação, conceitos relacionados e evolução;
- Efetuar um estudo de caso desenvolvido em ambiente Eclipse (Java) em combinação com a ferramenta de código aberto Hadoop *MapReduce*

utilizando dados reais disponibilizados pelo Instituto Nacional de Meteorologia, (INMET, 2016).

- Também é de interesse ampliar o escopo do trabalho através da pesquisa sobre as ferramentas mais representativas do ecossistema Hadoop, as quais permitem a automatização dos processos (por exemplo: serialização de dados e gestão de arquivos).

1.3 MOTIVAÇÃO

Segundo (IBM, 2016), nos últimos dois anos foram criados aproximadamente 90% dos dados armazenados no mundo. Estes dados provêm de dispositivos tais como: sensores climatológicos, postagens em redes sociais, imagens e vídeos digitalizados, dados de transações comerciais e sinais GPS (*Global Position System*) de dispositivos celulares, para citar alguns.

Seguindo essa tendência, em 2020, as empresas terão que administrar 10 vezes mais servidores, aumentando em 50 vezes o volume de dados, e com 75 vezes mais arquivos, tudo isto com apenas 1,5 vezes mais pessoas.

Entretanto, somente nos Estados Unidos faltam em torno a 4,4 milhões de profissionais da área de Ciência de Dados e somente 1/3 das vagas foi preenchida.

Dados estão sendo coletados constantemente o tempo todo. Porém a evolução histórica dos dados permite listar as mudanças verificadas nas últimas décadas:

- Nas décadas de 1950 e 1960 os dados eram considerados como um produto;
- Nas décadas de 1970 e 1980 os dados eram considerados como um subproduto;
- A partir da última década do século XX e durante a primeira década do atual, os dados começaram a ser utilizados pelas organizações no processo de tomada de decisão;
- Na presente década os dados são manejados como um bem intangível de grande valor, sendo essenciais no dia a dia das organizações.

Levando em consideração a evolução no volume e no valor dos dados, surgem importantes questões relacionadas diretamente com o assunto:

- O que pode ser feito com todo esse volume de informação?
- Como as decisões são efetuadas com base nos dados coletados pelas organizações?
- Nesse cenário de disponibilidade de dados como manter a produtividade nas organizações?

Estas questões são tratadas nos capítulos seguintes, na sequência natural em que é desenvolvido o presente trabalho.

1.4 ORGANIZAÇÃO DO TEXTO

A organização do texto no trabalho é realizada da seguinte forma: o capítulo inicial trata sobre a introdução, objetivos e motivação para o trabalho realizado. No capítulo 2 é efetuada uma fundamentação teórica utilizada na preparação do trabalho. No seguinte capítulo (o terceiro), é dedicado ao framework Apache Hadoop, suas características, arquitetura utilizada e as etapas de execução do modelo de programação *MapReduce*. O capítulo 4 trata sobre a configuração do ambiente de desenvolvimento do projeto, entanto que no capítulo 5 efetua-se o desenvolvimento do estudo de caso proposto para este trabalho. Já no capítulo 6 são efetuadas as conclusões referentes ao trabalho de pesquisa, e, finalmente, no capítulo 7 são indicadas as referências utilizadas para a elaboração do presente documento.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os principais conceitos pertinentes ao Big Data, a fim de prover as bases teóricas necessárias para a compreensão do tema abordado no presente trabalho.

2.1 PROCESSAMENTO DISTRIBUÍDO / PARALELO

Como consequência do avanço tecnológico e da redução dos custos na fabricação do hardware, os computadores, antes exclusivos das áreas científica e militar, assim como das grandes empresas e governos, foram se popularizando gradativamente ao longo das décadas, alcançando a maioria dos setores da sociedade o que abriu novas possibilidades no uso da tecnologia computacional.

Nesse sentido, foram desenvolvidas soluções que permitiram o tratamento de volumes de dados cada vez maiores, o que deu origem ao conceito de processamento distribuído.

Segundo Tanenbaum e Van Steer (2008, p. 2), um sistema distribuído pode ser visto como uma "coleção de computadores independentes entre si que se apresenta ao usuário como um sistema único e coerente".

Basicamente um sistema distribuído permite que uma tarefa qualquer pode ser dividida e distribuída em várias tarefas de menor tamanho para serem processadas por separado em paralelo.

2.1.1 Clusters e nós

Segundo Buyya (1999), o conceito de *cluster*, ou agregado de computadores, poder ser definido como “um tipo de sistema de processamento paralelo ou distribuído, que consiste em uma coleção de computadores autônomos interligados trabalhando como um único recurso de computação.”

Em continuidade com Buyya (1999), um nó (do inglês *node*) nada mais é do que um sistema de processamento simples (*single core*) ou múltiplo (*multi-core*, estações de trabalho, etc.) que possui seus próprios módulos de memória, recursos físicos de entrada e saída (I/O) e sistema operacional.

Assim, um *cluster* geralmente refere-se a dois ou mais computadores (nós) que operam conectados juntos. Os nós podem estar em um ambiente físico comum (gabinete, armário, bastidor) ou podem estar fisicamente separados, mas conectados via rede local (LAN). Entretanto, o usual é que este tipo de infraestrutura ocupe o mesmo edifício.

Independentemente da configuração, da perspectiva do usuário e das aplicações, um *cluster* de computadores conectados via LAN (*Local Area Network*) aparenta como um sistema único.

Um exemplo característico de *cluster* é encontrado no modelo físico de “máquinas de prateleira”, conhecido em inglês como *commodity machines*, que nada mais são do que máquinas de médio-baixo custo que operam interligadas.

A arquitetura típica de um sistema *cluster* é mostrada na Figura 1:

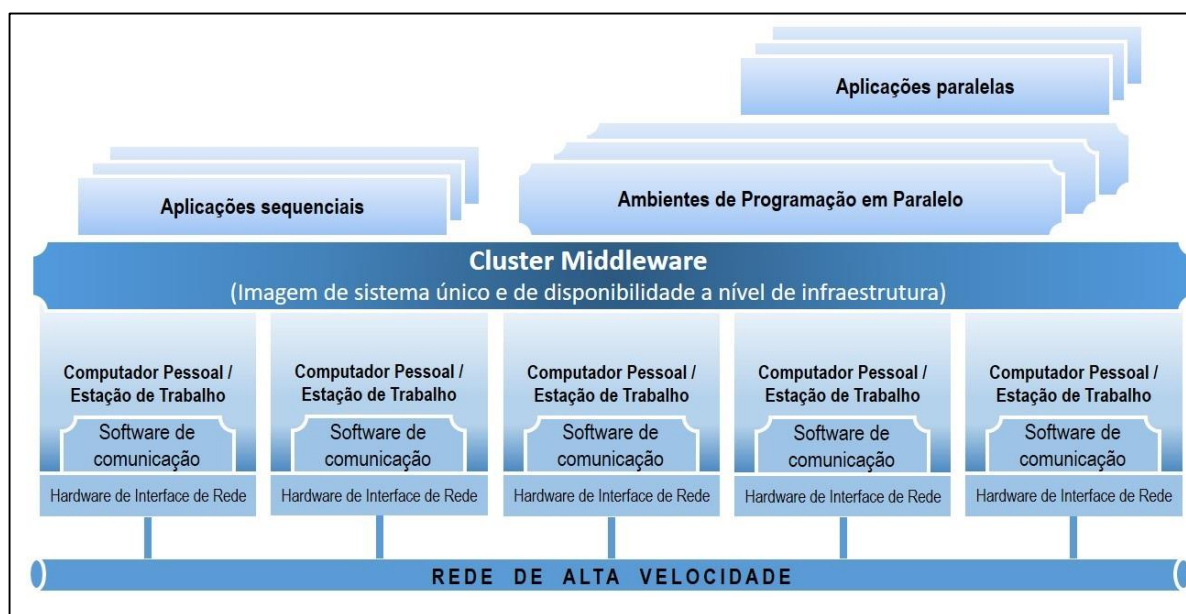


Figura 1 - Arquitetura típica de um *cluster*

Fonte: adaptado de Buyya (1999)

Observando a Figura 1, verifica-se que a arquitetura de *cluster* permite o escalonamento físico, isto é, a adição ou remoção de um nó sem afetar ou perturbar o sistema, o que é característico deste tipo de sistemas.

A seguir estão alguns componentes típicos de um sistema de cluster:

- Vários computadores de alto desempenho;

- Sistemas operacionais (em camada ou *kernel*);
- Redes de alta performance / *switches* (Gigabit Ethernet, etc.);
- Placas de interface de rede (*NICs*);
- Protocolos de comunicação rápida e de serviços;
- Serviço para infraestrutura de *cluster* (*single system image*), que fornece ao usuário a noção de “máquina única”.

Igualmente o escalonamento é verificado a nível de gerenciamento de tarefas, pois a flexibilidade do sistema permite maximizar a quantidade de aplicações executadas, assim como mesclar ordens administrativas com requisições de usuários.

2.2 INTRODUÇÃO AO BIG DATA

Big Data (BD) é uma área das Tecnologias da Informação e da Comunicação (TIC) cuja principal atividade é o armazenamento e tratamento de grandes quantidades de informação ou de conjuntos de dados.

De acordo com Zikopoulos (2013), BD integra o melhor das análises aplicadas em um espectro de dados muito mais abrangente, e que pode representar uma oportunidade para criar uma diferenciação mais expressiva entre as empresas tecnológicas.

No decorrer dos últimos anos a tecnologia BD tem alcançado uma importância maior ao se comparar à estatística e a mineração de dados, tal como pode ser verificado na Figura 2 (GOOGLE TRENDS, 2016). Em vermelho, corresponde ao interesse pela estatística, em azul à DM, e em amarelo o interesse pelo BD.

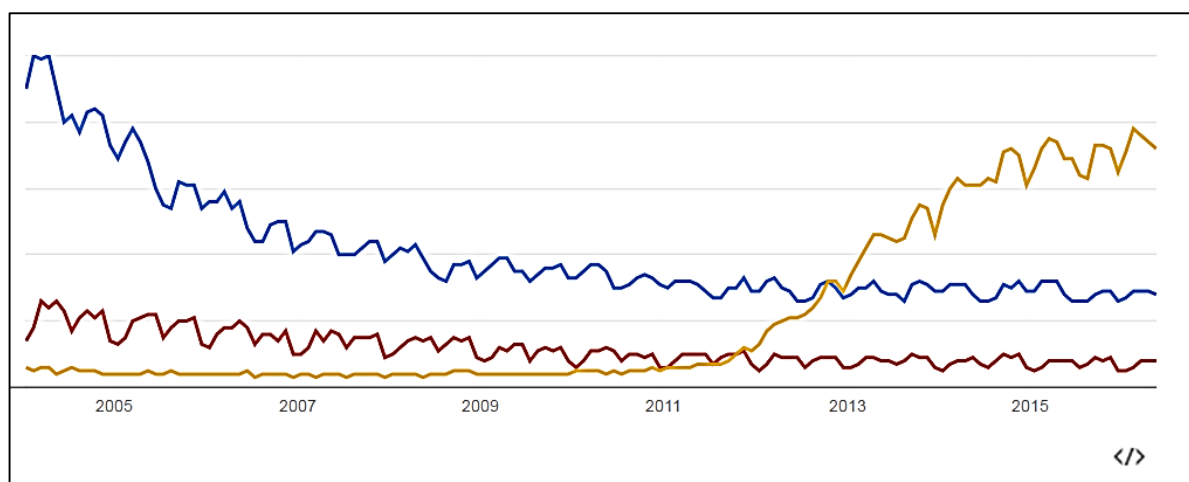


Figura 2 - Gráfico comparativo da média de interesse entre Big Data.

Fonte: *Google Trends* (2016).

De acordo com a seção 1.3, a informação caracteriza uma grande complexidade e igualmente, grande volume de dados de fontes diversas, o que sugere a necessidade de contar com um profissional que conheça e gere utilidade tangível de valor econômico e social. Segundo Zumel e Mount (2014, p.18), grande parte do embasamento teórico referente à Ciência de Dados, provem de dados estatísticos. Entretanto, vem sendo largamente influenciada pela tecnologia e pelas metodologias da engenharia de *software*, sendo desta maneira, amplamente desenvolvida por grupos de trabalho próprios da área de Tecnologia da Informação (TI).

Segundo Provost e Fawcet (2013), Cientista de Dados é uma profissão recente, cujo enfoque é direcionado ao BD, considerado como muito importante no âmbito da tecnologia. O Cientista de Dados requer formação multidisciplinar, isto é, o domínio da estatística e da matemática, assim como um sólido conhecimento em linguagens de programação e análise de sistemas. Nesse sentido pode ser definido como um híbrido entre tecnologia e análise quantitativa.

Em continuidade com Provost e Fawcet (2013), não menos relevante é o fato de que este profissional deverá contar com conhecimento e habilidade necessários para saber comunicar aquilo que vai descobrindo, identificando desta maneira novas oportunidades de negócio. A Figura 3 ilustra a conjunção de disciplinas consideradas como disciplinas chave para a Ciência de Dados.

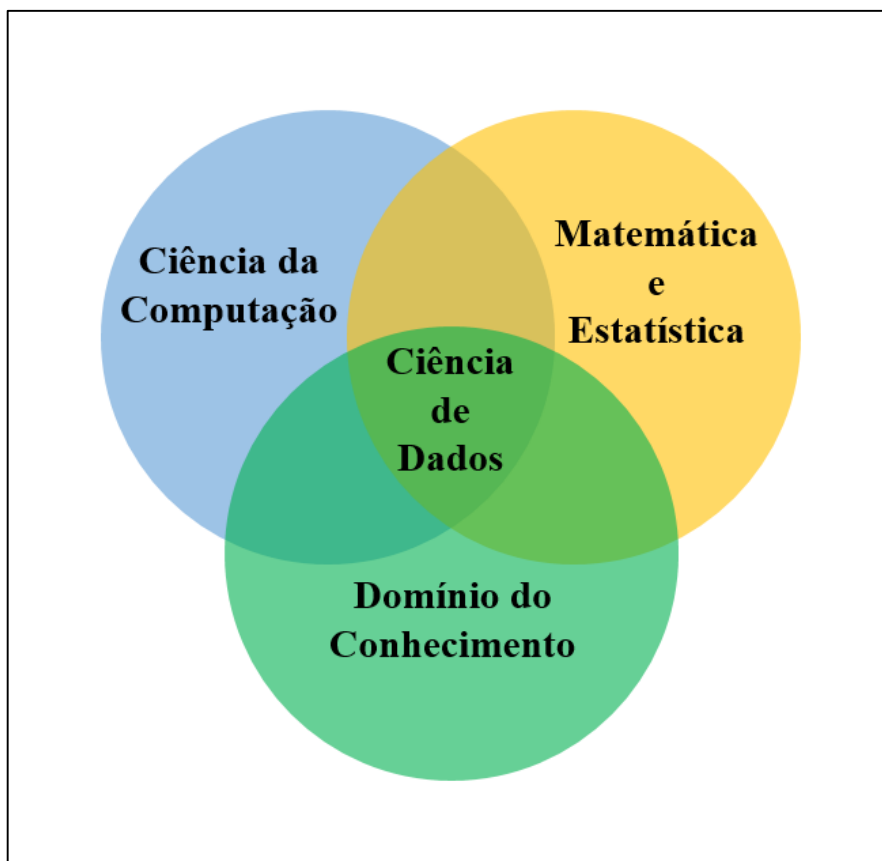


Figura 3 - Disciplinas chave para a Ciência de Dados¹

No que diz respeito do BD, de acordo com Hurwitz et al (2013, p. 25) o mesmo, “todavia não representa uma solução isolada, pois requer de uma infraestrutura que dê suporte à escalabilidade e o gerenciamento dos dados”. Entretanto, ainda com Hurwitz et al (2013), podem ser citadas a modo de exemplo algumas soluções verificadas utilizando ferramentas apropriadas a essa tecnologia tais como Hadoop MapReduce:

- Logística: Tomada de decisões antes da ocorrência de eventos catastróficos tais como desastres naturais;
- Medicina: Prevenção de epidemias;
- Economia: Adequação socioeconômica perante mudança de paradigmas econômicos ou de outra índole.

¹ Data Science and Open Source, Learn about open source tools for converting data into useful information, Jones, Tim Mm – Independent Author.

2.2.1 Os 5 V

De acordo com Zikopoulos et al.(2013, p. 19) ao falar sobre BD frequentemente são citadas as “5 V” –IBM foi quem definiu as primeiras três e logo depois foram adicionadas as outras duas– em inglês *volume*, *variety*, *velocity*, *veracity* e *value*, as quais definem objetivamente o propósito deste tipo de sistemas:

Volume: um sistema BD é capaz de armazenar uma grande quantidade de dados mediante infraestruturas escaláveis e distribuídas. Nos sistemas de armazenamento atuais começam a aparecer problemas de rendimento ao ter quantidades de dados de ordem e magnitude de petabytes ou superiores. Justamente BD esta pensado para trabalhar com estes volumes de dados.

Variiedade: as novas fontes de dados proporcionam novos e distintos tipos e formatos de informação aos já conhecidos (como dados não estruturados), que um sistema Big Data é capaz de armazenar e processar sem ter que realizar um processo prévio para estruturar ou indexar a informação.

Velocidade: uma das características mais importantes é o tempo de processamento e resposta sobre estes grandes volumes de dados, obtendo resultados em tempo real. E não apenas se trata de processar, senão, também de receber, hoje em dia as fontes de dados podem armazenar tal informação de maneira muito veloz.

Veracidade: é muito importante poder avaliar a autenticidade dos dados obtidos nos processos BD, e é por isso que as tecnologias que compõem uma arquitetura Big Data devem ser verificáveis na hora de adaptarem-se a novas mudanças no formato de dados (tanto na obtenção como no armazenamento) e seu processo. É um fato sabido que a evolução é uma constante na tecnologia de maneira que os novos sistemas devem estar preparados para aceita-los.

Valor: o objetivo final é gerar um valor de negócio de toda a informação armazenada a través de diferentes processos de maneira eficiente e com o custo mais baixo possível.

Desta forma, um sistema Big Data deve extrair valor (na forma de nova informação por exemplo) sobre grandes volumes de dados, da maneira mais rápida e eficiente possível.

2.3 ARQUITETURA BD

A arquitetura BD em geral é constituída por um modelo de cinco camadas (IBM, 2014):

1. coleta de dados,
2. armazenamento,
3. processamento de dados,
4. visualização,
5. administração.

Este modelo não é recente; sendo utilizado extensivamente nas soluções atuais de *Business Intelligence* (BI).

Na Figura 4 pode ser observado um exemplo de fluxo de informação em camadas dentro de um sistema BD, sendo a alimentação proveniente de diversas origens, tais como bases de dados, diferentes formatos de documentos, *data streaming*, etc., os quais são recebidos e armazenados através da camada de coleta, com ferramentas especificamente desenvolvidas para tal função. Os dados recebidos podem ser processados, analisados e visualizados segundo a necessidade de cada caso ou processo específico.

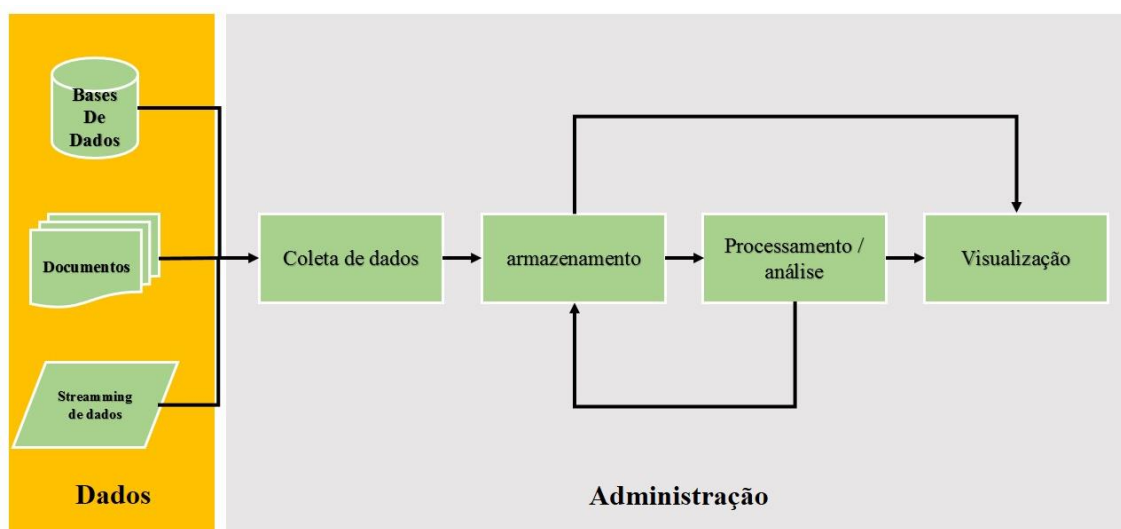


Figura 4 - Arquitetura Big Data em camadas.

Fonte: adaptado de IBM (2012).

2.3.1 Coleta de dados

Nesta camada o sistema deve se conectar com as fontes de informação a fim de obter essa informação. Em função de como seja efetuado o acesso às fontes, as ferramentas de aquisição dos dados podem ser divididas em dois grupos:

- **Por lotes ou *Batch*:** o sistema conecta a cada certo tempo com a fonte de dados na procura de nova informação. Geralmente este processo é utilizado nos sistemas de arquivos ou nas bases de dados, procurando mudanças desde a última conexão. Um exemplo é uma ferramenta para efetuar a migração periódica de dados, por exemplo, de um banco de dados a outro, caracterizando um processo de coleta de dados em lote.
- **Transmissão em tempo real (interativo) ou *Streaming*:** o sistema permanece conectado com a fonte de dados de maneira contínua, descarregando informação cada vez que esta é transmitida. Este processo é mais utilizado no monitoramento de um sistema a fim de aumentar a segurança e a detecção de falhas, por exemplo, no controle de sensores ou para obter informação em tempo real das redes sociais.

Para o estudo de caso que ocupa este trabalho, será utilizado o sistema similar ao do Batch.

Igualmente nesta etapa, de acordo com a necessidade da aplicação, os dados podem sofrer algum tipo de modificação ou alteração, por exemplo: aplicação de um filtro para descartar a informação não desejada ou a formatação da informação. Em qualquer caso as alterações serão salvas no sistema de armazenamento.

2.3.2 Armazenamento

A camada de armazenamento possui em um nível geral dois elementos fundamentais: o sistema de arquivos e a base de dados.

Até recentemente, os sistemas de tratamento da informação eram focados principalmente nas bases de dados, entretanto, sendo estas geralmente pouco flexíveis e como nos sistemas BD o que se procura é a maior variedade possível, em consequência os sistemas de arquivos tem adquirido uma importância cada vez maior.

2.3.3 Bases de dados

As bases de dados sempre tiveram uma importante presença nos sistemas de exploração da informação, principalmente as bases de dados relacionais, as quais estabeleceram um novo paradigma na década de 1970 em parte graças à facilidade de conceitualização de problemas.

Com a aparição dos sistemas relacionais surgiu também a especificação da linguagem SQL, desenhada para trabalhar neste paradigma (exemplos: PostgreSQL, SQLite, MySQL, MariaDB, etc.). Nas linguagens baseadas em SQL (Simple Query Language) é característico um sistema de consultas bastante simples, similar à linguagem humana, utilizando características algébricas e de cálculo relacional permitindo a recuperação da informação buscada de forma simplificada.

Embora os sistemas baseados em SQL sejam relativamente simples e rápidos na execução de consultas –graças à criação dos índices– os SGBD (Sistema de Gerenciamento de Bases de Dados) relacionais tradicionais possuem certos atributos que limitam o rendimento ao lidar com problemas em ambiente BD. Efetivamente, quando a informação armazenada supera o limite dos terabytes, manter a informação estruturada supõe um custo na criação e manutenção dos índices e no desempenho das consultas. Adicionalmente, são pouco flexíveis ao respeito da modificação na sua estrutura, uma vez montada; por exemplo: ao adicionar novas colunas numa tabela ou ao trocar o tipo de uma coluna.

Em decorrência dos problemas que envolvem bases de dados relacionais e sistemas BD, as bases de dados não relacionais *NoSQL* e as *Not only SQL* foram ganhando popularidade, principalmente entre as empresas cujo ramo de atividade é focado em internet e nas redes sociais.

As bases de dados não relacionais não seguem o modelo relacional e portanto não utilizam a linguagem SQL, aportando mais flexibilidade pois não requerem estruturas estáticas (tabelas).

Outra vantagem destes sistemas é que respondem ao quesito de escalabilidade já que não precisam de manter índices para a localização dos dados.

2.3.4 Processamento e análise

Uma vez tendo os dados armazenados, o passo seguinte em um sistema Big Data é explorar a informação para chegar aos resultados desejados. As ferramentas de análise e processamento de informação tem evoluído consideravelmente, especialmente aquelas que trabalham sobre dados não estruturados.

Em geral todas estas ferramentas desenvolvidas recebem a denominação genérica de “ecossistema”, que é visto no capítulo seguinte.

Nas arquiteturas Big Data mais recentes procura-se otimizar o paradigma de programação *MapReduce*. Se cria um sistema de armazenamento (seja um sistema de ficheiros distribuído ou uma base de dados NoSQL) para armazenar a informação não estruturada em grandes volumes de dados e, posteriormente, se armazenam os resultados dos processos e análises realizados sobre estes dados em um sistema SQL, obtendo uma maior velocidade de resposta ao consultar os resultados.

2.3.5 Visualização

As interfaces e modelos de visualização são as que experimentaram uma menor mudança a respeito as arquiteturas mais tradicionais. Como foi comentado no item 2.3.2, os resultados visualizados do processamento costumam ser consultados sobre as bases de dados relacionais ou SQL, já que são as que oferecem um menor tempo de resposta.

2.4 ARQUITETURA DO FRAMEWORK MAPREDUCE

Este *framework* é frequentemente relacionado com o conceito de *cluster* de computadores, ou seja, um sistema escalar com um grande poder de processamento de dados. Deste modo é possível trabalhar com uma grande quantidade de dados distribuindo-os simultaneamente em vários computadores.

O modelo *MapReduce* foi criado e implementado inicialmente em grande escala pela Google, através do projeto *Google MapReduce*. (DEAN, GHEMAWAT, 2008).

Uma das características marcantes deste modelo diz respeito da sua flexibilidade com relação aos diversos ambientes de trabalho sobre os que pode ser aplicado. Em consequência, o desempenho do *MapReduce* pode variar maneira de expressiva se comparado, por exemplo, entre um computador com um simples CPU (*Central Process Unit*) e um *cluster* com centenas de nós.

2.4.1 Função *Map* e função *Reduce*

Segundo Dean e Ghemawat (2008) a principal característica do *MapReduce* é a capacidade de separar os dados e distribuí-los junto ao código que deverá ser executado entre os diversos nós (*nodes*) do sistema.

Este modelo possui raízes na programação funcional, utilizando o conceito de mapear (*map*) e reduzir (*reduce*), as quais são primitivas presentes em várias linguagens funcionais, tradicionalmente mais próximas do ambiente acadêmico e de pesquisa do que da área de software comercial.

Cabe ao programador implementar adequadamente estas duas funções e definir suas propriedades, configurando a maneira em que operam (mapeamento e posterior redução) sobre os dados tratados (Hurwitz et al, 2013). Por outra parte, cabe ao próprio *framework* tarefas tais como a comunicação, a concorrência e o tratamento de erros, pois todos estes pertencem a camada de abstração da ferramenta.

A Figura 5 mostra o fluxo e a distribuição dos dados em um típico processo *MapReduce*, onde os dados “crus” ou metadados são separados em tamanhos menores a serem tratados com as funções, primeiramente a do mapeamento e na sequência com a redução e a posterior saída dos dados processados:

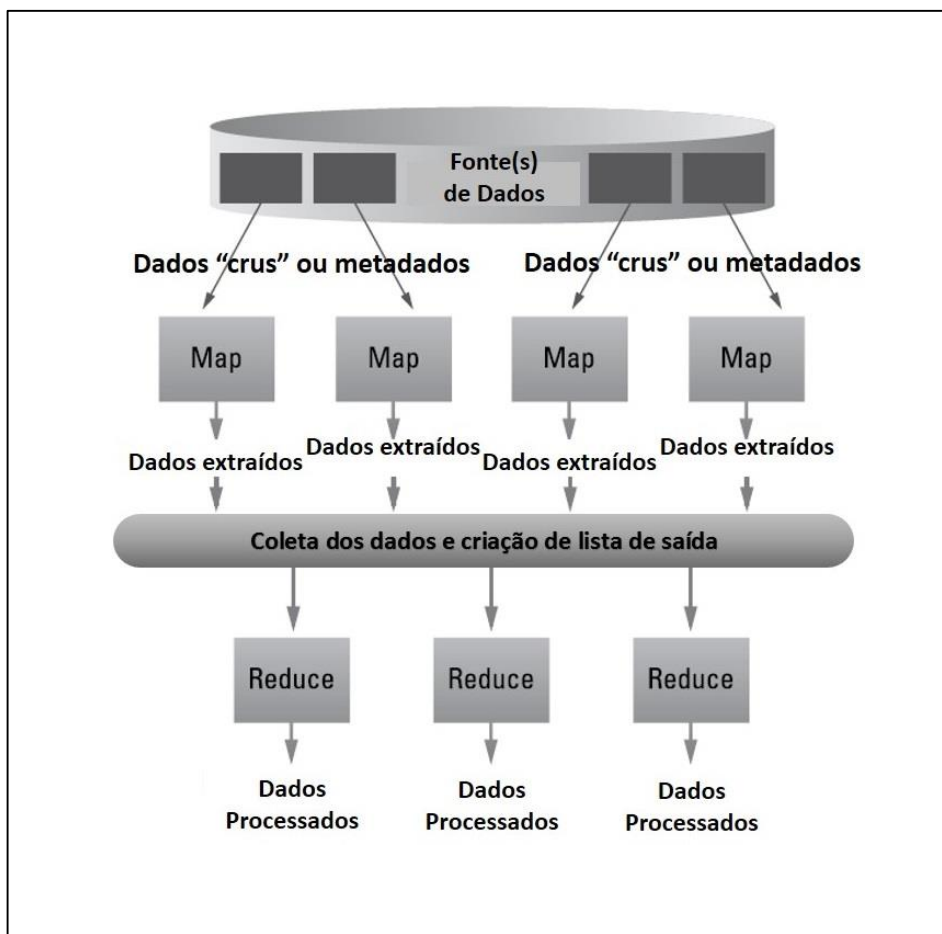


Figura 5 - Fluxo de Dados num processo típico MapReduce

Fonte: adaptado de Big Data for Dummies, p.106

O objetivo do *MapReduce* é o de melhorar o processamento de volumes expressivos de dados em sistemas distribuídos, principalmente com arquivos de tamanhos da ordem de gigabytes e terabytes. Também efetua a otimização no tratamento dos dados estruturados, pois opera à nível de sistema de arquivos.

O nome faz referência implícita à arquitetura do modelo, dividida em duas fases executadas sob uma infraestrutura formada por vários nós, conformando desta maneira um sistema distribuído, e que funciona da seguinte maneira:

Os nós do sistema são diferenciados em dois tipos: o *master* e o *worker* (em português: mestre e trabalhador, respectivamente), seguindo um modelo típico de arquitetura *master-slave*, ou seja mestre-escravo (White, 2015).

A função do nó mestre é a de atender a requisição do usuário, dividindo as *jobs* (requisições de execução) em várias *tasks* (tarefas) a serem distribuídas entre os nós

trabalhadores (ou seja os *workers*), os quais por sua vez as executam de acordo com a função definida pelo usuário (*map, reduce*).

O processo todo utiliza um sistema de arquivos próprio da arquitetura (no caso do Hadoop é o HDFS, *Hadoop Filesystem*), onde ficam armazenados os dados a serem utilizados pelas requisições.

De acordo com Dean e Ghemawat (2008), o processo *MapReduce* é dividido em duas fases:

Map: um dos nós, com o atributo “*master*”, se encarrega de dividir os dados de entrada (um ou vários ficheiros de tamanho grande) em vários blocos a serem tratados em paralelo pelos nós do tipo “*worker map*”. Cada bloco é processado independentemente do resto por um processo que executa uma função *map*. Esta função tem por objetivo realizar o processamento de dados e deixar os resultados em uma lista de pares chave-valor (ou seja, se encarrega de “mapear” os dados).

$$\text{map}(\text{input_key}, (\text{input_values})) \rightarrow \text{list}(\text{output_key}, \text{output_value})$$

Reduce: os nós worker do tipo reduce executam uma função reduce que recebe como entrada uma das chaves geradas na etapa de map junto com uma lista de valores correspondentes a essa chave. Como saída gera uma lista resultante de uma função com os valores recebidos. A união dos resultados pode corresponder a qualquer tipo de função (agregação, soma, máximo, etc.).

$$\text{reducer}(\text{input_key}, (\text{input_values})\text{iterator}) \rightarrow \text{list}(\text{output_key}, \text{output_value})$$

Entre as fases de *map* e de *reduce* existe uma etapa “interna” chamada de agregação (*agregation: shuffle, sort*), cujo objetivo é adicionar e ordenar as saídas dos maps para que na etapa do *reduce* se receba somente a estrutura do tipo chave valores (*key-values*) comentado anteriormente.

Um exemplo simples de um processo *MapReduce* pode ser visto na Figura 6, que conta o número de palavras no parágrafo inicial de Dom Casmurro, a mais famosa obra da literatura brasileira.

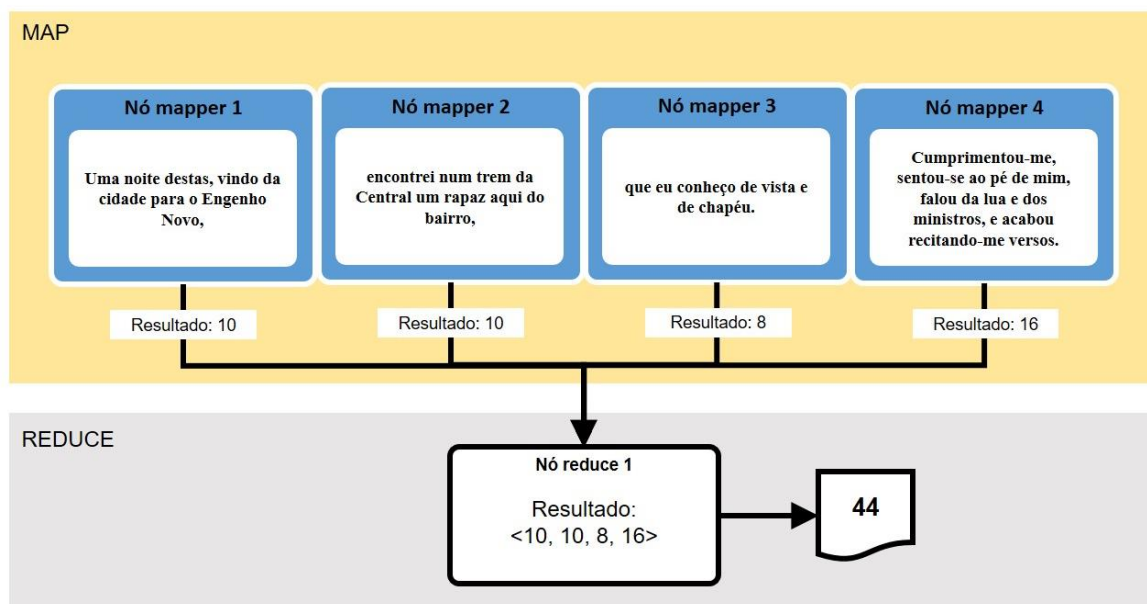


Figura 6 - Exemplo de um processo MapReduce, para contagem de palavras.

Fonte: adaptado de White (2012)

Este modelo de programação permite a implementação de processos que conferem à um sistema com importantes vantagens:

- A distribuição e paralelização são efetuadas de maneira automática.
- Permite trabalhar com dados de qualquer tipo de formato (estruturados, semiestruturados, não estruturados).
- Dispensa elevados requisitos de hardware.
- É escalável a nível do tamanho do *cluster*.
- Opera corretamente com grandes quantidades de dados, da ordem dos petabytes e mais, pois eles são divididos em blocos de tamanho menor independentes com relação ao tamanho original.
- Ao contrário dos sistemas distribuídos tradicionais, o algoritmo é deslocado até os dados, garantindo a integridade do processo.
- Funcionamento e manutenção transparentes aos desenvolvedores, os quais somente devem se preocupar com a lógica de negócio do algoritmo, pois cabe ao próprio sistema a gestão dos erros e os parâmetros da distribuição.

2.4.2 Algoritmos

A Figura 7 mostra um exemplo simples baseado em pseudocódigo, mais conhecido como *WordCount*, (Dean e Ghemawat, 2008, p. 2) que geralmente constitui uma introdução ao *MapReduce*. O objetivo desta aplicação não é outro do que contar a quantidade de ocorrências das palavras presentes em um texto.

```
1  Funcao map (Int key, String value):
2      #key      : n° da linha
3      #value    : texto na linha
4      listaDePalavras = split (valor)
5      for palavra in listaDePalavras:
6          emitir(palavra, 1)
7
8  Funcao reduce (String key, IntegerIterator values)
9      #key      : palavra emitida pela função map
10     #values   : conjunto de valores emitidos para a chave
11     total = 0
12     for v in values:
13         total = total +
14         emitir (palavra, total)
```

Figura 7 - Algoritmo da função *Map* e da função *Reduce* representado através de pseudocódigo *WordCount*.

Fonte: Dean e Ghemawat (2008).

O próprio framework Apache Hadoop possui uma série de exemplos para a prática em linguagem Java, entre os quais aparece a implementação em Java do *WordCount*.

A entrada dos dados corresponde a um trecho de documento de texto. No exemplo da Figura 7, cada chamada da função *map* recebe primeiro como valor (*value*) uma linha de texto do documento processado, e atrelado a este uma chave (*key*) com o número da linha. Assim, para cada palavra encontrada na correspondente linha é gerado um par chave/valor (*key/value*), onde a chave é a palavra em si, e o valor a constante 1 (um). Por sua vez, a função *reduce* recebe como entrada uma palavra (*key*), e um *iterator* para todos os valores emitidos pela função *map*, associados com a palavra tratada. Deste modo todos os valores são somados e um

par *key/value* (contendo a palavra e o total de ocorrências da mesma no documento de texto) é emitido como saída.

O fluxo lógico do processo de execução do exemplo em pseudocódigo pode ser visto com maior detalhe na Figura 8, onde o arquivo de entrada (documento de texto) é processado pela função *map* (configurada pelo usuário) gerando os correspondentes pares, os quais por sua vez são tratados pela função *reduce* (configurado pelo usuário), gerando finalmente um arquivo de saída com a palavra e um número de ocorrências da mesma no documento processado. Em seguida será explicado com maior detalhe as etapas que compõem o fluxo lógico de execução do *MapReduce*.

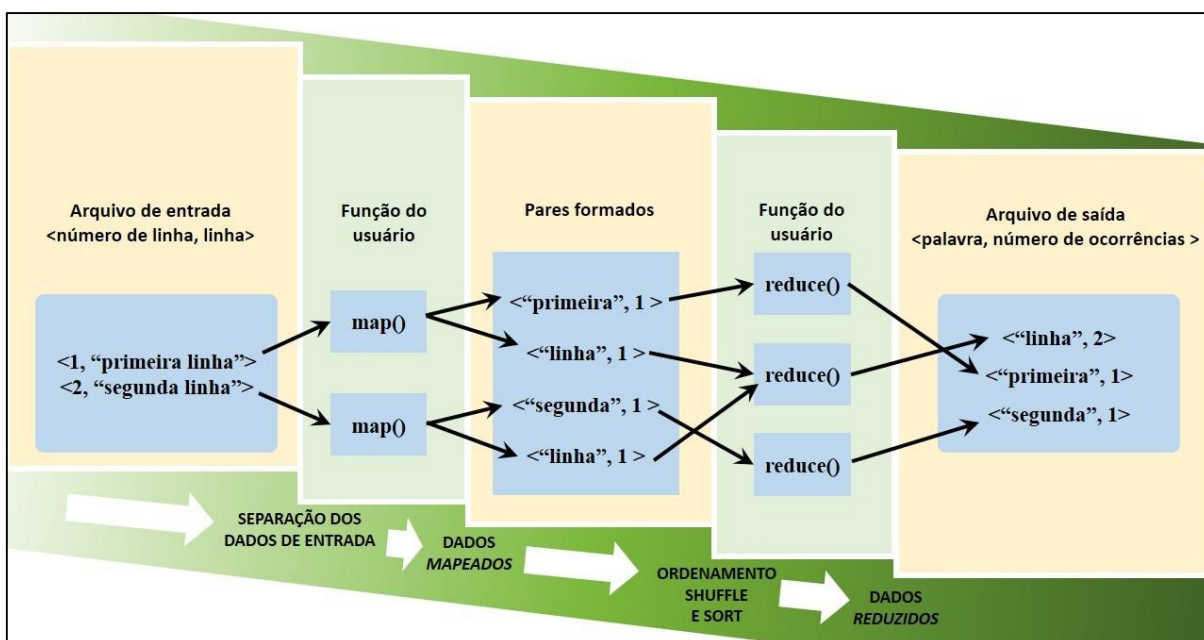


Figura 8 - Fluxo lógico num processo de execução do MapReduce.

Fonte: adaptado do Apache Hadoop (2016).

2.4.3 Limitações do MapReduce

Com relação às limitações do paradigma, segundo DeWitt e Stonebraker (2008), existem algumas bem documentadas, entre elas:

Depuração: sendo que o código do desenvolvedor é deslocado até os nós do *cluster*, fica mais difícil encontrar eventuais falhas no código. Neste sentido é importante testar corretamente o aplicativo antes de leva-lo até o sistema.

Implementação de outras funções: como o uso de *logs*, pois isto incide negativamente no desempenho dos processos MapReduce.

Latência: um *job MapReduce* demora tipicamente em torno de dez segundos. Em consequência, se o volume a processar for pequeno, tal vez o paradigma não seja a solução mais indicada.

Compatibilidade: existem algoritmos que não podem ser implementados utilizando este paradigma, como por exemplo o algoritmo de grafos Dijkstra, que requer um processamento sequencial.

Velocidade de processamento: a velocidade do processamento pode não ser tão elevada se comparado com a velocidade de uma base de dados.

No capítulo a seguir, será apresentada uma descrição sobre o Apache Hadoop, assim como os elementos relacionados a este framework e o ambiente de utilização do mesmo.

3 APACHE HADOOP

Esta seção descreve os conceitos relacionados com o framework Hadoop da Fundação Apache, sua implementação, e os elementos desta plataforma a serem utilizados no projeto do estudo de caso.

3.1 O QUÉ É HADOOP

Hadoop constitui-se em um projeto *open-source* do *MapReduce*, sendo o resultado da colaboração entre grandes empresas tais como Yahoo!, Facebook, Cloudera, Microsoft, Intel, Uber, entre outros (Hadoop, WhoWeAre, 2016).

Sua finalidade é a de um conjunto de ferramentas, aplicações e frameworks Java para o desenvolvimento de sistemas de computação escalável e distribuída.

Hadoop é baseado em dois trabalhos: o projeto original da Google para *MapReduce* e para o Google File System (GFS), o sistema de arquivos distribuído da mencionada empresa.

Atualmente o Hadoop vem sendo utilizado de maneira intensiva por um grande número de empresas em seus processos –tanto internos como os de serviços ao cliente– e dentre as mais expressivas são a IBM², Adobe³, eBay⁴, Spotify⁵, Twitter⁶.

Hadoop permite a criação de aplicações para o processamento de grandes volumes de informação distribuída através de um modelo de programação simples utilizando clusters. O framework é escalável e permite operar com armazenamento e processamento local, porém distribuído, funcionando tanto para clusters de apenas um nó como também para uma conformação de vários clusters com milhares de computadores

Outra característica importante do Hadoop é a detecção de erros a nível de aplicação, permitindo a gestão de erros nos nós com um bom nível de tolerância (Hurwitz et al, 2013, p. 107)

O projeto original do Hadoop, foi desenvolvido sobre três blocos fundamentais, que atualmente constituem os produtos principais:

² www.ibm.com;

³ www.adobe.com

⁴ www.ebay.com

⁵ www.spotify.com

⁶ www.twitter.com

- HDFS (Hadoop File System);
- Hadoop MapReduce: para o processamento em paralelo de grandes volumes de dados;
- Hadoop Common: conjunto das principais ferramentas e utilidades que são empregados pela maioria dos projetos.

Na versão 2.X é incluído o projeto YARN (*Yet Another Resource Negotiator*) que aporta um maior controle e gerenciamento de recursos do *cluster*. Para o estudo de caso que ocupa este trabalho será utilizada a versão 2.2.0 do framework Hadoop.

A tecnologia Hadoop tem gerado também outros subprodutos focados para ambiente de produção BD entre os que podem ser citados como principais (Zikopoulos et al, 2013):

Avro: para serialização de dados;

Chukwa: monitoramento de sistemas distribuídos;

Hbase: banco de dados não relacional, distribuído e escalável;

Hive: infraestrutura para as tecnologias relacionadas com data warehouse;

Pig: linguagem desenvolvida para fluxo de dados;

ZooKeeper: para a coordenação dos serviços.

A seguir, são indicadas as características mais importantes da tecnologia Hadoop.

3.2 CARACTERÍSTICAS

De acordo com Apache Hadoop (2016), os principais conceitos básicos que caracterizam a tecnologia Hadoop são os seguintes:

- As aplicações são codificadas em linguagem de alto nível (Java);
- A política de operação reduz ao mínimo a comunicação entre nós;
- Os dados, em princípio, são estendidos no sistema dividindo-os em blocos de 64MB ou 128MB;
- As tarefas *Map* (*map tasks*) trabalham com pequenas quantidades de dados, sendo o normal um bloco por tarefa;

- Durante a requisição de leitura, as tarefas *Map* são executadas nos nós onde se encontram os blocos;
- Caso um nó apresente falha, esta é detectada pelo nó mestre e o trabalho do nó com falha é repassado para outro nó.
- O nó que apresentou falha, uma vez reinicializado será automaticamente adicionado ao sistema para a realização de tarefas;
- A reinicialização de uma tarefa não afeta às restantes que estiverem sendo executadas em outros nós.
- Caso um nó apresente muita lentidão, o mestre poderá executar outra instância da mesma tarefa.

3.3 ARQUITETURA HDFS

HDFS acrônimo de *Hadoop File System*, é o sistema de arquivos distribuído – como explicado antes na seção 2.3.2 – sobre o qual são executadas as aplicações do Hadoop e que proporciona um bom número das características típicas do sistema.

Este sistema de arquivos foi desenvolvido especialmente para operar em hardware de baixo custo e para ser tolerante com relação as falhas.

Segundo Hurwitz et al (2013) o HDFS realiza a distribuição da informação (geralmente de grande volume) em blocos pequenos mais manejáveis e armazena estes blocos no sistema.

A importância deste processo reside na vantagem que oferece em caso de contar com arquivos de tamanho grande e que não permitam o manejo por um nó simples. Desta maneira também é possível verificar a quantidade de blocos em cada nó sem precisar gerar metadados e permissões de execução os quais poderiam limitar os recursos utilizados.

O sistema permite igualmente a replicação dos dados ao longo dos nós caracterizando assim um sistema dinâmico de alta disponibilidade e tolerante a falhas.

3.3.1 Características da arquitetura HDFS

As principais características da arquitetura HDFS são (HDFS Architecture Guide, 2016):

- Esquema amigável: similar aos sistemas de arquivos mais conhecidos, especialmente ao do UNIX (espaço para nomes, permissões de arquivos, segurança, etc.)
- Tolerante a falhas: uma instância HDFS pode alcançar até milhares de computadores operando como servidores, onde cada um deles pode armazenar uma parte do sistema de arquivos inclusive os replicados. Num ambiente deste tipo é bastante comum acontecer falhas, principalmente no hardware.
- Acesso via streaming: dada a natureza deste tipo de sistemas, o usuário precisa ter acesso aos dados com um rendimento elevado e constante.
- Volume de dados: Não apenas referente à capacidade total do sistema de arquivos e sim também ao volume individual dos arquivos que o compõem. Normalmente na ordem dos gigabytes até terabytes e inclusive –para sistemas de grande porte– até a ordem de petabytes ou mais.
- Modelo simples e coerente: o sistema HDFS foi projetado para gravar o arquivo em uma vez, e que depois de registrado não necessite de mudanças posteriores. Desta maneira são garantidos o rápido acesso e a coerência dos dados.
- Portabilidade: o sistema deve ser portátil para uma grande variedade de plataformas, tanto de hardware quanto de software.
- Sistema escalável: o HDFS permite a expansão em quente (*hot spot*) adicionando novos nós sem ter que cancelar ou pausar a execução dos processos no *cluster* e sem precisar de configuração prévia. Ou seja, o próprio sistema determina quais blocos irá armazenar e quais operações irá executar.

A seguir, é realizado um descritivo sobre os serviços utilizados pelo sistema de arquivos Hadoop.

3.3.2 Serviços ou *daemons* HDFS

A versão Hadoop 2.X (versão utilizada neste trabalho) utiliza um sistema de serviços de processos denominado *daemons*. Um *daemon* nada mais é do que um

processo que irá ficar em ativo ou pronto para a atividade enquanto o sistema HDFS estiver em funcionamento (YAHOO DEVELOPER NETWORK, 2016).

Os *daemons* encarregados do funcionamento do sistema são: *NameNode*, *SecondaryNameNode*, *JobTracker* e *TaskTracker*, cujas funções são comentadas na sequência.

A estruturação do sistema HDFS foi realizada observando a visão do *MapReduce* da Google e do Hadoop no referente à hierarquia dos nós. Existe o nó mestre, um *NameNode* e os nós escravos (*DataNodes* ou *workers*, nós de dados); conforme ilustrado na Figura 9.

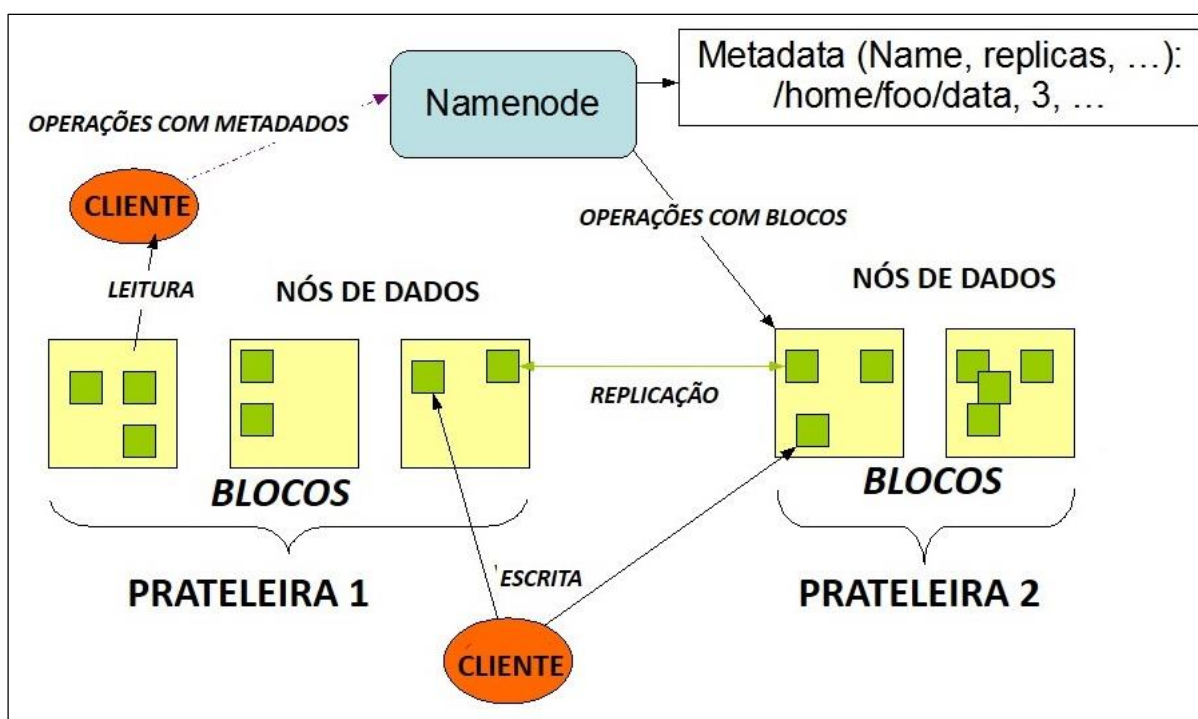


Figura 9 - Arquitetura HDFS

Fonte: HDFS Architecture Guide (2016).

O *NameNode* é o encarregado de gerenciar todo o sistema, mantendo as informações sobre todos os *DataNodes* (nós de dados) assim com a localização de cada bloco. O *NameNode* contém também uma estrutura do tipo árvore de arquivos onde são registradas todas as informações referentes a ele (Hadoop, HDFS Architecture Guide, 2016).

A existência de um único *NameNode* num cluster simplifica grandemente a arquitetura do sistema, concebido de tal modo que os dados de utilizador não são disseminados através do *NameNode*.

O *daemon* ou serviço *SecondaryNameNode* é utilizado como armazém de recuperação de dados do *NameNode*, entretanto sua utilização no *cluster* é opcional. Por outro lado, o *JobTracker* é um *daemon* cuja função é a de gestionar os *jobs* nas tarefas do *MapReduce*.

Finalmente o *TaskTracker* é o *daemon* encarregado de executar uma determinada tarefa em cada nó. As tarefas executadas são aplicações Java *MapReduce*.

3.4 ETAPAS DE UMA EXECUÇÃO MAPREDUCE HADOOP

Segundo White (2012, Cap. 6), a execução das funções implementadas pelo desenvolvedor através de linha de código são as que definem a lógica do fluxo dos dados no processo MapReduce. De acordo com a Figura 10, na fase *map* o par “chave-valor” é a entrada para a função de mapeamento (*mapper*) que gera um novo par “chave-valor” para a tarefa *reduce*.

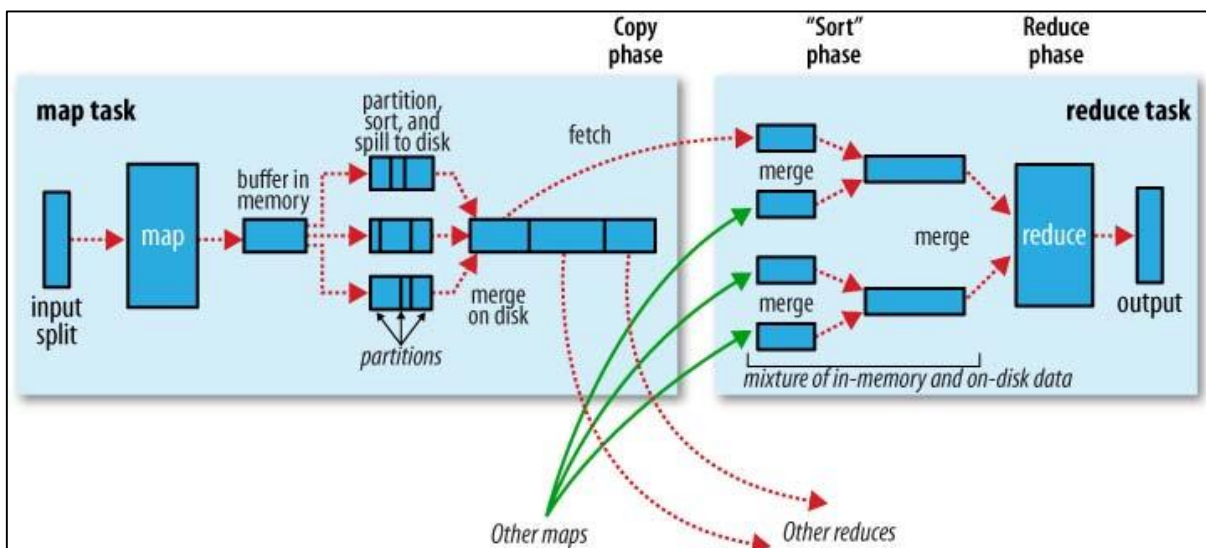


Figura 10 - Etapas do MapReduce.

Fonte: White (2012).

Entretanto, este volume de dados ainda permanece na memória volátil do computador dividido em tantas partições conforme o número de tarefas *reduce* houver. Esta parte do processo é controlada pelo *job* do *mapreduce* (`hadoop.mapreduce.Job` da biblioteca `hadoop` para desenvolvimento java), por exemplo: `Job.setNumReduceTasks(3)` define que haverá 3 tarefas). Em seguida, os dados gerados são alocados (copiados) no sistema de arquivos e passa-se à fase *reduce*.

Em continuidade com a Figura 10, assim que a tarefa *map* é concluída, na seguinte etapa a função *reduce* copia a chave e o valor ou conjunto de valores (que pode ter origem num processo *map* em outra máquina da rede); se o tamanho for pequeno então é armazenado na memória volátil ou RAM, senão, é alocado no sistema de arquivos do disco. Os dados do mapeamento, assim que disponíveis são coletados e ordenados por mesclagem (*merge sort*). Quando todas as partições (blocos) estiverem coletadas e ordenadas então é invocada a função *reduce*. Uma vez finalizado o processo *reduce*, é escrita a saída no sistema HDFS.

Como comentado na seção 3.1, o framework Apache Hadoop, para a versão 2.X (empregada no estudo de caso) utiliza-se tanto do Hadoop Common, como do Hadoop MapReduce e do YARN para a gestão do sistema através de comandos.

De acordo com a documentação do Apache Hadoop, os pacotes utilizados pela plataforma de desenvolvimento são ordenados em hierarquia por classes, interfaces e objetos do tipo *enum*. Entre os pacotes utilizados no projeto está o `conf`, `io` (para serialização), `fs` (`FileSystem`, `Path`), `mapreduce` (`job`, `Mapper`, `Reducer`, `lib`), `util` (`Tool`, `ToolRunner` para controle por linha de comando), entre os mais usualmente utilizados.

3.4.1 Serialização de dados

De acordo com Yahoo Developer Network (2016), os dados precisam ser transmitidos entre os diferentes nós do sistema em ambiente distribuído. Isso requer da serialização e deserialização dos dados para serem convertidos desde seu formato estruturado para o fluxo de bytes e vice-versa.

Em continuidade com Yahoo Developer Network (2016), Hadoop utiliza um protocolo de serialização de dados entre as fases *map* e *reduce* chamado `Writable`. Alguns dos tipos `Writable` por default mais utilizados são:

- `Text` para serialização de `String`;
- `IntWritable` para serialização de `Integer`;
- `FloatWritable` (**Float**), `LongWritable` (**Long**), `ByteWritable` (**Byte**), `DoubleWritable` (**Double**);
- `NullWritable` para emissão de valores do tipo `null`.

A lista de `Writables` é armazenada dentro do pacote `org.apache.hadoop.io`. No projeto de implementação deste trabalho, a cada processo *MapReduce* são recolhidos três elementos do *DataSet*: estação, ano e valor, sendo necessária a serialização dos dados mediante a utilização de um algoritmo `Writable` previamente codificado para o caso do projeto, como pode ser visto no Capítulo 5.

Em fase de produção (multiclusterizado), é desejável a utilização de um algoritmo de compressão dos dados saída, o que fará com que o tamanho dos dados seja menor, facilitando assim a transferência e armazenamento através da rede de clusters.

3.4.2 Comparação no fluxo de dados

Em continuidade com Yahoo Developer Network (2016), a implementação de um comparador `Comparator` permite uma maior eficiência da operação uma vez que realiza a comparação dos registros sem a necessidade da constante deserialização dos mesmos. A linha de código que realiza este processo pode ser vista na Figura 11.

```
public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)
```

Figura 11 – Comparação no fluxo de bytes.

Fonte: Yahoo Developer Network (2016).

Ou seja, através do `Comparator` é possível escolher o ordenamento dos dados buscados no fluxo de bytes sem a necessidade de analisar (*parsing*) todo o fluxo o que permite um maior desempenho do processo pois funciona a nível de byte.

3.4.3 Considerações finais do capítulo

Neste capítulo foi estudado o funcionamento de um processo de execução MapReduce, assim como o processo de serialização e comparação do fluxo de dados. No seguinte capítulo será apresentada a configuração do ambiente de desenvolvimento do projeto.

4 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Neste capítulo serão tratados os aspectos relacionados com o arcabouço prático do trabalho: a preparação dos materiais, obtenção dos dados, e a configuração apropriada das ferramentas de software a serem utilizados para levar à prática o desenvolvimento do projeto.

4.1 HARDWARE E SOFTWARE UTILIZADO

A implementação do estudo de caso é realizada num microcomputador com as seguintes configurações de hardware:

- Processador Dual Core @1.8Ghz, cache L3 2MB;
- Memória RAM 4 GB DDR3 @1,33Mhz;
- Disco rígido de 500 GB de capacidade.

O entorno de software computacional instalado é constituído pelos seguintes:

- SO Linux Ubuntu 14.04 LTS 64 bits;
- Java Open JDK 64-Bit versão 1.7.0_95;
- IDE Eclipse Mars 2;
- Apache Maven 3.2.1;
- Apache Hadoop 2.2.0.

Em Linux considera-se uma prática apropriada a criação de um usuário e um grupo específico para configurar e administrar a ferramenta, nesse caso com o fim de operar as atividades do cluster com maior segurança. Portanto é criado um usuário `hduser` e atribui-se ao mesmo as permissões necessárias via linha de comando.

Também deve ser instalado e configurado o protocolo SSH (*Secure Shell*) para efetuar uma autenticação segura entre os nós do cluster Hadoop e o sistema local. É recomendável não colocar senha de autenticação, caso contrário Hadoop exigirá a informação da senha a cada acesso ao nó ou nós do cluster.

4.2 CONFIGURAÇÃO HDFS

Uma vez instalado e atualizado o SO e configuradas as variáveis de ambiente para Java (JDK e Maven), procede-se à instalação e configuração do Hadoop, seguindo as instruções da página oficial (Apache Hadoop, 2016).

Via comando de terminal é autenticado com o usuário e senha do huser a fim de realizar a instalação e configuração do Hadoop, efetuado na pasta /usr/local/hadoop. Ressalta-se igualmente a importância de configurar corretamente as variáveis de ambiente do Hadoop assim como as configurações do jobtracker e do namenode. Na sequência são verificadas as versões dos softwares instalados, no console do sistema, de acordo com a Figura 12.

```
xiru@xiru-maquina:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.4 LTS
Release: 14.04
Codename: trusty
xiru@xiru-maquina:~$ java -version
java version "1.7.0_95"
OpenJDK Runtime Environment (IcedTea 2.6.4) (7u95-2.6.4-0ubuntu0.14.04.2)
OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)
xiru@xiru-maquina:~$ mvn -version
Apache Maven 3.2.1 (ea8b2b07643d1b84b6d16e1f08391b666bc1e9; 2014-02-14T15:37:52-03:00)
Maven home: /usr/share/maven3
Java version: 1.7.0_95, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
Default locale: pt_BR, platform encoding: UTF-8
OS name: "linux", version: "4.2.0-35-generic", arch: "amd64", family: "unix"
xiru@xiru-maquina:~$ hadoop version
Hadoop 2.2.0
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2.2.0.jar
```

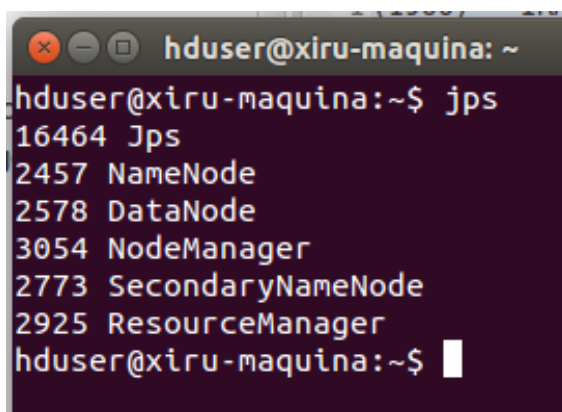
Figura 12 - Tela do Console exibindo as versões instaladas indicadas nos retângulos em vermelho

Fonte própria.

O cluster é configurado no modo pseudo distribuído, simulando um cluster de vários nós, porém rodando apenas numa máquina, ou seja, numa única JVM. Isso significa que haverá uma tarefa *map* e uma tarefa *reduce* para este estudo de caso.

Em seguida é formatado o sistema de arquivos HDFS através do console, pelo comando `hadoop namenode -format` do Hadoop Commons.

Na sequência são iniciados os serviços do cluster através do script `start-all.sh` localizado na pasta `/usr/local/hadoop/sbin` e depois através do comando `jps` verifica-se que todos os processos java estejam em ativo, de acordo com a Figura 13. Se tudo estiver correto na tela deverão ser exibidos os identificadores de processo com os *daemons* respectivos.



```

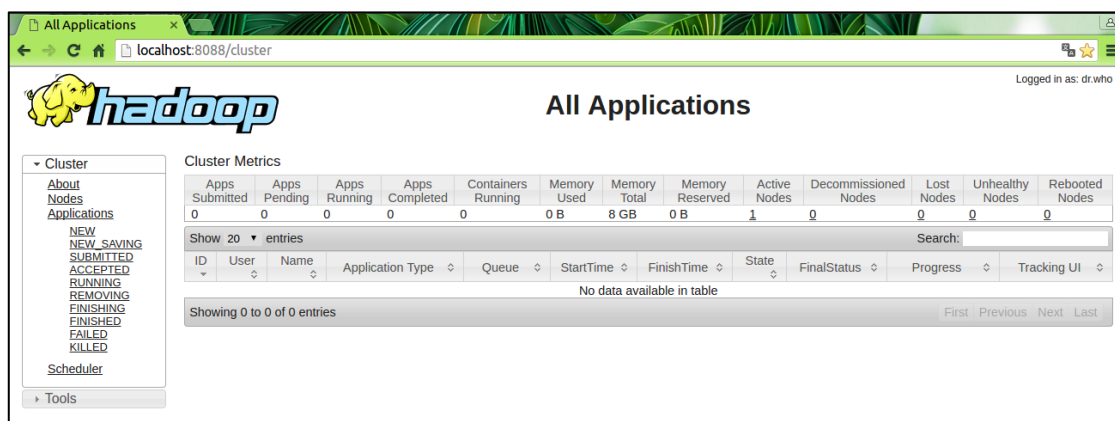
hduser@xiru-maquina: ~
hduser@xiru-maquina:~$ jps
16464 Jps
2457 NameNode
2578 DataNode
3054 NodeManager
2773 SecondaryNameNode
2925 ResourceManager
hduser@xiru-maquina:~$

```

Figura 13 - Console mostrando os processos Hadoop em ativo.

Fonte própria.

Hadoop disponibiliza também o acesso via navegador web em `http://localhost:8080` o que permite o acompanhamento das configurações, dos processos (*jobs*) executados no entorno, assim como os logs, o histórico do cluster entre outros (Figura 14).



The screenshot shows the Hadoop web interface at `localhost:8080/cluster`. The page title is "All Applications" and it is logged in as "dr.who".

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	8 GB	0 B	1	0	0	0	0

Showing 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
No data available in table										

Showing 0 to 0 of 0 entries

Figura 14 - Interface Web do cluster local.

Fonte: autoria própria.

A informação do *namenode* é acessada via HTTP onde são mostrados dados do *namenode* e do arquivo de sistema HDFS de maneira gráfica (Figura 15)

NameNode 'localhost:54310' (active)				
Started:	Thu Apr 21 17:33:24 BRT 2016			
Version:	2.2.0, 1529768			
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0			
Cluster ID:	CID-5a84966c-02c9-41a0-9154-c67993305883			
Block Pool ID:	BP-323269640-127.0.1.1-1460517282220			
Browse the filesystem NameNode Logs				
Cluster Summary				
Security is <i>OFF</i>				
20 files and directories, 12 blocks = 32 total.				
Heap Memory used 68.77 MB is 75% of Committed Heap Memory 90.50 MB. Max Heap Memory is 889 MB.				
Non Heap Memory used 28.05 MB is 96% of Committed Non Heap Memory 28.94 MB. Max Non Heap Memory is 214 MB.				
Configured Capacity	:	357.49 GB		
DFS Used	:	32.61 MB		
Non DFS Used	:	24.92 GB		
DFS Remaining	:	332.54 GB		
DFS Used%	:	0.01%		
DFS Remaining%	:	93.02%		
Block Pool Used	:	32.61 MB		
Block Pool Used%	:	0.01%		
DataNodes usages	:	Min %	Median %	Max %
		0.01%	0.01%	0.01%
Live Nodes	:	1 (Decommissioned: 0)		
Dead Nodes	:	0 (Decommissioned: 0)		

Figura 15 - Interface Web do NameNode.

Fonte: autoria própria.

4.3 OBTENÇÃO DOS DADOS (DATASET)

Antes de trabalhar na codificação de uma solução para o projeto, é imperativo obter o e analisar previamente (através da observação) o material que será utilizado como fonte de dados.

Para o caso, é utilizado uma série de dados reais, em formato estruturado obtidos através do Instituto Nacional de Meteorologia (INMET).

Segundo o próprio site do INMET, “o Banco abriga dados meteorológicos diários em forma digital, de séries históricas das várias estações meteorológicas

convencionais da rede de estações do INMET com milhões de informações, referentes às medições diárias, de acordo com as normas técnicas internacionais da Organização Meteorológica Mundial.

No banco de dados do INMET –o BDMEP– estão acessíveis os dados diários a partir de 1961 das estações para as quais se disponha, em forma digital, de pelo menos 80% dos dados que foram registrados naquele período. Os dados históricos referentes a períodos anteriores a 1961 ainda não estão em forma digital e, portanto, estão indisponíveis no BDMEP.

As variáveis atmosféricas disponibilizadas para consultas no BDMEP são: precipitação ocorrida nas últimas 24 horas; temperatura do bulbo seco; temperatura do bulbo úmido; temperatura máxima; temperatura mínima; umidade relativa do ar; pressão atmosférica ao nível da estação; insolação; direção e velocidade do vento” (INMET, 2016).

A fim de ter acesso aos dados, o portal exige um cadastro, onde são informados dados tais como nome, sobrenome, e-mail, dados relacionados com a instituição do interessado, entre outros.

Na janela de acesso (usuário e senha) o INMET solicita citar sempre a fonte dos dados, como "Fonte: Dados da Rede do INMET" e também fornecer cópias dos trabalhos e/ou teses/dissertações realizadas, que utilizam os dados do INMET.

Para este estudo de caso em particular, é de interesse acessar os dados meteorológicos do estado do Paraná. No menu do site acessa-se “Estações e dados” e no sub menu seleciona-se “BDMET - Dados Históricos”.

Depois de informar os dados de acesso, escolhe-se a “Série Histórica – Dados Diários” que constitui a informação utilizada para o projeto, como pode ser visto na Figura 16.

The image shows a screenshot of the INMET website's web interface for accessing historical data (BDMET). The page is titled "Dados Históricos" and features a search form for "Série Histórica - Dados Diários". The form includes the following fields and options:

- Período - Data início (dd/mm/aaaa):** 01/01/1966
- Período - fim:** 31/12/2015
- Região:** Sul
- (OU) Estado:** Parana
- Selecionar Variáveis:**
 - Precipitação(mm)
 - Temp Máxima(°C)
 - Temp Mínima(°C)
 - Insolação(horas)
 - Evaporação do Piche(mm)
 - Temperatura Compensada Média(°C)
 - Umidade Relativa Média(%)
 - Velocidade Vento Média(mps)

A "Pesquisa" button is located at the bottom of the form.

Figura 16 - Interface Web para o acesso aos dados do BDMET.

Fonte: INMET (2016).

Selecionamos todos os campos disponíveis e em seguida é mostrado o mapa do estado do Paraná com as estações disponíveis para o acesso aos dados, de acordo com a Figura 17.

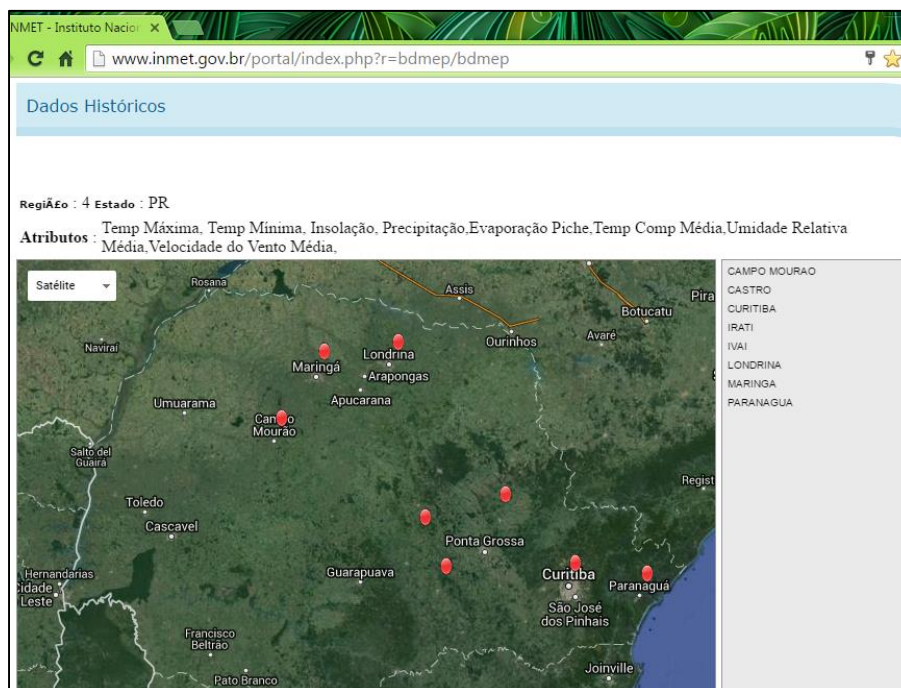


Figura 17 - Interface Web da INMET mostrando as estações meteorológicas do Paraná.

Fonte: INMET (2016).

São selecionadas na sequência cada uma das estações mostradas. Todos os dados são fornecidos diretamente via HTML em formato de texto separado por ponto e vírgula, tal como mostra o exemplo da Figura 18.

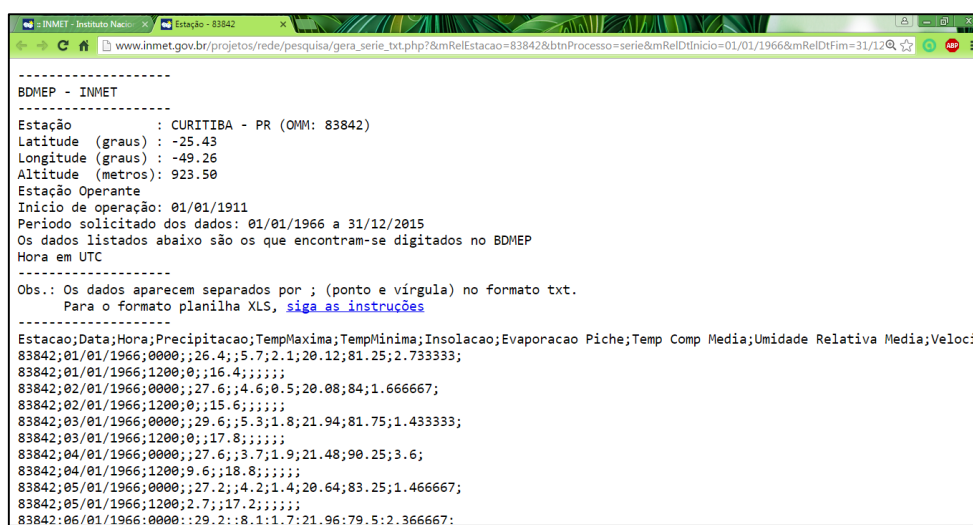


Figura 18 - Interface Web da INMET para série histórica da estação Curitiba – PR

Fonte: autoria própria.

A série histórica selecionada para cada caso inclui os dados dos últimos 50 anos começando no dia 1º de janeiro de 1966 e finalizando no dia 31 de dezembro de 2015.

Os dados de cada uma das estações são então salvos em formato CSV para posteriormente serem analisados e acondicionados de acordo com desenvolvimento do projeto.

4.4 ANÁLISE E PREPARAÇÃO DO DATASET

Observa-se que os dados mostram duas medições diárias em intervalos de 12 horas, totalizando 11 colunas. Para facilitar a interpretação é substituído o código da estação pelo nome do município correspondente, utilizando como ferramenta uma planilha eletrônica comum. Também é criado um novo arquivo de dados, desta vez utilizando os dados de todas as estações.

O *dataset* fica então constituído por nove arquivos CSV totalizando em conjunto 23,4 MB de dados que são copiados para a pasta `input` do usuário `hduser` dentro do sistema de arquivos HDFS, como mostrado na Figura 19.

Uma cópia destes arquivos também será utilizada posteriormente em Eclipse, dentro da pasta `input` do projeto.

Contents of directory `/user/hduser/input`

Goto : go

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
dados_bdmnet_1966_2015.csv	file	11.15 MB	1	128 MB	2016-04-21 13:49	rw-r--r--	hduser	supergroup
dados_bdmnet_CAMPOMOURAO_1966_2015.csv	file	1.68 MB	1	128 MB	2016-04-21 17:42	rw-r--r--	hduser	supergroup
dados_bdmnet_CASTRO_1966_2015.csv	file	1.36 MB	1	128 MB	2016-04-21 17:43	rw-r--r--	hduser	supergroup
dados_bdmnet_CURITIBA_1966_2015.csv	file	1.57 MB	1	128 MB	2016-04-21 17:46	rw-r--r--	hduser	supergroup
dados_bdmnet_IRATI_1966_2015.csv	file	1.38 MB	1	128 MB	2016-04-21 17:43	rw-r--r--	hduser	supergroup
dados_bdmnet_IVAI_1966_2015.csv	file	972.39 KB	1	128 MB	2016-04-21 17:46	rw-r--r--	hduser	supergroup
dados_bdmnet_LONDRINA_1966_2015.csv	file	1.41 MB	1	128 MB	2016-04-21 17:44	rw-r--r--	hduser	supergroup
dados_bdmnet_MARINGA_1966_2015.csv	file	1.31 MB	1	128 MB	2016-04-21 17:45	rw-r--r--	hduser	supergroup
dados_bdmnet_PARANAGUA_1966_2015.csv	file	1.52 MB	1	128 MB	2016-04-21 17:45	rw-r--r--	hduser	supergroup

[Go back to DFS home](#)

Figura 19 - Estrutura do diretório `input` (*dataset*) pela interface Web do Hadoop.

Fonte: autoria própria.

Com esses dados é possível realizar cálculos por meio de algoritmos que permitam a obtenção de dados que aportem algum valor, como por exemplo: evolução da temperatura regional em máximos, mínimos e média compensada ao longo dos últimos 50 anos, evolução da umidade relativa média, máximo de precipitação, etc.

4.5 DESENVOLVIMENTO DO PROJETO

Utilizando o Eclipse, é criado um projeto Maven com o nome `mapreduce_inmetPR_project`; em seguida é configurada toda a informação necessária (`name`, `groupId`, `artifactId`, `packaging`) dentro do arquivo `POM.xml`, de maneira que o próprio Maven será o responsável pela automatização dos processos de atualização das bibliotecas, assim como do *packaging* do projeto e das informações referidas ao processo *MapReduce*.

No seguinte capítulo é apresentado o desenvolvimento do projeto através de um estudo de caso.

5 DESENVOLVIMENTO DO ESTUDO DE CASO

Uma vez analisados e preparados os dados do INMET, assim como tendo configurado e testado todo o ambiente de desenvolvimento, procede-se a codificação do projeto em ambiente Eclipse Java.

5.1 IMPLEMENTAÇÃO JAVA

De acordo com as boas práticas de programação, é criado o projeto `mapreduce_inmetPR_project` e dentro do mesmo o pacote `br.edu.utfpr` que irá alojar as classes Java utilizadas no desenvolvimento. É importante que a informação contida no arquivo `POM.xml` (Maven) esteja em concordância com a configuração realizada no projeto Eclipse.

Como comentado na seção 3.4.1, a maneira efetiva de tratar os dados do *dataset*, é por meio do uso do *writable*. Ou seja, é necessário desenvolver código para conseguir que os dados tenham utilidade para o projeto. Em função do *DataSet* obtido será utilizado o `WritableComparable` que nada mais é do que a combinação dos métodos da interface `Writable` do Hadoop com a interface `Comparable` própria do Java.

O framework *MapReduce* precisa da instanciação do construtor para o `Writable`; sendo assim, são implementados os métodos `write` para a saída do mapper, e `readFields` para a leitura na tarefa *reduce*.

Também é utilizada a classe `HashMap<K, V>` própria da biblioteca `java.util` o que permite a utilização dos métodos `hashCode` e `equals`.

Para evitar desordem na apresentação dos dados o método `toString` fará com que sejam apresentados primeiramente o ano e a estação separados por hífen.

Na implementação do método `Comparable` também é utilizado o método `compareTo` para efetuar a ordenação das chaves na fase *shuffle and sort*.

Na classe `MedicaoWritable`, o `Comparator` estende `WritableComparator` do Hadoop sobrecarregado o método `compare`. A classe foi desenvolvida para atender a necessidade de tratamento dos dados do *dataset*, como comentado anteriormente na seção 3.5.1. Desde modo o `WritableComparator`

efetua a coleta e armazenamento dos dois dados padrão para o *DataSet* utilizado, ou seja, o ano e a estação.

5.2 A CLASSE DE GERENCIAMENTO

Nesta classe, *medicaoMeteorologica*, é implementado propriamente o *MapReduce* Hadoop. O *mapper* gera a saída *chave – valor*, onde a chave é composta pelo ano e estação e o valor será aquele solicitado ao momento de executar o processo (temperatura máxima, temperatura mínima, umidade, etc.), ou seja, por parâmetro em correspondência com as posições dos dados no *dataset*.

5.2.1 Mapper

No *Mapper* (Figura 20) são recebidos os blocos de dados que contém a informação, os quais são processados linha por linha. A classe *MedicaoMeteorologicaManager* estende da classe *Mapper* para definir os formatos de entrada da chave e valor assim como os tipos de saída da função. De acordo com o paradigma, a saída do *Mapper* será a entrada do *Reducer*. Neste caso, o valor da chave contém o ponteiro que percorre o *DataSet* extraindo cada uma das linhas do mesmo no campo *value*.

No campo *value*, a linha é separada pelo token ";" (do arquivo CSV no *DataSet*) selecionando somente os campos de interesse, de acordo com o argumento a informar no momento da execução. O *Mapper* também efetua a filtração dos dados, validando apenas os dados numéricos (*NumberUtils*) e descartando os nulos e valores de outro tipo no estudo de caso (Figura 20, linhas 57 e 58). Este filtro é importante pois existem linhas do *dataset* cuja informação não foi definida ou apresentam campos em branco.

Na fase *map* a informação que retorna o próprio método é registrada no objeto *context* através de uma tupla (chave, valor). Como comentado na seção 2.4.1, a chave é constituída pelos dados *ano* e *estação*, entanto que o valor é aquele informado no momento da execução, de acordo com o *writable*, sendo a própria tupla do tipo *writable*.

A tarefa do mapeamento é realizada pelo método `setup`, encarregado de recolher o parâmetro indicado para consulta (Figura 20, linhas 38, 39). Logo o parâmetro é invocado pelo método `run` no momento em que o `job` é criado.

```

32
33 public static class MedicaoMeteorologicaMapper extends Mapper<Object, Text, MedicaoWritable, FloatWritable> {
34
35     private static final String DATE_SEPARATOR = "/";
36
37     @Override
38     protected void setup(Context context) throws IOException, InterruptedException {
39         medicaoType = context.getConfiguration().get(MEDICAO_TYPE);
40     }
41
42     public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
43         final String[] values = value.toString().split(SEPARATOR);
44         final MedicaoWritable medicao = getMedicao(values, medicaoType);
45         //System.out.println("values: "+values.length+values[0]+" - "+values[1]+" - "+values[2]+" - "+values[3]);
46
47         final String medicaoValue;
48
49         if (MedicaoType.getOrder(medicaoType) >= values.length) {
50             medicaoValue = format("0");
51         }else{
52             medicaoValue = format(values[MedicaoType.getOrder(medicaoType)]);
53         }
54
55
56         //System.out.println(medicao+" - "+medicaoValue);
57         if (medicao != null && NumberUtils.isNumber(medicaoValue)) {
58             context.write(medicao, new FloatWritable(Float.valueOf(medicaoValue)));
59         }
60     }
61
62     private MedicaoWritable getMedicao(String[] values, String medicaoType) {
63         MedicaoWritable medicaoWritable = null;
64
65         final String date = format(values[DATE_ORDER]);
66
67         if (isValidData(date)) {
68             final String ano = date.split(DATE_SEPARATOR)[2];
69             final String estacao = format(values[ESTACAO_ORDER]);
70
71             medicaoWritable = new MedicaoWritable(ano, estacao);
72         }
73
74         return medicaoWritable;
75     }
76
77     private boolean isValidData(final String date) {
78         return date.contains(DATE_SEPARATOR);
79     }
80
81     private String format(String value) {
82         return value.trim();
83     }
84 }

```

Figura 20 – Código do *Mapper*.

Fonte: autoria própria.

Como parte das boas práticas de codificação, é criada uma classe `enum` que contém referencialmente os nomes das colunas no `dataSet`. Durante a execução, caso

não seja declarada a variável a procurar, por default é indicada a coluna PRECIPITACAO.

5.2.2 Reducer

A chave gerada pelo `writable` contém o *ano* e a *estação*, entretanto que o `mapper` contém a lista das medições recolhidas do *DataSet*. O *Reducer* recebe estes dados e na sequência busca o valor mais alto da lista, dando como resultado uma saída (Figura 21). Este valor é o resultado da aplicação de uma função matemática (`java.lang.Math`) que poderá ser alterado de acordo com o que se busca dentro do *DataSet* a ser trabalhado. Por padrão é deixado inicialmente em `Math.max`.

```

85
86 public static class MedicaoMeteorologicaReducer extends Reducer<MedicaoWritable,
      FloatWritable, MedicaoWritable, FloatWritable> {
87
88     public void reduce(MedicaoWritable key, Iterable<FloatWritable> values, Context
      context) throws IOException, InterruptedException {
89         float medicao = 0f;
90
91         for (FloatWritable medicaoValue : values) {
92             medicao = Math.max(medicao, medicaoValue.get());
93         }
94         context.write(key, new FloatWritable(medicao));
95     }
96 }
97

```

Figura 21 – Trecho de código do *reducer*.

Fonte: autoria própria.

5.2.3 Driver

O driver é a parte final do processo *MapReduce*. O driver inicializa o `job` e indica a plataforma do Hadoop para executar o código desenvolvido junto com o *DataSet*, assim como a gestão dos arquivos de saída (Yahoo Developer Network, 2016).

A classe `Configuration` é a encarregada de receber o argumento de entrada (tipo de medição) quando o `job` é criado. O argumento `inputPath` fornece os dados do *DataSet* contidos na pasta *input*. A saída do processo *reduce* é escrita em arquivos dentro do diretório identificado pelo `outputPath`.

A informação sobre a configuração é capturada no objeto `JobConf`, entanto que as funções de mapeamento e redução são identificadas pelos métodos `setMapperClass()` e `setReducerClass()`. O tipo de dados emitido pelo *reducer* é identificado pelo `setOutputKeyClass()` e pelo `setOutputValueClass()`.

O tipo de dados que alimenta ao `mapper` é controlado pelo uso do `InputFormat`. Por padrão o formato utilizado é gerenciado pelo `TextInputFormat` que carrega esses dados por pares (`LongWritable`, `Text`). O valor maior é o byte de saída na linha do arquivo. O objeto `Text` refere-se ao conteúdo da *String* na linha correspondente.

A chamada do `JobClient.runJob(conf)` irá submeter o `job` para o *MapReduce*. Esta chamada poderá ser bloqueada até que o processo seja finalizado (método `waitForCompletion(true)`). Caso o processo venha falhar, entrará numa `IOException`. Adicionalmente `JobClient` fornece uma versão sem bloqueio de processo chamada de `submitJob()` que não é utilizado neste projeto.

Na figura 22 pode ser visto o código do `driver run` e o `job` para a classe de gerenciamento.

```

98     @Override
99     public int run(String[] args) throws Exception {
100         if (args.length != 3) {
101             System.err.println("MedicaoMeteorologicaManager requer os parametros:
102             <input file> <output dir> <medicao type>");
103             System.exit(2);
104         }
105         deleteOutputFileIfExists(args);
106
107         final Configuration configuration = new Configuration();
108         configuration.set(MEDICAO_TYPE, args[2]);
109
110         @SuppressWarnings("deprecation")
111         final Job job = new Job(configuration);
112
113         job.setJarByClass(MedicaoMeteorologicaManager.class);
114         job.setInputFormatClass(TextInputFormat.class);
115         job.setOutputFormatClass(TextOutputFormat.class);
116
117         job.setMapOutputKeyClass(MedicaoWritable.class);
118         job.setMapOutputValueClass(FloatWritable.class);
119         job.setOutputKeyClass(MedicaoWritable.class);
120         job.setOutputValueClass(FloatWritable.class);
121
122         job.setMapperClass(MedicaoMeteorologicaMapper.class);
123         job.setReducerClass(MedicaoMeteorologicaReducer.class);
124
125         FileInputFormat.addInputPath(job, new Path(args[0]));
126         FileOutputFormat.setOutputPath(job, new Path(args[1]));
127
128         job.waitForCompletion(true);
129
130         return 0;
131     }

```

Figura 22 - job para o *MapReduce*

Fonte: autoria própria.

Finalmente é criado um main que através do `ToolRunner` permitirá a execução da classe de gerenciamento dentro do projeto.

5.3 EXECUÇÃO DO PROJETO

Neste estágio do trabalho, o projeto fica pronto para a fase de execução. Primeiramente deve ser criado o pacote *jar* com o Maven. Isto é feito por linha de comando (console) digitando `mvn package` indicando o caminho do projeto dentro do *workspace* do Eclipse. Utilizando o usuário `hduser`, criado especificamente para administrar o cluster, é verificado que todos os serviços Hadoop estejam efetivamente em andamento, através do comando `jps` no console do SO.

Caso os serviços não estejam inicializados, digita-se o comando `start-all.sh` no console do Ubuntu.

O projeto pode ser levado à execução de duas maneiras:

- Por padrão utilizando linha de comando do S.O. informando `hadoop jar <caminho do jar dentro do projeto> <classe manager dentro do mesmo projeto> <caminho do DataSet.csv dentro da pasta input no sistema HDFS> <saída padrão output> <coluna do DataSet para o writable>`.
- Pelo método *main* em ambiente eclipse, informando os argumentos na janela *run configurations*.

Para o primeiro caso, é possível acessar o arquivo de saída pelo comando `cat` do terminal indicando o caminho dentro do cluster; também é possível acessá-lo via HTTP (<http://localhost:50070>). Para o segundo caso simplesmente é aberto em Eclipse o arquivo gerado na pasta `output` do projeto.

Resulta importante indicar que toda vez que no código seja alterado o elemento buscado, antes de executar o processo *MapReduce*, necessariamente se deverá empacotar o projeto utilizando o Maven.

5.4 APRESENTAÇÃO DOS RESULTADOS

Modificando os parâmetros de busca pode-se obter todos os resultados que o DataSet consiga proporcionar. A seguir são analisados dois exemplos representativos do processo *MapReduce* para o projeto em estudo.

5.4.1 Temperatura máxima e mínima para Curitiba.

Executa-se o projeto buscando, por exemplo, a maior temperatura por cada ano do período ocorrido na estação INMET de Curitiba utilizando como *DataSet* o arquivo `dados_bdmet_CURITIBA_1966_2015.csv`.

Depois de alguns segundos, o sistema notifica o fim do processo. O resultado pode ser acessado via HTTP ou através da IDE do Eclipse, segundo o caso.

O *MapReduce* gerou uma saída de dados com 50 linhas, em concordância com o período de medições diárias durante 50 anos, gerando uma linha por ano de acordo com a Tabela 1, confeccionada a partir desses dados.

Tabela 1 - Máximos diários de temperatura por ano para Curitiba para o período 1966 – 2015

Ano	Temperatura máxima (°C)	Ano	Temperatura máxima (°C)	Ano	Temperatura máxima (°C)
1966	32,8	1983	31,1	2000	31,7
1967	31,8	1984	32,7	2001	33,1
1968	32,6	1985	35,2	2002	33,1
1969	32,0	1986	34,0	2003	33,5
1970	32,1	1987	32,2	2004	33,7
1971	34,0	1988	33,5	2005	33,9
1972	32,3	1989	29,8	2006	34,3
1973	32,3	1990	33,0	2007	33,6
1974	33,5	1991	32,4	2008	32,3
1975	34,8	1992	31,8	2009	33,8
1976	34,1	1993	33,0	2010	33,5
1977	32,2	1994	33,6	2011	32,6
1978	32,8	1995	32,2	2012	34,2
1979	32,4	1996	33,6	2013	33,5
1980	30,8	1997	33,2	2014	34,7
1981	31,5	1998	32,3	2015	33,6
1982	30,8	1999	31,9		

Fonte: autoria própria sobre dados extraídos do BDMET.

Após criar um gráfico para Curitiba com os dados das temperaturas máximas diárias (a temperatura mais alta de cada ano) ocorridas a cada ano entre 1966 e 2015, pode ser visto através da linha de tendência linear (em vermelho) que houve efetivamente um aumento gradativo ao longo do período observado.

Efetuada uma nova operação *MapReduce* sobre o mesmo *DataSet*, porém desta vez buscando os valores mínimos diários (modifica-se previamente no código e compila-se o *jar* com o Maven) é obtido um resultado mostrado na Tabela 2.

Tabela 2 - Mínimos diários de temperatura por ano para Curitiba para o período 1966 – 2015

Ano	Temperatura mínima (°C)	Ano	Temperatura mínima (°C)	Ano	Temperatura mínima (°C)
1966	-1,2	1983	0,0	2000	-3,5
1967	-2,2	1984	0,0	2001	0,0
1968	0,0	1985	0,0	2002	0,0
1969	-2,0	1986	-0,7	2003	0,0
1970	-5,2	1987	-1,3	2004	-1,6

1971	-4,0	1988	-1,3	2005	0,0
1972	-5,4	1989	-2,8	2006	0,0
1973	-4,0	1990	-2,1	2007	-1,3
1974	-1,6	1991	0,0	2008	-0,9
1975	-5,1	1992	-0,7	2009	-0,7
1976	0,0	1993	-1,2	2010	0,0
1977	-1,8	1994	-1,6	2011	-0,7
1978	-3,7	1995	0,0	2012	0,0
1979	-2,1	1996	0,0	2013	-2,1
1980	0,0	1997	0,0	2014	0,0
1981	-0,1	1998	0,0	2015	0,0
1982	0,0	1999	-1,9		

Fonte: Autoria própria sobre dados extraídos do BDMET.

Novamente é criado um gráfico com os dados das temperaturas mínimas diárias (a temperatura mais baixa do ano) ocorridas a cada ano entre 1966 e 2015 e percebe-se que a linha de tendência linear (em vermelho), mostra novamente um aumento gradativo ao longo do período observado, porém mais acentuado que no gráfico de temperatura máxima.

Em ambos casos (máxima e mínima diárias por ano para Curitiba) observa-se que o comportamento tende a aumentar os valores das temperaturas ao longo dos últimos 50 anos, notadamente nos valores mínimos. Portanto conclui-se que estes dados estariam aportando informações de valor para, por exemplo, os profissionais da área de climatologia, meteorologia e correlatos.

5.4.2 Máximo de precipitação por ano para o Paraná

Neste exemplo será determinada a precipitação máxima diária (em mm) por cada ano do período ocorrido em todas as estações INMET distribuídas no estado do Paraná utilizando como *DataSet* o arquivo `dados_bdmet_1966_2015.csv` que contém todos os dados das estações em um arquivo. Para isso é novamente alterada a função matemática `Math` dentro do código substituindo o `min` pelo `max` no nosso `Writable`. Logo de empacotar com o Maven, o código é executado pelo Hadoop obtendo-se os seguintes dados (Tabela 3):

Tabela 3 – Máximo de precipitação diária por ano para Paraná (INMET) no período 1966 – 2015

Ano - Estação	Precipitação máxima diária (mm)	Ano - Estação	Precipitação máxima diária (mm)
1966 - LONDRINA	124,2	1991 - IRATI	141,2
1967 - CASTRO	106,4	1992 - CASTRO	128,0
1968 - MARINGA	136,4	1993 - CAMPO MOURAO	170,3
1969 - PARANAGUA	160,8	1994 - IRATI	116,2
1970 - CURITIBA	94,5	1995 - CAMPO MOURAO	202,9
1971 - CASTRO	101,8	1996 - MARINGA	150,8
1972 - LONDRINA	120,1	1997 - IRATI	151,5
1973 - CURITIBA	107,6	1998 - PARANAGUA	129,5
1974 - LONDRINA	96,3	1999 - IRATI	146,2
1975 - PARANAGUA	134,7	2000 - LONDRINA	100,0
1976 - CAMPO MOURAO	117,0	2001 - PARANAGUA	81,0
1977 - LONDRINA	125,0	2002 - LONDRINA	122,2
1978 - MARINGA	134,4	2003 - IRATI	148,6
1979 - PARANAGUA	122,3	2004 - CAMPO MOURAO	295,8
1980 - IRATI	163,6	2005 - LONDRINA	150,6
1981 - IRATI	105,0	2006 - LONDRINA	110,6
1982 - LONDRINA	164,9	2007 - IRATI	128,2
1983 - IRATI	119,4	2008 - CAMPO MOURAO	146,4
1984 - LONDRINA	152,3	2009 - CASTRO	138,0
1985 - CAMPO MOURAO	111,7	2010 - MARINGA	109,7
1986 - MARINGA	190,5	2011 - PARANAGUA	184,0
1987 - CAMPO MOURAO	154,6	2012 - CASTRO	231,4
1988 - IRATI	136,8	2013 - LONDRINA	139,4
1989 - LONDRINA	137,2	2014 - CURITIBA	191,9
1990 - LONDRINA	117,6	2015 - IRATI	130,4

Fonte: autoria própria com dados extraídos do BDMET.

Com estes dados observa-se que durante os últimos 50 anos vem ocorrendo uma leve tendência ao aumento na máxima de precipitação. Também pode se observar que no ano de 2004 ocorreu uma medição acusando a maior quantidade de precipitação durante todo o período estudado, na estação de Campo Mourão com um registro de 295,8 mm em apenas um dia. No outro extremo pode se observar que a estação de Paranaguá registrou como precipitação máxima diária 81,0 mm ocorrido no ano de 2001, a mais baixa de todo o período.

Outro dado relevante poderia ser o número de ocorrências das máximas por estação; neste caso Londrina conta com o maior número, aparecendo em total 13 vezes durante o período estudado.

5.5 CONSIDERAÇÕES FINAIS

A traves do estudo de caso foi verificado que é possível obter informações importantes utilizando o Hadoop. Neste caso, trata-se de dados (*datasets*) reais, porém de tamanho relativamente pequeno, sendo que o arquivo de maior tamanho, utilizado em um dos exemplos, tem 11,7 MB de tamanho com pouco mais de 255 mil linhas distribuídas em 10 colunas.

No cenário deste estudo de caso, rodando em apenas uma máquina convencional, o Hadoop consegue processar e gerar uma saída em questão de poucos segundos (em média de 5 a 8 segundos).

Entretanto, tendo montado o projeto num ambiente real de produção (ambiente multiclusterizado) o Hadoop consegue trabalhar com *DataSets* milhares de vezes mais volumosos, com dados de diversa origem e com milhões de linhas a serem tratadas e processadas, reduzindo notadamente o tempo de processo graças a escalabilidade, tolerância a falhos e versatilidade que o framework possui.

6 CONCLUSÃO

Através deste trabalho foi realizado um estudo teórico sobre o framework Apache Hadoop, como também a implementação de um estudo de caso demonstrativo para mineração de dados. A aquisição do referencial teórico e o estudo das ferramentas mais representativas tem apresentado resultados que poderiam oferecer uma perspectiva de maior alcance visando um aprofundamento nos conhecimentos relacionados com esta tecnologia.

Hadoop permite a implementação de projetos de desenvolvimento e pesquisa em ambiente multiclusterizado (*hardware* de prateleira), utilizando equipamento convencional, o que na prática significa redução de custos de hardware, atendendo ao mesmo tempo à questões tais como: escalabilidade, variedade no hardware empregado, e versatilidade com relação ao volume e às características dos dados trabalhados. A través da utilização do framework Hadoop é possível obter informação de valor a partir de grandes volumes de dados, de maneira relativamente simples empregando um tempo relativamente curto. Entretanto, desde a perspectiva da escalabilidade obtém-se o diferencial mais importante que caracteriza tipicamente esta tecnologia.

6.1 TRABALHOS FUTUROS

Levando em consideração a importância adquirida recentemente por esta tecnologia na área de TI, marcada por uma evolução constante através da implementação de projetos de grande expressividade por parte das companhias mais proeminentes do setor; considera-se interessante uma possível continuidade deste trabalho por meio de pesquisas acadêmicas de maior alcance e o desenvolvimento de novos projetos conduzidos para outras áreas de interesse e que consigam gerar informações de utilidade.

A modo de exemplo, poderia se indicar a implementação na prática de um ambiente de produção Hadoop em máquinas de prateleira, para o tratamento de dados da rede acadêmica institucional, que poderia incluir os dados de todo o câmpus nas áreas mais importantes, inclusive no setor administrativo.

Estes dados poderiam revelar informações de valor que permitam um melhoramento na evolução das atividades relacionadas com a Instituição.

7 REFERÊNCIAS

APACHE HADOOP. **Wiki Apache Hadoop**. Disponível em: <<http://wiki.apache.org/hadoop>>. Acesso em: 27 de janeiro de 2016.

APACHE HADOOP. **Documentation**. Disponível em: <<https://hadoop.apache.org/doc.html>>. Acesso em 29 de janeiro de 2016.

APACHE HADOOP. **HDFS, Architecture Guide**. Disponível em: <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>. Acesso em 30 de janeiro de 2016.

APACHE HADOOP. **Who We Are**. Disponível em: <<https://hadoop.apache.org/who.html>>. Acesso em 31 de janeiro de 2016.

Buyya, R. **High-Performance Cluster Computing: Architectures and Systems**. (1st. ed., Vol. 1). Australia: Prentice Hall, 1999

Dean, J., Ghemawat, S., MapReduce: **Simplified Data Processing on Large Clusters**. Magazine Communications of the ACM, 107 - 113. 2008

DeWitt D, Stonebraker, M. **MapReduce: a major step backwards**. Disponível em: <https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html>. Acesso em 01 de fevereiro de 2016

GOOGLE TRENDS. Data Mining, Statistics and Big Data. Disponível em: <<http://www.google.com/trends/explore?hl=pt-BR#q=Data+Mining,+Estat%C3%ADstica,+Big+Data&cmpt=q>>. Acesso em 02 de fevereiro de 2016

Hilbert, M., & López, P. (2011). **The World's Technological Capacity to Store, Communicate, and Compute Information**. Science, 332(6025), 60–65. <<http://science.sciencemag.org/content/332/6025/60.full>>. Acesso em 15 de setembro de 2015.

Hurwitz, Judith et al. **Big Data For Dummies**. Wiley Brand: New Jersey, USA, 2013.

IBM. **What's Big Data.** Disponível em: <<http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>>. Acesso em 28 de janeiro de 2016.

INSTITUTO NACIONAL DE METEOROLOGIA – INMET. **BDMEP - Banco de Dados Meteorológicos para Ensino e Pesquisa.** Disponível em: <<http://www.inmet.gov.br/>>. Acesso em 30 de janeiro de 2016.

Provost, F., Fawcet, T. **Data Science For Business.** Sebastopol - CA, USA: O'Reilly Media Inc., 2013

Tanenbaum, A., Van Steer, M. **Distributed Systems: Principles and Paradigms,** 2nd. Ed., Pearson, 2006

White, Tom. **Hadoop, The Definitive Guide,** 3rd. Edition. Sebastopol - CA, USA: O'Reilly Media Inc, 2012.

YAHOO. **Yahoo Tutorial - YDN. Yahoo Developer Network.** Disponível em <<https://developer.yahoo.com/hadoop/tutorial/>>. Acesso em 03 de fevereiro de 2016.

Zikopoulos, Paul C. et al. **Harness The Power Of Big Data** (1st Ed., Vol. 1). New York: McGraw-Hill, 2013

Zummel, N., Mount, J. **Practical Data Science With R.** New York: Manning, 2014.