UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GUILHERME SANTIAGO RIBEIRO SILVA
LINCOLN MORO URBAN

IMPLEMENTAÇÃO DO PARADIGMA MAPREDUCE POR MEIO DO HADOOP INTEGRADO AO FRAMEWORK HIVE: UM ESTUDO PRÁTICO.

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2016

GUILHERME SANTIAGO RIBEIRO SILVA
LINCOLN MORO URBAN

IMPLEMENTAÇÃO DO PARADIGMA MAPREDUCE POR MEIO DO HADOOP INTEGRADO AO FRAMEWORK HIVE: UM ESTUDO PRÁTICO.

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Tarcizio Alexandre Bini

PONTA GROSSA
2016

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DO PARADIGMA MAPREDUCE POR MEIO DO HADOOP INTEGRADO AO FRAMEWORK HIVE: UM ESTUDO PRÁTICO

por

GUILHERME SANTIAGO RIBEIRO SILVA
LINCOLN MORO URBAN

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 24 de Maio de 2016 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Tarcizio Alexandre Bini
Prof. Orientador

_____
Simone de Almeida
Membro titular

_____
Richard Duarte Ribeiro
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Dedico este trabalho à minha família, pois mesmo na distância, se fizeram sempre presentes.
Guilherme

-

Dedico este trabalho à todas as pessoas próximas a mim e que de alguma forma, contribuiram com suas palavras de fé, luz e prosperidade.
Lincoln

AGRADECIMENTOS

Agradeço única e exclusivamente a Deus, primeiro por ter me concedido uma família extraordinária, esta que sempre me deu apoio incondicional ao longo desta caminhada. Por me conceder meus amigos e colegas, sem citar nomes para, eventualmente, não ser injusto com algum, por todo apoio e ensinamento que me concederam.

Agradeço a Deus por me conceder bons professores ao longo de toda minha vida acadêmica e consecutivamente pelo conhecimento oferecido por eles à mim. Um agradecimento especial ao meu orientador Tarcizio Alexandre Bini, que foi ponto de referência e lapidador.

Agradeço também a Deus por colocar em meu caminho todos os que duvidaram, sem sombra de dúvidas, estes também foram cruciais para que eu relutasse sempre que me deparava com a vontade de desistir ou com os rumores do fracasso. Por fim, agradeço a Deus por todos que, de alguma forma, contribuiram para que eu pudesse alcançar mais este objetivo e pudesse chegar onde estou.

Guilherme Santiago

Agradeço primeiramente a Deus, por me conceder saúde, sabedoria, luz, paz e tranquilidade no decorrer do desenvolvimento deste trabalho. Em meus momentos de reflexão, foi meu porto seguro, me amparando na angústia e me dando forças para seguir em frente, sua luz me ilumina e sua presença em minha vida é fundamental.

Agradeço a cada professor que colaborou com seu conhecimento, tempo e dedicação, para que ao longo da trajetória acadêmica, eu pudesse crescer e aprimorar ainda mais todo conteúdo visto em sala de aula. Um agradecimento especial ao orientador Professor Tarcizio Alexandre Bini, por todo amparo, dedicação e atenção a cada detalhe no decorrer do trabalho, agradeço sua paciência e muitas vezes a falta dela, juntamente com a cobrança, a qual foi motivo de inspiração e coragem para seguir em frente.

Por fim, agradeço pela família incrível que Deus me deu, cada um contribuiu da sua maneira e ajudou no que foi preciso para que eu pudesse chegar onde estou agora. Agradeço meus colegas de faculdade e amigos, todos que de alguma forma estiveram presentes, compartilhando conhecimento, auxiliando com uma palavra amiga e muitas vezes batalhando para superar os mesmos obstáculos. E à minha namorada Juliana, meus mais sinceros e profundos agradecimentos, por cada gesto de amor e carinho, pela atenção, pelas palavras de conforto e pela presença constante nos momentos mais importantes da minha vida, sempre me desejando o melhor, orando por mim onde quer que eu esteja.

Lincoln Moro Urban

RESUMO

SILVA, G. S. R.; URBAN, L. M. **Título do trabalho**: *Implementação do paradigma MapReduce por meio do Hadoop integrado ao framework Hive: Um estudo prático.* 2016. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal Do Paraná. Ponta Grossa, 2016.

Atualmente, com o avanço da tecnologia e a criação constante de novas aplicações, muitas empresas deparam-se com uma questão crucial para o segmento dos serviços de TI (Tecnologia da Informação), como por exemplo o armazenamento e manipulação de grandes volumes de dados. Empresas como o Facebook, Twitter, Google, entre outras, têm suas tecnologias e inovações pautadas sobre um novo conceito chamado Big Data. Essa nova tendência possibilitou o desenvolvimento de soluções que atendem a demanda do mercado, visto que os Sistemas Gerenciadores de Banco de Dados Relacionais, mesmo que ainda muito utilizados, deparam-se com problemas de desempenho, escalabilidade e processamento de bases de dados volumosas. Um dos conceitos mais utilizados atualmente, quando trata-se de Big Data, é o paradigma MapReduce. Este foi desenvolvido pela Google e tem seu funcionamento baseado no processamento e distribuição de dados em um conjunto de computadores (*cluster*), interligados por uma rede, possibilitando assim, uma maior flexibilidade na manipulação desses dados. Considerando o MapReduce um paradigma, algumas tecnologias foram criadas para implementar os seus conceitos, uma delas é o Hadoop, o qual possui módulos que realizam o gerenciamento e a distribuição de bases de dados entre diversas máquinas. O presente trabalho propõe a implementação e execução prática do paradigma MapReduce por intermédio do Hadoop em um ambiente virtualizado. Para tanto, fêz-se uso de um cenário experimental composto de tecnologias de virtualização e técnicas de *benchmark*, as quais simulam cargas de trabalho analíticas sobre bases de dados sintéticas. Os resultados por sua vez, apontam para a análise no tempo de execução das consultas submetidas a este cenário e também servem como base para trabalhos futuros e pesquisas relacionadas.

Palavras-chaves: Big Data. MapReduce. Hadoop. Virtualização. Benchmark.

ABSTRACT

SILVA, G. S. R.; URBAN, L. M. **Title of the working**: *Implementation of the MapReduce paradigm through Hadoop integrated into the framework Hive: A practical study.* 2016. Trabalho de Conclusão de Curso (Technology Analysis and Systems Development) - Federal Technology University - Paraná. Ponta Grossa, 2016.

Nowadays, with the advancement of technology and the constant creation of new applications, many companies are faced with a crucial issue for the segment of the IT (Information Technology) services as a storage and handling of large volumes of data. Companies like Facebook, Twitter, Google, among others, has its technologies and innovations guided per a new concept called Big Data. This new tendency allow the development of solutions that can meet the market demand, considering that the Relational Database Management Systems, although still widely used, encounter problems with regard to performance, scalability and processing of large databases. One of the most widely used concepts nowadays, when mention the Big Data, is the MapReduce paradigm. This was developed by Google and has its operation based on the processing and distribution of data in a set of computers (cluster), interconnected over a network, thus enabling greater flexibility in handling such data. Considering the MapReduce paradigm, some technologies were created to implement their concepts, one of them is Hadoop, which has modules that perform the management and distribution of databases between multiple machines. This paper proposes the implementation and practical implementation of the MapReduce paradigm through the Hadoop in a virtualized environment. Therefore was used an experimental environment, compound per virtualization technologies and benchmark techniques, which simulate analytical workloads on synthetic databases. The results in turn, point to the analysis in the time of execution of the queries submitted to this environment and also serve as a base for future work and related searches.

Key-words: Big Data. MapReduce. Hadoop. Virtualization. Benchmark.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

AaaS	*Analytics-as-a-Service*
BaaS	*Backup-as-a-Service*
CPU	*Central Processing Unit*
DaaS	*Database-as-a-Service*
ETL	Extract, Transform and Load
GB	*Gigabytes*
GFS	*Google File System*
GHz	Gigahertz
HDFS	*Hadoop Distributed File System*
HMR	*Hadoop MapReduce*
HQL	*Hive Query Language*
IaaS	*Infrastructure-as-a-Service*
IC	Intervalo de Confiança
IDC	*International Data Corporation*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
LTS	*Long Term Support*
MHz	*Megahertz*
ODBC	*Open Database Connectivity*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
PaaS	*Plataform-as-a-Service*
RAM	*Random Access Memory*
RPM	Rotações por Minuto
SaaS	*Software-as-a-Service*
SF	*Scale Factor*
SGBDRs	Sistemas Gerenciadores de Banco de Dados Relacionais
SPEC	*System Performance Evaluation Cooperative*

SQL *Structure Query Language*

TB *Terabytes*

TPC *Transaction Processing Performance Council*

TPS *Transações por segundo*

UDFs *User-Defined Functions*

VM *Virtual Machine*

VMM *Virtual Machine Monitor*

WAN *Wide Area Network*

YARN *Yet Another Resource Negotiator*

ZB *Zettabytes*

SUMÁRIO

1 INTRODUÇÃO

O aumento no uso de dispositivos móveis e a crescente difusão da Internet (redes banda larga, 3G, 4G) tem contribuído para a geração de dados digitais. Em um levantamento realizado pela IDC (*International Data Corporation* - Corporação Internacional de Dados), corporação que estuda e realiza previsões acerca de tecnologias, foi estimado que até 2020 o universo digital chegará a 40ZB (*zettabytes*) de dados. Este crescimento representa uma previsão que excede em 50 vezes as realizadas para o ano de 2010 (GANTZ; REINSEL, 2012).

A constante geração de informação faz com que seja necessário extrair conhecimento em massas de dados cada vez maiores. Com isso, tanto o armazenamento quanto o acesso aos dados torna-se muitas vezes dispendioso. Algumas tecnologias foram e têm sido criadas para trabalhar com as bases que possuam tamanho acima do que se tinha à alguns anos atrás, uma delas é o *Big Data* (IBM; ZIKOPOULOS; EATON, 2011).

Em paralelo com os estudos que visam encontrar soluções que suportem o crescente aumento no tamanho das bases de dados, outra área que tem ganho atenção ocupa-se com o acesso aos dados armazenados. Tem-se buscado criar opções para, de alguma forma, tornar a extração de informações das bases de grandes volumes mais rápida e fácil. Como exemplo destas soluções, pode-se citar tecnologias de otimização de consultas, implementações do algoritmo de MapReduce, a computação em Nuvem, os *clusters* de computadores, dentre outras (CHAUDHURI; NARASAYYA, 2002) (DEAN; GHEMAWAT, 2008).

Dean e Ghemawat (2008), definem a arquitetura MapReduce como um modelo de implementação de processamento dos registros para grandes conjuntos de dados, que trabalha com duas funções principais: *Map* e *Reduce*. A função *Map* processa os dados de forma a mapear os mesmos em conjuntos chave/valor com o objetivo de gerar um conjunto de pares de chave/valor intermediário. A função *Reduce* recebe o resultado da função *Map* e agrupa as chaves geradas de acordo com um valor frequente. Tudo isso com a finalidade de reduzir o processamento referente a uma rotina de recuperação de uma informação específica.

Um *framework* que implementa o MapReduce é o Hadoop (TM, 2015a). Além de utilizar a arquitetura MapReduce, o mesmo possui um sistema de arquivos próprio, o *Hadoop Distributed File System* - Sistema de Arquivos distribuidos Hadoop (HDFS). Estas duas características operam em conjunto, uma vez que o Hadoop possui um nó responsável pelas requisições dos usuários intitulado de *NameNode* e outros nós nomeados *DataNodes*, que são encarregados de armazenar os dados e executar o processamento dos mesmos.

A estrutura implementada pelo Hadoop possui um nó responsável por distribuir todas as requisições para os outros nós. Tal modelo arquitetural é conhecido como mestre e escravo. O nó mestre (*NameNode*) envia instruções aos nós escravos (*DataNodes*) e estes por sua vez executam as instruções recebidas. A fim de garantir o funcionamento e a integridade do sistema, existe uma comunicação entre os nós para que em eventuais falhas, o sistema possa contornar

tal indisponibilidade não atribuindo tarefas de processamento, de leitura e/ou escrita para aquele nó (SHVACHKO *et al.*, 2010).

É possível inserir, recuperar os dados no HDFS e trabalhar com os mesmos através de rotinas nativas do *framework*, seja inserção ou recuperação dos dados. Porém, essa rotina é vista como onerosa. A partir dessa dificuldade de se escrever instruções de manipulação dos dados utilizando o Hadoop, o Facebook optou por criar uma solução que integra-se com o *framework* e facilita a execução de algumas instruções sobre o mesmo. Esta solução é o Hive (TM, 2015d).

De acordo com Thusoo *et al.* (2009), o Hive suporta consultas em uma linguagem declarativa baseada na Linguagem de Consulta Estruturada (*Structured Query Language* - SQL). Ao serem executadas, estas consultas são compiladas nos *jobs* (serviços) do MapReduce, onde é feita a ”tradução” das instruções do Hive para instruções que o Hadoop possa interpretar. A partir do momento que essa conversão é finalizada, as instruções pertinentes às declarações escritas são executadas.

Não basta apenas possuir ferramentas para manipulação de dados e informações. É preciso ter acesso a dados e também instruções que possam ser submetidas à estes dados para então se observar o desempenho do cenário que eles contemplam. Existem vários ambientes e casos para se avaliar o comportamento de uma determinada rotina. Pode-se avaliar diferentes tipos de *hardware* para um mesmo propósito, diferentes tecnologias/arquiteturas e cada caso exige um tipo de métrica diferente.

Há cenários em que o tempo de resposta é a principal medida que se visa observar, outros, a confiabilidade é prioritária. Existem casos que avaliam a disponibilidade de aplicações quando submetidas a rotinas com diversas cargas de dados. Para isso foram criadas as técnicas de *benchmark*, que possuem como objetivo padronizar as avaliações em diferentes casos de testes.

A técnica de *benchmark* caracteriza-se pelo objetivo de mensurar o desempenho de sistemas computacionais por meio de uma metodologia sistêmica e padronizada dos ambientes, onde as avaliações são realizadas (BERRY; LARSON, 1991). Existem empresas que possuem o foco em definições de padrões para diversas áreas, *hardwares*, redes, *softwares*, bancos de dados e muitas outras. Uma das organizações que trabalha com análise de desempenho é o consórcio TPC (*Transaction Processing Performance Council* - Conselho de desempenho de processamento de transações) (TPC, 2015).

O consórcio TPC é uma organização sem fins lucrativos que desenvolve métricas e padrões para *benchmarks* voltados, inclusive, para bases de dados em suas diversas ramificações. Como exemplo, têm-se as bases de dados relacionais e bancos com propósito de serem implantados exclusivamente para negócios. Sendo assim, o presente trabalho fará uso de um dos *benchmarks* do TPC (TPC-H) com o intuito de observar o comportamento do cenário criado.

Uma vez que o propósito deste estudo não é modelar um segmento mercadológico e sim observar o desempenho alcançado pelo cenário proposto, tem-se a possibilidade de limitar as opções de avaliação de desempenho. Outro fator que contribui para o afunilamento das opções

é o fato do Hadoop não ser utilizado em transações com alto nível de inserção e atualização de dados. Esta característica é uma premissa básica das operações do tipo OLAP (*Online Analytical Processing*). Sendo assim, limita-se as possibilidades de escolha ao *benchmark* TPC-H.

Entretanto, como o Hive possui particularidades em suas consultas, algumas delas precisaram ser reescritas para serem executadas pelo TPC-H. Como exemplo destas diferenças apresenta-se o fato do Hive não trabalhar com o uso de seleção de múltiplas relações na construção de uma consulta para seleção de dados (mais detalhes são apresentados na subseção 2.7.2). O conjunto de reescritas das consultas do TPC-H geraram uma nova versão do *benchmark*. Esta nova versão é tratada por alguns como Hive 600 (JIA, 2009a).

1.1 OBJETIVOS

O propósito geral deste trabalho é criar um ambiente virtualizado que implemente o paradigma do MapReduce através do Hadoop. Além disso, busca-se-á a integração do mesmo com o *framework* Hive com o intuito de obter informações referentes à execução das consultas submetidas aos dados persistidos em uma base de dados.

1.1.1 Objetivo Geral

De forma geral, o objetido do presente trabalho consiste na implementação prática do paradigma MapReduce, bem como um estudo analítico acerca dos resultados alcançados por essa implementação. Para tal, será utilizado o *framework* Hadoop, executado sobre um ambiente virtualizado. Também objetiva-se legitimar o ambiente proposto através do uso de técnicas de *benchmark* ajustadas para o cenário de processamento e armazenamento distribuído.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho consistem em:

- Criar um cenário que possibilite implementar o paradigma MapReduce através do Hadoop;
- Integrar o *framework* Hive ao Hadoop e assim possibilitar, através de linguagens declarativas, criar consultas a serem submetidas aos dados armazenados no HDFS;
- Submeter as consultas do *benchmark* à uma base sintética gerada pelo DBGen e obter os dados dos resultados das execuções.

1.2 JUSTIFICATIVA

A difusão e a facilidade do acesso à Internet implicam no aumento da quantidade de dados gerados. Muitas empresas oferecem serviços como o Office 365[1], Google Docs[2], Facebook[3], YouTube[4] entre tantos outros que são utilizados constantemente por muitos usuários e por isso, possuem bases de dados que aumentam de tamanho a cada nova interação com o usuário.

Com o crescimento das bases de dados, outras opções aos bancos de dados relacionais foram desenvolvidas e/ou ganharam mais espaço. As opções foram criadas com a finalidade de diminuir o tempo de resposta das requisições. Alguns exemplos de alternativas foram as bases de dados distribuídas e os bancos NoSQL (SADALAGE; FOWLER, 2013).

As soluções que propõem novas arquiteturas para as bases de dados têm como objetivo otimizar a recuperação dos mesmos, uma vez que esta rotina de recuperação torna-se mais onerosa a medida que as bases tendem a serem mais volumosas. Isso se dá pelo aumento no número de aplicações que os usuários utilizam e também pelo aumento das interações por parte do usuário com as mesmas.

Sendo assim, ter acesso à dados de forma rápida é algo imprescindível, uma vez que, ao pensar na perspectiva de um usuário que acessa um *site* em busca de informações, o tempo de resposta às suas necessidades irá influenciar sua decisão de continuar a utilizar o sistema ou não. Com isso, as empresas são forçadas a oferecer aplicações que implementem sistemas eficientes e que os dados que o mesmo usa, possam ser obtidos com o menor tempo possível.

Portanto, o propósito deste trabalho é realizar um estudo de caso através de um cenário que implemente o paradigma MapReduce por meio do Hadoop. As rotinas que compreendem a manipulação de dados serão compostas pela submissão de consultas através do *framework* Hive para recuperação de dados. Isso se dá com a finalidade de se observar como o cenário se comporta através dos resultados que serão obtidos por meio da execução do *benchmark* TPC-H de forma adaptada para o Hive. A adaptação se dá, basicamente, pela reescrita das consultas.

[1] https://outlook.office365.com/
[2] https://apps.google.com/docs
[3] https://facebook.com
[4] https://youtube.com

2 ARMAZENAMENTO E PROCESSAMENTO DE DADOS DISTRIBUÍDOS

Esta seção trata do embasamento teórico utilizado para o desenvolvimento do trabalho em questão. É apresentada a estrutura das tecnologias que, juntas, visam diminuir o tempo de resposta de acesso aos dados de uma base sintética persistida no HDSF, bem como seus conceitos e implementações. Sendo assim, faz-se necessário contextualizar o estado da arte das tecnlogias de geração e processamento de dados distribuídos.

Recentemente Zadrozny e Kodali (2013) realizaram um levantamento sobre os dados gerados na *Web*, em redes sociais, dispositivos móveis e por sensores presentes em diversos ambientes como plantações, rios, e outros. Como resultado, notou-se um aumento considerável no tamanho das bases que persistem estas informações. Dentre os números levantados têm-se:

- **Facebook** - O Facebook atualmente possui uma base com mais de 1 (um) bilhão de usuários sendo que desse número, 618 milhões são ativos e compartilham cerca de 2.5 bilhões de *posts* por dia, gerando outros 2.7 bilhões de curtidas. Estes números correspondem à geração de aproximadamente 500 *terabytes* de informação ao final de 1 (um) dia. Ao se atentar à arquivos de imagens, são enviadas uma média de 300 milhões de novas imagens por dia e para armazenar tais fotos são necessários 7 (sete) *petabytes* em disco;
- **Twitter** - Possui cerca de 500 milhões de usuários, e esse número cresce em uma progressão diária de 150 mil novos usuários. A cada dia, estima-se que 200 milhões de usuários geram 500 milhões de *tweets*;
- **LinkedIn** - Possui mais de 200 milhões de membros e a cada segundo 2 (dois) novos usuários cadastram-se no sistema. Isso, em 2012, gerou cerca de 5.7 bilhões de buscas aos perfis dos profissionais cadastrados;
- **Instagram** - Os usuários do Instagram enviam 40 milhões de fotos todos os dias, curtem 8500 fotos a cada segundo e geram 1000 comentários no mesmo intervalo de 1 (um) segundo.

2.1 BIG DATA

Segundo Matturdi *et al.* (2014), o termo Big Data deve ser usado para descrever qualquer grande quantidade de dados, sejam eles estruturados ou não estruturados. Entretanto, o termo não possui nenhuma relação com qualquer quantidade de dados e tampouco se resume a somente essa característica. Os dados desta tecnologia apresentam uma ampla variedade e usualmente os bancos de dados relacionais não suportam trabalhar com tais dados, devido a sua forma semi-estruturada ou desestruturada.

Garlasu *et al.* (2013) pondera que os dados são processados de forma diferente e com uma variação que depende da análise que será aplicada sobre os dados iniciais. As análises realizadas devem utilizar tecnologias modernas que permitem manipular de forma rápida e eficiente esses grandes aglomerados de dados. Como exemplo das tecnologias que possibilitam a implementação do Big Data tem-se o MapReduce através do Hadoop, utilização da Computação em Grade (*Grid*), Computação em Nuvem e Computação em Aglomerados (*clusters*).

O Big Data não é uma tecnologia independente, ela consiste da integração de tecnologias para o propósito de manipular grandes volumes de dados díspares, ou seja, não estruturados, com um tempo de resposta adequado, a fim de garantir uma análise em tempo real. Alguns autores tipicamente ressaltam que o Big Data possui 3 (três) características fundamentais (Figura 1), a saber (HURWITZ *et al.*, 2013) (MCAFEE; BRYNJOLFSSON, 2012) (MATTURDI *et al.*, 2014) (KATAL; WAZID; GOUDAR, 2013):

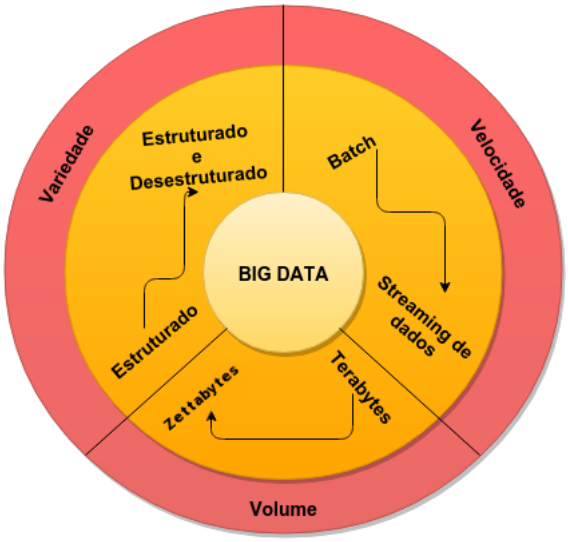

Figura 1 – Características do Big Data
Fonte: Adaptação de (IBM; ZIKOPOULOS; EATON, 2011)

- **Volume** - Esta característica leva em conta a quantidade de dados gerados a todo instante. As redes sociais produzem *terabytes*[1] de dados todos os dias. As aplicações que trabalhavam com essa quantidade, hoje manipulam bases na casa de *zettabytes*[2]. Essa quantidade de dados é definitivamente difícil de se tratar através dos sistemas tradicionais, como os bancos de dados relacionais;

- **Variedade** - Os dados persistidos não são da mesma categoria. As bases podem ser constituídas por dados relacionais e dados semiestruturados (dados que não possuem um modelo estrutural formal tal como dados persistidos em relações). Estes dados são provenientes de diversas fontes como páginas *Web*, *e-mails*, documentos, sensores, dispositivos móveis entre outros.

[1] 1 Terabyte (TB) = 10^{12} bytes
[2] 1 Zettabytes (ZB) = 10^{21} bytes

- **Velocidade** - É uma característica essencial não apenas para a entrada dos dados na base, mas também para recuperação dos mesmos, processamento e saídas provenientes da execução das análises sobre os dados. Por manipular bases maiores, essa caracterísica é um grande desafio.

Alguns autores propõem outras características para o Big Data. Para Grolinger *et al.* (2014), além das citadas como *Variedade*, *Volume* e *Velocidade*, tem-se também a *Variabilidade* como propriedade importante. Já para Katal, Wazid e Goudar (2013) deve-se considerar, além das 3 (três) características citadas, a *Variabilidade*, *Complexidade* e o *Valor*. Demchenko, Laat e Membrey (2014) consideram 5 V's: *Variedade*, *Velocidade*, *Volume*, *Valor* e *Veracidade*.

A *Variabilidade* considera as inconsistências do dados. A *Complexidade* está relacionada com o grande esforço para vincular, corresponder, preparar e transformar dados, visto que eles são provenientes de várias fontes. E por fim o *Valor*, que a partir de uma análise sobre os dados, encontra-se valores e tendências de negócio para melhorar as estratégias de uma empresa (HURWITZ *et al.*, 2013).

O Big Data não é uma tecnologia única. Ela pode ser implementada de várias maneiras e com várias outras tecnologias como por exemplo, a computação em Nuvem, Computação em Grade (*grid*) (GARLASU *et al.*, 2013), Computação em Aglomerados (*clusters*), implementações do MapReduce (PANDEY; TOKEKAR, 2014) (GROLINGER *et al.*, 2014), Hadoop (MATTURDI *et al.*, 2014), entre outras. A seguir tem-se uma breve descrição sobre cada uma destas tecnologias.

2.2 COMPUTAÇÃO EM GRADE

A Computação em Grade (*Grid Computing*) é bastante abrangente e pode ser compreendida como uma infraestrutura de *software* que conecta e faz a gestão de recursos computacionais distribuídos de forma a deixar transparente ao usuário o acesso a tais recursos. Os recursos vão desde armazenamento, *softwares*, processamento ou mesmo acesso à periféricos. Em uma das primeiras definições de Foster e Kesselman (1999), o termo Computação em Grade é visto como uma infraestrutura distribuída que permite a gestão e acesso remoto a recursos computacionais.

Os sistemas de uma grade não precisam estar alocados no mesmo espaço físico e tampouco precisam estar permanentemente inter-conectados. Apesar de fazerem parte de um mesmo ambiente, os recursos trabalham de forma independente uns dos outros e ainda assim, são sistemas capazes de realizar processamento em paralelo. Buyya *et al.* (2002) listam algumas características sobre as grades, a saber:

- **Heterogeneidade** -Uma grade é constituída por um ambiente heterogêneo com várias tecnologias distintas, como servidores, dipositivos móveis, *hardwares* de laboratórios, sensores etc;

- **Escalabilidade** - Uma grade pode trabalhar com alguns poucos ou milhões de recursos e isso deve acontecer sem perda de desempenho. Vale ressaltar que pelo fato dos recursos estarem dispersos geograficamente, problemas como latência devem ser considerados no projeto da grade;
- **Compartilhamento de Recursos** - Os recursos de uma grade são compartilhados entre os nós e não podem ser alocados para uma aplicação específica;
- **Domínios Administrativos Variados** - Com a possibilidade de implementar vários ambientes heterogêneos em uma grade, cada ambiente deve possuir seu domínio administrativo com regras e restrições próprias;
- **Controle Distribuído** - Uma grade pode funcionar se um nó ficar indisponível em algum momento, isso porque, não existe um nó central que realiza a gestão total do sistema. Então cada nó da grade deve ser autogerido;
- **Dinamicidade e Adaptabilidade** - As aplicações e os gerenciadores de uma grade devem adaptar-se à indisponibilidade de um recurso mudando seu comportamento sem interferir nos processos ativos.

Os sistemas/ambientes de uma grade são distribuídos em categorias como mostra a Figura 2. A categoria de **Grade Computacional** engloba recursos diversos que são integrados com a finalidade de prover uma capacidade de processamento agregado. O sistema ainda pode ser subdividido em sistemas de *Super Computação Distribuída* e de *Alto Desempenho*. A supercomputação é usada para distribuir o processamento de uma aplicação a fim de reduzir o tempo de resposta. Já a grade de alto desempenho é utilizada para aumentar a taxa de conclusão de um fluxo de trabalho em aplicações com cálculos complexos (KRAUTER; BUYYA; MAHESWARAN, 2002).

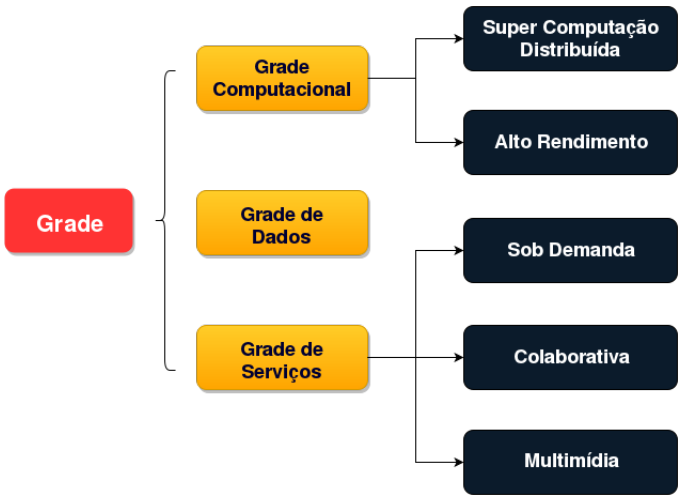

Figura 2 – Taxonomia dos sistemas de uma grade
Fonte: Adaptação de (KRAUTER; BUYYA; MAHESWARAN, 2002)

A categoria de **Grade de Serviços** abrange sistemas que executam aplicações que não podem ser executadas em uma única máquina. Ela se subdivide em *Serviços Sob Demanda*, *Colaborativos* e *Multimídia*. Os serviços sob demanda agregam, dinamicamente, recursos de acordo com a necessidade dos mesmos. Já os serviços colaborativos conectam usuários e aplicações em grupos para trabalhos colaborativos. E por fim, os serviços multimídia fornecem uma infraestrutura para aplicações de multimídia em tempo real (KRAUTER; BUYYA; MAHESWARAN, 2002).

A categoria **Grade de Dados** considera sistemas que fornecem uma infraestrutura de acesso e classificação de dados em bibliotecas digitais ou *data warehouses* em uma WAN (*Wide Area Network* - rede de área ampla). Como exemplo de serviços de uma Grade de Dados temos aplicações para mineração de dados, estas que manipulam várias fontes de dados diferentes em sua execução (KRAUTER; BUYYA; MAHESWARAN, 2002).

O modelo arquitetural de uma grade é extensível e aberto a várias soluções. As camadas da Figura 3 agrupam os componentes que possuem características comuns. A arquitetura apresentada segue os princípios de um modelo conhecido por Ampulheta (*hourglass model*). A ”garganta” ou gargalo define um conjunto de abstrações e protocolos de conectividade que facilitam a partilha de recursos. O topo define comportamentos de alto nível e a base apresenta tecnologias distintas mapeadas (FOSTER; KESSELMAN; TUECKE, 2001). A seguir é apresentada uma descrição sucinta de cada uma das camadas da Figura 3:

Figura 3 – Modelo de Arquitetura da Computação em Grade
Fonte: Adaptação de (FOSTER; KESSELMAN; TUECKE, 2001)

- **Aplicação** - A camada de aplicação diz respeito à interface que o usuário interage com a grade. É nela que estão todas as ferramentas para gestão da grade;
- **Coletividade** - A segunda camada refere-se aos *middlewares*. São eles que possibilitam a integração e comunicação de sistemas heterogêneos através de protocolos e serviços;
- **Recursos** - A camada de recursos provê os protocolos de compartilhamento, gerenciamento e de controle de recursos locais;
- **Conectividade** - Define de forma física a autenticação e comunicação dos recursos da grade;

- **Ambiente** - Contempla os recursos locais disponíveis no nó, bem como o monitoramento dos mesmos.

Com a computação em grade um usuário pode utilizar um recurso em outra estação de trabalho (em outra localidade) e armazenar os resultados da utilização do recurso em um terceiro ambiente distinto. Em um cenário colaborativo, um outro usuário poderia usar os resultados armazenado pelo primeiro para dar continuidade ao serviço iniciado.

Algumas aplicações trabalham com a capacidade ociosa das máquinas da grade para diminuir o tempo necessário para resposta à uma instrução. Isso só é possível pela arquitetura da grade, que possibilita essa heterogeneidade e esse grau de comunicação.

2.3 COMPUTAÇÃO EM NUVEM

O termo computação em Nuvem foi utilizado pela primeira vez em 2006 em uma conferência do Google, na ocasião Eric Schmidt apresentava a forma de operação e gerenciamento dos *data centers* da empresa (GILDER, 2006). Porém, foi depois de 2008 que o conceito começou a ser mais difundido. Para Gilder (2006) a arquitetura empregada na computação em nuvem mantém os dados residentes em servidores e as aplicações que manipula tais dados são executadas tanto em um servidor nuvem, quanto no navegador do usuário que utiliza a aplicação.

> *"Nesta arquitetura, os dados são em sua maioria, residentes em servidores 'em algum lugar na Internet' e o aplicativo é executado tanto nos 'servidores cloud' quanto no navegador do usuário.*" Eric Schmidt em *"The Information Factories"*[3]

Apesar do termo ter sido cunhado em 2006, foi em 2011 que Mell e Grance (2011) publicaram a definição final de computação em nuvem. Segundo eles, a computação em nuvem tende a ser um modelo computacional que acessa uma rede de recursos quando necessário, sob demanda, e de qualquer localização. Tais recursos podem ser redes, servidores, armazenamento, serviços entre outros. Outra característica deste modelo é que, necessariamente, a disponibilidade dos recursos mencionados deve ser realizada de maneira rápida e com o mínimo de esforço por parte do cliente.

Vaquero *et al.* (2009) também definem de forma clara o que para ele é Computação em Nuvem. Eles caracterizam este paradigma como um *pool* (série) de recursos virtualizados tais como plataformas de *hardware*, de desenvolvimento e serviços facilmente acessados, reconfigurados e ajustados conforme a necessidade de escala. A forma como os recursos são oferecidos usualmente é através de um modelo *pay-per-use*, isto é, o cliente pagará apenas pelo recurso utilizado.

[3] Tradução livre

Existem vários modelos de serviços na computação em nuvem, alguns são mais difundidos tais como os serviços IaaS, PaaS e SaaS. A Figura 4 exemplifica algumas das variações dos modelos arquiteturais dos serviços oferecidos pelas nuvens. Através dela é possível ver que mesmo que em níveis diferentes de atuação, ainda assim as variações podem trabalhar em conjunto. Cada nível traz os recursos que pode-se explorar. A seguir é apresentada uma breve descrição de cada modelo citado (ARMBRUST *et al.*, 2010) (CHIEU *et al.*, 2009b) (RIMAL; CHOI; LUMB, 2009) (VAQUERO *et al.*, 2009):

Figura 4 – Infraestrutura da pilha *aaS
Fonte: Adaptação de (BOJANOVA; SAMBA, 2011)

- ***Infraestrutura como serviço*** *(IaaS - Infrastructure-as-a-Service)* - Os provedores de infraestrutura, através da virtualização, dividem, atribuem e redimensionam dinamicamente os recursos a fim de construir sistemas *ad-hoc* voltados para a necessidade de cada cliente.

- ***Plataforma como serviço*** *(PaaS - Plataform-as-a-Service)* - A tecnologia de computação em nuvem oferece plataformas de *software* onde os sistemas dos clientes serão executados. Os recursos necessários para a utilização dos sistemas ficam transparentes ao cliente. Um exemplo conhecido é o serviço da Google denominado Google AppEngine[4].

- ***Software como serviço*** *(SaaS - Software-as-a-Service)* - O modelo de SaaS consiste em uma alternativa para as aplicações que necessitam de execução local. Os serviços são oferecidos através da Internet e podem ser acessados de qualquer lugar e qualquer dispositivo,

[4] www.developers.google.com/appengine

independente de sua arquitetura. Como exemplo, tem-se os aplicativos de escritório *Google Docs*[5].

Além destes 3 (três) níveis de serviço citados, existem outros como o *Database-as-a-Service (DaaS)* que é um banco de dados por demanda, em que a capacidade do banco vai variar de acordo com a necessidade de cada usuário. Tem-se também o serviço de *Backup-as-a-Service (BaaS)*, que oferece ao usuário a opção de armazenar seus dados com segurança e ter acesso aos mesmos em qualquer momento e lugar.

Um serviço que trabalha com análises de dados em nuvem é o *Analytics-as-a-Service (AaaS)* (LOMOTEY; DETERS, 2014). O serviço utiliza a disponibilidade de uma nuvem combinada com ferramentas como o Hadoop, para gerar análises sobre os dados conforme a necessidade do usuário. É um serviço semelhante ao *SaaS* porém, com foco em análises.

A computação em nuvem é um modelo arquitetural que vai além do processamento e armazenamento de dados. É um paradigma que oferece soluções para diversas áreas mas que precisa de planejamento a partir do momento em que se decide pela sua utilização. Faz-se necessário a elaboração de um cenário que permita a implementação de tal tecnologia a fim de alcançar bons resultados.

Existe certa confusão entre a computação em nuvem e *clusters*, ou computação em *grid*. Entretanto, pelas características apresentadas, pode-se notar que o alcance da computação em nuvem vai além do compartilhamento de processamento e/ou armazenamento. Uma implementação de uma nuvem pode ser composta por um *cluster* ou um *grid* computacional. Porém vale ressaltar que apesar das tecnologias possuirem traços semelhantes, cada uma possui suas particularidades e aplicações. As principais características de uma nuvem são:

- **Agrupamento (*pooling*) de recursos** - Recursos físicos e virtuais são agrupados para atender simultaneamente às requisições dos clientes;
- **Amplo acesso à rede** - Assim que um dispositivo conecta-se à rede, os recursos devem estar disponíveis imediatamente e prontos para serem usados;
- **Autoatendimento sob demanda** - As necessidades de um cliente podem variar, sendo assim, um cliente deve conseguir variar os recursos em uso quando desejar e sem interferência humana;
- **Elasticidade rápida** - A medida que uma alteração seja feita nos recuros, os mesmos devem se tornar disponíveis novamente com o mínimo de tempo de espera;
- **Medição do serviço** - Como dito, o cliente deve ser taxado de acordo com os recursos que utilizar.

[5] www.docs.google.com

Como descrito anteriormente, a computação em nuvem possui vários modelos arquiteturais que atendem diversos propósitos. É válido dizer que em uma mesma malha computacional possa existir uma ou mais nuvens que por sua vez, podem executar diversos serviços distintos. Esta multi disponibilidade é possível pois um único *hardware* pode oferecer várias pseudo instâncias computacionais através de virtualização. A seguir é apresentado uma contextualização sucinta sobre a virtualização e suas abordagens.

2.3.1 Virtualização

Conforme apresentado na Seção 2.1, os serviços de Big Data podem ser implementados em mais de uma arquitetura (*Clusters*, Computação em Nuvem, etc), e as arquiteturas apresentadas possuem algo em comum: a possibilidade de serem implementadas em ambientes virtualizados. Isso se dá pela estratégia de se economizar recursos ociosos de um sistema e também como alternativa para pequenas empresas aumentar seu poder computacional sem o investimento de um alto valor monetário.

Chieu *et al.* (2009a) definem a virtualização como a criação de um ambiente computacional simulado chamado Máquina Virtual (VM - *Virtual Machine*). Essa virtualização é feita através de um *software* que controla o *hardware* subjacente possibilitando a interação entre o *hardware* físico com a VM. O *software* de controle é conhecido como *hypervisor* ou Monitor de Máquina Virtual (VMM - *Virtual Machine Monitor*).

Já segundo Bini (2014), a virtualização pode ser compreendida como uma simulação de estações de trabalho/servidores através de uma camada de *software* sobreposta à camada de *hardware*. Com isso tem-se a possibilidade de uma única máquina partilhar seus recursos com uma ou mais VMs.

Um ambiente virtualizado é consituído por 3 (três) conceitos básicos (MATTOS, 2010) (FIGUEIREDO; DINDA; FORTES, 2005). O primeiro conceito representa os recursos físicos de *hardware* e *software* e é denominado como sistema hospedeiro (*host system*). O segundo baseia-se em um sistema virtual (*guest system*) que pode ser representado por vários sistemas virtuais distintos sendo executados sobre o ambiente real.

Outro conceito primordial é o conceito de VMM. É a camada de *software* que abstrai os recursos físicos para que as VMs possam utilizá-los. Também é o componente de *software* que hospeda as VMs. Outra de suas característcas trata da função escalonar o acesso à recursos físicos entre VMs (ROSE, 2004).

Algumas abordagens acerca da virtualização foram propostas e através disso surgiram implementações distintas de se virtualizar sistemas e criar Máquinas Virtuais (VMs *Virtual Machines*). A seguir são apresentados algumas das abordagens mais discutidas (BINI, 2014) (ROSE, 2004):

- **Virtualização Total** - A virtualização total provisiona VMs idênticas ao sistema hospedeiro. Desta forma o sistema operacional e as aplicações contidas nestas VMs são executadas com a alusão de estarem utilizando diretamente os recursos físicos de *hardware*.

- **Paravirtualização** - A paravirtualização é uma otimização da *Virtualização Total*. Neste tipo, o sistema operacional convidado tem acesso direto ao *hardware* físico porém, quando é necessário realizar instruções sensíveis tal como gestão de memória e interrupções, as requisições são direcionadas ao VMM e ele então as processa.

- **Virtualização em Nível de Sistema Operacional (*Operating System Level Virtualization*)** - Este tipo provê uma virtualização a nível de sistema operacional que resulta na possibilidade de um mesmo hospedeiro executar uma ou mais VMs de forma isolada e segura.

- **Virtualização Assistida por *Hardware* - *Hardware-Assisted Virtualization*** - O tipo de virtualização por *Hardware* é delineado por extensões nos processadores (processadores específicos) que permitem a execução de VMs em seu sistema operacional sem a necessidade de adequar o SO e tampouco traduzir as instruções da VM sobre o *hardware* físico.

A virtualização tem como uma das principais características, ser uma tecnologia que pode ser oferecida como serviço sobre demanda. Com isso, pode-se aumentar ou diminuir a quantidade de recursos a medida que eles são necessários. Este cenário retrata algumas das características dos serviços das nuvens como exemplo, a rápida elasticidade. No entanto, uma nuvem pode ser composta por instâncias físicas também. A forma como é implantada varia do investimento destinado à nuvem.

Ao se criar uma malha computacional física dá-se margem para algumas pessoas confundirem o serviço de nuvem com a computação em aglomerados. Entretanto, apesar das características em comum, são arquiteturas distintas. Na Seção 2.4 é apresentado uma definição da computação em aglomerados com a finalidade de dar base teórica para se compreender a distinção de cada um dos tipos computacionais já citados.

2.4 COMPUTAÇÃO EM AGLOMERADOS (*CLUSTERS*)

Para Baker e Buyya (1999) a computação em aglomerados, de forma bem sucinta, nada mais é que um conjunto de computadores interligados através de uma mesma tecnologia de rede trabalhando em uma mesma tarefa. Ku, Min e Choi (2010) classificam como uma técnica computacional para superar a limitação de um sistema unitário e autônomo. Já Sharma, Kumar e Gupta (2009) tratam a computação em aglomerados como a combinação de múltiplos recursos computacionais em uma configuração paralela.

Com a finalidade de se compreender a computação em aglomerados é necessário entender o que é um aglomerado/*cluster*. De certa forma, a definição converge com a definição da tecnologia em si. Um *cluster* é um conjunto de computadores com baixo acoplamento (dependência uns dos outros) e que trabalham de forma colaborativa, deixando transparente para quem usa a aplicação, que se trata apenas de um computador (SHARMA; KUMAR; GUPTA, 2009). Eles podem ser classificados em três formas (KAUSHIK, 2008):

- **Aglomerados para Alta Disponibilidade** - Como o próprio nome diz, estes aglomerados possuem o objetivo de deixar um sistema sempre disponível. Ele trabalha com o sistema de redundância de nós (computadores). No caso de um nó falhar, a indisponibilidade é contornada através do outro nó. Sendo assim, se o serviço para, logo é reestabelecido através do outro computador redundante.

- **Aglomerados para Balanceamento de carga** - Este tipo de agrupamento distribui as requisições de entrada entre os vários nós. Cada nó executa os mesmos programas ou processam o mesmo conteúdo. Com isso, eles conseguem tratar solicitações para o mesmo conteúdo ou aplicativo. No caso da indisponibilidade de um nó, as requisições são redistribuídas entre os demais nós disponíveis.

- **Aglomerados para Alto Desempenho** - Esta forma de aglomerado faz uso de vários nós de forma simultânea. Esta simultaneidade se dá com a finalidade de solucionar uma tarefa computacional complexa e que demanda recursos muito superiores se fosse executada em apenas um computador. Neste caso, cada nó executa uma sub-tarefa que compõe um todo.

A arquitetura básica de um aglomerado é mostrada na Figura 5. Os principais componentes de um *cluster* incluem, múltiplos computadores individuais (PCs, *workstations* - estações de trabalho), um sistema operacional, interfaces de rede, uma conexão de alto desempenho, *softwares* de comunicação, um *middleware* e aplicações sequenciais e/ou paralelas.

As principais características da computação em aglomerados são (SHARMA; KUMAR; GUPTA, 2009):

- **Alto desempenho** - Ao integrar os vários nós do aglomerado, se aumenta o poder computacional e assim pode-se resolver problemas que apenas super computadores outrora conseguiriam;

- **Alta disponibilidade** - Provê a disponibilidade dos serviços mesmo quando um computador falha;

- **Alta confiabilidade** - Esta característica está muito relacionada com a disponibilidade. Uma vez que o sistema tem suporte para estar sempre disponível, pode-se confiar na disponibilidade quando esta for necessária;

- **Alta escalabilidade** - Facilidade de adicionar ou remover algum nó da rede;

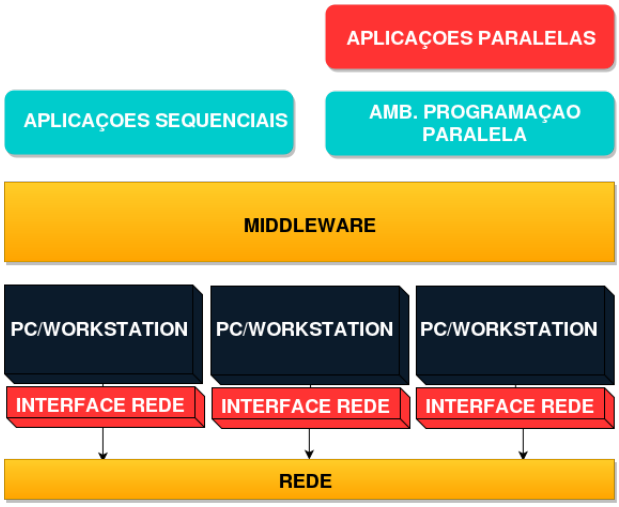

Figura 5 – Arquitetura de um *cluster*
Fonte: Adaptação de (BUYYA, 1999)

- **Simplificação** - Facilita a partilha de recursos dos nós sem que os usuários dos sistemas ou os gestores precisem se envolver de forma direta.

Os aglomerados foram criados para atender propósitos desafiadores através de uma forma onde, ao se agrupar componentes menos robustos como resultado obtem-se características que, se fossem encontradas em um único dispositivo, este possuiria um custo muito superior a todos os componentes.

Como exemplo de aplicações suportadas por aglomerados, pode-se citar aplicações metereológicas, simulações nucleares, processamento de imagem, mineração de dados, astrofísica dentre outras. Nestes tipos de aplicações são exigidas tarefas de cálculos e processamento intensivo. Já no cenário de alta disponibilidade tem-se como exemplos do uso de aglomerados as aplicações comerciais derivadas do setor bancário e de *backup* (SADASHIV; KUMAR, 2011).

Ku, Min e Choi (2010) dizem que a computação em aglomerados foi a base para várias tecnologias, como por exemplo a computação em nuvem, o HDFS e o GFS (*Google File System* - Sistema de arquivos Google). No caso do HDFS e do GFS isso se dá porque ambos trabalham como uma tecnologia de armazenamento virtualizado baseado em aglomerados, os quais são acessados pelo MapReduce. O MapReduce é plataforma computacional destinada a processamento e que será tratada na próxima Seção deste trabalho.

2.5 MAPREDUCE

O conceito de escalabilidade de aplicações e a necessidade de processar e armazenar grandes volumes de dados têm acarretado desafios para as tecnologias que não estavam adequadas a essa realidade. Os novos cenários têm saturado os sistemas pela quantidade de informações

que os mesmos passaram a manipular. As aplicações não conseguem atender a demanda de requisições dos serviços criando-se um gargalo que faz com que o tempo de resposta às mesmas aumente.

Esta realidade serviu de motivação para a criação e emprego de arquiteturas alternativas que visam reduzir o tempo de espera das informações obtidas no processamento dos dados. Como exemplo dessas soluções têm-se as tecnologias em nuvem como os serviços AaaS, implementações do algoritmo conhecido por MapReduce, utilização de *clusters*, bases de dados distribuídas, dentre outras.

O MapReduce é uma arquitetura e uma implementação para o processamento e armazenamento de grandes volumes de dados de forma distribuída. Foi apresentada pela Google no OSDI'04: *Sixth Symposium on Operating System Design and Implementation*, em dezembro de 2004 (DEAN; GHEMAWAT, 2004). É uma solução que apresenta como característica de funcionamento, possuir como entrada, dados que são mapeados em chave/valor, organizar os dados por chaves que sejam iguais e com isso reduzí-los a conjuntos mapeados.

De acordo com a própria Google, o MapReduce é um modelo computacional capaz de trabalhar de forma paralela com os dados sem que eles estejam armazenados de forma centralizada. Essa arquitetura visa a organização e o gerenciamento das diversas máquinas de um *cluster* e tem a capacidade de distribuir o processamento sobre os dados.

Dean e Ghemawat (2008) resumem em 5 (cinco) as fases de uma implementação do MapReduce. A Figura 6, representa a estrutura de funcionamento da arquitetura MapReduce e suas fases:

1. Iteração sobre entradas;
2. Cálculo dos pares chave-valor para cada parte da entrada;
3. Agrupamento de todos os valores intermediários pelas chaves;
4. Iteração sobre os grupos resultantes;
5. Redução de cada grupo.

O MapReduce apresenta o fluxo de execução paralela dos processos de forma transparente quando aplicada à ambientes com vários *clusters*. O próprio sistema responsável pela execução dos processos trata dos detalhes do particionamento dos dados de entrada, do agendamento das execuções dos programas, das falhas que a máquina apresentar e gerencia a comunicação entre os computadores que operam em conjunto (DEAN; GHEMAWAT, 2010).

Para Dean e Ghemawat (2010) as funções de *Map* e *Reduce* possuem uma baixa complexidade de entendimento e muitas vezes podem ser representadas por equivalentes na linguagem SQL simples. Entretanto, essa representação nem sempre pode ser feita, em certos casos este processo se torna complicado e/ou sem resultado.

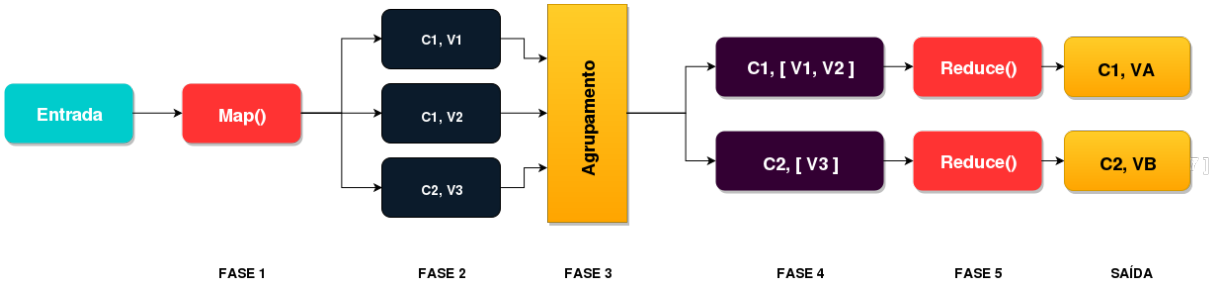

Figura 6 – Estrutura de funcionamento do MapReduce
Fonte: Adaptação de (KAWA; KREWSKI, 2014)

Uma implementação do algoritmo de MapReduce, conforme exemplificado pelo Algoritmo 1, segue algumas características como a existência das funções ***Map*** e ***Reduce***. A função Map() cria um intermediário para cada ”palavra” de uma entrada juntamente com uma contagem associada de ocorrências (valor fixo de ’1’ apenas para fins demonstrativos). Já a função Reduce() agrupa todas as ocorrências criadas para um determinado valor à chave que ela corresponde recebendo como entrada as saídas geradas na função Map() (DEAN; GHEMAWAT, 2008).

Algoritmo 1 Map e Reduce - Adaptação de (DEAN; GHEMAWAT, 2008)

```
1:
2:  //CHAVE(Nome do documento)
3:  //VALOR(Conteúdo do documento)
4:  function MAP(String chave, String Valor)
5:      for each palavra p do in  valor
6:          criarIntermediario(p, "1")
7:
8:  //CHAVE(Uma palavra)
9:  //VALORES(Uma lista de contadores)
10: function REDUCE(String chave, Iterator valores)
11:     resultado = 0
12:     for each v do in  valores
13:         result+ = ParseInt(v)
14:         criar(AsString(result))
```

Como visto, o nome MapReduce revela de forma prática sua lógica de funcionamento, por meio de suas duas funções principais: *Map* e *Reduce* (SADALAGE; FOWLER, 2013). A fim de entender o fluxo principal do seu funcionamento, define-se um exemplo didático, que parte do princípio em que tem-se uma estrutura de dados com várias informações referentes a experimentos agrícolas. As informações foram obtidas mediante um sensor alocado em uma plantação que em determinado intervalo de tempo, registra a temperatura do dia.

Com esses dados persistidos, em uma base fragmentada em diversas máquinas, deseja-se encontrar o ano em que a temperatura alcançou seu pico máximo. Os dados persistidos não necessariamente estão estruturados e a base já armazena temperaturas por 15 anos com registros

a cada hora. A partir do objetivo definido, é necessário submeter os dados às funções *Map()* e *Reduce()* com a finalidade de encontrar o valor requisitado.

A primeira etapa do processo é a função *Map()* (Mapeamento), a qual recebe como entrada uma estrutura (também chamada de agregado). A partir dela, são gerados alguns pares do tipo: chave-valor. No exemplo ilustrado na Figura 7, tem-se a entrada de várias linhas de um registro e assim que os dados são processados pela função, tem-se como saída pares chave-valor de um ano e valor de uma temperatura registrada.

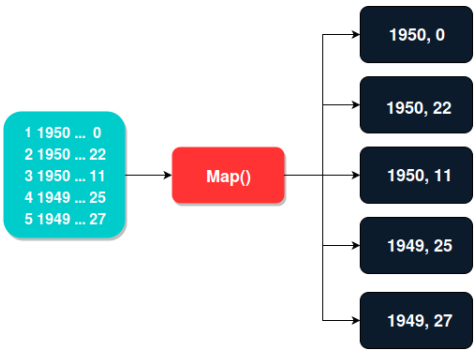

Figura 7 – Exemplo da função Map
Fonte: Autoria própria

Depois de realizada a função *Map()*, todos os pares chave-valor retornados são agrupados e passados como entrada à função *Reduce()* (Figura 8). A função *Reduce()* por sua vez agrupa os valores de uma mesma chave e gera uma única saída. A diferença entre as duas funções é que a função *Map()* depende de apenas um registro para execução, ao passo que a função *Reduce()* pode receber múltiplos resultados. O MapReduce define muito bem cada tarefa de mapeamento e as direciona para os nós corretos de modo que, todos os dados sejam passados para a função *Reduce()* (SADALAGE; FOWLER, 2013)

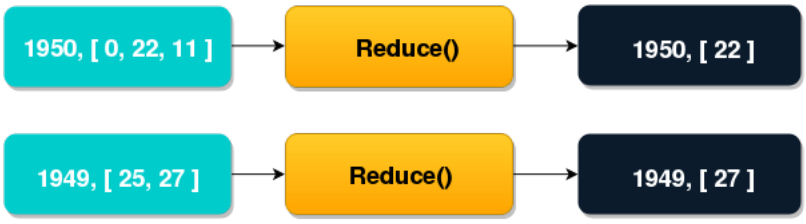

Figura 8 – Exemplo da função Reduce
Fonte: Autoria própria

Após os dados serem processados pela função *Reduce()* tem-se como saída a máxima temperatura de cada ano. A partir disso é possível consultar de maneira mais simples qual a temperatura máxima de modo geral (no exemplo está relacionada ao ano de 1949 e obteve-se 27°) ou por ano. A Figura 9 mostra de forma integral a execução das 5 (cinco) fases do MapReduce aplicadas ao exemplo criado.

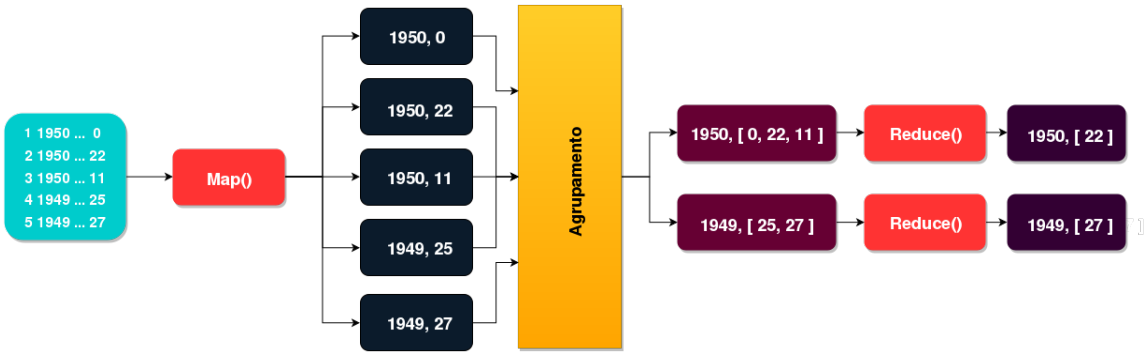

Figura 9 – Exemplo completo da execução do MapReduce
Fonte: Autoria própria

O Hadoop é uma implementação do MapReduce que trabalha com um ou vários nós em um *cluster* que divide tanto os dados quanto o processamento sobre os mesmos. A seguir é apresentado com mais detalhes as características desta tecnologia, com a finalidade de oferecer embasamento teórico e contextualizá-la frente aos objetivos deste trabalho.

2.6 HADOOP

Devido a grande quantidade de dados gerados com a difusão da Internet, dos dispositivos móveis, dos aplicativos que operam nestas plataformas, bem como os dados gerados por grandes corporações diariamente, problemas como armazenamento, indexação e gerencimento dos mesmos são cada vez mais comuns. O aumento do tamanho das bases dificulta o acesso aos registros pois, mais dados resulta em mais possibilidades de se encontrar um valor.

Em função deste problema no acesso aos dados, as empresas são forçadas a encontrar soluções que minimizem o tempo de resposta no acesso aos registros processados pelas aplicações fornecidas pelas mesmas. A partir deste propósito a Yahoo desenvolveu uma implementação do MapReduce e deu-lhe o nome de Hadoop.

O Hadoop é uma plataforma computacional distribuída desenvolvida em Java, voltada para *clusters* e como já mensionado, criada com a finalidade de processar grandes volumes de dados. Depois de sua criação e consolidação, vários setores começaram a adotá-la, como exemplo, pode-se citar: os setores de finanças, mídia e entretenimento, governo, saúde, serviços de informação, varejo e outras indústrias com requisitos de Big Data, além de empresas como Facebook e Twitter.

Desde o momento de sua criação, o Hadoop tem ganho mercado (tenha visto as empresas que optaram por trabalhar com esta plataforma) e em Janeiro de 2008, o Hadoop passou a ser um projeto de alto nível mantido pela comunidade Apache (WHITE, 2009). A seguir são apresentadas algumas características do Hadoop segundo (TECHNOLOGIES, 2015):

- **Organização Simplificada** - Pelo fato da não obrigatoriedade de estruturação dos dados tampouco a adoção de esquemas, a organização dos dados fica transparente no Hadoop.
- **Tolerância à falhas** - Se um nó do *cluster* falhar, o processamento é executado em outros nós disponíveis. No caso de acesso aos dados, a redundância dos dados minimiza a indisponibilidade de um nó responsável pelo armazenamento dos dados e execução de processos (*data node*).

Basicamente sua estrutura divide-se em dois sub-sistemas: o *Hadoop Distributed File System* (HDFS) e o *Hadoop MapReduce* (HMR). Porém, além desses principais sub-sistemas, o Hadoop é composto por vários sub-projetos, alguns nativos e outros que podem ser integrados conforme a necessidade do responsável pela Implementação do Hadoop. A Figura 10 apresenta alguns componentes integrados ao Hadoop de forma nativa.

Cada componente nativo pode ser visto como um módulo do Hadoop. Eles possuem seu propósito particular, porém operam de forma integrada e visam alcançar o objetivo geral que é manipular os dados de forma distribuída e eficiente. A partir da versão 2.0 do Hadoop uma nova versão do MapReduce foi implementada, esta nova versão chama-se YARN (*Yet Another Resource Negotiator* - Mais um negociador de Recursos [6]) no *framework*.

Com o aumento da popularidade do Hadoop outros projetos acabaram sendo criados e integrados ao mesmo, seja de forma nativa ou como um complemento, cada um com sua particularidade. A seguir são listados alguns dos componentes e suas respectivas funções (GASPAROTTO, 2014), (TM, 2015), (TM, 2015e), (TM, 2014):

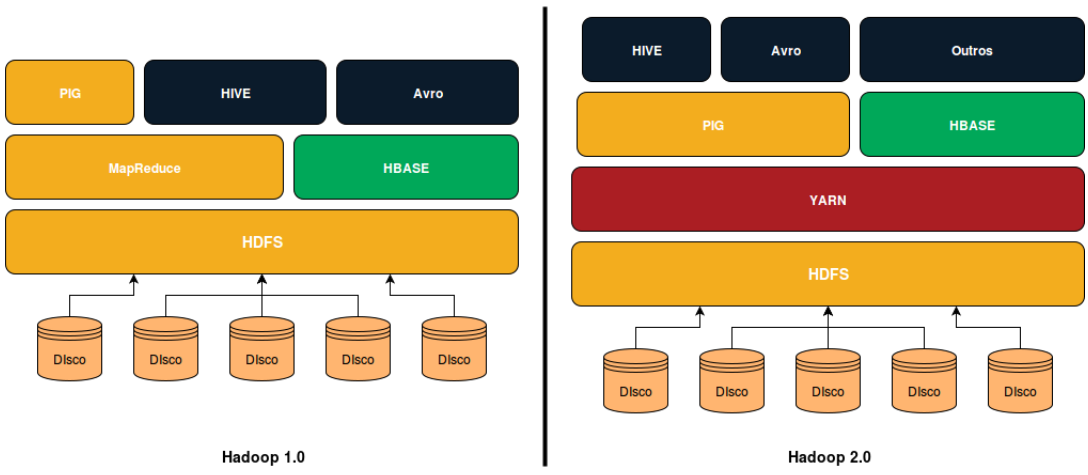

Figura 10 – Organização dos Componentes do Hadoop
Fonte: Adaptação de (MURTHY, 2013) e (CIURANA, 2011)

- **HDFS** - Sistema de Arquivos distribuídos Hadoop;
- **MapReduce** - *Framework* para processamento de informações em grandes bases.

[6] Tradução livre feita pelos autores do trabalho

- **Pig** - Plataforma para análise de grandes conjuntos de dados que consiste em uma linguagem de alto nível para expressar análises de dados em programas;
- **HBase** - Sistema gerenciador de banco de dados orientados à colunas.
- **Avro** - É um *framework* de serialização de dados;
- **Yarn** (Hadoop 2.0) - Versão recente do MapReduce.

O Hadoop possui uma estrutura (*skeleton*) que é executada sob uma arquitetura **Mestre/Escravo** (Figura 11), ou seja, existe um nó que é responsável pela gestão, pelo controle de acesso aos dados e também controle do processamento dos mesmos, este nó é conhecido por *NameNode*. Existem também os *DataNodes* que podem ser um ou mais nós e que são responsáveis por processar e armazenar os dados. A seguir, apresenta-se um detalhamento sobre as características dos nós mestres e nós escravos.

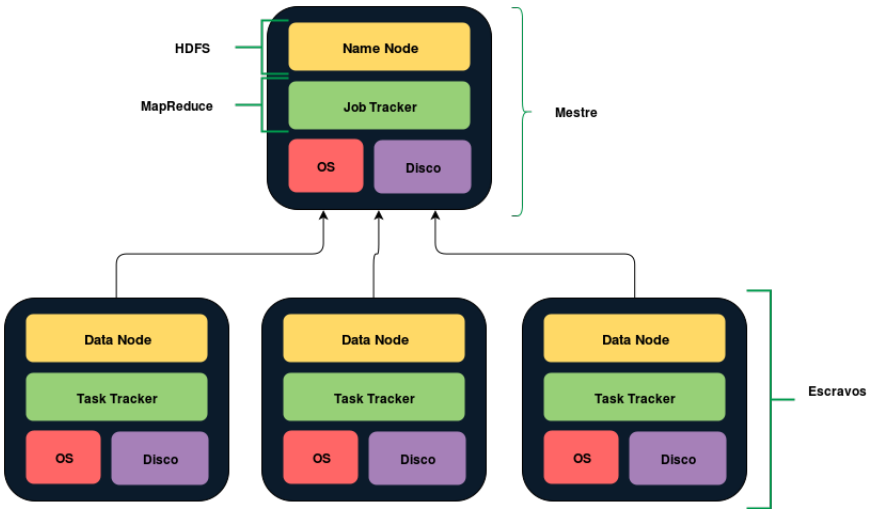

Figura 11 – Topologia organizacional do Hadoop
Fonte: Adaptação de (HEDLUND, 2011)

2.6.1 Nós mestres (*NameNodes*)

Os *NameNodes* são instâncias representadas por *inodes* (nós de índices), ou seja, todas as informações do *NameNode* como blocos de localização, metadados e outras características de sua estrutura básica são armazenadas neste nó. Também armazenam as permissões, horário de acessos e modificações, *namespace* (hierarquia de arquivos e diretórios) e o espaço ocupado em disco pelos mesmos (SHVACHKO *et al.*, 2010).

Além das informações sobre os blocos de dados alocados nos nós do *cluster*, o nó mestre também possui um serviço chamado *Job Tracker*. Este serviço é responsável pela gestão

do processamento sobre os dados. O processamento realizado pelos nós escravos é realizado de forma distribuídas e é o *Job Tracker* que indica qual *DataNode* processará os dados envolvidos em uma transação(VENNER, 2009). O *NameNode* não se comunica diretamente com trocas de mensagens com os nós escravos, ele apenas identifica a disponibilidade dos *DataNodes* e envia para os mesmos instruções como:

- Replicação de blocos para outros blocos;
- Remoção de réplicas de um bloco;
- Ativação ou desativação de um nó.

2.6.2 Nós Escravos (*DataNodes*)

O Hadoop pode trabalhar com uma arquitetura com apenas um nó, porém o *cluster* pode se estender à centenas ou milhares de máquinas. Os nós são constituídos pelos *DataNodes*, que são as máquinas que armazenam e processam de forma distribuída os dados. Quando uma aplicação escreve um bloco em um *DataNode*, de forma transparente ele estará gerando 2 (dois) arquivos. Um com o conteúdo do documento e outro composto pelos metadados referentes ao mesmo (SHVACHKO *et al.*, 2010).

White (2009) diz que um *NameNode* tem informações sobre quais *DataNodes* estão ativos no *cluster* devido a um sistema chamado *Heartbeats* (Batimentos cardíacos). Cada escravo envia periodicamente as mensagens (*Heartbeats*) para o nó mestre a cada 3 (três) segundos (HEDLUND, 2011).

Se o *NameNode* parar de receber as mensagens de um nó escravo após 10 (dez) minutos assume-se que o *DataNode* não está disponível e assim não pode-se designar escrita ou acesso a dados àquele nó. Ainda assim, o nó mestre cria réplicas dos dados daquele nó em outros, para manter o número de réplicas dos blocos presentes naquela máquina (BORTHAKUR, 2007), (SHVACHKO *et al.*, 2010).

Nos nós escravos também estão presentes os serviços de *Task Tracker*. Segundo Venner (2009), este serviço é o responsável por executar as tarefas de *Map* e *Reduce* de forma individual. O *DataNode* recebe as instruções do *Job Tracker* e com isso as executa e retorna o resultado encontrado naquele nó para que, se necessário, as informações sejam integradas com os resultados dos outros nós e assim se forme o resultado esperado em sua totalidade.

2.6.3 *Hadoop Distributed File System* (HDFS)

O HDFS é um sistema de arquivos distribuídos, otimizado para atuar em dados não estruturados. É um sistema que foi desenvolvido tomando como base o GFS (GHEMAWAT HOWARD GOBIOFF, 2003). Seu desenvolvimento tem como base o conceito *write-once*, *read-many-times*, ou seja, escreva uma vez, leia muitas vezes. Esse conceito é tido como essencial para o Hadoop, uma vez que normalmente, em aplicações com grande volume de informações, esses dados serão processados várias vezes.

Outra definição é a de White (2009), que trata o HDFS como um sistema de arquivos desenvolvido com a finalidade de armazenar grandes conjuntos de dados e que possui padrões para acesso aos mesmos. O autor ainda pondera que ao categorizar os conjuntos de dados como grandes implica que o tamanho dos mesmos sejam com centenas de *megabytes, gigabytes ou terabytes*.

Mais que um conceito, o Hadoop em sua implementação nativa não permite a edição dos dados persistidos. A única forma de alterar um arquivo é adicionar dados no final do mesmo. Outro ponto a ser levado em consideração é que o Hadoop é uma tecnologia desenvolvida com a finalidade de manipular grandes arquivos. Se o ambiente que o mesmo for implementado trabalhar apenas com muitos dados, porém dados com tamanhos considerados pequenos (*megabytes* ou menores), o Hadoop pode não ser a melhor escolha (WHITE, 2009).

Comparado a sistemas de arquivos tradicionais, o HDFS têm as mesmas operações básicas e permissões de escrita, leitura e execução. Porém, o diferencial encontra-se em um arquivo especial que nada mais é do que um *log* de edição de dados. Este *log* serve para que haja um controle mais apurado sobre as modificações nos dados (GASPAROTTO, 2014). Com isso é possível saber em que momento o bloco de dados foi criado/alterado bem como informações sobre todas as eventuais modificações ocorridas.

A Figura 12 exemplifica como é realizado uma operação de escrita de um arquivo no HDFS. Este processo se dá pelo seguinte fluxo:

- Primeiramente o cliente (aplicação) solicita ao *NameNode* a alocação de um conjunto de três *DataNodes* para escrever as réplicas de cada blocos. Cada arquivo é dividido, por padrão, em blocos de 128 *megabytes*, porém esse valor é configurável conforme necessário, basta alterar a configuração *dfs.blocksize* no arquivo **hdfs-default.xml** (TM, 2015c);
- Em resposta à solicitação do cliente, o *NameNode* informa em quais *DataNodes* os blocos devem ser escritos;
- O cliente escreve diretamente o conteúdo do bloco em um nó;
- O *DataNode* que recebeu a escrita replica o conteúdo para outros nós de acordo com a configuração de replicação;

- O próximo bloco é então escrito no local designado.

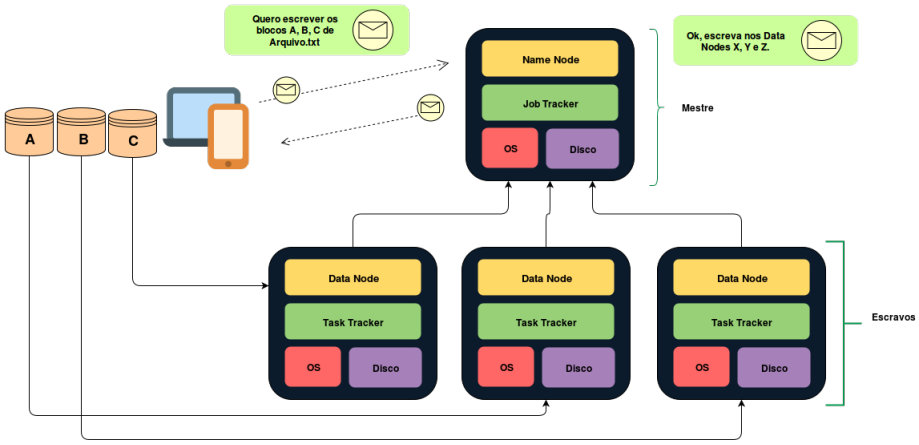

Figura 12 – Escrita de dados no HDFS
Fonte: Adaptação de (HEDLUND, 2011)

Ao se atentar sobre os fatores de replicação no Hadoop, tem-se como valor padrão 3 (três), ou seja, esse fator indica que um bloco é persistido 3 (três) vezes em diferentes nós do *cluster*. No entanto, o fator de replicação pode ser alterado de acordo com a necessidade do cliente. Para isso, é preciso alterar o parâmetro *dfs.replication* no arquivo de configurações do HDFS e com isso serão geradas cópias na escala informada como valor do parâmetro (TM, 2015c).

2.6.4 *Hadoop MapReduce* (HMR)

O HMR é o responsável pelo processamento dos dados, e implementa dessa forma todos os conceitos do MapReduce. O *framework* se destaca pelo fato de não ser realizado nenhuma programação para que os processos sejam executados paralelamente. Pode-se justificar a utilização do Hadoop por ser uma ferramenta: rentável, flexível, tolerante à falhas e escalável. Além de suportar a manipulação de grandes massas de dados distribuídas, como é o objetivo do presente trabalho.

O Hadoop tem seu próprio meio de manipular os dados, tanto para inserção de novos dados quanto para acessar os dados inseridos, bem como instruções para gerir as alterações de permissões dos mesmos. Os comandos de manipulação de arquivos do Hadoop possuem uma relação com os mesmos comandos usados em *Shell Script*. A Tabela 1 apresenta alguns exemplos de instruções através de linhas de comando:

Existem comandos para processar os dados diretamente no Hadoop, porém esses comandos não são muito simples e seu uso torna-se demorado senão complicado, e por isso surgiu o Hive, que visa facilitar o trabalho com os dados persistidos no HDFS. A Seção 2.7 traz uma

Tabela 1 – Rotinas em linha de comando do Hadoop

Operação	Descrição
$ hdfs dfs -ls /caminho/	Listar o conteúdo de um diretório
$ hdfs dfs -mkdir novaPasta	Criar um diretório
$ hdfs dfs -mv arquivo.txt caminho/	Move um arquivo para um determinado diretório
$ hdfs dfs -rm -r pasta/	Exclui um diretório do HDFS
$ hdfs dfs -put arquivo.txt /caminho/	Envia um arquivo para o HDFS
$ hdfs dfs -cat /caminho/arquivo.txt	Lê o conteúdo de um arquivo
$ hdfs dfs -chmod 700 /caminho/arquivo.txt	Altera a permissão de um arquivo
$ hdfs dfs -setrep -w 4 /caminho/arquivo.txt	Atribui o fator de replicação do arquivo para 4
$ hdfs dfs -du -h /caminho/arquivo.txt	Verifica o tamanho do arquivo
$ hdfs hadoop namenode -format	Cria um novo nó

Fonte: Autoria própria

descrição com mais detalhes para que se possa ter uma dimensão do que é o Hive e como ele se comporta.

2.7 HIVE

Cada solução que trabalha com a implementação de um sistema para bases de dados possui suas características como arquitetura, sistema operacional, método de recuperação dos dados, dentre outras. Ao limitar as implementações contidas no trabalho faz-se necessário apresentar as características do Hadoop e MapReduce.

O Hadoop é uma implementação do MapReduce que trabalha com bases de dados distribuídas e de larga escala. Mas além de armazenar os dados é preciso recuperar os mesmos e a partir disso foi desenvolvida uma tecnologia que trabalha sob o Hadoop e que oferece este serviço, esta tecnologia é o Hive (TM, 2015d).

O Hive é uma solução de recuperação de dados de código aberto que suporta consultas no *cluster* do Hadoop em uma linguagem declarativa baseada no SQL. A linguagem do mesmo é o *Hive Query Language* (HiveQL ou HQL) (CAPRIOLO; WAMPLER; RUTHERGLEN, 2012) (TM, 2015d).

Já para os autores Wang *et al.* (2014) e Thusoo *et al.* (2009) o Hive é uma solução para *Warehouse* de código aberto para gerenciar e analisar grandes bases. Ele é uma interface de processamento em lote e ETL (Extração - *Extract*, Transformação - *transform*, Carregamento - *Load*) que tem sido utilizado por grandes empresas como Yahoo, Facebook e Google.

O Hive não foi projetado para cargas de trabalho OLTP (*Online Transaction Processing* - Processamento de Transações Online) isso por que seu propósito é trabalhar com grandes volumes de dados e não a níveis de atualizações de linhas/registros simples e conjunto de dados com latência não significativa entre a emissão de uma consulta e o retorno do resultado da mesma

(TM, 2015b). Isto por que, segundo os autores Capriolo, Wampler e Rutherglen (2012), o Hive é uma tecnologia desenvolvida para aplicações de *data warehouse*, onde o processamento se dá em dados com constância e estaticidade.

Com a utilização do Hive tem-se ganhos em escalabilidade, isso por que caso alguma máquina seja adicionada ao *cluster* onde o Hadoop está implementado, essa mudança fica transparente para o Hive. Outra vantagem é a extensibilidade, uma vez que o Hive trabalha com funções definidas pelos usuários (UDFs - *User-Defined Functions*), o mesmo é ainda tolerante a falhas e possui baixo acoplamento (TM, 2015b).

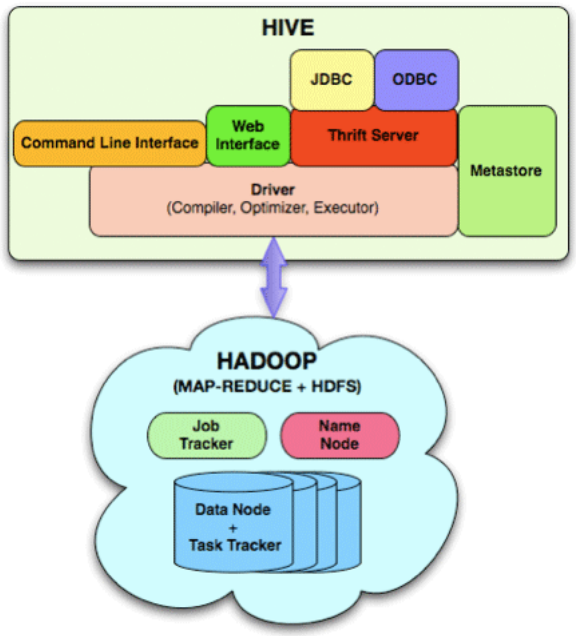

Figura 13 – Arquitetura Hive
Fonte: (THUSOO *et al.*, 2010)

A arquitetura básica do Hive pode ser vista na Figura 13. Os autores Thusoo *et al.* (2010) apresentam esta arquitetura que tem como principais componentes:

- ***Driver*** - Gerencia o ciclo de vida de uma declaração HQL, bem como as sessões de identificação e estatísticas;
- ***Metastore*** - Armazena os catálogos dos sistemas, metadados sobre tabelas, colunas entre outros;
- ***Compiler*** - Componente que compila as declarações HQL através de um grafo acíclico;
- ***Executor*** - Executa as tarefas provenientes do compilador e interage com outras instâncias do Hadoop;
- ***Hive Server*** - Componente que permite a interação e/ou integração do Hive com outras aplicações;

- ***Command Interface*** - Interface web e driver JDBC/ODBC;
- **Interfaces de Extensibilidade** - Permite aos usuários definerem suas próprias funções.

Análogo aos bancos de dados relacionais, os dados no Hive são armazenados em relações/tabelas e estas por sua vez possuem tuplas e cada tupla possui um número de colunas tipadas, isto é, cada coluna contempla dados de um tipo específico (THUSOO *et al.*, 2010). A Tabela 2 mostra os tipos de dados suportados pelo Hive segundo Capriolo, Wampler e Rutherglen (2012):

Tabela 2 – Tipos de dados suportados pelo Hive

Tipo	Descrição
TINYINT	Inteiro de 1 byte
SMALLINT	Inteiro de 2 bytes
INT	Inteiro de 4 bytes
BIGINT	Inteiro de 8 bytes
BOOLEAN	Valor booleano (verdadeiro ou falso)
FLOAT	Ponto flutuante simples
DOUBLE	Ponto flutuante de dupla precisão
STRING	Sequência de caracteres
TIMESTAMP	Tempo (Inteiro, Float ou String)
BINARY	Array de bytes
STRUCT	Análogo às estruturas da Linguagem C
MAP	Coleção de entradas de chave-valor
ARRAY	Sequencia de valores do mesmo tipo e ordenados

Fonte: Autoria própria

2.7.1 HiveQL

Segundo Gruenheid, Omiecinski e Mark (2011) a HiveQL é uma linguagem declarativa baseada no SQL que é utilizada para manipular os dados com o Hive. Com ela é possível fazer uso de comandos como CREATE e DROP em bases, relações internas/externas e partições e utilizar o comando SELECT para recuperar os dados.

Um exemplo de criação de uma relação através da HQL é apresentada a seguir:

```
CREATE TABLE clientes (
key STRING,
nome STRING,
idade INT,
salario FLOAT,
```

```
dependentes ARRAY<STRING>,
documentos MAP<STRING, STRING>,
endereco STRUCT<rua:STRING, cidade:STRING, estado:STRING, cep:INT>)
PARTITIONED BY (estado STRING);
```

O exemplo cria uma tabela chamada **clientes** e a particiona de acordo com o campo estado. Como já dito anteriormente, o Hive não suporta transações de atualização, exclusão e inserção de registros, assim a forma de se inserir dados nas ”relações” é através de operações **LOAD**. Para popular a relação criada anteriormente utiliza-se algo próximo à declaração:

```
LOAD DATA LOCAL INPATH 'caminho'
OVERWRITE INTO TABLE clientes
PARTITION (estado = 'MG');
```

Para recuperar informações de uma relação utiliza-se a operação SELECT. A seguir apresenta-se um exemplo básico de sua utilização. Esta declaração é análoga à operação de seleção das bases de dados relacionais.

```
SELECT cli.nome, cli.idade, ped.key
FROM clientes cli
JOIN pedidos ped ON cli.key = ped.clikey;
```

2.7.2 Hive 600

Ao se propor executar o *benchmark* TPC-H em uma base de dados presente em um cenário onde o mesmo possui integração com o Hive, chega-se no impasse de que o HQL não possui algumas características do SQL como por exemplo, a seleção em múltiplas relações, consultas dependendo de valores de subconsultas, dentre outras diferenças. Com isso, o autor Jia (2009b) propôs a reescrita das 22 consultas do *benchmark* adaptando-as para que possam ser executadas com sucesso neste novo ambiente.

Para exemplificar algumas das diferenças entre SQL e HQL, apresenta-se a seguir a consulta Q3 (*Shipping priority query*) do *benchmark* TPC-H escrita em SQL e sua adaptação para o HQL (JIA, 2009b):

```
SELECT ...
FROM costumer, orders, lineitem
```

```
WHERE
  c_mktsegment = '[SEGMENT]'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '[DATE]'
    and l_shipdate > date '[DATE]'
...
```

Como citado, as consultas HQL não suportam seleção em múltiplas relações sendo assim, a consulta anterior foi reescrita e como resultado obteve-se:

```
SELECT ...
FROM
  costumer c JOIN orders o
  ON
      c.c_mktsegment = '[SEGMENT]'
     and c.c_custkey = o.o_custkey
     and o.o_orderdate < date '[DATE]'
    JOIN lineitem l
  ON
      l.l_orderkey = o.o_orderkey
       and l.l_shipdate > date '[DATE]'
...
```

A reescrita das consultas do TPC-H, que foram escritas originalmente para bancos de dados que suportam nativamente a SQL, permite que sejam executadas consultas em um ambiente Hadoop populado pela base do *benchmark* TPC-H e obtenha o mesmo resultado no caso de bases de dados relacionais. A reescrita trouxe pequenas alterações em 11 consultas, alterações consideráveis em 6 (seis) e em 5 (cinco) consultas foi preciso grandes alterações a fim de se obter resultados equivalentes.

O ambiente Hadoop tem sua própria forma de trabalhar com os dados presentes no mesmo, mas o Hive veio para dar velocidade a este processo. No entanto, para que isso aconteça é preciso além de tudo, de um planejamento de adequação para os cenários a fim de se obter os melhores resultados.

3 CENÁRIO EXPERIMENTAL E RESULTADOS

Este Capítulo apresenta a metodologia empregada no desenvolvimento do cenário experimental proposto pelo trabalho para a submissão das consultas do *benchmark*, assim como os resultados obtidos. Isso inclui, por exemplo, as especificações de *hardware* da máquina onde o cenário foi elaborado, bem como sistema operacional, dentre outros detalhes listados nas subseções a seguir.

Inicialmente faz-se necessário estabelecer padrões para se executar experimentos. Não se pode executar em um cenário particular, pois se necessário comparações futuras torna-se difícil recriar o ambiente. Existem algumas formas para se executar e metrificar experimentos computacionais, como mostra a Seção 3.1. Ela aborda a definição de padrões quando se decide por um estudo exploratório.

3.1 BENCHMARK

A tecnologia é algo que se renova e traz muitas opções para a solução de um mesmo problema. Muitas vezes surge a necessidade de avaliar qual a melhor solução para um determinado cenário. Fundamentado nesta necessidade surgiu a técnica de análise de desempenho. Ciferri (1995) apresenta 3 (três) diferentes modelos de execução de análises, a saber:

- **Modelo analítico -** Este que relaciona medidas de desempenho à parâmetros da ferramenta analisada;
- **Modelo de simulação -** Reproduz as atividades da ferramenta analisada através de um conjunto de hipóteses e condições;
- **Modelo experimental -** Utiliza a própria ferramenta analisada para se obter os resultados de desempenho.

Dentre os modelos acima citados, dar-se-a atenção ao modelo experimental que utiliza a técnica de *benchmark* que traz uma padronização dos testes, estes por sua vez invariáveis, bem definidos e que suportam diferentes ferramentas (BORAL; DEWITT, 1984).

Para Ciferri (1995) a técnica de *benchmark* baseia-se em uma análise experimental representativa com uma malha de testes funcionais e de desempenho que são executados sobre um sistema a fim de mensurar o desempenho do mesmo. Tanto ele quanto Gray (1992) citam 4 (quatro) características básicas da técnica de *benchmark*, a saber:

- **Relevância** - Deve ser uma análise representativa para a aplicação a qual ela trabalha;
- **Portabilidade** - Deve suportar diferentes arquiteturas e sistemas operacionais.

- **Escalabilidade** - Deve suportar a execução desde computadores pessoais até mesmo *main-frames*;
- **Simplicidade** - Precisa ser uma análise simples para que esta não perca sua credibilidade.

Há *benchmarks* disponíveis para diversas áreas, mas a fim de limitar o tema à área da informática existem alguns cenários que Gray (1992) define como os principais, a saber:

- **Comparar diferentes *softwares* em um *hardware*** - Comparar o desempenho de diferentes sistemas ao executar a mesma aplicação;
- **Comparar diferentes *softwares* em uma máquina** - Avaliar o desempenho de dois *software* diferentes ao serem submetidos a um mesmo *hardware*;
- **Comparar máquinas diferentes mas que sejam compatíveis** - Avaliar o desempenho de computadores dentro de uma família;
- **Comparar diferentes versões de um mesmo produto em uma máquina** - Usado para avaliar as melhorias de uma nova versão de um produto.

Dentre os quatro modelos básicos de avaliações citados por Gray (1992), existem alguns consórcios que definiram padrões e métricas para alguns destes modelos. Para padrões de referência de domínios científicos e estações de trabalho, existe o consórcio SPEC (*System Performance Evaluation Cooperative* - Cooperativa de Avaliação de Desempenho do Sistema) (SPEC, 2015).

Além do SPEC, outro consórcio que apresenta um conjunto de *benchmarks* com foco em domínios científicos e que trabalham com ênfase em arquiteturas de computadores é o *Perfect Club* (BERRY *et al.*, 1988). Entretanto, tratando-se de *benchmarks* no cenário de banco de dados, uma referência de destaque é o TPC (TPC, 2015). O TPC é um orgão sem fins lucrativos que define, regulamenta e mantém uma série de *benchmark* voltados para a área de banco de dados, a saber (COUNCIL, 2015a):

- **Processamento de transações OLTP**
  - **TPC-C** - é um *benchmark* complexo para transações OLTP, isto por que suporta vários tipos de transação, banco de dados mais complexo e estrutura de execução globais, além de suportar cinco transações simultâneas de diferentes tipos e complexidade, seja executado on-line ou de forma programada (COUNCIL, 2010).
  - **TPC-E** - é um *benchmark* que possui como foco mensurar o desempenho de bases de dados centralizadas que executem transações relacionadas a contas de clientes em empresas/comércios (COUNCIL, 2015d).

- **Suporte à decisão**
  - **TPC-H** - Consiste em um conjunto de consultas *ad-hoc* (especificadas apenas para essa finalidade) voltadas para negócios. Esse *benchmark* serve como base para sistemas de apoio à decisão que processam grandes volumes de dados, executam consultas com um alto grau de complexidade (COUNCIL, 2014b).
  - **TPC-DS** - Modela e define modelos de vários aspectos de aplicações gerais de um sistema de apoio à decisão, incluindo consultas e dados de manutenção. Trabalha com bases de dados volumosas e suporta trabalhar com ambientes ad-hoc, OLAP, minereação de dados, dentre outros (COUNCIL, 2015c).
  - **TPC-DI** - É um *benchmark* que avalia o desempenho de ferramentas que trabalham com dados entre sistemas. Ele manipula e carrega dados de diversar fontes e os prepara para serem usados em um *data warehouse* (COUNCIL, 2014a).
- **Virtualização**
  - **TPC-VMS** - Este *benchmark* implementa os benchmarks TPC-C, TPC-E, TPC-H e TPC-DS adaptando-os à metodologia e aos os requisitos para a execução e geração de relatórios de métricas de desempenho para bancos de dados virtualizados (COUNCIL, 2013).
- **Big Data**
  - **TPCx-HS** - É um *benchmark* desenvolvido para fornecer informações objetivas acerca de *hardware*, sistema operacional e distribuição do Hadoop que sejam compatíveis e que ofereçam métricas de desempenho verificável e custo-benefício (COUNCIL, 2015b).
- **Características comuns**
  - **TPC-Energy** - Visa mensurar o aumento no consumo de energia nos servidores ao implementarem um benchmark. Suas métricas possuem valores particulares para cada *benchmark* (COUNCIL, 2012).
  - **TPC-Pricing** - *Benchmark* destinado a fornecer uma comparação justa de várias implementações de aplicações que realizam tarefas idênticas, controladas e repetitivas com a finalidade do cliente poder avaliar e optar pela solução que é mais viável para sua realidade (COUNCIL, 2011).

Existem outros benchmarks definidos pelo TPC porém eles, na data deste trabalho, foram classificados pelo consórcio TPC como obsoletos. São eles o TPC-A, TPC-App, TPC-B, TPC-D, TPC-R e o TPC-W.

Como visto, existem diversos tipos de áreas que podem ser avaliadas e estas se ramificam em outras opções com propósitos específicos. Cada tipo de *benchmark* possui suas características e métricas de avaliação que são específicas aos objetivos de cada atividade. Ao se tratar de bases de dados, alguns autores corroboram sobre algumas métricas de desempenho.

Segundo Kohler e Hsu (1990), os *benchmarks* para análise de bases de dados baseiam seus resultados em duas métricas básicas: *Transações por segundo* (TPS), que avalia a taxa de transferência de dados pelas transações submetidas no intervalo de 1 (um) segundo e *Custo por TPS* (KS/TPS). Já para o consórcio TPC, além das duas métricas citadas, é necessário levar em conta a disponibilidade dos dados. Segundo o consórcio é de suma importância mensurar a taxa de disponibilidade de acesso aos dados persistidos na base (COUNCIL, 2015a).

Como o cenário deste trabalho resume-se a ferramentas de armazenamento e obtenção de resultados no acesso aos dados em bases, limitar-se-a a escolha de *benchmarks* destinados a transações de seleção em grandes massas de dados. Entretanto, com esse propósito existem dois tipos de *benchmarks*, para transaçõse OLTP e outro para transações OLAP.

Os *benchmarks* OLPT são destinados a realizar a avaliação de cenários com sistemas que possuem alta taxa de transações de inserção e atualização de registros, como aplicações para o setor financeiro (TYLER; FISHER, 1995). Já os *benchmarks* OLAP avaliam transações que são executadas mas que não fazem alterações no estado da base. Esse tipo de *benchmark* trabalham, em sua essência, com bases que servem apenas para análises e prospecções.

Uma vez que o Hadoop não trabalha manipulação e atualização de dados nas bases tem-se que descartar os benchmarks que trabalham com o tipo de cargas OLTP. Sendo assim, o *benchmark* que trabalha com cargas OLAP e que se encaixa no propósito do trabalho é o TPC-H.

Apresenta-se a seguir uma abordagem mais específicas sobre o *benchmark* TPC-H a fim de entender com mais detalhes suas características e como o mesmo realiza a avaliação das tecnologias avaliadas.

3.1.1 TPC-H

O TPC-H é um *benchmark* que faz uso de cargas de trabalho que simulam um ambiente de tomada de decisão em aplicações de banco de dados. Criado pela organização TPC em 1999 (COUNCIL, 2014b), o TPC-H é constituído por um conjunto de consultas e de um esquema de dados, afim de estabelecer comparativos entre produtos, *softwares*, *hardwares*, etc.

A principal característica do TPC-H é trabalhar com aplicações de grande volume de dados e cargas de trabalho do tipo OLAP, ou seja, são realizadas operações de consulta aos dados armazenados. Por fim, o desempenho de cada solicitação é tomado por base no tempo de resposta.

Considerando a sua estrutura, o TPC-H é composto por 22 consultas, tendo como carac-

terísticas: alto grau de complexidade, natureza *ad-hoc*, vários tipos de acesso aos dados, custo elevado de processamento, dentre outras (COUNCIL, 2014b). Além disso, o *benchmark* emprega um esquema de dados, com tamanho variável, conhecido por SF (*Scale Factor* - Fator de escala), podendo assumir valores de 1 GB (GigaByte) a 100.000 GB (COUNCIL, 2014b). O SF possui relação com o tamanho da base, uma vez que algumas relações possuem tamanho definido através da multiplicação do mesmo pelo número de tuplas que ela dispõe. A Figura 14 mostra a representação do esquema da base de dados do TPC-H.

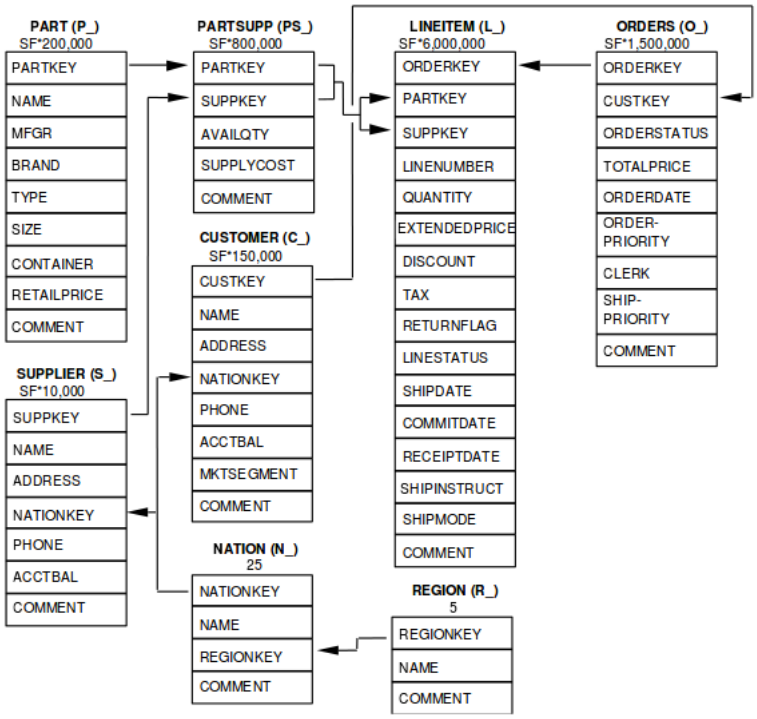

Figura 14 – Representação do Esquema do TPC-H
Fonte: (COUNCIL, 2014b)

O TPC-H também possui um gerador de dados, chamado DBGen (COUNCIL, 2014b). Este gerador tem por objetivo popular a base de dados e sua configuração varia de acordo com o SF utilizado. O DBGen compreende ainda, duas formas de geração dos dados: uma através de arquivos do tipo texto e outra diretamente no banco de dados. Na primeira opção, é possível criar arquivos separadamente para cada uma das tabelas do banco e o tempo gasto para gerar os arquivos não é considerado. Já na segunda opção, o tempo para gerar os dados e carregá-los no banco é contabilizado.

Após o processo de geração dos dados no banco e de posse das 22 consultas do TPC-H, algumas funções para manipulação dos dados podem ser executadas, segundo (POESS; FLOYD, 2000):

- **RF1 (*Refresh Function 1*)** - Função destinada a inserção de dados nas maiores tabelas da base (ORDERS e LEINEITEM). A quantidade de registros inseridos, equivale a 0,1% do

total de tuplas das tabelas.

- **RF2 (*Refresh Function 2*)** - Função que realiza a exclusão de dados nas maiores tabelas. A quantidade de registros excluídos, equivale a 0,1% do total de tuplas das tabelas.

- ***Power*** - Função de teste composta pela execução da função RF1 seguida das 22 consultas mais a função RF2.

- ***Throughput*** - Função de teste definida pela execução de sessões compostas pelas 22 consultas do TPC-H. Cada sessão é chamada de *query-streams* (número de vezes que aquela consulta foi submetida), e junto a ela são executadas as funções RF1 e RF2. A quantidade de sessões a serem efetuadas no banco de dados depende do SF utilizado. A Tabela 3 define o *query-streams* para cada SF (COUNCIL, 2014b):

Tabela 3 – Número de *streams* para cada fator de escala

Fator de Escala (SF)	Número de *Streams*
1	2
10	3
30	4
100	5
300	6
1000	7
3000	8
10000	9
30000	10
100000	11

Fonte: (COUNCIL, 2014b)

- ***Performance*** - Função que baseia-se na execução das funções *Power* e *Throughput* para apresentar a capacidade de processamento de cada consulta.

Para cada execução da função RF1, deve-se obrigatoriamente executar a função RF2. Assim, o banco de dados sempre estará com o mesmo tamanho e a mesma quantia de dados armazenados. Todas essas funções fazem parte do TPC-H e fornecem dados para cálculos das métricas do *benchmark*.

3.2 *HARDWARE* E PLATAFORMA OPERACIONAL

Os experimentos foram executados em 4 (quatro) VMs alocadas em um computador disponibilizado pelo Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná Câmpus Ponta Grossa.

O computador em questão possui um processador Quad Core Intel Core i7-3770k 3.5 GHz x 8 com *clock* mínimo de 1600.019 MHz e máximo de 2082.910 MHz executando sob a arquitetura de 64 *bits*. O mesmo possuía, na ocasião deste trabalho, 8 GB de memória RAM e um disco rígido de 1 TB com 7200 RPM (Rotações por minuto). O sistema operacional utilizado era o Linux Ubuntu 14.04 LTS (*Long Term Support* - Suporte a Longo Prazo) com *kernel* 3.16.0-38-generic X86_64. Para a implementação do ambiente virtualizado, foi empregado o qemu-kvm versão 2.0.0 (QEMU-KVM, 2016), onde cada uma das VMs era gerenciada pelo ambiente gráfico *Virtual Machine Manager* versão 0.9.5 (MANAGER, 2016).

O cenário experimental faz uso da arquitetura mestre/escravo. Para isso foi criado um nó mestre (*NameNode*) e 3 *DataNodes*. O *NameNode* é o nó responsável por gerir os nós com dados (*DataNodes*). A seguir serão detalhadas suas configurações.

3.2.1 Nó Mestre - *NameNode*

Para este nó foi criado uma VM com um processador com 2 (dois) núcleos. Também possui 2 GB de memória RAM e um total de 80 GB de espaço em disco. O sistema operacional instalado na mesma era o Linux Ubuntu 14.04 Server Edition LTS com *kernel* 3.16.0.30 *generic*.

3.2.2 Nós Escravos - *DataNodes*

Para os nós escravos foram criadas três VMs com 1 (um) processador cada. Também possui 1 GB de memória RAM e um total de 90 GB de espaço em disco. O sistema operacional instalado na mesma era o Linux Ubuntu 14.04 *Server Edition* LTS com *kernel* 3.16.0.30 *generic*.

3.3 AMBIENTE HADOOP E HIVE

A versão do Hadoop utilizada no cenário experimental foi a 2.7.1, lançada em 06 de Julho de 2015, a qual apresenta uma série de melhorias feitas desde a última versão (2.6.0), algumas delas são listadas a seguir, segundo a documentação do projeto, localizada no *site* da fundação Apache (TM, 2015a):

- A versão aboliu o suporte ao pacote JDK (*Java Development Kit*) 6 do Java e agora trabalha apenas com a versão 7 ou superior da ferramenta;
- Suporte para arquivos com tamanhos de blocos variáveis;
- A versão traz também o suporte à plataforma Microsoft Azure, para execução de aplicati-

vos e serviços utilizando a computação em nuvem.

A versão do Hive utilizada no cenário experimental foi a 0.13.0, lançada em 21 de Abril de 2014. Dentre as melhorias mais significativas, algumas estão relacionadas à linguagem HQL listadas a seguir, segundo o *site* da IBM (IBM, 2016):

- A versão inclui o tipo de dado CHAR;
- Suporte às cláusulas IN/NOT IN para subconsultas e também as cláusulas EXISTS / NOT EXISTS;
- Suporte à precisão para o tipo de dado DECIMAL.

3.4 RESULTADOS E TESTES

Os testes realizados no cenário experimental compõem a parte final deste trabalho, contextualizando desta forma, todos os tópicos apresentados nas seções anteriores e apresentando os resultados obtidos na execução das ferramentas aplicadas ao caso prático proposto.

Inicialmente, foi necessário instalar e configurar as 4 (quatro) VMs, uma *Master* e 3 (três) (*Slaves*) conforme descrito na Seção 3.2. Feito isso, o Hadoop foi instalado em todas as máquinas do *cluster* e inicializado com os comandos: ***start-dfs.sh*** e ***start-yarn.sh*** (Apêndice A). Na continuidade dos processos, o Hive também foi instalado, porém somente na máquina *Master* (Apêndice B), a qual irá gerenciar e executar todas as consultas na base de dados.

O próximo passo para a execução dos testes foi a geração da base de dados, realizada pela ferramenta DBGen (Apêndice C), fornecida pelo *site* da organização TPC (COUNCIL, 2014b). Para o experimento foi utilizada uma base de dados com fator de escala 1 (1 GB), tendo em vista que o objetivo do trabalho é simplesmente executar as 22 consultas fornecidas pelo *benchmark* TPC-H e extrair o tempo de resposta de cada uma.

Por fim, a ferramenta Hive 600 foi instalada e configurada, onde foi possível realizar a distribuição dos dados entre todos os nós do *cluster*, através do arquivo: **tpch_prepare_data.sh**. Também foi definida a quantidade de vezes que cada consulta seria submetida (5 (cinco) vezes), configurando o arquivo: **benchmark.conf**. Assim, foi possível executar o experimento através do arquivo: **tpch_benchmark.sh** (Apêndice C).

Os resultados obtidos são apresentados na Tabela 4, onde encontram-se dispostas as consultas, a quantidade de vezes que cada uma foi executada e o tempo de execução (em segundos). Estas informações foram arranjadas a fim de verificar se houve alguma variação significativa no tempo de resposta de cada uma.

Tabela 4 – Tabela de resultados do experimento

	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
Consulta 1	6,62	7,19	6,26	6,50	6,48
Consulta 2	6,66	6,26	6,33	6,55	6,17
Consulta 3	7,44	7,21	6,88	6,49	6,48
Consulta 4	6,77	6,62	6,59	6,92	6,62
Consulta 5	6,44	6,48	6,87	6,70	6,55
Consulta 6	6,86	6,65	6,52	6,58	6,71
Consulta 7	6,46	6,30	6,41	7,33	6,61
Consulta 8	6,83	6,56	6,87	6,61	6,70
Consulta 9	6,76	7,28	6,98	6,65	6,74
Consulta 10	7,78	6,64	6,68	6,47	6,53
Consulta 11	6,54	6,70	7,21	6,34	6,66
Consulta 12	6,70	6,54	6,72	6,64	6,43
Consulta 13	6,79	6,81	6,49	6,75	6,50
Consulta 14	6,59	6,55	6,70	6,69	6,55
Consulta 15	6,56	6,53	6,34	6,77	6,52
Consulta 16	6,45	6,67	7,16	6,56	6,54
Consulta 17	6,58	6,76	6,32	6,56	6,53
Consulta 18	6,53	6,65	6,90	6,71	6,53
Consulta 19	7,60	6,66	6,37	6,32	6,88
Consulta 20	6,54	6,58	7,97	6,72	6,59
Consulta 21	6,35	6,60	7,36	6,28	6,70
Consulta 22	7,37	6,59	6,63	6,60	6,50

Fonte: Autoria Própria

A partir do tempo de execução das consultas obtem-se algumas informações relevantes e que podem auxiliar em algumas conclusões. Pode-se obter a média dos tempos de execução de cada consulta, o desvio padrão e o intervalo de confiança (IC). O desvio padrão (σ) é encontrado conforme exemplificado na Equação 3.1 e neste caso, indica a variação que existe no tempo de execução das consultas em relação à média aritmética encontrada.

$$\sigma_i = \sqrt{\frac{\sum_{i=1}^{n} \left(x_i - \overline{x_i}\right)^2}{n-1}} \tag{3.1}$$

Em que:

σ_i = Desvio padrão. $\overline{x}_i$ = Média aritmética. x_i = Série de valores. n = Núm. de amostras.

Segundo Silva (2013), o intervalo de confiança representa os limites inferior e superior que um determinado intervalo de valores pode assumir em novos testes para obtenção de novos valores seguindo os mesmos cenários. Em uma adaptação para o cenário deste trabalho, representa os limites inferior e superior que o tempo de execução pode assumir em outros casos em que se submeter uma consulta à base de dados, de acordo com um valor de confiança. No presente trabalho, o valor de confiança empregado é de 95% (IC 95%). Isto quer dizer que, para as médias obtidas na execução das consultas o IC calculado inclui a verdadeira média dos tempos

em 95% dos casos. A Equação 3.2 mostra como é alcançado o valor do IC.

$$IC_i = \overline{x}_i \pm z * \frac{\sigma_i}{\sqrt{n}} \tag{3.2}$$

Em que:

$\overline{x}_i$ = Média σ_i = Desvio padrão z = Valor de confiança n = Número de amostras

Uma vez encontrado o IC para cada série de execução das consultas pode-se encontrar os limites inferiores e superiores para as mesmas. O limite inferior (I_i) representa o valor mínimo de tempo que uma consulta poderá alcançar se executada, isto com uma confiança de 95 %. Seu valor é obtido através da Equação 3.3. Em contrapartida, o limite superior indica o tempo máximo que uma consulta poderá levar para ser executada. Seu valor é obtido pela Equação 3.4

$$I_i = \overline{x}_i - IC_i \tag{3.3}$$

$$L_i = \overline{x}_i + IC_i \tag{3.4}$$

Em que:

$\overline{x}_i$ = Média aritmética IC_i = Intervalo de confiança

Tendo como base a Tabela 5 e com o intuito de exemplificar de forma gráfica as informações encontradas, apresenta-se na Figura 15 a distribuição das informações das consultas. Este gráfico distribui, em seu eixo x, as 22 consultas e no eixo y a média do tempos de execução de cada uma delas. Neste gráfico também é possível visualizar os intervalos de confiança através dos limites inferior e superior sobrepostos aos tempos.

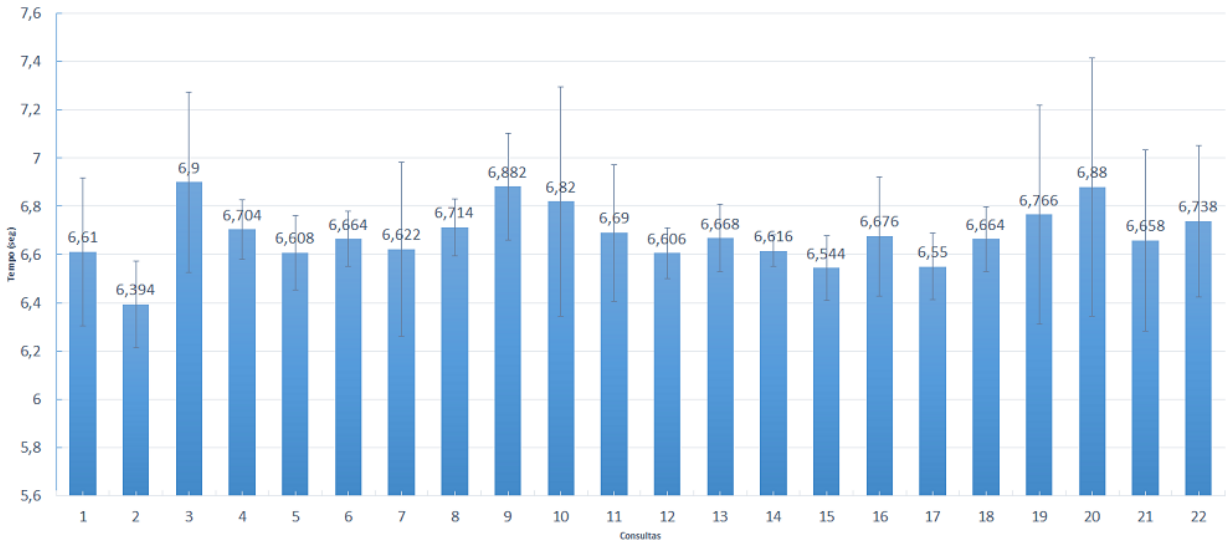

Figura 15 – Gráfico das médias dos tempos de execução das consultas
Fonte: Autoria própria

Com a finalidade de facilitar a visualização e entendimento de todas estas informações, a Tabela 5 apresenta os valores obtidos e descritos anteriormente para as 22 consulta do TPC-H que foram submetidas à base de dados.

Tabela 5 – Informações Adicionais

	Média ($\overline{x}$)	**Des. Padrão (σ)**	**IC**	**Lim. Inferior (I_i)**	**Lim. Superior (L_i)**
Consulta 1	6,610	0,349	0,306	6,303	6,916
Consulta 2	6,394	0,204	0,179	6,214	6,573
Consulta 3	6,900	0,427	0,375	6,524	7,275
Consulta 4	6,704	0,139	0,122	6,581	6,826
Consulta 5	6,608	0,176	0,154	6,453	6,762
Consulta 6	6,664	0,130	0,114	6,549	6,778
Consulta 7	6,622	0,411	0,360	6,261	6,982
Consulta 8	6,714	0,134	0,118	6,595	6,832
Consulta 9	6,882	0,253	0,222	6,659	7,104
Consulta 10	6,820	0,543	0,476	6,343	7,296
Consulta 11	6,690	0,322	0,282	6,407	6,972
Consulta 12	6,606	0,120	0,105	6,500	6,711
Consulta 13	6,668	0,159	0,139	6,528	6,807
Consulta 14	6,616	0,074	0,064	6,551	6,680
Consulta 15	6,544	0,153	0,134	6,409	6,678
Consulta 16	6,676	0,281	0,246	6,429	6,922
Consulta 17	6,550	0,156	0,137	6,412	6,687
Consulta 18	6,664	0,153	0,134	6,529	6,798
Consulta 19	6,766	0,518	0,454	6,311	7,220
Consulta 20	6,880	0,613	0,537	6,342	7,417
Consulta 21	6,658	0,428	0,375	6,282	7,033
Consulta 22	6,738	0,356	0,312	6,425	7,050

Fonte: Autoria própria

Com base na Tabela 4, Tabela 5 e no gráfico apresentado na Figura 15 apresenta-se no Capítulo 4 algumas conclusões acerca da série de execuções das consultas no cenário apresentado neste trabalho.

4 CONCLUSÃO E TRABALHOS FUTUROS

Esta seção contém as considerações finais do presente trabalho, bem tópicos relacionados à trabalhos futuros que podem ser desenvolvidos com base no tema proposto.

4.1 CONSIDERAÇÕES FINAIS

Em todos esses anos da era da computação em nossa sociedade, é notável o avanço constante e a criação de novas tecnologias para diversas áreas do mercado. Contudo, há uma tecnologia que perdurou durante todo esse tempo, conhecida por SGBDRs (Sistemas Gerenciadores de Banco de Dados Relacionais). Essa tendência tradicional oferece uma série de vantagens como: a estabilidade no armazenamento de dados, estruturas bem definidas, consistência e integridade dos dados armazenados. Porém, alguns pontos cruciais como escalabilidade das aplicações e grande acesso às unidades de memória, tornaram-se um verdadeiro ”gargalo” para os SGBDRs.

Atualmente, o grande desafio está na manipulação de grandes volumes de dados e muitas tecnologias têm seu funcionamento baseado na distribuição de bases de dados em *clusters* de servidores. O presente trabalho teve como objetivo principal, a elaboração de um cenário experimental, onde foi possível distribuir uma base de dados em um ambiente virtualizado, criando-se um *cluster* de 4 (quatro) máquinas e com o auxílio das ferramentas Hadoop e Hive, executar consultas em uma base sintética fornecida pelo *benchmark* TPC-H.

Os resultados obtidos demonstraram que o tempo de execução de cada consulta foi relativamente rápido e que não houve uma variação muito significativa entre uma execução e outra. Outro ponto observado foi que, utilizando uma base de dados de 1GB, o tempo para a geração dos dados e distribuição dos mesmos também foi rápido, visto que as VMs dividem os recursos de memória, disco e CPU (*Central Processing Unit* - Unidade Central de Processamento) da máquina física. Contudo, pode-se concluir que, o cenário atendeu aos requisitos propostos no desenvolvimento deste trabalho, trazendo resultados para avaliação do tempo de resposta e servindo como base para futuras pesquisas, sejam elas em ambientes virtualizados, ou fazendo uso de máquinas físicas.

É evidente, que por se tratar de uma tecnologia recente, ainda há muito o que ser desenvolvido e pesquisado com as ferramentas em questão, sendo assim, abaixo seguem algumas contribuições que são proporcionadas por este trabalho:

- Criação de um cenário experimental para fins de testes em bases de dados distribuídas, utilizando a virtualização. Para isso fez-se necessário:
  - Aplicar os conceitos da arquitetura MapReduce através dos *frameworks* Hadoop e

Hive, o que possibilitou a distribuição da base de dados no cenário, bem como a execução dos testes;

  - Utilizar uma ferramenta que adapta as consultas (antes escritas em linguagem SQL) do *benchmark* TPC-H, reescrevendo-as para a linguagem HQL (utilizada pelo Hive);
  - Realizar a geração da base de dados utilizando a ferramente *DBgen*, para o *benchmark* TPC-H, a qual foi disponibilizada pela organização TPC;
  - Capturar e analisar o tempo de resposta de cada consulta executada na base de dados.

- Comprovação da agilidade do processo de execução das consultas submetidas a base de dados.

4.2 TRABALHOS FUTUROS

O presente trabalho possibilita e contribui para novas linhas de pesquisa e trabalhos que, futuramente, venham a ser desenvolvidos, estes com o intuito de difundir as tecnologias apresentadas para fins acadêmicos e de mercado. Abaixo são listadas algumas possibilidades:

- Criação do cenário experimental utilizando máquinas físicas para efeitos de comparação com o ambiente virtualizado;
- Utilização de SF maior para a base de dados, verificando o comportamento e funcionamento das ferramentas utilizadas sobre cargas de dados mais volumosas;
- Comparação de resultados com outros SGBDs, visto que o Hadoop apresenta compatibilidade com outras ferramentas e plataformas;
- Adicionar uma quantidade maior de máquinas ao *cluster* para verificação do desempenho e distribuição dos dados.

REFERÊNCIAS

ARMBRUST, B. *et al.* A view of cloud computing. **Communications of the ACM**, ACM, v. 53, n. 4, p. 50–58, 2010. Disponível em: <http://portal.acm.org/citation.cfm?id=1721672>.

BAKER, M.; BUYYA, R. Cluster computing: The commodity supercomputer. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 29, n. 6, p. 551–576, maio 1999. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/(SICI)1097-024X(199905)29:6<551::AID-SPE248>3.0.CO;2-C>.

BERRY, G. C. M.; LARSON, J. Scientific benchmarks characterization. **Parallel Computing**, v. 17, n. 10, 1991.

BERRY, M. *et al.* The perfect club benchmarks: Effective performance evaluation of supercomputers. **International Journal of Supercomputer Applications**, v. 3, p. 5–40, 1988.

BINI, T. A. **Análise da aplicabilidade das regras de ouro ao tuning de sistemas gerenciadores de bancos de dados relacionais em ambiente de computação em nuvem**. Tese (Doutorado) — Universidade Federal do Paraná, Setor de ciências exatas, Curso de Pós-Graduação em Informática, Curitiba, 2014.

BOJANOVA, I.; SAMBA, A. Analysis of cloud computing delivery architecture models. In: **Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications**. Washington, DC, USA: IEEE Computer Society, 2011. (WAINA '11), p. 453–458. ISBN 978-0-7695-4338-3. Disponível em: <http://dx.doi.org/10.1109/WAINA.2011.74>.

BORAL, H.; DEWITT, D. J. A methodology for database system performance evaluation. In: **Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 1984. (SIGMOD '84), p. 176–185. ISBN 0-89791-128-8. Disponível em: <http://doi.acm.org/10.1145/602259.602283>.

BORTHAKUR, D. The hadoop distributed file system: Architecture and design. **Hadoop Project Website**, v. 11, n. 2007, p. 21, 2007.

BUYYA, R. **High Performance Cluster Computing: Architectures and Systems**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130137847.

BUYYA, R. *et al.* Economic models for resource management and scheduling in grid computing. In: . [S.l.]: Wiley Press, 2002. p. 1507–1542.

CAPRIOLO, E.; WAMPLER, D.; RUTHERGLEN, J. **Programming Hive**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2012. ISBN 1449319335, 9781449319335.

CHAUDHURI, S.; NARASAYYA, V. **Identifying essential statistics for query optimization for databases**. Google Patents, 2002. US Patent 6,363,371. Disponível em: <http://www.google.com/patents/US6363371>.

CHIEU, T. *et al.* Dynamic scaling of web applications in a virtualized cloud computing environment. In: **e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on**. [S.l.: s.n.], 2009. p. 281–286.

CHIEU, T. C. *et al.* Dynamic scaling of web applications in a virtualized cloud computing environment. In: **Proceedings of the 2009 IEEE International Conference on e-Business Engineering**. Washington, DC, USA: IEEE Computer Society, 2009. (ICEBE '09), p. 281–286. ISBN 978-0-7695-3842-6. Disponível em: <http://dx.doi.org/10.1109/ICEBE.2009.45>.

CIFERRI, R. R. **Um benchmark voltado à análise de desempenho de sistemas de informações geográficas**. Dissertação (Mestrado) — Universidade Estadual de Campinas - UNICAMP, Cidade Universitária Zeferino Vaz - Barão Geraldo, Campinas - SP, 13083-970, 7 1995.

CIURANA, E. **Apache Hadoop Deployment: A Blueprint for Reliable Distributed Computing**. [S.l.], 2011. 7 p.

COUNCIL, T. P. P. **TPC Benchmark™C Standard Specification Revision 5.11**. [S.l.], 2010. 132 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5-11.pdf>.

____. **TPC Pricing Specification - Standard Specification Revision 1.7.0**. [S.l.], 2011. 49 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/price_v1.7.0.pdf>.

____. **TPC Energy Specification - Standard Specification Revision 1.5.0**. [S.l.], 2012. 60 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpc%20energy%20specification%201.5.0.pdf>.

____. **TPC Virtual Measurement Single System TPC-VMS Standard Specification Revision 1.2.0**. [S.l.], 2013. 62 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-vms-v1.2.0.pdf>.

____. **TPC Benchmark™DI Standard Specification Revision 1.1.0**. [S.l.], 2014. 117 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpcdi-v1.1.0.pdf>.

____. **TPC Benchmark™H Standard Specification Revision 2.17.1**. [S.l.], 2014. 136 p. Disponível em: <www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf>.

____. **Active TPC Benchmarks**. 2015. Disponível em: <http://www.tpc.org/information/benchmarks.asp>.

____. **TPC Benchmark™DI Standard Specification Revision 1.3.0**. [S.l.], 2015. 38 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-hs_specification_1.3.0.pdf>.

____. **TPC Benchmark™DS Standard Specification Revision 1.4**. [S.l.], 2015. 156 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpcds_1.4.pdf>.

____. **TPC Benchmark™E Standard Specification Revision 1.14.0**. [S.l.], 2015. 287 p. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpce-v1.14.0.pdf>.

DEAN, J.; GHEMAWAT, S. **MapReduce: Simplified Data Processing on Large Clusters**. 2004. Disponível em: <http://research.google.com/archive/mapreduce.html>.

____. Mapreduce: Simplified data processing on large clusters. **Commun. ACM**, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

____. Mapreduce: A flexible data processing tool. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 1, p. 72–77, jan. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1629175.1629198>.

DEMCHENKO, Y.; LAAT, C. de; MEMBREY, P. Defining architecture components of the big data ecosystem. In: **Collaboration Technologies and Systems (CTS), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 104–112.

FIGUEIREDO, R. J. O.; DINDA, P. A.; FORTES, J. A. B. Guest editors' introduction: Resource virtualization renaissance. **IEEE Computer**, v. 38, n. 5, p. 28–31, 2005. Disponível em: <http://dblp.uni-trier.de/db/journals/computer/computer38.html#FigueiredoDF05>.

FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid: Blueprint for a New Computing Infrastructure**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-475-8.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **Int. J. High Perform. Comput. Appl.**, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 15, n. 3, p. 200–222, ago. 2001. ISSN 1094-3420. Disponível em: <http://dx.doi.org/10.1177/109434200101500302>.

GANTZ, J.; REINSEL, D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. **IDC IVIEW**, EMC Corporation, p. 1–16, Dec 2012.

GARLASU, D. *et al.* A big data implementation based on grid computing. In: **Roedunet International Conference (RoEduNet), 2013 11th**. [S.l.: s.n.], 2013. p. 1–4. ISSN 2068-1038.

GASPAROTTO, H. M. **Hadoop MapReduce: Introdução a Big Data**. 2014. Disponível em: <http://www.devmedia.com.br/hadoop-mapreduce-introducao-a-big-data/30034>.

GHEMAWAT HOWARD GOBIOFF, S.-T. L. S. **The Google File System**. 2003. Disponível em: <http://research.google.com/archive/gfs.html>.

GILDER, G. Information factories. In: . [S.l.]: Wired, 2006.

GRAY, J. **Benchmark Handbook: For Database and Transaction Processing Systems**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN 1558601597.

GROLINGER, K. *et al.* Challenges for mapreduce in big data. In: **Services (SERVICES), 2014 IEEE World Congress on**. [S.l.: s.n.], 2014. p. 182–189.

GRUENHEID, A.; OMIECINSKI, E.; MARK, L. Query optimization using column statistics in hive. In: **Proceedings of the 15th Symposium on International Database Engineering & Applications**. New York, NY, USA: ACM, 2011. (IDEAS '11), p. 97–105. ISBN 978-1-4503-0627-0. Disponível em: <http://doi.acm.org/10.1145/2076623.2076636>.

HEDLUND, B. **Understanding Hadoop Clusters and the Network**. 2011. Acessado em 18 de Setembro de 2015. Disponível em: <http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/>.

HURWITZ, J. *et al.* **Big Data For Dummies**. 1st. ed. [S.l.]: For Dummies, 2013. ISBN 1118504224, 9781118504222.

IBM. 2016. Disponível em: <https://www.ibm.com/support/knowledgecenter/SSPT3X_4.0.0/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/bi_hive.html>.

IBM; ZIKOPOULOS, P.; EATON, C. **Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data**. 1st. ed. [S.l.]: McGraw-Hill Osborne Media, 2011. ISBN 0071790535, 9780071790536.

JIA, Y. **[Hive-600] Running TPC-H queries on Hive**. 2009. Disponível em: <https://issues.apache.org/jira/browse/HIVE-600>.

____. **Running the TPC-H Benchmark on Hive**. [S.l.], 2009. 26 p. Disponível em: <https://issues.apache.org/jira/secure/attachment/12416257/TPC-H_on_Hive_2009-08-11.pdf>.

KATAL, A.; WAZID, M.; GOUDAR, R. Big data: Issues, challenges, tools and good practices. In: **Contemporary Computing (IC3), 2013 Sixth International Conference on**. [S.l.: s.n.], 2013. p. 404–409.

KAUSHIK, K. **Cluster Computing**. 2008. Computer Science and Engineering, Cochin University of science and technology, Kochi, Kerala 682022, India.

KAWA, A.; KREWSKI, P. **Getting Started with Apache Hadoop**. [S.l.], 2014. 7 p.

KOHLER, W. H.; HSU, Y.-P. A benchmark for the performance evaluation of centralized and distributed transaction processing systems. In: **Distributed Computing Systems, 1990. Proceedings., Second IEEE Workshop on Future Trends of**. [S.l.: s.n.], 1990. p. 488–493.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 32, n. 2, p. 135–164, fev. 2002. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.432>.

KU, M.; MIN, D.; CHOI, E. Scarex: A framework for scalable, reliable, and extendable cluster computing. In: **Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on**. [S.l.: s.n.], 2010. p. 966–972.

LOMOTEY, R.; DETERS, R. Analytics-as-a-service (aaas) tool for unstructured data mining. In: **Cloud Engineering (IC2E), 2014 IEEE International Conference on**. [S.l.: s.n.], 2014. p. 319–324.

MANAGER, V. M. 2016. Disponível em: <https://virt-manager.org/>.

MATTOS, D. M. F. **Virtualização: VMWare e Xen**. [S.l.], 2010. 13 p. Acessado em: 11 de Outubro de 2015. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/artigo.pdf>.

MATTURDI, B. *et al.* Big data security and privacy: A review. **Communications, China**, v. 11, n. 14, p. 135–145, Supplement 2014. ISSN 1673-5447.

MCAFEE, A.; BRYNJOLFSSON, E. Big Data: The management revolution. **Harvard Business Review**, v. 90, n. 10, p. 60–68, 2012.

MELL, P.; GRANCE, T. The nist definition of cloud computing. **NIST Special Publication**, v. 800-145, p. 1–3, 2011.

MURTHY, A. **Apache Hadoop 2 is now GA!** 2013. Disponível em: <http://br.hortonworks.com/blog/apache-hadoop-2-is-ga/>.

PANDEY, S.; TOKEKAR, V. Prominence of mapreduce in big data processing. In: **Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on**. [S.l.: s.n.], 2014. p. 555–560.

POESS, M.; FLOYD, C. New tpc benchmarks for decision support and web commerce. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 29, n. 4, p. 64–71, dez. 2000. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/369275.369291>.

PRAJAPATI, V. 2015. Disponível em: <http://pingax.com/install-apache-hadoop-ubuntu-cluster-setup/>.

PREM., O. 2014. Disponível em: <http://doctuts.readthedocs.org/en/latest/hadoop.html>.

QEMU-KVM. 2016. Disponível em: <http://wiki.qemu.org/MainPage>.

RIMAL, B. P.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: **Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC**. Washington, DC, USA: IEEE Computer Society, 2009. (NCM '09), p. 44–51. ISBN 978-0-7695-3769-6. Disponível em: <http://dx.doi.org/10.1109/NCM.2009.218>.

ROSE, R. **Survey of System Virtualization Techniques**. [S.l.], 2004.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial, Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. [S.l.]: Novatec, 2013. ISBN 1558601597.

SADASHIV, N.; KUMAR, S. M. D. Cluster, grid and cloud computing: A detailed comparison. In: **Computer Science Education (ICCSE), 2011 6th International Conference on**. [S.l.: s.n.], 2011. p. 477–482.

SHARMA, P.; KUMAR, B.; GUPTA, P. An introduction to cluster computing using mobile nodes. In: **2009 Second International Conference on Emerging Trends in Engineering Technology**. [S.l.: s.n.], 2009. p. 670–673. ISSN 2157-0477.

SHVACHKO, K. *et al.* The hadoop distributed file system. In: **Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)**. Washington, DC, USA: IEEE Computer Society, 2010. (MSST '10), p. 1–10. ISBN 978-1-4244-7152-2. Disponível em: <http://dx.doi.org/10.1109/MSST.2010.5496972>.

SILVA, G. S. R. **USO INTEGRADO DE COMPUTAÇÃO EM NUVEM E GOOGLE ANDROID NO PROCESSO DE DESCOBERTA DE CONHECIMENTO**. Trabalho de conclusão de curso, Maio 2013.

SPEC. **Standard Performance Evaluation Corporation**. 2015. Disponível em: <https://www.spec.org/spec/>.

SRIDHARAN, A. 2014. Disponível em: <http://documents.mx/documents/tpc-honhive.html>.

TECHNOLOGIES, M. **What is Apache Hadoop?** 2015. Disponível em: <https://www.mapr.com/products/apache-hadoop>.

THUSOO, A. *et al.* Hive: A warehousing solution over a map-reduce framework. **Proc. VLDB Endow.**, VLDB Endowment, v. 2, n. 2, p. 1626–1629, ago. 2009. ISSN 2150-8097. Disponível em: <http://dx.doi.org/10.14778/1687553.1687609>.

____. Hive - a petabyte scale data warehouse using hadoop. In: LI, F. *et al.* (Ed.). **ICDE**. IEEE, 2010. p. 996–1005. ISBN 978-1-4244-5444-0. Disponível em: <http://dblp.uni-trier.de/db/conf/icde/icde2010.html#ThusooSJSCZALM10>.

TM, A. H. **Apache Hadoop**. 2015. Disponível em: <https://hadoop.apache.org>.

____. **Apache Hive**. 2015. Disponível em: <https://cwiki.apache.org/confluence/display/Hive/Home/>.

____. **Arquivo de configurações do HDFS Hadoop**. 2015. Acessado em: 19 de Setembro de 2015. Disponível em: <https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>.

____. **Hive**. 2015. Disponível em: <https://hive.apache.org/>.

____. **Hive**. 2015. Disponível em: <http://hbase.apache.org/>.

TM, A. P. **Apache Avro**. 2014. Disponível em: <https://avro.apache.org/>.

____. **Apache Pig**. 2015. Disponível em: <https://pig.apache.org/>.

TPC. **Transactional Processing Performance**. 2015. Disponível em: <http://www.tpc.org/>.

TYLER, T.; FISHER, D. Using distributed oltp technology in a high performance storage system. In: **Mass Storage Systems, 1995. 'Storage - At the Forefront of Information Infrastructures', Proceedings of the Fourteenth IEEE Symposium on**. [S.l.: s.n.], 1995. p. 45–53. ISSN 1051-9173.

VAQUERO, L. M. *et al.* A break in the clouds: Towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, 2009. ISSN 0146-4833.

VENNER, J. **Pro Hadoop**. 1st. ed. Berkely, CA, USA: Apress, 2009. ISBN 1430219424, 9781430219422.

WANG, K. *et al.* Simulating hive cluster for deployment planning, evaluation and optimization. In: **Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on**. [S.l.: s.n.], 2014. p. 475–482.

WHITE, T. **Hadoop: The Definitive Guide**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2009. ISBN 0596521979, 9780596521974.

ZADROZNY, P.; KODALI, R. **Big Data Analytics Using Splunk: Deriving Operational Intelligence from Social Media, Machine Data, Existing Data Warehouses, and Other Real-Time Streaming Sources**. 1st. ed. Berkely, CA, USA: Apress, 2013. ISBN 143025761X, 9781430257615.

APÊNDICE A - Processo de instalação do ambiente Hadoop

PROCESSO DE INSTALAÇÃO DO AMBIENTE HADOOP

Este apêndice apresenta o passo-a-passo da instalação do ambiente Hadoop nas VMs do *cluster*, bem como os arquivos de configuração, comandos executados e inicialização dos serviços da ferramenta (Obs: Todos os procedimentos descritos a seguir devem ser executados primeiro na máquina *Master* e posteriormente repetidos em todas as máquinas *Slaves*).

- **Pré-requisitos**

  - **Pacote Java JDK:** Após todas as máquinas do *cluster* terem sido criadas e inicializadas, a primeira coisa a ser feita é a instalação do pacote *Sun Java X JDK*, utilizando o comando:

    * **sudo apt-get install sun-javaX-jdk**

    Para verificar se a versão do Java foi instalada corretamente, basta executar o comando:

    * **java -version**

  - **Configuração de chaves SSH:** Para que o Hadoop possa funcionar corretamente e gerenciar todos os nós do *cluster*, é necessário gerar chaves SSH para comunicação entre as máquinas, isso permite o acesso da máquina *Master* às máquinas *Slave* e vice-versa.
    No terminal, o comando a ser executado é:

    * **ssh-keygen -t rsa -P ”**

    Feito isso o sistema fornecerá um par de chaves, contendo uma chave pública e uma privada, localizadas no diretório **/home/<nome-da-máquina>/.ssh/**. Contudo, é necessário copiar a chave pública recém gerada para o diretório de chaves autorizadas, através do comando:

    * **cat /home/<nome-da-máquina>/.ssh/id_rsa.pub »**
      /home/<nome-da-máquina>/.ssh/authorized_keys

    O arquivo **authorized_keys** de cada máquina deve conter a sua própria chave incluindo a chave de todos os outros nós do *cluster*.

- **Instalação do Hadoop**

  - **Download:** Após toda a configuração dos pré-requisitos, o *download* do Hadoop pode ser realizado através do *site* da ferramenta (http://www.apache.org), o qual irá direcionar para um repositório contendo o arquivo: **hadoop-X.X.X.tar.gz**. Feito isso, o arquivo deve ser descompactado em algum diretório dentro da máquina (para o experimento deste trabalho, o diretório escolhido foi: **/usr/local/**).

– **Variáveis de Ambiente:** O Hadoop possui ainda algumas variáveis de ambiente que precisam ser configuradas nas máquinas do *cluster*, para tanto, deve-se editar o arquivo **.bashrc**, através do comando:

  * **nano /home/<nome-da-máquina>/.bashrc**.

  Ao final desse arquivo, adiciona-se as linhas exibidas na Figura 16:

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Add Hadoop bin/ directory to PATH
export PATH= $PATH:$HADOOP_HOME/bin
```

Figura 16 – Variáveis de Ambiente Hadoop
Fonte: (PREM., 2014)

– **Variável JAVA_HOME:** No começo da instalação do ambiente Hadoop, o pacote Java precisou ser instalado. Agora é necessário configurar a variável de ambiente *JAVA_HOME* que irá apontar para a pasta do *Sun Java 6 JDK*. Para isso, o arquivo **hadoop-env.sh** deve ser editado, utilizando o comando:

  * **nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh**.

  As linhas apresentadas na Figura 17 devem ser adicionadas:

```
# export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64  (for 64 bit)
# export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64  (for 32 bit)
```

Figura 17 – Variável JAVA_HOME
Fonte: (PREM., 2014)

– **Arquivo Core-site.xml:** O próximo arquivo a ser configurado é o **core-site.xml**. Devemos alterá-lo conforme a Figura 18, através do comando:

  * **nano /usr/local/hadoop/etc/hadoop/core-site.xml**

  Com isso, adiciona-se o bloco de tags **<property>** dentro das tags **<configuration>** e substitui-se a palavra *localhost* pelo nome da máquina *Master* (configurado no arquivo **hosts**):

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit core-site.xml

## Paste these lines into <configuration> tag OR Just update it by replacing localhost with maste
r
<property>
  <name>fs.default.name</name>
  <value>hdfs://HadoopMaster:9000</value>
</property>
```

Figura 18 – Exemplo de um arquivo core-site configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Hosts:** O arquivo Hosts deve ser configurado com o objetivo de dar nomes às máquinas do *cluster*, antes representadas somente através do endereço *IP*. Deve-se executar o comando:

  * **nano /etc/hosts**.

  A partir do momento que o arquivo é aberto, adiciona-se as linhas conforme o exemplo da Figura 19, onde são especificados os nomes referentes a cada endereço *IP* (esse arquivo deve ser alterado em todos os nós do *cluster*:

```
# Edit the /etc/hosts file with following command
sudo gedit /etc/hosts

# Add following hostname and their ip in host table
192.168.2.14    HadoopMaster
192.168.2.15    HadoopSlave1
192.168.2.16    HadoopSlave2
```

Figura 19 – Exemplo de um arquivo Hosts configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Hdfs-site.xml** O arquivo **hdfs-site.xml** é onde será configurado o grau de replicação dos dados gerados, ou seja, é a quantidade de máquinas do *cluster* que o Hadoop irá realizar a distribuição. No caso deste experimento, os dados serão distribuídos em 3 nós escravos (*Slaves*) e como no exemplo da Figura 20 as tags **<property>** devem ser inseridas entre as tags **<configuration>**, o comando para edição do arquivo é:

  * **nano usr/local/hadoop/etc/hadoop/hdfs-site.xml**

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit hdfs-site.xml

## Paste/Update these lines into <configuration> tag
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```

Figura 20 – Exemplo de um arquivo hdfs-site configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Yarn-site.xml:** O arquivo **yarn-site.xml** deve ser configurado como mostra o exemplo da Figura 21, colocando as tags **<property>** entre as tags **<configuration>** e alterando a palavra *localhost* para o nome da máquina mestre (*Master*). Para editar o arquivo, basta executar o comando:

  * **nano /usr/local/hadoop/etc/hadoop/yarn-site.xml**

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit yarn-site.xml

## Paste/Update these lines into <configuration> tag
<property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>HadoopMaster:8025</value>
</property>
<property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>HadoopMaster:8035</value>
</property>
<property>
        <name>yarn.resourcemanager.address</name>
        <value>HadoopMaster:8050</value>
</property>
```

Figura 21 – Exemplo de um arquivo yarn-site configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Mapred-site.xml:** O arquivo **mapred-site.xml** (**/usr/local/hadoop/etc/hadoop/**) deve ser configurado como mostra o exemplo da Figura 22 colocando as tags **<property>** entre as tags **<configuration>** e alterando a palavra *localhost* para o nome da máquina mestre (*Master*), para editar o arquivo, basta executar o comando:

  * **nano mapred-site.xml**

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit mapred-site.xml

## Paste/Update these lines into <configuration> tag
<property>
        <name>mapreduce.job.tracker</name>
        <value>HadoopMaster:5431</value>
</property>
<property>
        <name>mapred.framework.name</name>
        <value>yarn</value>
</property>
```

Figura 22 – Exemplo de um arquivo mapred-site configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Masters:** O arquivo masters identifica para o Hadoop o nome da máquina mestre (*Master*) e é o único arquivo que não deve ser configurado nas máquinas escravo (*Slave*). A Figura 23 apresenta a estrutura deste arquivo. Para editá-lo, o comando é:

  * **nano /usr/local/hadoop/etc/hadoop/masters**

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit masters

## Add name of master nodes
HadoopMaster
```

Figura 23 – Exemplo de um arquivo masters configurado
Fonte: (PRAJAPATI, 2015)

– **Arquivo Slaves:** O arquivo slaves (**/usr/local/hadoop/etc/hadoop/**) identifica para o Hadoop o nome de todas as máquinas escravo (*Slave*) e assim como o arquivo masters ele não pode ser configurado nas máquinas escravo *Slave*. A Figura 24 mostra a estrutura do arquivo. Para editá-lo, o comando é:

  * **nano slaves**

```
## To edit file, fire the below given command
hduser@HadoopMaster:/usr/local/hadoop/etc/hadoop$ sudo gedit slaves

## Add name of slave nodes
HadoopSlave1
HadoopSlave2
```

Figura 24 – Exemplo de um arquivo slaves configurado
Fonte: (PRAJAPATI, 2015)

– **Diretórios tmp e namenode** (Somente na máquina mestre): Após toda a configuração dos arquivos do Hadoop deve-se criar um diretório para o nó mestre (*namenode*), como exemplificado na Figura 25:

```
sudo mkdir -p /usr/local/hadoop_tmp/
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
```

Figura 25 – Criando diretório tmp e datanode na máquina mestre
Fonte: (PRAJAPATI, 2015)

– **Diretórios tmp e datanode** (Somente nas máquinas escravo): Após toda a configuração dos arquivos do Hadoop deve-se criar um diretório para o *datanode* dos Slaves, como exemplificado na Figura 26:

```
sudo mkdir -p /usr/local/hadoop_tmp/
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
```

Figura 26 – Criando diretório tmp e datanode nas máquinas escravo
Fonte: (PRAJAPATI, 2015)

– **Formatando o Namenode** (Somente na máquina *Master*): Finalizadas todas as configurações do Hadoop em todas as máquinas do *cluster*, podemos formatar o Namenode para que as alterações sejam realmente efetivadas, para tanto, basta ir até a pasta do Hadoop e executar o comando responsável pela formatação através dos comandos:

1. **cd /usr/local/hadoop**
2. **hdfs namenode -format**

A Figura 27 exemplifica esta ação.

```
# Run this command from Masternode
hduser@HadoopMaster: usr/local/hadoop/$ hdfs namenode -format
```

Figura 27 – Formatando o namenode
Fonte: (PRAJAPATI, 2015)

– **Inicializando o Hadoop:** Após a formatação do namenode, o Hadoop está pronto e configurado para ser inciado, para tanto, deve-se executar os dois comandos listados abaixo:

* **/usr/local/hadoop/sbin/start-dfs.sh** - Esse comando inicializa todo o sistema de arquivos distribuído do **Hadoop**, incluindo os nós Master e Slaves.
* **/usr/local/hadoop/sbin/start-yarn.sh** - Esse comando inicializa os serviços que gerenciam todos os rescursos computacionais do *cluster*.

APÊNDICE B - Processo de instalação do ambiente Hive

PROCESSO DE INSTALAÇÃO DO AMBIENTE HIVE

Este apêndice apresenta o processo de instalação do ambiente Hive, o qual se assemelha à instalação do Hadoop, porém com menos requisitos e arquivos de configuração (Obs: todos os procedimentos descritos a seguir devem ser executados somente na máquina mestre).

- **Instalação**

  - **Download:** O primeiro passo para a instalação do Hive é ter o Hadoop previamente instalado em todas as máquinas do *cluster*. Feito isso, deve-se acessar o *site* da ferramenta: (**http://apache.claz.org/hive/stable/**) e realizar o *download* do arquivo: **apache-hive-X.X.X-bin.tar.gz**. Logo após o *download* é necessário descompactar o arquivo em um diretório da máquina (para esse experimento, o diretório utilizado foi: **/usr/lib/**) e executar os comandos listados na Figura 28:

```
user@ubuntu:~$  cd  /usr/lib/
user@ubuntu:~$  sudo mkdir hive
user@ubuntu:~$  cd Downloads
user@ubuntu:~$  sudo mv apache-hive-0.13.0-bin /usr/lib/hive
```

Figura 28 – Instalação do Hive - Download
Fonte: (PREM., 2014)

  - **Variáveis de Ambiente:** Assim como no Hadoop, o Hive também possui algumas variáveis de ambiente que devem ser configuradas no arquivo: **.bashrc**, para isso, basta acessar o arquivo e inserir as linhas listadas na Figura 29:

```
user@ubuntu:~$  cd
user@ubuntu:~$  sudo gedit  ~/.bashrc


# Set HIVE_HOME
export HIVE_HOME="/usr/lib/hive/apache-hive-0.13.0-bin"
PATH=$PATH:$HIVE_HOME/bin
export PATH
```

Figura 29 – Instalação do Hive - Configuração
Fonte: (PREM., 2014)

– **Variável de ambiente Hadoop no Hive:** Para que haja a comunicação do Hadoop com o Hive, é necessário configurar a variável **$HADOOP_HOME** dentro do arquivo **hive-config.sh**. Para isso, deve-se realizar as ações descritas na Figura 30.

```
user@ubuntu:~$ cd  /usr/lib/hive/apache-hive-0.13.0-bin/bin
user@ubuntu:~$ sudo gedit hive-config.sh


# Allow alternate conf dir location.
HIVE_CONF_DIR="${HIVE_CONF_DIR:-$HIVE_HOME/conf"
export HIVE_CONF_DIR=$HIVE_CONF_DIR
export HIVE_AUX_JARS_PATH=$HIVE_AUX_JARS_PATH


export HADOOP_HOME=/usr/local/hadoop    (write the path where hadoop file is there)
```

Figura 30 – Configuração variável Hadoop
Fonte: (PREM., 2014)

– **Inicialização do Hive:** Após a configuração ter sido realizada, basta executar o comando listado na Figura 31. O Hive será executado e um terminal irá aparecer (**hive>**) para que seja inserido qualquer código de criação de base de dados e comandos relacionados:

```
user@ubuntu:~$  hive
```

Figura 31 – Inicialização do Hive
Fonte: (PREM., 2014)

APÊNDICE C - Roteiro de execução das consultas

ROTEIRO DE EXECUÇÃO DAS CONSULTAS

Este apêndice apresenta o processo da geração da base de dados utilizando o *benchmark* TPC-H (ferramenta DBGen), bem como a execução de todas as consultas no ambiente experimental criado (Obs: Todo processo descrito a seguir é executado somente na máquina mestre).

- **Geração da base de dados**

  - **Download do DBGen:** Para a geração da base de dados, o primeiro passo é realizar o *download* da ferramente *DBgen* diretamente no *site* da organização TPC (http://www.tpc.org/tpch/spec/tpch_X_X_X.zip). Após baixar o arquivo, basta descompactá-lo em algum diretório da máquina mestre.

  - **Configuração do arquivo makefile.suite:** Após realizado o *download* é necessário editar o arquivo: **makefile.suite** localizado dentro da pasta da ferramenta, a Figura 32 exemplifica os parâmetros que devem ser alterados:

```
CC  = gcc
...
...
DATABASE = SQLSERVER <Aceita qualquer valor, mas recomenda-se preenchê-lo >
MACHINE = LINUX
WORKLOAD = TPCH
...
...
```

Figura 32 – Configuração do arquivo makefile
Fonte: (SRIDHARAN, 2014)

  - **Compilando a ferramenta:** Após configurar o arquivo makefile.suite, deve-se executar o comando:

    * **make -f makefile.suite**

    Como resultado será gerada a ferramenta para criação das tabelas e dos dados do *benchmark* TPC-H.

  - **Gerando a base com o SF:** Após todos estes passos, basta executar o comando:

    * **./dbgen -s 1**

    Onde **'-s'** é o parâmetro para SF e o número 1 (um) representa o tamanho da base de dados (no caso 1 GB). Para verificar se todas as tabelas foram geradas, pode-se executar o comando:

    * **ls -l *.tbl**

    A Figura 33 apresenta os comandos executados, bem como todos os arquivos **.tbl**, localizados na pasta raiz da ferramenta:

```
[training@localhost DBGen]$ ./dbgen -s 0.1
TPC-H Population Generator (Version 2.8.0)
Copyright Transaction Processing Performance Council 1994 - 2008
[training@localhost DBGen]$ ls -l *.tbl
-rw-rw-r-- 1 training training  2426114 Sep 21 21:46 customer.tbl
-rw-rw-r-- 1 training training 74246996 Sep 21 21:46 lineitem.tbl
-rw-rw-r-- 1 training training     2224 Sep 21 21:46 nation.tbl
-rw-rw-r-- 1 training training 16893122 Sep 21 21:46 orders.tbl
-rw-rw-r-- 1 training training 11728193 Sep 21 21:46 partsupp.tbl
-rw-rw-r-- 1 training training  2398643 Sep 21 21:46 part.tbl
-rw-rw-r-- 1 training training      389 Sep 21 21:46 region.tbl
-rw-rw-r-- 1 training training   139625 Sep 21 21:46 supplier.tbl
[training@localhost DBGen]$
```

Figura 33 – Geração e verificação dos dados
Fonte: (SRIDHARAN, 2014)

- **Execução das consultas com o Hive 600 e coleta dos resultados**

  Os procedimentos que serão executados a seguir só irão funcionar corretamente se os serviços do Hadoop estiverem em execução em todas as máquinas do *cluster*, para isso é necessário executar os comandos:

  1. **start-dfs.sh**
  2. **start-yarn.sh**

  Feito isso, tem-se como ações os seguintes passos:

  – ***Download* da ferramenta:** Após a geração de todas as tabelas do *benchmark* TPC-H com o DBGen, é necessário realizar o *download* do Hive 600. Essa ferramenta possibilitará submeter as consultas à base de dados do cenário experimental. Ela encontra-se disponível para *download* no endereço a seguir e pode ser descompactada em qualquer diretório da máquina mestre:

    * **https://issues.apache.org/jira/secure/attachment/12416615**

  – **Copiando as tabelas para o Hive 600** Após descompactada a pasta do Hive 600, as tabelas geradas que encontram-se na pasta do *DBGen* devem ser copiadas para a pasta **/data** que está dentro da ferramenta Hive 600. Feito isso, é necessário rodar o arquivo **tpch_prepare_data.sh**, através do comando:

* **/<caminho-até-a-pasta-data>/tpch_prepare_data.sh**

A Figura 34 apresenta os comandos contidos nesse arquivo que irão criar as pastas no sistema de arquivos do Hadoop (HDFS) e em seguida executar o comando:

* **-put**

Este comando distribui toda a base de dados entre os nós do *cluster* (mestre e escravos):

```
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/

$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/customer
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/lineitem
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/nation
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/orders
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/part
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/partsupp
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/region
$HADOOP_HOME/bin/hadoop fs -mkdir /tpch/supplier

$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/customer.tbl /tpch/customer/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/lineitem.tbl /tpch/lineitem/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/nation.tbl /tpch/nation/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/orders.tbl /tpch/orders/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/part.tbl /tpch/part/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/partsupp.tbl /tpch/partsupp/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/region.tbl /tpch/region/
$HADOOP_HOME/bin/hadoop fs -put /<caminho-onde-encontra-se-a-tabela>/supplier.tbl /tpch/supplier/
```

Figura 34 – Comandos do arquivo prepare-data.sh
Fonte: (SRIDHARAN, 2014)

– **Verificação dos dados entre os nós do *cluster*:** Para verificar se os dados foram carregados no sistema de arquivos do Hadoop, basta executar o comando a seguir em todas as máquinas no cluster.

* **$HADOOP_HOME/bin/hadoop fs -ls /tpch**

Os dados serão apresentados conforme a Figura 35:

```
[training@localhost ~]$ $HADOOP_HOME/bin/hadoop fs -ls /tpch
Found 8 items
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/customer
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/lineitem
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/nation
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/orders
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/part
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/partsupp
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/region
drwxr-xr-x   - training supergroup          0 2011-09-21 21:55 /tpch/supplier
[training@localhost ~]$
```

Figura 35 – Verificação dos dados nas máquinas
Fonte: (SRIDHARAN, 2014)

– **Configurando o arquivo benchmark.conf:** Para que as consultas possam ser executadas, é necessário configurar o arquivo **benchmark.conf**, nele é definida a quantidade de vezes que cada consulta será submetida a base de dados. A opção a ser

configurada dentro do arquivo é a **NUM_OF_TRIALS** (Por *default* essa opção é definida como 6 (seis), mas pode ser alterada para qualquer valor desejado).

A Figura 36 representa o conteúdo do arquivo, que além dessas configurações, apresenta onde será gerado o arquivo de *log*, contendo o tempo de resposta de cada consulta (Obs: as variáveis de ambiente **$HADOOP_HOME** e **$HIVE_HOME** também devem ser definidas):

```
#!/usr/bin/env bash

BASE_DIR=`pwd`

TIME_CMD="/usr/bin/time -f Time:%e"

NUM_OF_TRIALS=6

LOG_FILE="benchmark.log"

LOG_DIR="$BASE_DIR/logs"

# hadoop
HADOOP_CMD="$HADOOP_HOME/bin/hadoop"

# hive
HIVE_CMD="$HIVE_HOME/bin/hive"

# hive tpch queries
# hive all benchmark queries
HIVE_TPCH_QUERIES_ALL=( \
        "tpch/q1_pricing_summary_report.hive" \
        "tpch/q2_minimum_cost_supplier.hive" \
        "tpch/q3_shipping_priority.hive" \
        "tpch/q4_order_priority.hive" \
        "tpch/q5_local_supplier_volume.hive" \
        "tpch/q6_forecast_revenue_change.hive" \
        "tpch/q7_volume_shipping.hive" \
        "tpch/q8_national_market_share.hive" \
        "tpch/q9_product_type_profit.hive" \
        "tpch/q10_returned_item.hive" \
        "tpch/q11_important_stock.hive" \
        "tpch/q12_shipping.hive" \
        "tpch/q13_customer_distribution.hive" \
        "tpch/q14_promotion_effect.hive" \
        "tpch/q15_top_supplier.hive" \
        "tpch/q16_parts_supplier_relationship.hive" \
        "tpch/q17_small_quantity_order_revenue.hive" \
        "tpch/q18_large_volume_customer.hive" \
        "tpch/q19_discounted_revenue.hive" \
        "tpch/q20_potential_part_promotion.hive" \
        "tpch/q21_suppliers_who_kept_orders_waiting.hive" \
        "tpch/q22_global_sales_opportunity.hive" \
)
```

Figura 36 – Arquivo benchmark.conf
Fonte: (SRIDHARAN, 2014)

– **Executando as consultas na base de dados:** Após todas as configurações definidas, as consultas podem ser executadas através do arquivo **tpch_benchmark.sh** com o comando:

  * **/<caminho-até-a-pasta-do-hive-600/tpch_benchmark.sh**

A partir daí as consultas serão submetidas e o tempo de resposta de cada uma delas irá aparecer na tela, conforme a Figura 37:

```
*********************************************
*           PC-H benchmark on Hive          *
*********************************************

Running Hive from
Running Hadoop from /usr/lib/hadoop
See benchmark.log for more details of query errors.

Executing Trial #1 of 1 trial(s)...
Running Hive query: tpch/q1_pricing_summary_report.hive
Time:39.02
Running Hive query: tpch/q2_minimum_cost_supplier.hive
Time:133.48
Running Hive query: tpch/q3_shipping_priority.hive
Time:96.27
Running Hive query: tpch/q4_order_priority.hive
Time:67.66
Running Hive query: tpch/q5_local_supplier_volume.hive
Time:115.78
Running Hive query: tpch/q6_forecast_revenue_change.hive
Time:26.20
Running Hive query: tpch/q7_volume_shipping.hive
Time:153.21
Running Hive query: tpch/q8_national_market_share.hive
Time:161.01
Running Hive query: tpch/q9_product_type_profit.hive
Time:154.61
```

Figura 37 – Execução dos testes e apresentação dos resultados
Fonte: (SRIDHARAN, 2014)