Universidade do Minho
Escola de Engenharia

João Filipe Freitas Martins

Apache Kudu: Vantagens e Desvantagens na Análise de Vastas Quantidades de Dados

Dissertação de Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação da

Professora Doutora Maribel Yasmina Santos

Outubro de 2018

"You're not living life in you comfort zone,

You're just wasting valuable time"

Benjamin Sahba

Resumo

Durante os últimos anos, temos assistido a um aumento exponencial da quantidade de dados produzidos. Este aumento deve-se, principalmente, à enorme utilização de sensores, assim como à massificação da utilização das redes sociais e de dispositivos móveis que, em permanência, recolhem dados de diversos tipos e contextos. O tratamento e análise destes dados por parte das organizações traduz-se numa inegável vantagem competitiva nos mercados, cada vez mais exigentes. Por este motivo, o estudo e desenvolvimento de novas ferramentas para a exploração destes dados tem atraído a atenção das organizações e também da comunidade científica, uma vez que as técnicas e tecnologia tradicionais se têm mostrado incapazes de lidar com dados de tal natureza. Neste contexto, surge o termo *Big Data*, utilizado para definir este tipo de dados de grande volume, diferentes graus de complexidade e, por vezes, não estruturados ou com um modelo de dados pré-definido. Associado ao termo *Big Data* surgem novos repositórios de dados com modelos lógicos próprios, denominados de bases de dados *NoSQL*, que vêm substituir as bases de dados relacionais baseadas no paradigma relacional. Estes repositórios estão integrados num ecossistema vasto de ferramentas e tecnologias para lidar com este tipo de dados, o *Hadoop*. Neste seguimento, esta dissertação tem por objetivo estudar uma das muitas ferramentas associadas ao projeto *Hadoop*, o *Kudu*. Esta nova ferramenta, de arquitetura híbrida, promete preencher a lacuna existente entre as ferramentas de acesso a dados de forma sequencial e as ferramentas de acesso a dados de forma aleatória, simplificando, por isso, a arquitetura complexa que a utilização destes dois tipos de sistemas implica. Para cumprir os objetivos da dissertação foram realizados testes de desempenho com dois modelos de dados distintos, ao *Kudu* e a outras ferramentas destacadas na literatura, para possibilitar a comparação de resultados.

Palavras-Chave: *Big Data*, *NoSQL*, *Hadoop*, *Kudu*

ABSTRACT

Over the last few years we have witnessed an exponential increase in the amount of data produced. This increase is mainly due to the huge use of sensors, as well as the mass use of social networks and mobile devices that continuously collect data of different types and contexts. The processing and analysis of these data by the organizations translates into an undeniable competitive advantage in the increasingly competitive markets. For this reason, the study and development of new tools for the exploration of these data has attracted the attention of organizations and scientific community, since traditional techniques and technology have been unable to deal with data of this nature. In this context, the term Big Data appears, used to define this type of data of great volume, different degrees of complexity, and sometimes unstructured or disorganized. Associated with the term Big Data arise new data repositories with own logical models, denominated of databases NoSQL, that replace the traditional models. These repositories are integrated into a vast ecosystem of tools and technologies to handle this type of data, Hadoop. In this follow-up, this dissertation aims to study one of the many tools associated with the Hadoop project, Kudu. This new hybrid architecture tool promises to fill the gap between sequential data access tools and random data access tools, thereby simplifying the complex architecture that the use of these two types of systems implies. To fulfill the objectives of the dissertation, performance tests were performed with two different data models, over Kudu and other tools highlighted in the literature, to allow the comparison of results.

Keywords: Big Data, NoSQL, Hadoop, Kudu

ÍNDICE

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas, Siglas e Acrónimos

Este documento utiliza um conjunto de abreviaturas, siglas e acrónimos listados de seguida:

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

BASE - Basic Availability, Soft State, Eventual Consistency

BDW - Big Data Warehouse

CAP - Consistency, Availability, Partition tolerance

CPU - Central Processing Unit

DSRM - Design Science Research Methodology

DW - Data Warehouse

ETL - Extract Transform Load

GFS - Google File System

HDD - Hard Disk Drive

HDFS - Hadoop Distributed File System

JSON - JavaScript Object Notation

NoSQL - Not Only Structured Query Language

NTFS - New Technology File System

OLAP - Online Analytical Processing

ORC - Optimized Row Columnar

RAM - Random Access Memory

SF - Scale Factor

SGBD - Sistema de Gestão de Bases de Dados

SQL - Structured Query Language

SSB - Star Schema Benchmark

1. INTRODUÇÃO

Neste capítulo são apresentados o enquadramento e a motivação que levaram à realização desta dissertação, bem como os objetivos, a abordagem metodológica, o plano de trabalho e os contributos do mesmo. Além disso, é ainda descrita a organização do presente documento.

1.1 Enquadramento e Motivação

Os últimos anos marcaram um momento decisivo na história dos dados, com o surgimento dos motores de pesquisa, redes sociais, *tablets* e *smartphones*. Todas estas fontes de dados contribuíram para uma massificação dos dados, nas perspetivas de criação, aquisição e consumo (Krishnan, 2013).

Esta massificação levou diversas organizações a investirem em soluções tecnológicas para gestão de dados, conduzindo, também, ao aumento da popularidade do termo *Big Data* definido por Krishnan (2013) como "volumes de dados disponíveis em vários níveis de complexidade, gerados a diferentes velocidades e com vários níveis de ambiguidade que não podem ser processados através da utilização de tecnologias, métodos de processamento ou algoritmos tradicionais, ou por qualquer solução comercial já disponível".

Os custos associados à análise destes vastos volumes de dados tornam essa análise quase proibitiva. Tendo em consideração diversas soluções possíveis, várias pessoas e organizações tentaram resolver este problema. Atualmente, a solução mais popular é um projeto *open source* chamado *Hadoop* (Zikopoulos, Eaton, DeRoos, Deutsch, & Lapis, 2011).

Hadoop é um projeto da *Apache Software Foundation* inspirado no projeto *Google (distributed) File System* (GFS) e no paradigma de programação *MapReduce*. Este pode ser definido como "um ambiente de computação construído em cima de um sistema de ficheiros agrupados e distribuídos que foi projetado especificamente para operações de dados em grande escala" (Zikopoulos et al., 2011).

Associado ao ecossistema *Hadoop* estão diversos projetos, nomeadamente, o *Kudu*, um novo sistema de armazenamento projetado para preencher a lacuna existente entre sistemas de armazenamento de acesso sequencial de alto *throughput*, como o *Hadoop Distributed File System* (HDFS), e sistemas de acesso aleatório e de baixa latência, como o *HBase* ou o *Cassandra* (Cloudera, 2016).

Assumindo estes conceitos e tecnologias, a motivação inerente ao presente projeto de investigação é compreender os cenários e contextos em que o *Kudu* pode ser usado, destacando as vantagens e desvantagens identificadas em cada um dos cenários testados.

1.2 Objetivos

Esta dissertação tem como objetivo investigar se o *Kudu* permite preencher a lacuna existentes entre sistemas de armazenamento de acesso sequencial de alto *throughput* e sistemas de acesso aleatório de baixa latência, mesmo em *commodity hardware*, usando como termo de comparação uma arquitetura que usa uma base de dados *NoSQL* e o HDFS (com SQL-*on*-*Hadoop*) para responder a consultas sobre os dados.

Esta investigação propõe-se cumprir os seguintes objetivos:

- Compreender a tecnologia e os principais contextos de utilização;
- Avaliar o seu desempenho utilizando um *benchmark* de referência;
- Caracterizar as vantagens e desvantagens do *Kudu* na análise de vastas quantidades de dados que sofrem diversas alterações ao longo do tempo.

1.3 Abordagem Metodológica

Tendo em conta o enquadramento e objetivos da dissertação, e estando a mesma inserida na área de Sistemas de Informação, a metodologia a ser utilizada durante a investigação será a "*Design Science Research Methodology* (DSRM) *for Information Systems*" de (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). Esta metodologia divide um projeto de investigação em seis fases sequenciais, admitindo a realização de diversas iterações com objetivo de melhorar os resultados obtidos. (Figura 1)

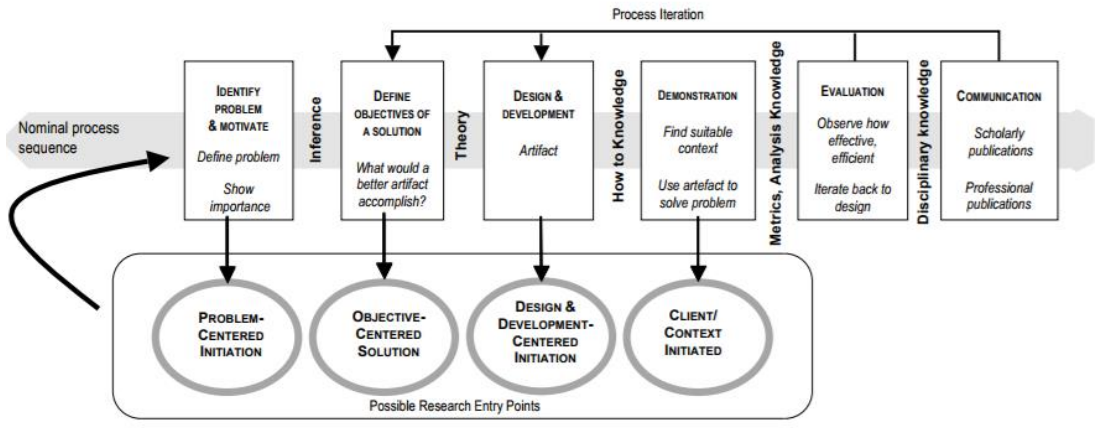

Figura 1 - *Design Science Research Methodology for Information Systems*. Retirado de (Peffers et al., 2007).

O modelo considera quatro possíveis pontos de partida de um projeto de investigação (Figura 1). O ponto de partida desta dissertação será o primeiro, ou seja, início centrado no problema, sendo, por isso, a fase inicial dedicada à identificação do problema e da motivação inerente ao mesmo.

1.3.1 Fases da Metodologia

Tendo por base a metodologia a utilizar e as imposições associadas à realização da dissertação, foram identificadas as seguintes tarefas a realizar:

1. Plano de Trabalho – elaboração do presente documento, que se destina a dar uma imagem global da dissertação, com a descrição e identificação do enquadramento, motivação, principais objetivos, abordagem metodológica, tarefas a realizar e calendarização das mesmas;
2. Identificação do Problema e Motivação – definição do problema da investigação e identificação da motivação associada à sua resolução, descrevendo e justificando o valor associado à solução encontrada;
3. Definição dos Objetivos da Solução – identificação dos objetivos a atingir, tendo em consideração o problema e as possíveis limitações da investigação;
4. Revisão de Literatura – identificação, seleção, análise e compreensão da literatura pertinente para o tema da dissertação;
5. Enquadramento Tecnológico – descrição das tecnologias a utilizar durante a realização da dissertação;
6. Conceção e Desenvolvimento – criação do artefacto que deve conter modelos e métodos, assim como instanciações desses modelos e métodos. Neste caso em particular, será dado particular ênfase à criação e definição das regras a seguir no processo de avaliação da ferramenta Apache *Kudu*;
7. Demostração – aplicação das regras definidas no artefacto produzido na etapa anterior num caso de demostração;
8. Avaliação – medição dos resultados obtidos na demostração, análise e validação das regras criadas. Esta fase será executada com a utilização de métricas de *performance* e comparação de resultados, podendo resultar em alterações às regras e realização de novas avaliações;
9. Comunicação – Escrita e apresentação da dissertação, sendo também possível a publicações de artigos relacionados em revistas/conferências da área.

1.3.2 Processo de Revisão de Literatura

O processo de recolha da literatura relevante é essencial na manutenção de um método de trabalho consistente, pelo que, numa fase inicial, definiram-se as palavras-chave, as bases de dados de referência e o método de seleção dos artigos.

No que diz respeito ao enquadramento conceptual houve várias limitações, desde a sua contextualização, uma vez que a literatura revista se limita apenas ao conjunto de artigos pertinentes disponíveis para a comunidade científica da Universidade do Minho, mas também limitações a nível temporal, tendo sido selecionados artigos com data igual ou superior a 2010, salvo raras exceções referentes a autores ou artigos relevantes na área.

A identificação e seleção da literatura foram realizadas no período compreendido entre Setembro de 2017 e Fevereiro de 2018, utilizando as seguintes palavras-chave: Big *Data*, *NoSQL*, *Hadoop* e *Kudu*, individualmente ou combinadas entre si, nas bases de dados de referência que se seguem: *Google Scholar*; *Scopus*; *IEEE Xplore*; *ACM Digital Library*. Após este intervalo, a procura de novas referências bibliográficas relevantes foi restrita a conceitos, técnicas ou tecnologias particulares, com o objetivo de complementar aspetos concretos no seguimento do tema que estava a ser desenvolvido.

Foram também utilizados artigos fornecidos pela orientadora da dissertação, assim como, os sites das tecnologias e alguns artigos referenciados em outros artigos.

1.4 Organização do Documento

O documento encontra-se dividido em seis capítulos. Neste primeiro capítulo são definidos o enquadramento e a motivação do tema da dissertação, bem como os seus objetivos e a abordagem metodológica a utilizar.

O segundo capítulo apresenta a revisão de literatura. Nele é explorado o conceito *Big Data*, assim como as suas vantagens competitivas, os seus desafios e problemas. São também descritos os conceitos *NoSQL* e *MapReduce* e identificadas as diferenças entre o processamento tradicional de dados e o processamento de dados *Big Data*.

No terceiro capítulo são descritas as principais tecnologias associadas ao termo *Big Data*, nomeadamente, o ecossistema *Hadoop*, e as suas principais ferramentas com especial destaque para as ferramentas HDFS, *Yarn, Hive*, *Presto*, *Impala*, ORC, *Parquet* e *Kudu*.

O quarto capítulo descreve a infraestrutura utilizada, assim como, os conjuntos de dados e os cenários de teste em que estes são utilizados.

De seguida, no quinto capítulo, são apresentados os resultados obtidos na realização dos testes, as considerações relevantes e as análises consideradas pertinentes.

De seguida, expõe-se a conclusão da dissertação, são discriminadas as referências bibliográficas que foram utilizadas como suporte à redação do presente documento e, por último, são disponibilizados os apêndices da dissertação.

Esta página foi deixada intencionalmente em branco.

2. ENQUADRAMENTO CONCEPTUAL

Neste capítulo, são apresentados os principais conceitos associados ao termo *Big Data* e descritos os seus principais paradigmas de modo a facilitar a leitura e compreensão da restante dissertação.

2.1 *Big Data*

Segundo Krishnan (2013), *Big Data* é "o fenómeno que mais tem capturado a atenção da indústria da computação moderna desde a *Internet*". Nos últimos anos, o termo tem sido amplamente publicado e discutido entre a comunidade científica e, para isso, muito têm contribuído os avanços na recolha e armazenamento de dados.

Entre as diversas opiniões dos autores evidencia-se a ideia de que *Big Data* descreve um problema de dados que é irresolúvel usando ferramentas tradicionais (DeRoos, Zikopoulos, Melnyk, Brown, & Coss, 2014). Tratando-se de um conceito abstrato (Chen, Mao, & Liu, 2014) a sua definição continua ambígua. "Nós guardamos tudo: dados ambientais, financeiros, médicos, de vigilância e a lista continua" (Zikopoulos et al., 2011), mas não é só a quantidade de dados que define *Big Data*.

2.1.1 Características do *Big Data*

O modelo dos 3V's, uma das definições mais populares, foi apresentado em 2001 pelo grupo META (atual Gartner) com o objetivo de descrever os desafios e as oportunidades que surgem pelo aumento massivo de dados. O conceito, que ainda hoje é utilizado, foi popularizado uma década mais tarde pela IBM (Chen et al., 2014).

Este modelo baseia-se em três características principais (Chen et al., 2014):

- Volume – quantidade de dados de diferentes tipos e tamanhos gerados continuamente pelas mais variadas fontes;
- Velocidade – fluxo contínuo de dados cujo objetivo é a obtenção de um conjunto de resultados esperados a partir da análise e processamento dos mesmos, rápida e atempadamente de forma a servir o seu propósito;
- Variedade – vários formatos possíveis que incluem dados semi-estruturados e não estruturados, como áudio, vídeo, páginas web, texto, entre outros, bem como dados estruturados tradicionais.

A partir da contextualização anterior, é possível perceber o porquê da intensa complexidade associada ao processamento de *Big Data*, pelo que Krishnan (2013) admite a existência de

características adicionais resultantes da interseção entre volume, velocidade e variedade, tal como se pode observar na Figura 2.

- Ambiguidade – os dados podem assumir diversos significados dependendo do contexto em que se inserem, pelo que esta propriedade resulta da indisponibilidade de metadados, que definem as propriedades dos dados;
- Viscosidade – traduz a dificuldade de integração e análise de vastos volumes de dados, característica que é potenciada pelo constante aumento do volume de dados gerado a velocidade elevada;
- Viralidade – está associada à rapidez na partilha da informação, característica que é potenciada pela variedade de dados e velocidade a que são gerados.

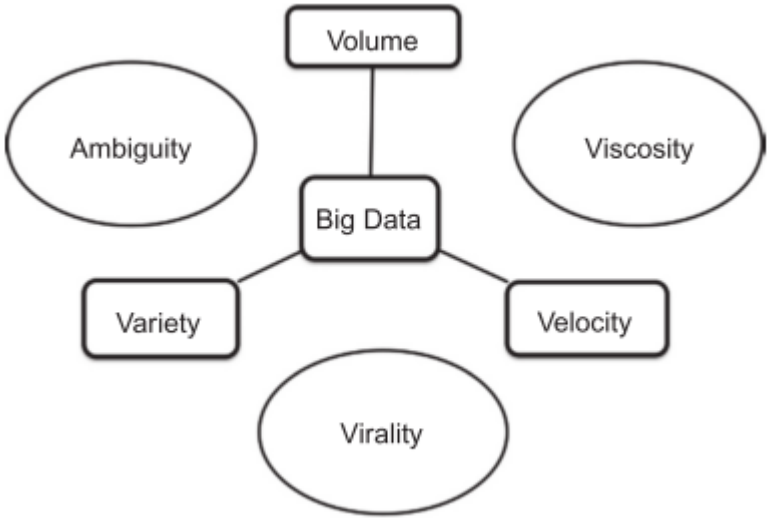

Figura 2 - Características adicionais resultantes da interseção entre volume, velocidade e variedade. Retirado de (Krishnan, 2013).

Impõe-se, assim, a necessidade de compreender, analisar, limpar e processar vastos quantidades de dados de forma a podermos utilizar a informação para fins específicos. A importância da informação extraída, isto é, os resultados esperados do processamento dos dados, pode assim ser entendida como o valor dos dados (Khan, Uddin, & Gupta, 2014).

O valor dos dados é considerado por vários autores como a característica fundamental para a compreensão do potencial do *Big Data* na resolução dos problemas do mundo real (Khan et al., 2014).

Apesar do consenso entre os autores no que diz respeito ao valor como característica relevante do *Big Data*, as opiniões divergem quando o assunto são as restantes características que levam ao valor. De seguida apresenta-se um sumário das características encontradas na literatura.

- Veracidade – precisão e relevância dos dados. Os dados perdem valor e significância se não forem precisos, isto é, imprecisão pode levar a decisões erradas (Patgiri & Ahmed, 2017);
- Validade - pode parecer semelhante à veracidade dos dados, no entanto, são conceitos diferentes. Os dados podem não ter problemas de veracidade, mas ser inválidos se não forem devidamente compreendidos. O mesmo conjunto de dados pode ser válido ou invalido dependendo do momento em que são analisados (Khan et al., 2014);
- Visualização – processo para tornar visíveis os dados ocultos do *Big Data*, pois não faz sentido armazenar dados se estes não puderem ser vistos ou mostrados (Patgiri & Ahmed, 2017);
- Variabilidade – refere-se à mudança, mutação ou modificação natural dos dados, isto é, os dados podem mudar de acordo com o tempo, o espaço e a utilização ou, então, tornarem-se obsoletos (Patgiri & Ahmed, 2017);
- Complexidade – refere-se ao facto de que *Big Data* é gerado através de imensuráveis fontes de dados. Isto leva a um desafio crítico: conectar, combinar, limpar e transformar dados recebidos de diferentes fontes (Gandomi & Haider, 2015).

2.1.2 A Vantagem Competitiva do *Big Data*

Hoje em dia, em quase todos os ramos da indústria, as empresas estão focadas em explorar os dados de modo a obter vantagens competitivas, isto é, de modo a extrair valor dos dados. Em contrapartida, o volume e as restantes características dos dados ultrapassaram a capacidade das bases de dados tradicionais e da análise manual (Charles & Gherman, 2013). Neste seguimento, surge *Big Data* que permite, principalmente, eficazes análises de dados, agregando dados de fontes internas e externas, de modo a identificar relacionamentos até aqui escondidos e que de outra forma não seriam encontrados. Destas potencialidades advêm diversos benefícios, entre os quais (Barham, 2015):

- Aumentar a probabilidade de os gestores tomarem a decisão correta no momento certo – por exemplo, no setor da aviação, uma companhia aérea pode utilizar *Big Data* para analisar os dados históricos de vendas, a situação económica atual, as previsões meteorológicas e o *feedback* das redes socias, de modo a prever os destinos com mais passageiros num futuro próximo e, portanto, executar mais viagens ou alocar aviões com mais lugares para esses destinos. Além disso, *Big Data* pode permitir ao consumidor tomar melhores decisões, por exemplo, um paciente pode comparar a eficácia de um

medicamento alternativo para tratar a sua doença, não apenas em geral, mas também com pacientes com idade, raça e género semelhantes;

- Fazer uma melhor gestão de risco – ao executar análises de cenários preditivos, por exemplo, uma empresa petrolífera pode usar *Big Data* para gerar cenários futuros dependendo da taxa de produção atual, procura de mercado, tendência histórica do preço e outros fatores, para prever o preço futuro do petróleo e, assim, fazer uma melhor orçamentação e previsão financeira;
- Identificar com mais precisão nichos de mercado – o que permitirá responder de forma mais rápida a essa necessidade do mercado. Por exemplo, *Big Data* pode revelar que os idosos de uma determinada cidade passam mais tempo no quintal que o habitual e que, por consequência, plantam mais vegetais em vez de flores ao contrário do que acontece nas outras cidades e, portanto, redirecionar uma empresa para a venda destes produtos nesta cidade ao invés de flores;
- Melhorar o acompanhamento e monitorização dos produtos – *Big Data* pode, de imediato, informar as empresas sobre defeitos e novos usos dos seus produtos assim que estejam no mercado. Por exemplo, uma empresa farmacêutica pode utilizar Big Data para analisar os comentários sobre um medicamento recentemente lançado e descobrir novos efeitos colaterais, bons ou maus, que não eram conhecidos nos testes pré-lançamento;
- Detetar fraudes e atuar imediatamente - por exemplo, os bancos podem usar *Big Data* para detetar transações anormais que podem ser o resultado de contas bancárias roubadas ou analisar melhor os riscos de um empréstimo e, portanto, tomar melhores decisões;
- Melhorar a eficiência dos processos produtivos – *Big Data* pode identificar exatamente onde estão a ser causados desperdícios, onde estão a ocorrer atrasos ou onde estão os maiores custos do processo produtivo. Por exemplo, uma grande empresa com fábricas espalhadas pelo mundo pode saber em tempo real quais as matérias-primas mais caras de cada fábrica, devido à sua localização, e ajustar assim a produção.

2.1.3 Desafios e Problemas

Embora os benefícios do *Big Data* sejam factuais e substanciais, existe um conjunto de desafios que devem ser assimilados de modo a compreender completamente o potencial do *Big Data*. Alguns destes desafios são causados pelas características do *Big Data*, uns pelos seus métodos e modelos de

análise, e outros pelas limitações dos atuais sistemas de processamento de dados (Jin, Wah, Cheng, & Wang, 2015).

Na literatura destaca-se a atenção dada pelos autores a desafios como: entender a real noção e conceito do *Big Data*, problemas de privacidade e considerações éticas associadas à exploração dos dados. Além disso, classificar os dados de modo a construir informação relevante, requer, muitas vezes, a intervenção humana. Apesar de as tecnologias de *Big Data* continuarem em constante evolução, o conhecimento humano e a experiência dos líderes empresarias necessários para tirar partido do *Big Data* não têm acompanhado esta tendência evolutiva, o que se traduz num outro grande desafio (Sivarajah, Kamal, Irani, & Weerakkody, 2016).

Segundo Zicari (2014), os desafios do *Big Data* podem ser agrupados em três categorias principais, como representado na Figura 3, com base no ciclo de vida dos dados:

- Desafios dos dados - relacionados com as suas características, já exploradas neste documento;
- Desafios dos processos - relacionados com as técnicas, isto é, como recolher os dados, como os integrar, como os transformar, como selecionar o modelo de análise correto ou como apresentar os resultados;
- Desafios de gestão - relacionados com a privacidade, segurança e administração dos dados e os aspetos éticos.

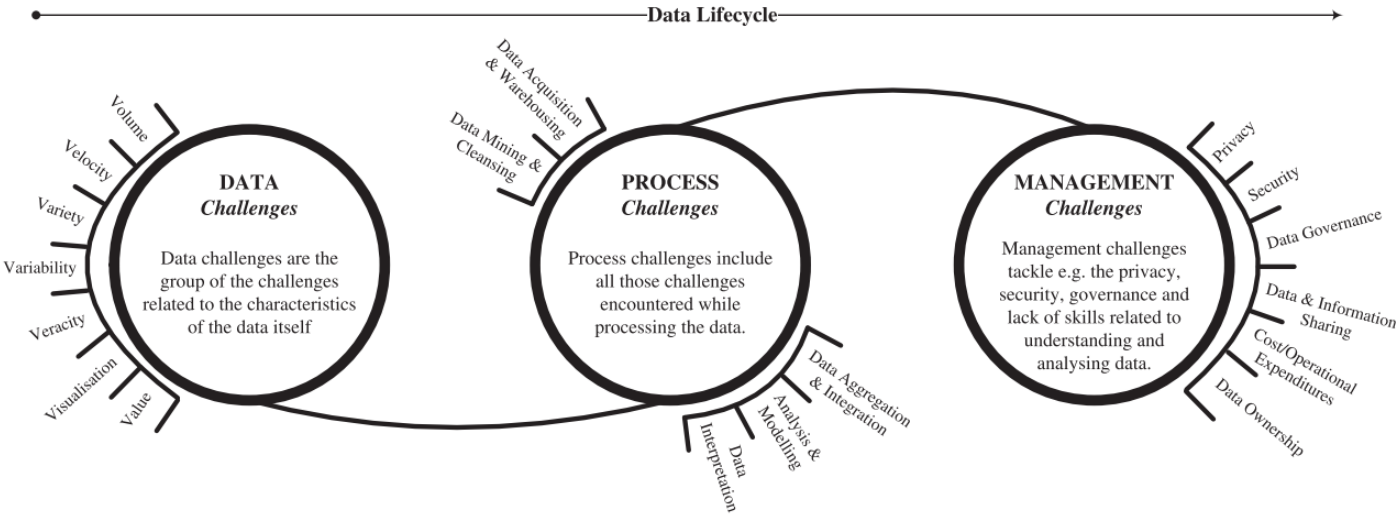

Figura 3 - Classificação conceptual dos desafios do *Big Data*. Retirado de (Sivarajah et al., 2016) .

2.1.4 Processamento de Dados

Segundo Krishnan (2013), o processamento tradicional de dados é definido como a recolha, o processamento e a gestão de dados, resultando na extração de informação para o utilizador final, como representado na Figura 4.

Devido à natureza estruturada e ao volume reduzido ou moderado dos dados, as tarefas de gestão como *data quality* e *data cleansing* têm pouca relevância neste processo, sendo, normalmente, executadas no sistema de origem dos dados.

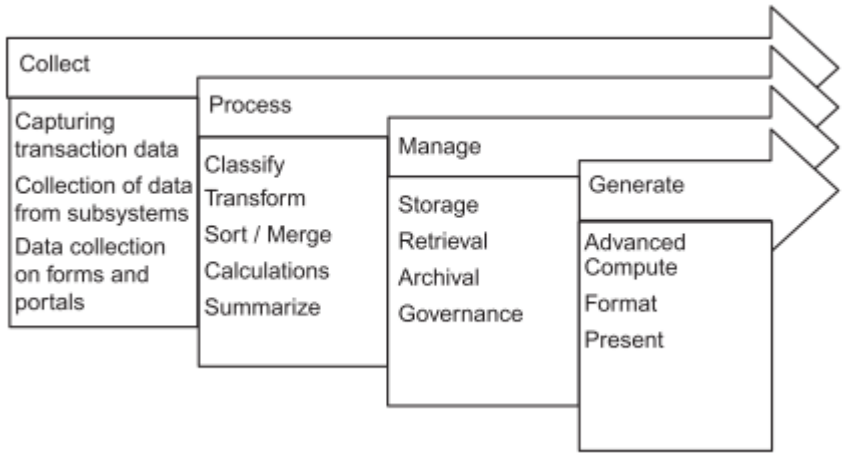

Figura 4 - Processamento tradicional de dados. Retirado de (Krishnan, 2013).

No processamento tradicional de dados, os dados são primeiramente analisados, sendo definido um conjunto de requisitos que leva à criação de um modelo de dados e, por sua vez, à criação de uma estrutura de base de dados. No entanto, o processamento de *Big Data* (Figura 5) segue um ciclo amplamente diferente, uma vez que os dados são inicialmente recolhidos e carregados na plataforma de destino, sendo posteriormente aplicada a camada de metadados, definida a estrutura dos dados e, por fim, os dados são transformados e analisados (Krishnan, 2013).

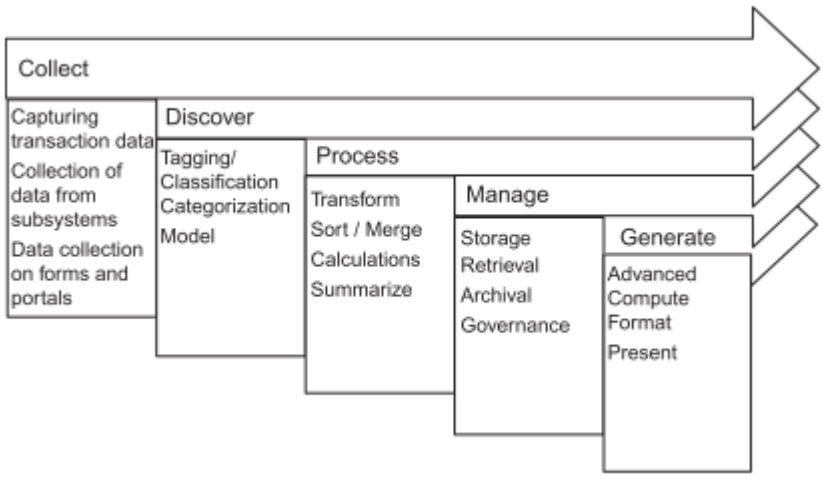

Figura 5 - Processamento de *Big Data*. Retirado de (Krishnan, 2013).

Devido ao seu volume e complexidade, não é aconselhável a utilização de uma estrutura de base de dados tradicional. Deste modo, Krishnan (2013) sugere a utilização de uma arquitetura baseada em ficheiros em detrimento da estrutura tradicional.

Krishnan (2013) realça, também, a necessidade de compreender o fluxo do processamento de *Big Data* (Figura 6), de modo a definir uma infraestrutura e uma arquitetura de processamento eficientes. Este fluxo está dividido em 4 fases, descritas a seguir:

- Recolha – os dados são recebidos de diferentes fontes e carregados para um sistema de ficheiros aqui denominado de *landing zone*. Normalmente, os dados são distribuídos em subdiretorias de acordo com o seu tipo. Alterações aos ficheiros, como nome ou extensão, devem ser realizadas nesta fase.
- Carregamento – os dados são carregados com a aplicação dos metadados e preparados para a transformação. Este processo divide os grandes ficheiros iniciais em pequenos conjuntos de dados e é criado um catálogo de ficheiros. Nesta fase, os dados podem, também, ser divididos horizontalmente ou verticalmente, dependendo dos requisitos do utilizador ou de processamento.
- Transformação – os dados são transformados pela aplicação das regras de negócio. Esta fase tem diversos passos o que pode resultar numa difícil e complexa gestão. As etapas de processamento produzem resultados intermédios que podem ser armazenados para análise posterior. Tipicamente, os resultados desta fase são um conjunto de metadados e as respetivas métricas.

- Extração – os dados resultantes podem ser extraídos para posterior processamento, onde se incluem: análise, relatórios operacionais, integração em *data warehouse* e visualização.

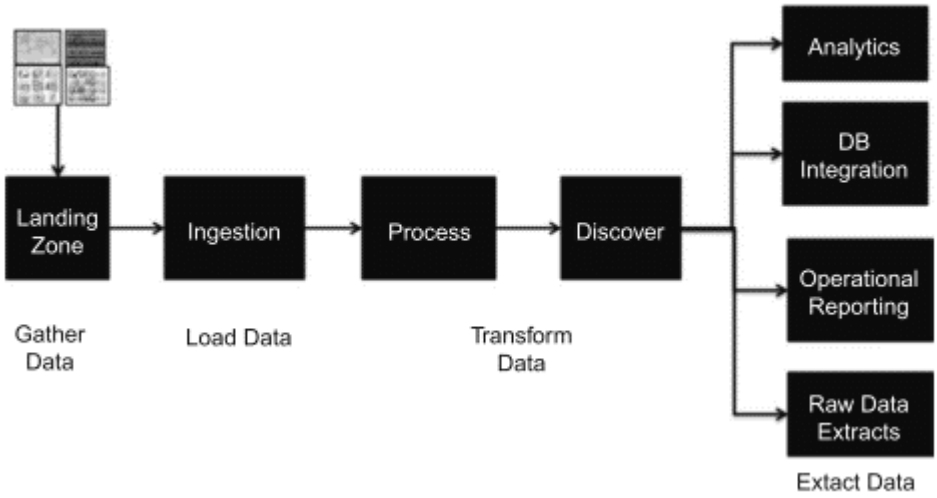

Figura 6 - Fluxo de processamento de *Big Data*. Retirado de (Krishnan, 2013).

2.2 *Big Data Warehouse*

Data Warehouses (DWs) são há muito reconhecidos como um ativo empresarial fundamental, fornecendo suporte a decisões baseadas em factos para as organizações. Tradicionalmente, os sistemas de gestão de bases de dados (SGBDs) são usados para armazenar dados históricos, fornecendo diferentes perspetivas analíticas sobre vários processos de negócios (Costa & Santos, 2017b).

Com os atuais avanços nas técnicas e tecnologias de *Big Data*, o conceito de *Big Data Warehouse* (BDW) surge para superar várias limitações dos DWs tradicionais. Apesar do *Big Data* ser um tópico científico e técnico relativamente recente já existem alguns esforços de padronização de construção e componentes lógicos para os sistemas de *Big Data* em geral. No entanto, o conceito de *Big Data Warehousing* é ainda mais recente, com ambiguidades ainda mais preocupantes e falta de abordagens padronizadas. O BDW pode ser definido pelas suas caraterísticas (Costa & Santos, 2018):

- Processamento massivo em paralelo;
- Tarefas analíticas complexas e mistas (por exemplo, consultas *ad hoc*, *data mining*, *text mining*, análise exploratória, vistas materializadas)
- Armazenamento flexível para suportar dados de várias fontes;
- Operações em tempo real (processamento de *streams* de dados, atualizações de baixa latência e alta frequência)

- Alto desempenho com respostas quase em tempo real;
- Escalabilidade para alocar o aumento de dados, utilizadores e análises;
- Uso de *commodity hardware* para reduzir custos;
- Interoperabilidade num conjunto de múltiplas tenologias.

O BDW representa uma mudança de paradigma para as organizações que enfrentam vários desafios com as plataformas tradicionais de *Data Warehousing* nomeadamente congestionamentos na recolha, armazenamento, processamento e análises de dados, devido à grande exigência de entradas/saídas eficientes, escalabilidade e elasticidade, que podem ser superados através de armazenamento e processamento distribuídos. A restrita modelação dos DWs tradicionais também é um fator que levanta algumas preocupações o que só destaca ainda mais a relevância dos BDWs. No entanto, o estado de arte em BDW reflete o estado embrionário do conceito, bem como a ambiguidade e a falta de abordagens comuns para a construção de BDWs de acordo com as suas características (Costa & Santos, 2018).

2.3 Bases de Dados *NoSQL*

Com o contínuo desenvolvimento da Internet e da computação na *cloud,* têm emergido diversas aplicações que exigem cada vez mais das tecnologias de base de dados, sobretudo, no que diz respeito a: alta concorrência de leitura e escrita com baixa latência, armazenamento eficiente, alta escalabilidade e disponibilidade, bem como baixos custos operacionais e de gestão.

Apesar de ocuparem uma posição predominante no mercado, as bases de dados relacionais possuem algumas limitações quando enfrentam as exigências referidas, entre as quais: leitura e escrita lenta, capacidade de armazenamento limitada e escalabilidade limitada.

Deste modo, e para colmatar estas necessidades, surgiram diversos novos tipos de bases de dados que, devido às suas características diferenciadoras, são comummente chamados de bases de dados *NoSQL* (Han, Haihong, Le, & Du, 2011). Este termo é também interpretado como a abreviatura para "*Not Only* SQL", destacando-se assim que o SQL não é o objetivo principal destes sistema (Grolinger et al., 2014).

Ao contrário das bases de dados tradicionais, que seguem a filosofia de *design* ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*), as bases de dados *NoSQL* perdem esta característica em troca de outra, BASE (*Basic Availability*, *Soft state*, *Eventual consistency*) (Tudorica & Bucur, 2011), também uma filosofia de *design*, desenvolvida por Brewer (2012). A estes dois conceitos é, normalmente, associado o teorema CAP (*Consistency*, *Availability*, *Partition tolerance*), que é da autoria do mesmo autor.

A ideia principal do teorema CAP, representado na Figura 7, é que um sistema distribuído pode ter no máximo duas das três propriedades referenciadas (Krishnan, 2013):

- Todos os dados estão disponíveis em todos os nós do sistema;
- Todos os pedidos ao sistema têm resposta;
- O sistema funciona independentemente da disponibilidade, particionamento ou perda de dados ou comunicação.

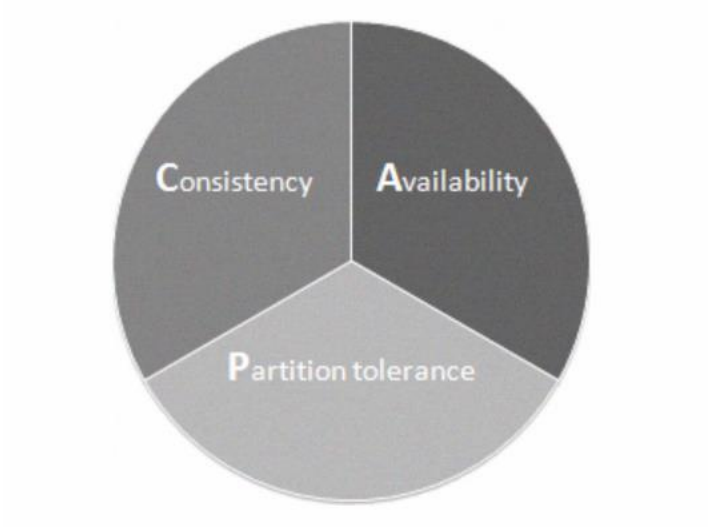

Figura 7 - Teorema CAP. Retirado de (Han et al., 2011)

Tendo por base estes princípios, e para satisfazer as necessidades já enumeradas, diversas bases de dados *NoSQL* foram desenvolvidas ao longo do tempo. Atualmente são conhecidas mais de 225 bases de dados *NoSQL* ("NOSQL Databases," 2018). Portanto, enumerá-las e avaliá-las tornou-se uma tarefa quase impossível. Tendo isto em consideração, as bases de dados *NoSQL* são, normalmente, divididas em quatro modelos de dados descritos a seguir (Costa & Santos, 2017a):

- Modelo *key-value* – os valores são, geralmente, armazenados em pares chave-valor. A chave identifica exclusivamente um valor de um tipo arbitrário. Estes modelos de dados são conhecidos por serem isentos de esquema, mas podem não ter capacidade para representar adequadamente relações ou estruturas, uma vez que consultas e indexação são asseguradas através da chave. Cada chave é única e as consultas estão diretamente ligadas com as chaves. Exemplos: *Redis*; *Memcached*; *BerkeleyDB*; *Voldemort*; *Riak*; *Dynamo*.

- Modelo *column-oriented* - o modelo de dados em coluna pode ser visto como uma extensão do modelo *key-value*, adicionando colunas e famílias de colunas, e fornecendo indexação e consulta mais poderosas devido a esta adição. Este *design* foi amplamente inspirado pelo projeto *Bigtable*, o que não significa necessariamente que todas as bases de dados orientadas por coluna sejam totalmente inspiradas por este. Exemplos: *Bigtable*; *HBase*; *Cassandra*; *Hypertable*.
- Modelo *document* - adequado para representar dados no formato de documento. A *JavaScript Object Notation* (JSON) é frequentemente usada. Este modelo pode conter estruturas complexas, como objetos aninhados, e, normalmente, também inclui índices secundários, proporcionando assim mais flexibilidade de consulta do que o modelo de dados de *key-value*. Exemplos: *MongoDB*; *CouchDB*; *Couchbase*.
- Modelo *graph* - baseado na teoria de grafos, em que os objetos podem ser representados como nós e as relações entre eles como arestas. Os grafos são apropriados no tratamento de dados interconectados com várias relações. Exemplos: Neo4j; *InfiniteGraph*; *GraphDB*; *AllegroGraph*; *HyperGraphDB*.

2.4 O Paradigma *MapReduce*

MapReduce é um paradigma de programação, que foi projetado para lidar com o processamento paralelo de grandes conjuntos de dados. Este paradigma foi proposto pela Google, em 2004, como uma abstração que permite realizar computação simples, enquanto esconde os detalhes do paralelismo, armazenamento distribuído, balanceamento de cargas e a tolerância a falhas (Inoubli, Aridhi, Mezni, & Jung, 2016). As características centrais do *MapReduce* são as suas duas funções: *Map*, que executa operações de filtragem e ordenação, e *Reduce*, que executa operações de agregação (Grolinger et al., 2014).

O principal contributo do paradigma MapReduce é a escalabilidade que permite a execução paralela e distribuída por um grande número de nós do sistema. As tarefas de *Map* e *Reduce* são divididas num grande número de processos que, posteriormente, são atribuídos aos nós da rede. Se um dos nós de rede falhar, os processos a si atribuídos são distribuídos pelos restantes nós do sistema, o que torna o paradigma tolerante a falhas (Grolinger et al., 2014).

Como representado na Figura 8, as fases do paradigma *MapReduce* são as seguintes:

- *Data reading* - nesta fase, os dados são convertidos num conjunto de pares chave-valor. Estes dados são então divididos em conjuntos mais pequenos e, por sua vez, estes novos conjuntos são atribuídos às instâncias da função *Map*;
- *Map phase* - para cada conjunto é acionada a função *Map* correspondente que produz um conjunto intermédio de pares chave-valor;
- *Combine phase* - este passo permite agrupar os pares intermédios criados na fase anterior;
- *Partitioning phase* - os resultados provenientes da *combine phase* são então distribuídos pelas funções *Reduce* correspondentes;
- *Reduce phase* - a função agrupa novamente os pares com a mesma chave e calcula o resultado final.

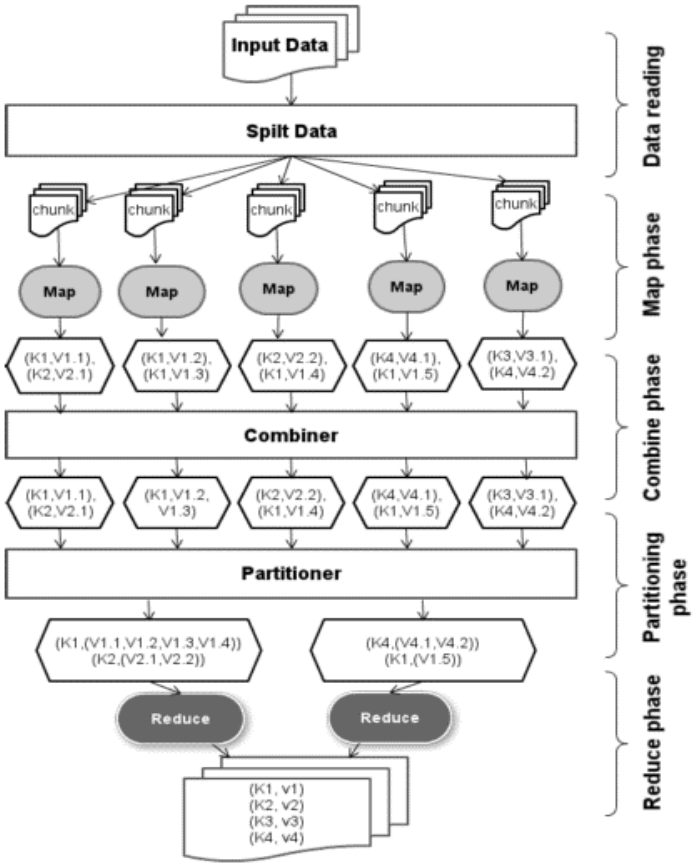

Figura 8 - Arquitetura do *MapReduce*. Retirado de (Inoubli et al., 2016).

Embora o *MapReduce* se tenha provado eficaz, ao estudar a literatura é possível encontrar diversos estudos e projetos *open-source* focados em aumentar a sua *performance*. Neste contexto, os desafios associados ao paradigma podem ser divididos em duas áreas principais:

- *Data management* – para realizar tarefas *MapReduce* é necessário definir a estrutura dos dados e fluxos de trabalho, nos quais os dados vão ser processados e geridos. Antes, os sistemas de gestão de bases de dados tradicionais eram os mais utilizados. No entanto, estes sistemas têm-se mostrado incapazes de escalar para uma quantidade considerável de dados. Para colmatar esta necessidade, um vasto conjunto de novas tecnologias de bases de dados tem surgido, como sistemas de ficheiros distribuídos e bases de dados *NoSQL*. Mesmo assim, escolher o sistema adequado para um problema específico continua a ser um desafio.
- *Scheduling* - é um aspeto importante do *MapReduce*. Durante a execução de uma tarefa, várias decisões precisam de ser tomadas. Essas decisões devem ter em conta várias informações, tais como: localização de dados, disponibilidade de recursos, entre outras. Por exemplo, durante a execução de uma tarefa *MapReduce* é necessário superar problemas como o *Map-Skew*, que se refere à má divisão da carga de trabalho atribuída a cada nó do sistema.

Esta página foi deixada intencionalmente em branco.

3. ENQUADRAMENTO TECNOLÓGICO

Neste capítulo é feita uma descrição do ecossistema Hadoop e das suas principais ferramentas dando principal destaque à ferramenta *Kudu.*

3.1 *Apache Hadoop*

O *Apache Hadoop* tem atraído uma atenção substancial por parte da indústria e da comunidade académica. Na verdade, este tem sido o principal suporte do movimento *Big Data* (Hu, Wen, Chua, & Li, 2014).

Hadoop é um projeto da *Apache Software Foundation* projetado para operar sobre dados de grande escala. Este ambiente de computação, escrito em Java, foi desenvolvido sobre um sistema de ficheiros distribuídos inspirado no *Google File System* (GFS) e no paradigma de programação *MapReduce*. Em Outubro de 2007, uma pesquisa conjunta entre a IBM e a Google estuda a possibilidade da utilização conjunta do GFS e do *MapReduce* para resolução de problemas de grande escala da Internet. No seguimento desta pesquisa, e devido ao seu sucesso, surge o Hadoop, com o objetivo de adicionar casos de uso aos da investigação original (Zikopoulos et al., 2011).

Trata-se de um projeto *open-source* que suporta o armazenamento e processamento de quantidades massivas de dados não necessitando, para isso, de hardware caro. O Hadoop permite o processamento distribuído através de hardware convencional de baixo custo.

Segundo Hu et al. (2014), entre as diversas vantagens do *Hadoop* é necessário realçar as características que o tornam particularmente adequado para a gestão e análise de grandes volumes de dados, entre as quais:

- Escalabilidade: o *Hadoop* permite que a estrutura de hardware seja reduzida ou ampliada sem necessidade de alterar o formato dos dados. O sistema redistribui automaticamente os dados e o processamento para acomodar as alterações de hardware;
- Eficiência de custo: *Hadoop* traz a computação massiva para servidores de baixo custo, o que reduz os custos de armazenamento e mantém este processamento acessível para o volume crescente do *Big Data*;
- Flexibilidade: o *Hadoop* é capaz de armazenar qualquer tipo de dados de qualquer fonte. Além disso, diferentes tipos de dados de múltiplas fontes podem ser agregados no

Hadoop para análise posterior. Assim, muitos desafios do *Big Data* podem ser abordados e resolvidos;

- Tolerância a falhas: Falhas computacionais e consequentes perdas de dados são comuns em Big *Data*. O Hadoop consegue recuperar destas falhas causadas, habitualmente, por falha de um nó ou por congestionamento da rede.

3.1.1 Ecossistema *Hadoop*

Como já referido, o *Hadoop* é um projeto *open-source* da *Apache Software Foundation* baseado no GFS e no *MapReduce*. O Hadoop contém dois componentes principais: o *Hadoop Dristributed File System* (HDFS) e o *Hadoop MapReduce* que, devido à necessidade de aumentar a sua escalabilidade, evolui para a sua versão 2.0, também denominada de *Yarn*. Estes dois componentes já foram anteriormente descritos.

Para além destes componentes, o *Hadoop* possui diversos outros projetos associados, como aparece demonstrado na Figura 9, onde estão também representadas as suas principais características (Costa & Santos, 2017a).

- *Spark* – é composto pelos módulos *SparkSQL*, *Spark Streaming*, *MLlib* e *GraphX*, que lhe permitem o processamento de dados estruturados, em *streaming* e gráficos, assim como *machine learning;*
- *Hive* – ferramenta utilizada nesta dissertação e que se encontra descrita mais à frente;
- *Avro* – é um sistema de serialização de dados;
- *Tez* – é uma estrutura extensível para a aplicações de processamento de dados em *batch* ou interativos;
- *Cassandra* – sistema de base de dados distribuída, altamente escalável e tolerante à falha;
- *Pig* – plataforma para criação de programas de alto nível de análise de dados;
- Hbase – é uma base dados distribuída e escalável para armazenamento *Big Data*;
- *Ambari* – serviço web para gestão e monotorização de *clusters Hadoop*;
- *Zookeeper* – serviço centralizado para sistemas distribuídos de coordenação de serviços;
- *Mahout* – projeto para criação de algoritmos de *machine learning* distribuídos e escaláveis;
- *Chukwa* – sistema de recolha de dados para monitorização de sistemas distribuídos.

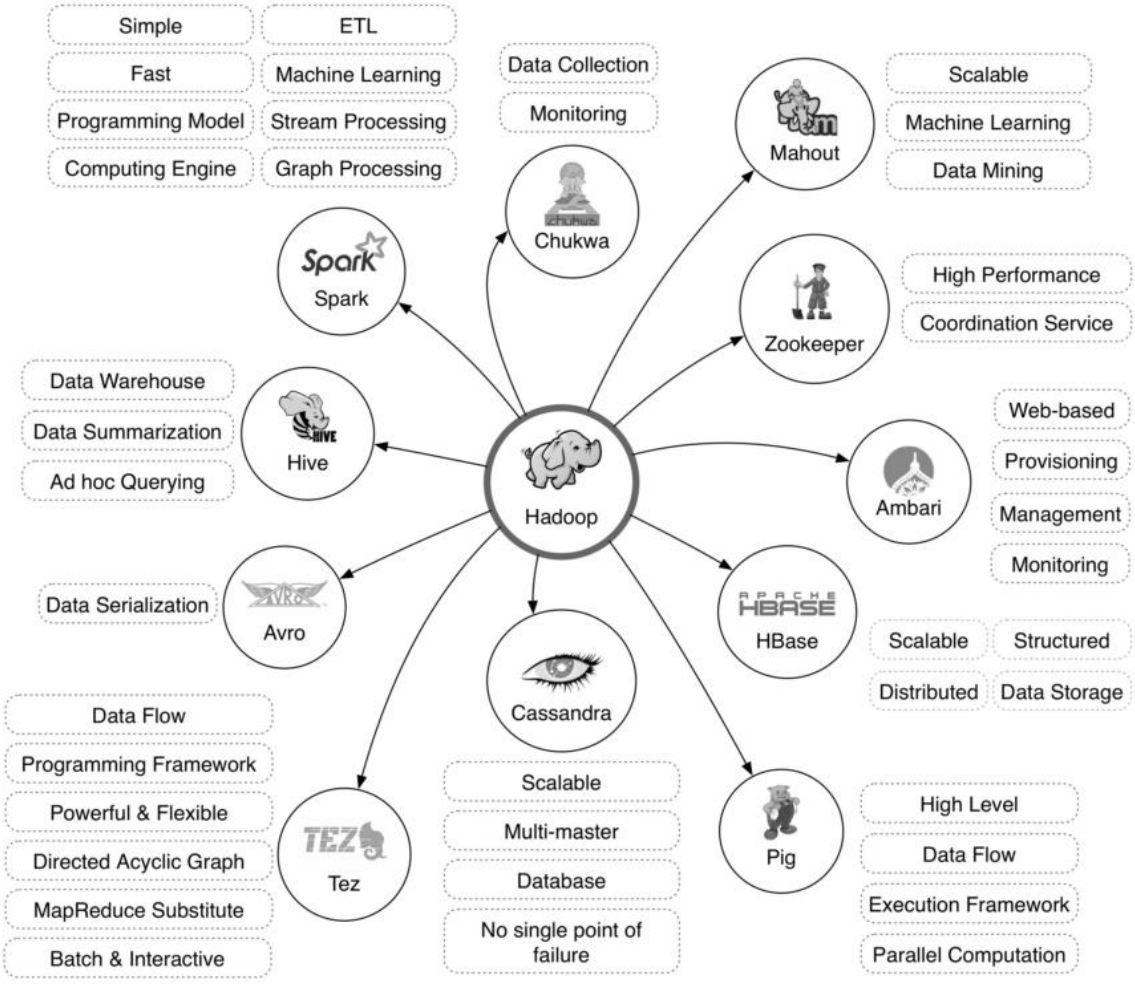

Figura 9 - Ecossistema *Apache Hadoop*. Retirado de (Costa & Santos, 2017a).

Muitos outros projetos relacionados não se encontram representados na Figura 9, entre os quais se incluem:

- *Flume* – serviço de recolha, agregação e deslocação de grandes quantidades de dados;
- *Oozie* – serviço de coordenação de *workflow's*;
- *HCatalog* – camada de metadados para os dados armazenados;
- *Sqoop* – conetor para integração de dados de outras plataformas;
- *Storm* – sistema de computação em *real-time* de alto *throughput* e baixa latência;
- *Kafka* – sistema de mensagens e consultas para enviar e receber mensagens entre processos;
- *Drill* – sistema distribuído para análise interativa de *datasets*;
- *Kerberos*, *Knox*, *Ranger* – projetos com o objetivo de assegurar a segurança do ambiente *Hadoop;*

- *Kudu* – projeto em estudo na presente dissertação que se encontra descrito mais à frente nesta dissertação.

3.1.2 HDFS

Um componente chave do *Hadoop* é o seu sistema de ficheiros HDFS (*Hadoop Distributed File System*), que é utilizado para guardar todos os dados de entrada e saída das aplicações (Sogodekar, Pendey, Tupkari, & Manekar, 2016). O HDFS fornece acesso global aos ficheiros do cluster, deste modo, para otimizar a portabilidade é implementado como um sistema de ficheiros ao nível do utilizador em Java que, por sua vez, explora o sistema de ficheiros nativo de cada nó, como ext3 ou NTFS (*New Technology File System*), para armazenar os dados. Os ficheiros do HDFS são divididos em blocos, normalmente de 64MB, e cada bloco armazenado como um ficheiro independente no sistema de ficheiros local (Shafer, Rixner, & Cox, 2010).

O HDFS é implementado por dois serviços, como representado na Figura 10: *NameNode* e *DataNode*. O *NameNode* é responsável por gerir a árvore de diretorias do HDFS e é um serviço centralizado no *cluster* que opera num único nó. Este serviço executa operações comuns do sistema de ficheiros, como abrir, fechar, alterar nome e apagar. O *NameNode* não armazena dados do HDFS, apenas mapeia os ficheiros, mantendo uma lista dos blocos e do *DataNode* onde estes se encontram armazenados.

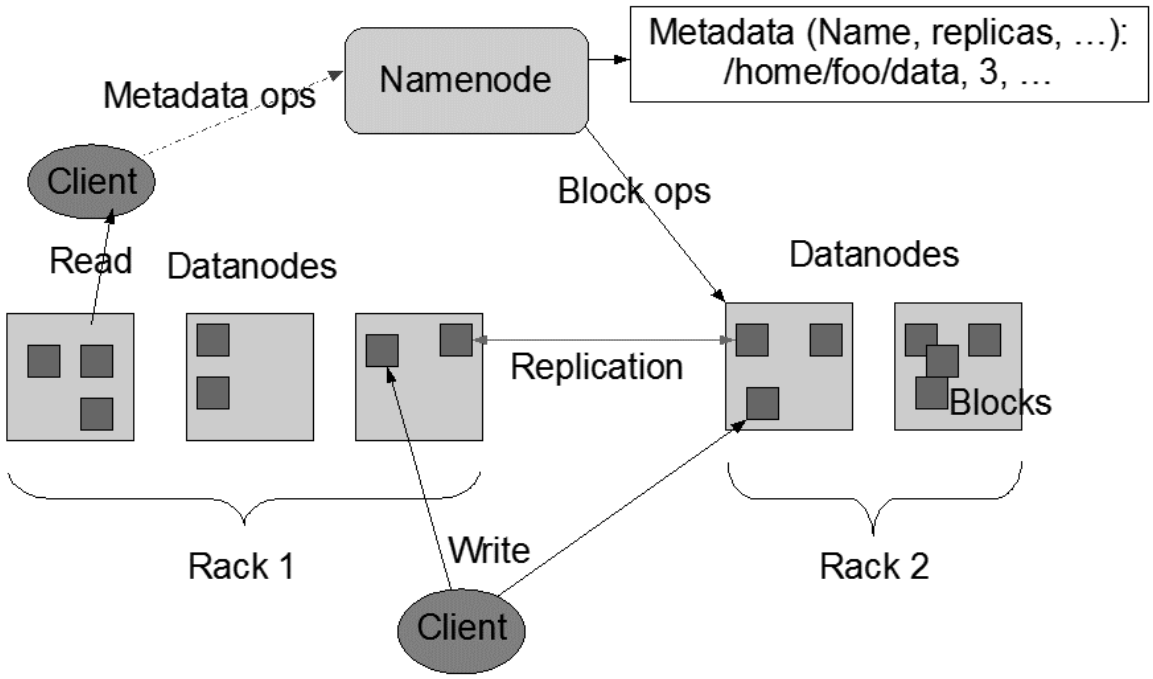

Figura 10 - Arquitetura do HDFS. Retirado de (Borthakur, 2013).

Além de um *NameNode* centralizado, todos os restantes nós do *cluster* fornecem o serviço *DataNode*. Cada *DataNode* armazena blocos HDFS e cada um desses blocos é armazenado separadamente no sistema de ficheiros local do nó. Estes, por sua vez, podem utilizar sistemas de ficheiros diferentes. Os blocos são criados ou eliminados no *DataNode* a pedido do *NameNode*, que valida e processa os pedidos dos utilizadores. Embora o *NameNode* seja responsável pela gestão de todo o sistema de ficheiros, nas operações de leitura e escrita a ligação é estabelecida diretamente entre o utilizador e o *DataNode* (Shafer et al., 2010).

O HDFS disponibiliza um sistema de replicação automático. Por defeito, são guardadas duas copias de cada bloco em diferentes *DataNode's*. Este sistema de replicação é transparente para o cliente, isto é, quando é escrito um bloco o cliente estabelece ligação apenas com um *DataNode*, este, por sua vez, estabelece uma ligação com um segundo *DataNode* para lhe enviar um "eco" dos dados (Shafer et al., 2010).

3.1.3 *Yarn*

O *Yarn* ou *MapReduce* 2.0 é uma das evoluções do *Hadoop*. Como representado na Figura 11, na arquitetura *Yarn* cada *DataNode* possui um *NodeManager*, enquanto o *NameNode* possui um *RecourceManager*. O objetivo é ter um gestor de recursos global, assim como um gestor de recursos de cada nó.

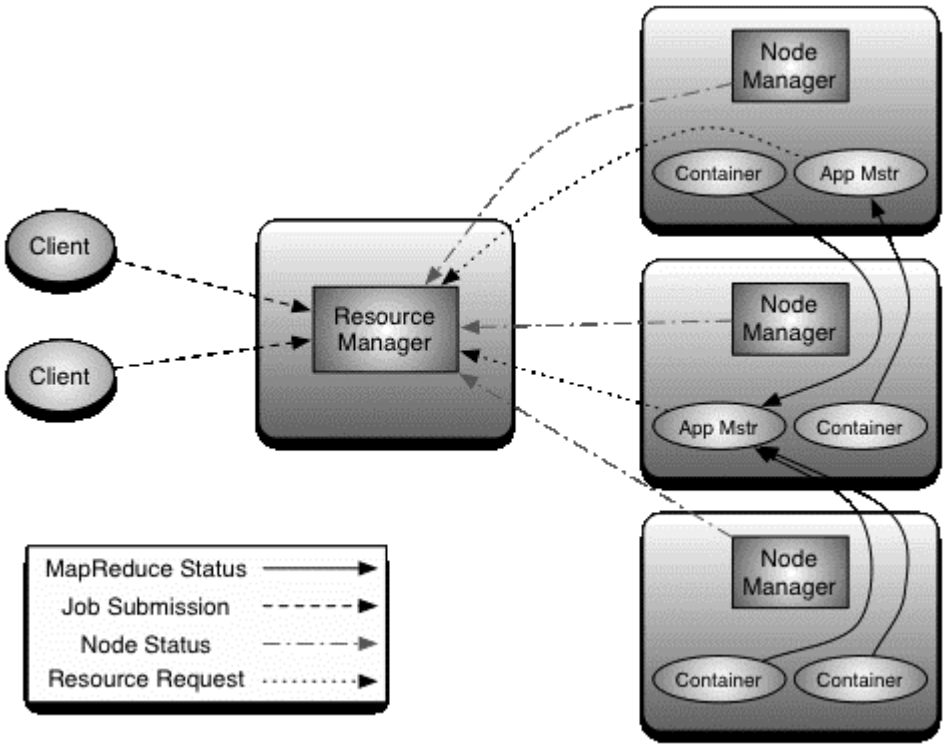

Figura 11 - Arquitetura *Yarn*. Retirado de (Apache, 2017).

Espalhados pelos *DataNode's* estão *Container's* e *ApplicationMaster's.* Para cada tarefa é criado um novo *ApplicationMaster* que tem como função coordenar a sua tarefa no cluster. Este é o "supervisor" dos *Container's*, que têm como função realizar as tarefas designadas. A tarefa de *MapReduce* é realizada pelos *ApplicationMaster's*, que reportam diretamente ao *ResourceManager,* enquanto os *NodeManager's* são responsáveis por garantir que as tarefas não sobrecarregam o seu nó. Com a informação recebida dos *ApplicationMaster's* e dos *NodeManager's,* o *ResourceManager* decide a distribuição dos *Container's* e *ApplicationMaster's* pelos *DataNode's* tendo em conta os recursos disponíveis em cada um deles (Raj, Nivash, Nirmala, & Dhinesh Babu, 2015).

3.2 SQL-*on-Hadoop*

Uma vez que, o *Hadoop* é o sistema padrão de armazenamento e processamento de *Big Data*, tanto para dados estruturados como não estruturados, e que o SQL é a linguagem padrão de armazenamento e manipulação de base de dados relacionais, tornou-se cada vez mais pertinente o desenvolvimento de ferramentas que disponibilizem as funcionalidades de análise do SQL em dados armazenados no HDFS.

Neste seguimento, o *Hive* foi o sistema pioneiro de suporte a análises tipo SQL em dados armazenados no HDFS. No entanto a performance das primeiras versões do Hive não era satisfatória o que levou ao surgimento de muitos outros sistemas SQL-*on-Hadoop* (Qin et al., 2017).

Outros sistemas SQL-*on-Hadoop* que se destacam na literatura são o Presto e o *Impala* (Santos et al., 2017; Floratou, 2014), ambos utilizados nesta dissertação e descrito a seguir.

3.2.1 *Hive*

Apache Hive é um sistema de data warehouse para o *Apache Hadoop*. Tem sido amplamente utilizado em organizações para gerir e processar vastos volumes de dados como eBay, Facebook, LinkedIn, Spotify, Taobao, Tencent e Yahoo.

O Hive foi originalmente desenvolvido como uma camada de tradução no topo do *MapReduce*, tem a sua própria linguagem SQL (*HiveQL*) que converte consultas em processos *MapReduce*. Com esta interface SQL, os utilizadores não necessitam de escrever programas cansativos e, por vezes, difíceis de mapear para manipular dados no HDFS (Huai et al., 2014).

3.2.2 *Presto*

O *Presto* é um projeto de open source que foi desenvolvido pelo Facebook, é uma ferramenta projetada para consultar com eficiência vastas quantidades de dados usando consultas distribuídas. Foi criado como uma alternativa às ferramentas SQL-*on*-*Hadoop* que utilizam o *MapReduce*, como o *Hive*. Mas além disso, o *Presto* pode ser utilizado em diferentes tipos de fontes de dados, incluindo bases de dados relacionais tradicionais e outras fontes de dados, como o *Kudu*.

O *Presto* foi projetado para lidar com *data warehousing* e *analytics*: análise de dados, agregando vastas quantidades de dados e produzindo relatórios normalmente classificadas como *Online Analytical Processing* (OLAP) (Presto, 2018).

3.2.3 *Impala*

O *Impala* é um projeto *open source* desenvolvido especificamente para aproveitar a flexibilidade e escalabilidade do *Hadoop*. É integrado no ambiente *Hadoop* e utiliza inúmeros componentes padrão do mesmo, com o objetivo de manter a sua flexibilidade e proporcionar ao utilizador uma experiência de utilização semelhante à utilização do SQL em bases de dados relacionais.

Para reduzir a latência, que advém, por exemplo, pela utilização do *MapReduce* ou pela leitura de dados remotamente, o *Impala* implementa uma arquitetura distribuída que é responsável por todos os aspetos na execução de consultas e que é utilizada na mesma infraestrutura do *Hadoop* (Kornacker et al., 2015).

3.3 Armazenamento de dados

Nesta secção é feita uma descrição dos formatos de armazenamento de dados mais destacados na literatura.

3.3.1 *Optimized Row Columnar* (ORC)

O ORC é um formato de ficheiros baseado em colunas, optimizado para leitura de *streams* de dados, mas que suporta a localização rápida de linhas. O armazenamento em colunas permite a leitura, descompactação e processamento apenas dos dados necessários para uma dada consulta. A sua codificação é variável, de acordo com o tipo de dados guardados.

O formato ORC foi desenvolvido em Janeiro de 2013 com o objetivo de aumentar consideravelmente a velocidade do *Hive* (Apache Software Foundation, 2015).

3.3.2 *Parquet*

Apache Parquet é um formato de armazenamento baseado em colunas, otimizado para trabalhar com datasets com várias colunas. Os casos de uso do *Parquet* geralmente envolvem o trabalho com subconjuntos dessas colunas em vez de registos completos (Plase, Niedrite, & Taranovs, 2017). Este oferece uma compactação eficiente dos dados e esquemas de codificação com desempenho melhorado para manipular dados complexos em massa (Floratou, 2014).

O *Parquet* é similar a outros formatos de armazenamento baseado em colunas disponíveis no *Hadoop*, por exemplo o ORC, e é compatível com a maioria das ferramentas do ecossistema *Hadoop* (Patil & Kshirsagar, 2018).

Na maioria dos casos de uso, o *Impala* recomenda a utilização do *Parquet* (Kornacker et al., 2015).

3.4 *Kudu*

O armazenamento estruturado no ecossistema Hadoop é, tipicamente, alcançado de duas maneiras, como se explica a seguir. Para um *dataset* estático, os dados normalmente são armazenados no HDFS, usando um formato de dados binário, como *Apache Parquet*. No entanto, nem o HDFS, nem esses formatos têm capacidade para atualizar registos ou para acesso aleatório eficiente. Por outro lado, *Datasets* mutáveis são, normalmente, guardados em armazenamentos semiestruturados, como *Apache Hbase* ou *Apache Cassandra*. Estes sistemas permitem leituras e escritas de baixa latência, mas ficam muito atrás dos formatos estáticos em termos de throughput de leitura sequencial para aplicações como análise baseada em SQL ou *machine learning*.

A lacuna entre o desempenho analítico oferecido por *datasets* estáticos no HDFS e o acesso aleatório de baixa latência no *HBase* e *Cassandra* exigem a utilização de arquiteturas complexas para atingir ambos os padrões. Normalmente, são desenvolvidos *pipelines* de dados para consumo e atualização em *streaming* no *HBase*, seguidos de exportações periódicas das tabelas para o *Parquet* para posterior análise. Tais arquiteturas têm diversas desvantagens (Lipcon et al., 2015):

- Os arquitetos de software têm a necessidade de desenvolver código complexo para gerir o fluxo e a sincronização de dados entre os dois sistemas;
- É necessário gerir *backups*, políticas de segurança e monotorização para diversos sistemas distintos;
- A arquitetura pode apresentar atrasos significativos entre a chegada dos dados à *staging area* e a disponibilização de novos dados para análise;

- Em contexto de mundo real, os sistemas normalmente necessitam de receber dados antigos em atraso ou corrigir registos que já se encontram alugados no armazenamento estático. Para alcançar estes objetivos é necessário reescrever e alterar partições, o que normalmente implica a intervenção manual no sistema.

O *Kudu* é um novo sistema de armazenamento projetado e implementado desde o início para preencher esta lacuna entre os sistemas de sistema sequencial de alto *throughput* e os sistemas de acesso aleatório de baixa latência. Embora os sistemas existentes continuem a ter vantagens em algumas situações, o *Kudu* oferece uma alternativa intermédia que pode simplificar drasticamente as arquiteturas normalmente utilizadas. Em particular, o *Kudu* oferece uma API simples para inserir, atualizar e apagar registos, enquanto fornece um throughput semelhante ao do Parquet (Lipcon et al., 2015).

Da perspetiva do utilizador, o *Kudu* é um sistema de armazenamento para tabelas de dados estruturados. Um cluster *Kudu* pode ter tantas tabelas quanto as desejadas, cada uma delas com uma estrutura definida, que consiste num número finito de colunas. Cada coluna tem nome, tipo, e nulidade opcional, contendo uma delas a chave primária da tabela. A chave primária deve ser única e é o único elemento que permite atualizar ou apagar linhas de forma eficiente. Este modelo de dados é semelhante ao modelo relacional, mas difere de muitos modelos de base de dados distribuídas, como *Cassandra*, *MongoDB*, *Riak*, *BigTable*, etc. Tal como acontece com as bases de dados relacionais, o utilizador deve definir o esquema de uma tabela no momento da sua criação, não sendo permitido inserir dados em colunas não definidas. O utilizador pode a qualquer momento alterar a tabela, adicionando ou removendo colunas, à exceção da coluna que contém a chave primária.

Após a criação de uma tabela, o utilizador pode alterar o conteúdo da tabela através da utilização das APIs *Insert*, *Update* e *Delete*. Estas APIs apenas permitem a alteração de um registo por transação. Nenhuma API para alteração de múltiplos registos em simultâneo é disponibilizada pelo Kudu.

O *Kudu* oferece apenas uma operação de *scan* para leitura de dados. No *scan*, o utilizador pode definir tantas condições quantas desejar, apesar de apenas serem permitidos dois tipos de condições: comparação entre colunas e valores constantes e intervalos de chaves primárias. Estas condições são interpretadas tanto pela API do cliente, como pela do servidor, de modo a reduzir de forma eficiente a quantidade de dados transferidos. O cliente pode, também, especificar o subconjunto de colunas que pretende consultar, uma vez que o armazenamento em disco é feito em coluna. A especificação deste subconjunto pode melhorar substancialmente o desempenho.

O *Kudu* não fornece nenhuma consola ou linguagem SQL-*on*-*Hadoop* sendo o único suporte para operações SQL através da integração com outras ferramentas como o *Impala* ou o *Presto* (Lipcon et al., 2015).

Seguindo o design de muitos outros projetos, e de acordo com o ilustrado na Figura 12, o *Kudu* conta com um único *MasterServer*, responsável pelos metadados, e um número maior de *TabletServers*, responsáveis pelos dados. O *MasterServer* pode ser replicado para suportar a tolerância a falhas, sendo a recuperação do sistema muito rápida após uma falha. Normalmente, todo o sistema pode ser implementado em *commodity hardware*, sem a necessidade de requisitos especiais para o *MarterServer*.

Como a maioria dos sistemas de bases de dados distribuídos, as tabelas em *Kudu* são particionadas horizontalmente. Assim como no *BigTable*, estas partições horizontais são chamadas de *tablets*. Qualquer registo pode ser mapeado para exatamente um *tablet* através da sua chave primária, garantindo assim que as operações de acesso aleatório, como atualização ou remoção, afetem apenas um único *tablet*.

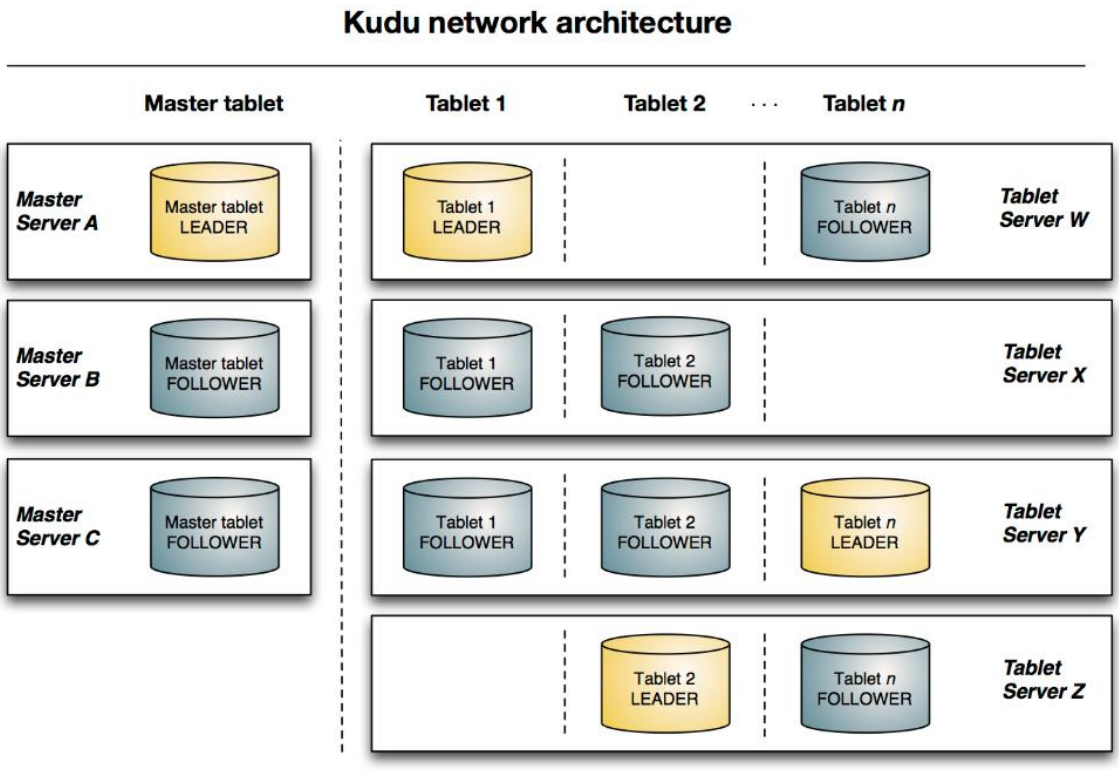

Figura 12 - Arquitetura do *Kudu*. Retirada de (Apache Software Foundation, 2016).

A fim de proporcionar alta disponibilidade e durabilidade durante a execução, o *Kudu* replica todos dados das tabelas por diversas máquinas, como se pode verificar na Figura 12. Ao criar uma tabela é necessário especificar o fator de replicação, normalmente 3 ou 5 dependendo dos requisitos definidos,

sendo da responsabilidade do *MasterServer* que o número de réplicas pretendidas seja mantido permanentemente.

Com o objetivo de garantir todas estas características em simultâneo, o *Kudu* não reutiliza nenhum mecanismo de armazenamento pré-existente. Em contrapartida, é implementada uma nova arquitetura híbrida de armazenamento em colunas (Lipcon et al., 2015).

Esta página foi deixada intencionalmente em branco.

4. INFRAESTRUTURA E CONJUNTO DE DADOS UTILIZADO

Tendo em conta os objetivos da dissertação, que consistem em compreender os cenários e contextos em que o *Kudu* pode ser usado, destacando as vantagens e desvantagens identificadas, neste capítulo será apresentada a infraestrutura utilizada na realização dos testes, assim como os conjuntos de dados utilizados e o protocolo de testes seguido.

4.1 Infraestrutura

A infraestrutura utilizada para a realização dos testes está configurada na plataforma *Google Cloud Platform* [1] e é composta por 5 nós (1 HDFS *NameNode* e 4 HDFS *DataNodes*) com as seguintes características:

- vCPU (virtual CPU) de 4 núcleos (*Intel Xenon* de 2.2 Ghz);
- 32GB de *Random Access Memory* (RAM);
- *Hard Disk Drive* (HDD) de 256GB;
- *Solid State Drive* (SSD) de 300GB (apenas nos HDFS *DataNodes*);
- Interface *Ethernet* 10 *Gigabits*.

É utilizado o sistema operativo CentOS 7.5.1804 *Minimal* em todos os nós do sistema, e a distribuição *Hadoop* utilizada é *Cloudera Express* 5.14.3.

O *cluster* tem também instalado o Presto, distribuído em 1 *master* alojado no *NameNode* e 4 *workers* cada um alojado em um dos *DataNodes*, configurado para utilizar 16GB de RAM.

4.2 Conjunto de Dados

Para a realização dos testes foram utilizados dois conjuntos de dados, o TPC-H, um modelo relacional de um *dataset* tradicional de vendas, este modelo foi desnormalizado como será explicado na secção seguinte, e o *Star Schema Benchmark* (SSB), um modelo baseado no TPC-H modelado numa estrutura multidimensional (estrela). A utilização destes *datasets*, tem como objetivo testar a viabilidade da utilização do *Kudu* na implementação de um BDW em detrimento de outras ferramentas com o *Hive*.

[1] https://cloud.google.com/

4.2.1 TPC-H

Um dos *datasets* utilizados nos testes foi o TPC-H versão 2.17.3, um *dataset* de suporte à decisão com um conjunto de *queries* orientadas ao negócio. Este *benchmark* disponibiliza um mecanismo que permite gerar *datasets* de diferentes tamanhos, o que facilita a criação e avaliação de cenários com diferentes volumes de dados.

Embora os sistemas SQL-*on*-*Haddop* possam executar operações *join*, os resultados obtidos a partir de um modelo normalizado tendem a exigir mais recursos e aumentar o tempo de execução de *queries* (Floratou, 2014). Portanto, em vez de utilizar o modelo TPC-H normalizado, foi utilizado um modelo desnormalizado para armazenar os dados. Assim, para realizar a desnormalização, foi desenvolvido um processo ETL, que transforma o *dataset* TPC-H em uma tabela desnormalizada armazenada no *Hive*, e um outro, para copiar a tabela armazenada em *Hive* para uma outra armazenada em *Kudu.* Estas duas tabelas foram então utilizadas como fontes de dados para a realização dos testes.

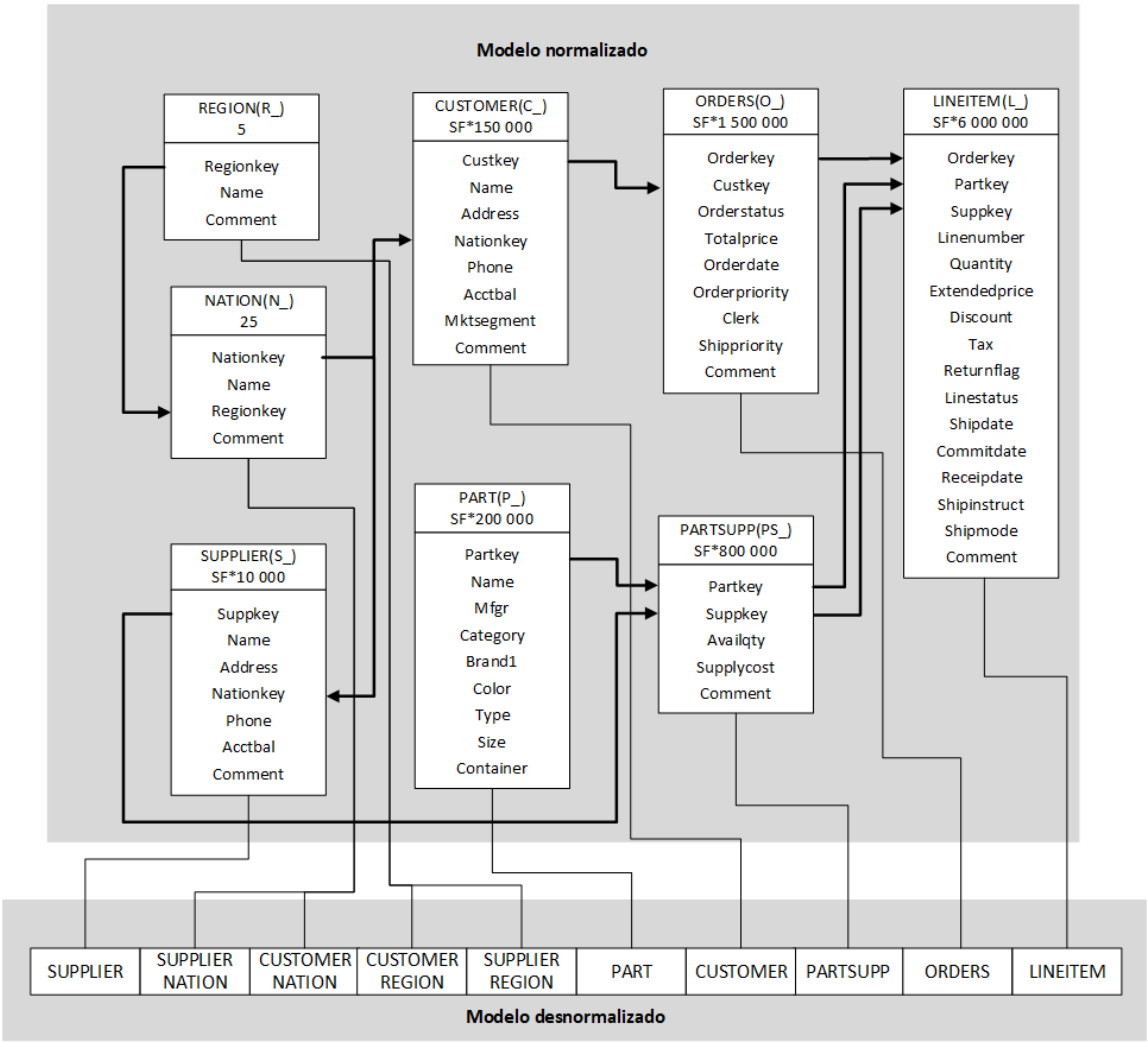

Figura 13 - Modelo de dados TPC-H. Adaptado de ("TPC Benchmark H," 2017)

Posteriormente, na secção 4.3, será explicado o protocolo de testes e a justificação para a utilização destas diferentes tecnologias.

A Figura 13 mostra o esquema normalizado TPC-H e a sua transposição para uma tabela desnormalizada.

Utilizando o conjunto de dados pré-definido é possível definir o tamanho do *dataset* desejado, sendo este gerado tendo em conta o fator de escala (*scale factor* (SF)) definido. Deste modo são geradas as tabelas com os tamanhos correspondentes, calculadas de acordo com as fórmulas da Figura 13.

Para este *dataset* foram definidas 22 *queries* ("TPC Benchmark H," 2017) com diferentes níveis de complexidade e diversos operadores SQL. Nos testes realizados nesta dissertação foram utilizadas 21 das *queries* uma vez que a *query* 22 não é possível executar no modelo desnormalizado. A *query* 22 visa selecionar os clientes que não efetuaram compras, dados que são excluídos no processo de desnormalização uma vez que o modelo desnormalizado apenas representa factos, isto é, os dados que não são representativos de factos não são armazenados no modelo desnormalizado. Estas *queries* estão disponíveis no Apêndice 1.

4.2.2 SSB

O SSB, como já referido, é um modelo baseado no TPC-H. As alterações realizadas por O'Neil, O'Neil, & Chen (2009) visam transformar o modelo relacional TPC-H num modelo multidimensional em estrela e encontram-se descritas no mesmo trabalho. Nesta dissertação não foram realizadas quaisquer alterações ao modelo SSB, pelo que, o modelo de dados utilizado mantém a sua tabela de factos (*Lineorder*) assim como as suas 4 dimensões (*Customer*, *Supplier*, *Part* e *Date*) Nenhum atributo foi adicionado ou removido e o tamanho das tabelas foi mantido como descrito na Figura 14.

À imagem do que o que acontece no TPC-H, no SSB é possível definir o tamanho do *dataset* desejado, bastando para isso alterar o SF. O tamanho das tabelas geradas é calculado de acordo com as fórmulas da Figura 14.

Os autores do SSB disponibilizam também um conjunto de 13 *queries*, sendo estas *queries* as utilizadas na realização dos testes. Estas *queries* estão disponíveis no Apêndice 2.

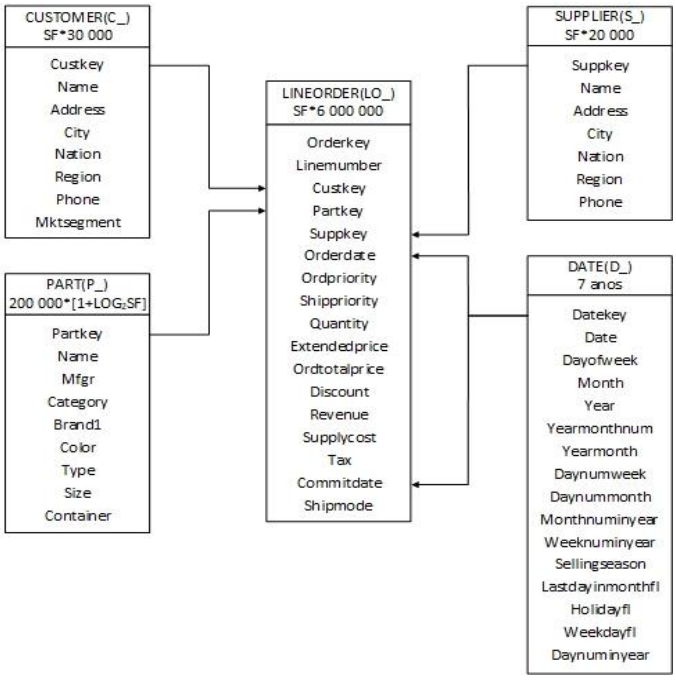

Figura 14 - Modelo de dados SSB. Adaptado de (O'Neil et al., 2009)

4.3 Protocolo de Testes

De acordo com o referido anteriormente, a presente dissertação tem como objetivo compreender os cenários e contextos em que o *Kudu* pode ser usado, utilizando, para isso, duas fontes de dados distintas. Estas fontes serão utilizadas em 2 cenários:

1. Cenário 1, este cenário tem como objetivo testar o *Kudu* num ambiente em que é utilizado um modelo de dados desnormalizado, sendo para isso medido o tempo de processamento das *queries* em diferentes repositórios de dados e utilizando diferentes ferramentas de consulta;
2. Cenário 2, este cenário tem como objetivo testar o *Kudu* num ambiente em que é utilizado um modelo de dados multidimensional em estrela. Testando assim o seu desempenho numa possível implementação num BDW. O método para avaliar o desempenho do *Kudu* é similar ao do Cenário 1.

O objetivo dos testes é perceber quais as vantagens e desvantagens da utilização do *Kudu* em cada um dos cenários, assim como, o impacto da utilização ou não de replicação de dados e perceber ainda o impacto do aumento do volume de dados, utilizando em todos os cenários 2 fatores de replicação (1,3), que representam a não utilização de replicação no fator 1 e a utilização de 3 replicações no fator 3, e 3 fatores de escala (10GB, 30GB, 100GB).

Como repositórios de dados são utilizados o *Kudu*, o *Hive* (em formato ORC) e o *Parquet* e como ferramentas de consulta são utilizados o *Presto* e o *Impala*. Relativamente às escolhas do *Hive* e do *Presto*, estas foram fundamentadas pelo estudo realizado por Santos et al. (2017) que destaca as vantagens da utilização do *Hive* em contextos de aumento do volume dos dados e o melhor desempenho do *Presto* em detrimento de outras ferramentas de consulta como o *Spark* e o *Drill*. A escolha do *Impala* foi devida ao facto de ambas as ferramentas, *Kudu* e *Impala*, terem sido inicialmente desenvolvidas pela Cloudera (Lipcon et al., 2015),(Kornacker et al., 2015) sendo por isso pertinente comparar e perceber a diferença de comportamento do *Kudu* quando utilizado em conjunto com ferramentas que não sejam desenvolvidas pela mesma empresa. Uma vez que o formato ORC não é suportado pelo *Impala* e visto que o formato de ficheiros recomendado para utilizar em conjunto com o *Impala* é o *Parquet* (Kornacker et al., 2015), a escolha recaiu sobre esse formato.

A Figura 15 sumariza os cenários de teste anteriormente descritos.

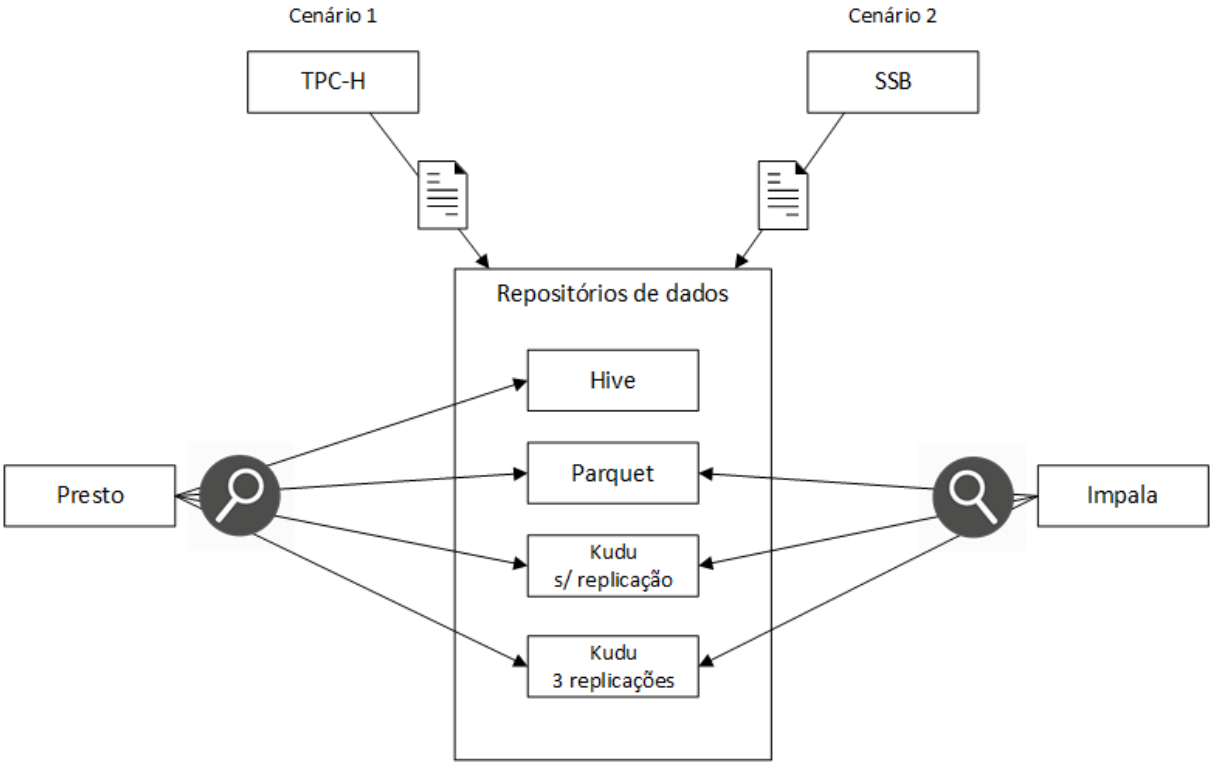

Figura 15 - Cenários de teste

Como representado na figura, ambos o *datasets* foram carregados para 4 repositórios de dados, nomeadamente *Hive*, *Parquet*, *Kudu* sem replicação e *Kudu* com três replicações. Relativamente às ferramentas de consulta, com o *Presto*, as *queries* foram executadas nos 4 repositórios. Enquanto que com o *Impala*, as *queries* não foram executadas no *Hive,* uma vez que o formato de armazenamento utilizado (ORC) não é suportado pelo *Impala*, como referido anteriormente.

Esta página foi deixada intencionalmente em branco.

5. *Apache Kudu*: Vantagens e Desvantagens na Análise de Vastas Quantidades de Dados

Neste capítulo são apresentados os resultados obtidos nos dois cenários de teste descritos no capítulo anterior. Os resultados encontram-se divididos pelo fator de escala (10GB, 30GB, 100GB). Nas próximas secções são apresentados os resultados de cada *query* em cada um dos repositórios de dados (*Hive*, *Parquet*, *Kudu*) executada através das diferentes ferramentas de consulta (*Presto*, *Impala*). Estes resultados correspondem à média do tempo de execução (em segundos) de quatro execuções seguidas. Verificar-se-á em alguns casos a menção "ERRO" que significa uma falha na execução da *query* por falta de memória, assim como, o destaque a negrito do melhor tempo de execução de cada *query*.

5.1 Cenário 1 – TPC-H desnormalizado

Esta secção contém os resultados obtidos nos testes realizados com o conjunto de dados TPC-H divididos pelo fator de escala (10GB, 30GB, 100GB), assim como todas as apreciações consideradas pertinentes.

De acordo com os resultados obtidos no fator de escala 10GB (Tabela 1) verifica-se que a combinação com melhores resultados é armazenamento *Kudu* sem replicação questionado pelo *Impala, com* um tempo de execução total de 114,13s. No total das 21 *queries* esta combinação obteve melhor desempenho em 14, uma das quais em empate com o *Kudu* com três replicações questionado pelo *Impala*, sendo o melhor desempenho das restantes *queries* dividido pelas restantes combinações, à exceção do armazenamento *Parquet* questionado pelo *Presto*, que não obtiveram o melhor desempenho em nenhuma *query*.

Observando os resultados é, também, possível perceber que o tempo total de execução é sempre inferior em qualquer um dos armazenamentos quando questionados pelo *Impala*. Em termos de comparação, o melhor resultado obtido pelo *Presto* (armazenamento *Hive*) é 15% mais lento que o pior resultado obtido pelo *Impala* (armazenamento *Kudu* com três replicações).

Numa análise mais focada no Kudu constata-se que a utilização de replicação piora o seu desempenho, devido ao aumento do tempo gasto pelo *Kudu Client* para recolher a localização das tabelas junto do *Kudu Master*. Questionado pelo *Presto*, o tempo total de execução aumenta em 12% quando se utilizam três replicações, tendo melhor desempenho apenas em duas das 21 *queries*. No caso da utilização do *Impala* os resultados são ainda mais significativos, a utilização de três replicações piora em

27% o tempo total de execução obtendo melhor desempenho em apenas uma das *queries* a Q8, e o mesmo tempo na Q6.

Tabela 1 – Resultados TPC-H SF=10

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Q1	6,11	17,10	37,92	38,95	4,58	**3,38**	3,66
Q2	6,55	23,91	5,31	5,00	7,92	**1,67**	1,74
Q3	7,31	12,72	**4,18**	4,66	17,68	15,18	15,82
Q4	4,60	9,70	3,70	4,05	2,97	**1,74**	2,40
Q5	4,11	9,77	3,49	3,67	3,43	**1,55**	1,67
Q6	3,39	7,30	2,24	2,31	2,22	**1,18**	**1,18**
Q7	3,99	8,53	**2,03**	2,23	2,79	3,08	3,16
Q8	3,98	9,09	1,95	2,12	3,12	1,29	**1,27**
Q9	**5,07**	12,45	41,77	38,40	3,83	5,88	6,72
Q10	12,59	16,33	6,96	18,51	7,37	**3,39**	3,52
Q11	4,57	12,47	4,37	4,68	4,40	**2,67**	3,01
Q12	4,62	9,62	5,64	6,08	3,25	**2,10**	3,11
Q13	16,24	18,40	55,24	62,73	**6,26**	9,61	18,65
Q14	3,19	7,58	2,25	2,39	2,01	**1,15**	1,21
Q15	8,06	16,77	7,08	7,35	5,35	**3,84**	4,14
Q16	**7,06**	11,28	61,77	70,72	**7,06**	9,03	12,48
Q17	4,56	12,95	4,32	4,56	2,60	**1,22**	1,82
Q18	26,13	29,81	58,10	62,38	18,14	**11,27**	13,94
Q19	3,84	9,82	2,56	2,97	3,63	**1,94**	2,07
Q20	6,85	16,93	13,87	14,84	7,42	**5,40**	7,72
Q21	24,02	37,42	47,95	60,48	**14,57**	27,56	35,55
Total	166,87	309,95	372,71	419,08	130,60	**114,13**	144,82

Outra informação relevante extraída dos resultados obtidos, é o facto de o desempenho do *Kudu* piorar consideravelmente quando questionado pelo *Presto* em comparação com o *Impala*, sendo o tempo de execução total das *queries* quase três vezes superior. Aliás, é de salientar o facto de o *Kudu*, quando questionado pelo *Presto*, obter os piores resultados de todas as combinações em análise.

Olhado agora para a Figura 16, que representa os tempos de processamento das *queries* executadas pelo *Impala*, ou seja, as combinações com melhores resultados, é possível constatar que existe um padrão comum nos resultados obtidos, apesar de os dados serem armazenados em diferentes formatos. Quando comparado o armazenamento em *Kudu* com e sem replicação, as únicas diferenças relevantes são nas *queries* Q13, Q16, Q18 e Q21, não sendo, mesmo assim, alterado o padrão. Quando examinado o armazenamento *Parquet*, este obtém melhores resultados que o *Kudu* com e sem replicação nas queries Q7, Q9, Q13, Q16 e Q21. Destaca-se, e para realçar a similaridade das três

combinações, que o tempo total de execução do armazenamento *Parquet* é quase igual à média do tempo total de execução do armazenamento em *Kudu* com e sem replicação.

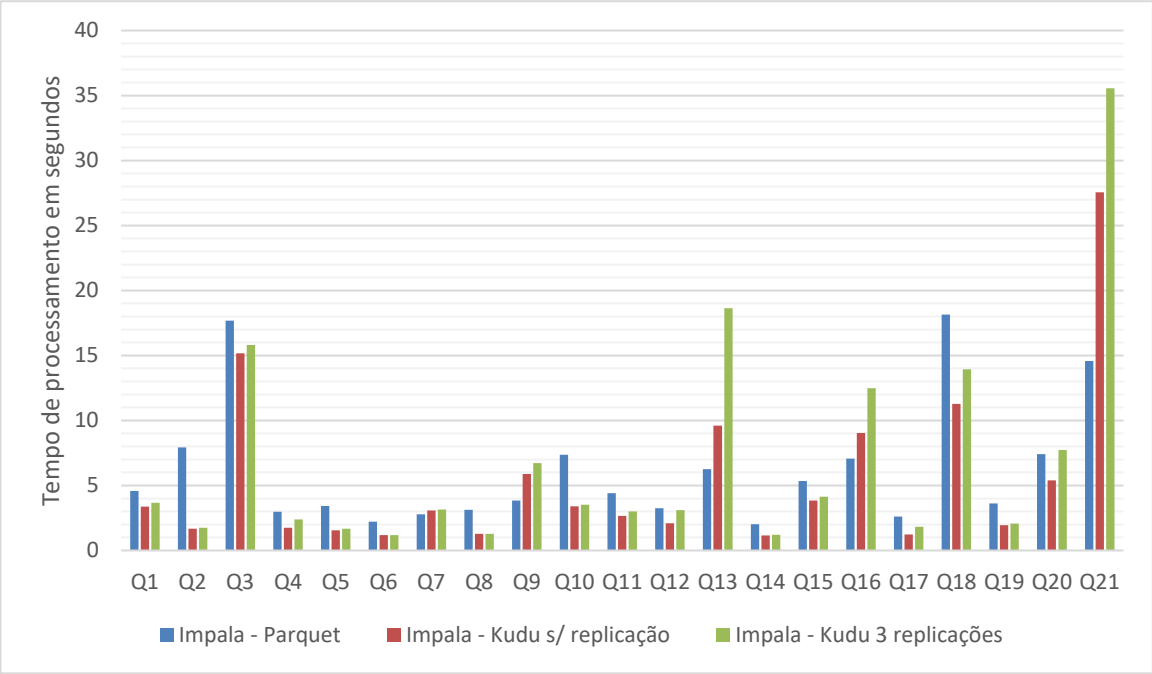

Figura 16 - Padrão do Processamento das *queries* para SF=10

Analisando resultados obtidos no fator de escala 30GB (Tabela 2) verifica-se que neste caso os resultados não são tão esclarecedores quanto no fator de escala 10GB. Enquanto que a combinação com melhor tempo total de execução é armazenamento *Parquet* questionado pelo *Impala, com* um tempo de execução total de 431,31s, a combinação com melhor desempenho num maior número de *queries* é armazenamento *Kudu* sem replicação questionado pelo *Impala*, com melhor desempenho em 9 das 21 *queries*. De destacar também que a combinação com melhor tempo total de execução, tem também o melhor desempenho em 7 queries, estando o melhor desempenho das restantes 5 *queries* distribuído entre o armazenamento *Kudu* com três replicações questionado pelo *Presto* e pelo *Impala*.

Tendo em conta o impacto causado pela replicação no desempenho do *Kudu*, é possível perceber que ao contrário do que acontece no fator de escala 10, em que a replicação piora o desempenho, neste caso, no fator de escala 30 a replicação melhora o desempenho. Quando questionado pelo Presto e utilizando três replicações, o tempo de execução total diminui 21%, obtendo também, melhor desempenho em 19 das 21 *queries*. No caso da utilização do *Impala* os resultados não são tão expressivos, a melhoria no tempo total de execução, quando utilizando três replicações, é de apenas 6%

e a melhoria de desempenho só se verifica em 7 *queries*. Este ponto será avaliado de uma outra perspetiva posteriormente nesta secção.

Tabela 2 - Resultados TPC-H SF=30

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Q1	31,97	39,54	135,52	130,40	14,59	**14,21**	16,11
Q2	69,17	57,90	120,24	47,46	**26,75**	47,43	94,37
Q3	43,32	32,11	12,19	**10,96**	53,33	56,25	61,22
Q4	17,80	23,66	11,83	10,92	8,65	**5,29**	7,27
Q5	19,79	25,23	11,62	10,51	10,78	**5,95**	5,97
Q6	8,33	17,38	5,13	4,45	5,92	**3,20**	3,59
Q7	9,44	20,62	4,24	**3,68**	7,19	10,95	10,59
Q8	15,05	22,81	4,67	3,67	8,23	2,78	**2,62**
Q9	21,22	32,56	149,60	32,54	**10,56**	24,57	24,22
Q10	64,99	60,48	127,72	30,85	**29,47**	105,95	72,02
Q11	11,54	30,76	17,17	11,32	10,67	10,36	**9,04**
Q12	28,65	24,90	23,65	22,09	9,25	**7,24**	13,33
Q13	60,07	56,46	242,63	224,39	**21,01**	39,39	40,58
Q14	17,28	18,22	5,42	4,75	5,63	**3,22**	3,31
Q15	27,78	47,13	54,77	45,54	**16,47**	35,81	37,26
Q16	17,95	28,30	281,44	248,85	**15,37**	33,12	33,26
Q17	13,58	33,91	13,77	13,38	7,07	**3,42**	4,48
Q18	115,84	102,76	257,69	249,03	77,14	68,06	**63,19**
Q19	13,55	25,95	6,91	7,04	12,23	**6,80**	7,67
Q20	30,94	50,08	59,85	69,96	27,11	**24,69**	35,13
Q21	104,39	120,07	185,94	184,34	**53,89**	399,41	310,43
Total	742,66	870,84	1731,97	1366,13	**431,31**	908,09	855,66

Olhando em detalhe para os resultados obtidos pelo armazenamento *Kudu* sem replicação questionado pelo *Impala*, a combinação com melhor desempenho num maior número de *queries*, é possível perceber que apenas uma *query*, a Q21, é responsável por quase 44% do tempo total de execução das *queries*. Sendo este valor tão discrepante dos restantes é relevante perceber o seu impacto na avaliação geral dos resultados obtidos no fator de escala 30. Deste modo a Tabela 3 apresenta o tempo total de execução das *queries* de cada combinação assim como o mesmo excluindo a Q21.

Analisando a Tabela 3 é possível destacar que apesar da exclusão da Q21, a combinação com o armazenamento *Parquet* questionado pelo *Impala* mantém o melhor tempo total de execução. É de destacar também o facto de que com esta exclusão a utilização de replicação piora o desempenho do *Kudu* quando este é questionado pelo *Impala*.

Tabela 3 - Total TPC-H SF=30 vs. Total TPC-H SF=30 excluindo Q21

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Tempo total de execução	742,66	870,84	1731,97	1366,13	**431,31**	908,09	855,66
Tempo total de execução excluído a Q21	638,27	750,77	1546,03	1181,79	**377,42**	508,68	545,22

De acordo com resultados da Tabela 3, e tendo em perspetiva o impacto da replicação sobre o Kudu, é possível concluir que a exclusão da Q21 não tem impacto nos resultados obtidos quando utilizando o Presto. Nesse caso a melhoria de desempenho referente à replicação era de 21% e passa a ser de 24%. No caso do *Impala*, em que se verificava uma ligeira melhoria de desempenho, agora é possível constatar uma redução de 7% do desempenho. Deste modo conclui-se que no fator de escala 30 a replicação tem um impacto positivo quando utilizada com o *Presto* e um impacto não relevante quando utilizada com o *Impala*.

Focando ainda na exclusão da Q21, é possível perceber que o tempo total de execução é sempre inferior em qualquer um dos armazenamentos quando questionados pelo *Impala*. Em termos de comparação, o melhor resultado obtido pelo *Presto* (armazenamento *Hive*) é 17% mais lento que o pior resultado obtido pelo *Impala* (armazenamento *Kudu* com três replicações). Quando esta análise é realizada incluindo a Q21 é percetível o impacto que a mesma tem no armazenamento *Kudu* questionado por *Impala.* Neste caso, o armazenamento *Hive* questionado pelo *Presto* consegue um melhor desempenho que o *Kudu* com e sem replicação questionado pelo *Impala*.

Outra informação relevante extraída dos resultados obtidos, é o facto de o desempenho do *Kudu* piorar consideravelmente quando questionado pelo *Presto* em comparação com o *Impala*. Aliás, é de salientar o facto de o *Kudu*, quando questionado pelo *Presto*, obter os piores resultados de todas as combinações em análise, mesmo incluindo ou excluindo a Q21.

Relativamente ao fator de escala 100 (Tabela 4) pode-se verificar que, ao contrário do que acontece nos fatores de escala 10 e 30, não foram realizados testes ao armazenamento *Kudu* com três replicações, uma vez que o espaço disponível no *cluster* para armazenamento de dados não é suficiente.

Também é possível constatar que a Q21 não foi executada pelo *Presto*, dado que, devido à complexidade da *query* o *Presto* não dispõe de memória suficiente para a executar com sucesso.

De acordo com o que se verifica no fator de escala 30, os resultados obtidos não são tão esclarecedores como no fator de escala 10. Mais uma vez, a combinação com melhor tempo total de

execução é armazenamento *Parquet* questionado pelo *Impala,* com um tempo de execução total de 1638,63s, e a combinação com melhor desempenho num maior número de *queries* é armazenamento *Kudu* sem replicação questionado pelo *Impala*, com melhor desempenho em 10 das 21 *queries*. De destacar também que a combinação com melhor tempo total de execução, tem também o melhor desempenho em 7 queries, estando o melhor desempenho das restantes 4 *queries* distribuído entre o armazenamento *Kudu* sem replicação e armazenamento *Hive* ambos questionados pelo *Presto*.

Tabela 4 - Resultados TPC-H SF=100

	Presto			**Impala**	
	Hive	Parquet	Kudu s/ replicação	Parquet	Kudu s/ replicação
Q1	53,96	98,96	366,24	41,84	**34,41**
Q2	91,16	203,95	267,21	**79,61**	240,62
Q3	100,90	162,83	**93,85**	183,77	160,05
Q4	53,14	111,44	94,11	31,53	**18,23**
Q5	78,10	143,60	30,80	33,29	**16,46**
Q6	51,76	57,76	11,38	18,13	**8,21**
Q7	48,94	87,47	**8,15**	22,27	28,59
Q8	77,23	119,38	9,32	26,28	**5,44**
Q9	75,90	138,45	424,46	**33,41**	61,77
Q10	**155,59**	266,36	357,64	175,34	302,43
Q11	24,64	119,32	35,47	35,61	**14,71**
Q12	90,28	112,52	112,58	39,71	**28,50**
Q13	163,34	203,13	644,10	**85,80**	121,37
Q14	61,78	79,00	14,87	19,87	**10,11**
Q15	111,43	192,20	178,64	**54,06**	103,37
Q16	45,05	116,68	675,59	**41,44**	84,56
Q17	99,14	119,63	34,80	21,76	**10,12**
Q18	317,86	408,95	736,77	368,96	**289,02**
Q19	56,09	105,14	**16,47**	46,43	20,72
Q20	118,40	216,11	371,86	**85,40**	165,27
Q21	ERRO	ERRO	ERRO	**194,10**	1293,28
Total	1874,68	3062,88	4484,3	**1638,63**	3017,23

À semelhança do que acontece no fator de escala 30 é possível perceber que os resultados obtidos na Q21 são discrepantes dos restantes, não sendo esta executada com sucesso pelo *Presto* e representando o seu tempo de execução quase 43% do tempo total de execução das *queries* obtido pelo armazenamento *Kudu* sem replicação questionado pelo *Impala*, a combinação com melhor desempenho num maior número de *queries*. Assim sendo, a Tabela 5 apresenta o tempo total de execução das *queries* de cada combinação assim como o mesmo excluindo a Q21.

Analisando a Tabela 5 é possível verificar que apenas as combinações questionadas pelo *Impala* sofrem alteração, uma vez que, as restantes já não incluíam o tempo de execução da Q21 devido ao erro ocorrido.

Mantendo os resultados do fator de escala 10 e 30, excluindo a Q21, o tempo total de execução é sempre inferior em qualquer um dos armazenamentos quando questionados pelo *Impala*. Mais uma vez, o melhor resultado obtido pelo *Presto,* é no armazenamento *Hive* que mesmo assim é 9% mais lento que o pior desempenho do *Impala*.

Tabela 5 - Total TPC-H SF=100 vs. Total TPC-H SF=100 excluindo Q21

	Presto			**Impala**	
	Hive	Parquet	Kudu s/ replicação	Parquet	Kudu s/ replicação
Tempo total de execução	1874,68	3062,88	4484,3	**1638,63**	3017,23
Tempo total de execução excluído a Q21	1874,68	3062,88	4484,3	**1444,53**	1723,95

De destacar a dependência percetível que o *Kudu* tem relativamente ao *Impala*, visto que, à semelhança do que acontece nos fatores de escala 10 e 30, o *Kudu*, quando questionado pelo *Presto,* obtém os piores resultados de todas as combinações em análise.

Focando a análise na escalabilidade das ferramentas, as Tabela 6 Tabela 7 apresentam a comparação de resultados obtidos entre os SF 10 e 30 e os SF 10 e 100 pelas diversas ferramentas quando questionadas pelo *Presto* e *Impala* respetivamente. Os resultados apresentados representam o rácio entre o tempo de execução das *queries* nos SF 30 ou 100 e o tempo de execução das *queries* no SF 10, isto é, o número de vezes que aumentou o tempo de execução relativamente ao SF 10. Os valores a negrito representam os casos em que se verificou um aumento de desempenho associado ao aumento do fator de escala. Ou seja, como o SF 30 corresponde a 3 vezes mais dados que o SF 10, valores inferiores a 3 nas colunas que comparam estes dois fatores de escala representam uma melhoria de desempenho. Na mesma linha de pensamento valores inferiores a 10 nas colunas que comparam os fatores de escala 10 e 100 representam também uma melhoria de desempenho.

Como referido anteriormente, não foram realizados testes ao armazenamento *Kudu* com 3 replicações no SF 100, deste modo não é possível fazer a comparação de resultados com o SF 10 pelo que a coluna das Tabela 6 Tabela 7 referente a esses valores encontra-se vazia. Os rácios entre o SF 10 e 100 da Q21 também não se encontram disponíveis na Tabela 6 uma vez que o *Presto* não conseguiu

executar essa *query* com sucesso no SF 100, assim sendo o rácio total apresentado nestes casos exclui o tempo de execução da Q21 no SF 10.

Observando a Tabela 6, é possível perceber que a ferramenta com melhores resultados de escalabilidade é o *Parquet*, sendo a única a obter uma melhoria de desempenho total quando comparando o SF 10 com o SF 30, verificando uma melhoria em 17 das 21 *queries* nessa comparação. Apesar de não constatar uma melhoria de desempenho na comparação entre os SF 10 e 100, o Parquet obtém os melhores resultados entre as ferramentas.

Tabela 6 – Escalabilidade TPC-H usando *Presto*

	Presto							
	Hive		Parquet		Kudu s/ replicação		Kudu 3 replicações	
	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100
Q1	5,2	**8,8**	**2,3**	**5,8**	3,6	**9,7**	3,3	-
Q2	10,6	13,9	**2,4**	**8,5**	22,6	50,3	9,5	-
Q3	5,9	13,8	**2,5**	12,8	**2,9**	22,5	**2,4**	-
Q4	3,9	11,6	**2,4**	11,5	3,2	25,4	**2,7**	-
Q5	4,8	19,0	**2,6**	14,7	3,3	**8,8**	**2,9**	-
Q6	**2,5**	15,3	**2,4**	**7,9**	**2,3**	**5,1**	**1,9**	-
Q7	**2,4**	12,3	**2,4**	10,3	**2,1**	**4,0**	**1,6**	-
Q8	3,8	19,4	**2,5**	13,1	**2,4**	**4,8**	**1,7**	-
Q9	4,2	15,0	**2,6**	11,1	3,6	10,2	**0,8**	-
Q10	5,2	12,4	3,7	16,3	18,4	51,4	**1,7**	-
Q11	**2,5**	**5,4**	**2,5**	**9,6**	3,9	**8,1**	**2,4**	-
Q12	6,2	19,5	**2,6**	11,7	4,2	20,0	3,6	-
Q13	3,7	10,1	3,1	11,0	4,4	11,7	3,6	-
Q14	5,4	19,3	**2,4**	10,4	**2,4**	**6,6**	**2,0**	-
Q15	3,4	13,8	**2,8**	11,5	7,7	25,2	6,2	-
Q16	**2,5**	**6,4**	**2,5**	10,3	4,6	10,9	3,5	-
Q17	**3,0**	21,7	**2,6**	**9,2**	3,2	**8,0**	**2,9**	-
Q18	4,4	12,2	3,4	13,7	4,4	12,7	4,0	-
Q19	3,5	14,6	**2,6**	10,7	**2,7**	**6,4**	**2,4**	-
Q20	4,5	17,3	**3,0**	12,8	4,3	26,8	4,7	-
Q21	4,3	-	3,2	-	3,9	-	**3,0**	-
Total	4,5	13,1	**2,8**	11,2	4,6	13,8	3,3	-

Olhando para o *Kudu*, observa-se uma melhor escalabilidade quando utilizando 3 replicações, neste caso, o *Kudu* com 3 replicações apresenta melhoria de desempenho em 13 das 21 *queries* e uma perda de desempenho pouco significativa (10%) na comparação entre os SF 10 e 30, de destacar que o *Kudu* sem replicação apresenta os piores resultados totais entre todas as ferramentas.

Na Tabela 7, em que são apresentados os resultados de escalabilidade do *Impala*, é possível perceber que, mais uma vez, a ferramenta com melhores resultados é o *Parquet*. Apesar de não apresentar melhorias de desempenho total, o *Parquet* obtém um melhor desempenho em 10 das 21 *queries* comparando o SF 10 com o SF 30 e melhor desempenho também em 10 *queries* quando a comparação é feita com o SF 100.

Tabela 7 – Escalabilidade TPC-H usando *Impala*

	Impala					
	Parquet		Kudu s/ replicação		Kudu 3 replicações	
	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100
Q1	3,2	**9,1**	4,2	10,2	4,4	-
Q2	3,4	10,1	28,3	143,8	54,2	-
Q3	3,0	10,4	3,7	10,5	3,9	-
Q4	**2,9**	10,6	3,0	10,5	3,0	-
Q5	3,1	**9,7**	3,8	10,6	3,6	-
Q6	**2,7**	**8,2**	**2,7**	**7,0**	3,0	-
Q7	**2,6**	**8,0**	3,6	**9,3**	3,4	-
Q8	**2,6**	**8,4**	**2,2**	**4,2**	**2,1**	-
Q9	**2,8**	**8,7**	4,2	10,5	3,6	-
Q10	4,0	23,8	31,3	89,2	20,4	-
Q11	**2,4**	**8,1**	3,9	**5,5**	3,0	-
Q12	**2,9**	12,2	3,4	13,6	4,3	-
Q13	3,4	13,7	4,1	12,6	**2,2**	-
Q14	**2,8**	**9,9**	**2,8**	**8,8**	**2,7**	-
Q15	3,1	10,1	9,3	26,9	9,0	-
Q16	**2,2**	**5,9**	3,7	**9,4**	**2,7**	-
Q17	**2,7**	**8,4**	**2,8**	**8,3**	**2,5**	-
Q18	4,3	20,3	6,0	25,6	4,5	-
Q19	3,4	12,8	3,5	10,7	3,7	-
Q20	3,7	11,5	4,6	30,6	4,5	-
Q21	3,7	13,3	14,5	46,9	8,7	-
Total	3,3	11,1	8,0	15,1	5,9	-

Relativamente ao *Kudu*, pode-se verificar que à imagem do que acontece com o *Presto*, a utilização de 3 replicações melhora a sua escalabilidade. De destacar que em termos de escalabilidade o *Presto* obtém melhores resultados que o Impala quando questionando o *Kudu* utilizando ou não replicação.

5.2 Cenário 2 – SSB

Esta secção contém os resultados obtidos nos testes realizados com o conjunto de dados SSB divididos pelo fator de escala (10GB, 30GB, 100GB), assim como todas as apreciações consideradas pertinentes.

Conforme os resultados obtidos no fator de escala 10 (Tabela 8) verifica-se que a combinação com melhores resultados é o armazenamento *Parquet* questionado pelo *Impala,* com um tempo de execução total de 21,34s. De salientar que esta combinação obtém melhor desempenho em todas a *queries*.

Focando a análise no *Kudu,* verifica-se que a utilização de replicação melhora o seu desempenho. Questionado pelo *Presto*, o tempo total de execução diminui em 3% quando se utilizam três replicações, tendo melhor desempenho em 6 das 13 *queries*. No caso da utilização do *Impala,* os resultados são mais significativos, a utilização de três replicações melhora em 13% o tempo total de execução e, neste caso, obtém melhor desempenho em 12 *queries*.

Tabela 8 - Resultados SSB SF=10

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Q1.1	3,95	8,61	3,85	4,70	**1,01**	1,42	1,41
Q1.2	3,52	5,16	2,64	3,00	**1,02**	1,38	1,26
Q1.3	3,51	4,97	2,60	2,94	**0,96**	1,33	1,20
Q2.1	10,26	16,59	20,61	20,90	**1,73**	6,05	5,57
Q2.2	10,00	14,50	19,27	19,82	**1,60**	3,78	3,13
Q2.3	10,23	14,02	19,15	19,49	**1,59**	3,79	3,04
Q3.1	9,12	14,01	19,20	19,72	**1,82**	17,58	18,76
Q3.2	7,09	11,97	19,32	18,72	**1,82**	16,75	15,45
Q3.3	6,88	10,54	18,25	17,38	**1,72**	16,96	13,75
Q3.4	6,64	10,71	18,56	17,36	**1,47**	1,81	1,74
Q4.1	22,35	30,44	36,27	33,36	**2,26**	20,87	17,05
Q4.2	11,78	18,12	28,03	26,22	**2,32**	18,88	15,07
Q4.3	11,87	17,65	27,92	25,50	**2,02**	18,70	15,13
Total	117,20	177,28	235,67	229,11	**21,34**	129,29	112,56

Outro dado importante retirado dos resultados obtidos, é o facto de o desempenho do *Kudu* piorar consideravelmente quando questionado pelo *Presto* em comparação com o *Impala*, sendo o tempo de execução total das *queries* quase duas vezes superior. Aliás, é de salientar o facto de o *Kudu*, quando questionado pelo *Presto*, obter os piores resultados de todas as combinações em análise.

Tendo em consideração exclusivamente os resultados obtidos pelo *Presto*, o armazenamento com melhor desempenho é o *Hive*. O seu tempo total de execução é praticamente metade do tempo de execução do *Kudu*. Comparando estes resultados do *Hive* com os resultados obtidos pelo *Impala*, percebe-se que apesar de ter um desempenho muito inferior ao *Parquet*, o seu desempenho é muito similar ao desempenho do *Kudu*.

Analisando resultados obtidos no fator de escala 30 (Tabela 9) verifica-se que neste caso os resultados são ainda mais esclarecedores do que no fator de escala 10. A combinação com melhor tempo total de execução é novamente armazenamento *Parquet* questionado pelo *Impala,* com um tempo de execução total de 35,72s, o que corresponde a uma redução de 89% no tempo total de execução relativamente ao melhor desempenho do *Kudu*. O armazenamento Parquet obtém, também, melhor desempenho em todas as *queries*.

Tendo em conta o impacto causado pela replicação no desempenho do *Kudu*, este mostra-se irrelevante, com alterações no tempo total de execução inferiores a 1%, tanto utilizando o *Presto* como utilizando o *Impala* na execução das *queries*. Relativamente ao desempenho obtido em cada *query*, quando questionado pelo *Presto,* o *Kudu* com três replicações obtém melhor desempenho em 7 das 13 *queries*. Quando questionado pelo *Impala* esse número reduz para 5 das 13 *queries*.

Tabela 9 - Resultados SSB SF=30

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Q1.1	9,62	9,15	11,48	8,60	**1,56**	2,79	2,65
Q1.2	6,66	8,43	5,93	5,54	**1,46**	2,41	2,29
Q1.3	6,39	8,63	5,64	5,55	**1,42**	2,25	2,26
Q2.1	28,70	35,12	64,60	58,35	**2,62**	9,19	10,14
Q2.2	27,22	34,66	60,88	55,68	**2,55**	7,23	7,52
Q2.3	26,60	35,00	57,80	56,10	**2,49**	6,99	7,24
Q3.1	26,56	33,65	57,73	61,47	**2,95**	53,51	51,96
Q3.2	21,06	29,26	53,30	55,00	**2,89**	48,95	48,09
Q3.3	20,34	27,66	50,83	55,76	**2,72**	47,78	47,32
Q3.4	20,48	25,87	53,18	53,48	**2,13**	3,09	3,13
Q4.1	71,26	84,19	100,96	104,62	**4,85**	55,13	55,68
Q4.2	35,00	48,00	82,23	84,01	**5,08**	48,03	49,04
Q4.3	34,65	48,59	78,73	78,17	**3,01**	48,00	48,23
Total	334,54	428,20	683,29	682,33	**35,72**	335,35	335,55

Mais uma vez o *Kudu* mostra-se muito dependente do *Impala*, voltando a obter os piores resultados de todas as ferramentas em análise quando questionado pelo *Presto*. Comparativamente com

o *Hive*, que obteve o melhor desempenho entre as ferramentas questionadas pelo *Presto*, o tempo total de execução das *queries* obtido pelo *Kudu* é mais de duas vezes superior. Quando a comparação é feita com o *Parquet* questionado pelo *Impala* esse tempo é mais de 19 vezes superior.

Relativamente ao fator de escala 100 (Tabela 10) pode-se verificar que as Q4.1 Q4.2 Q4.3 não foram executada pelo *Presto*, dado que, devido à complexidade das *queries,* o *Presto* não dispõe de memória suficiente para as executar com sucesso.

Tabela 10 - Resultados SSB SF=100

	Presto				**Impala**		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Q1.1	24,09	33,18	51,34	27,73	**6,71**	8,73	11,18
Q1.2	17,12	26,19	20,89	16,55	**2,95**	6,08	6,46
Q1.3	17,42	24,96	19,38	16,85	**2,91**	5,81	5,91
Q2.1	92,52	123,84	240,23	195,63	**9,37**	26,40	26,78
Q2.2	88,90	119,02	255,54	210,43	**5,65**	23,83	23,33
Q2.3	87,19	121,64	207,10	192,13	**5,71**	22,66	22,80
Q3.1	79,79	121,80	179,64	198,48	**8,74**	230,67	235,88
Q3.2	61,28	97,66	171,80	189,71	**6,92**	216,57	227,04
Q3.3	57,81	93,22	151,18	195,52	**6,37**	216,99	224,23
Q3.4	58,19	93,33	166,31	180,80	**4,69**	13,45	17,16
Q4.1	ERRO	ERRO	ERRO	ERRO	**18,39**	372,79	363,11
Q4.2	ERRO	ERRO	ERRO	ERRO	**21,26**	369,43	351,26
Q4.3	ERRO	ERRO	ERRO	ERRO	**7,25**	369,67	346,87
Total	584,32	854,83	1463,41	1423,82	**106,92**	1883,07	1862,00

De acordo com o que se verifica no fator de escala 30, a combinação com melhor tempo total de execução é novamente armazenamento *Parquet* questionado pelo *Impala,* com um tempo de execução total de 106,92s, o que corresponde a uma redução de 94% no tempo total de execução relativamente ao melhor desempenho do *Kudu*. A combinação *Parquet Impala* apresenta também melhor desempenho em todas as *queries,* sendo o seu tempo de execução total inferior, mesmo incluindo as *queries* do grupo 4 (Q4.1, Q4.2, Q4.3) que não são executadas com o *Presto*.

Relativamente ao impacto da replicação no *Kudu* é percetível que apesar de uma ligeira melhoria no tempo total de execução, os resultados apresentam pouca relevância uma vez que, a variação existente é reduzida e por vezes contraditória, visto que em alguns casos utilizando o Presto a replicação melhora o desempenho enquanto utilizando o Impala piora e vise versa.

De modo a facilitar a comparação de resultados, dado que as *queries* do grupo 4 não foram executadas pelo *Presto,* a Tabela 11 apresenta o tempo total de execução das *queries* de cada combinação assim como o mesmo excluindo as *queries* do grupo 4.

Tabela 11 - Total SSB SF=100 vs. Total SSB SF=100 excluindo Q4

	Presto				Impala		
	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Parquet	Kudu s/ replicação	Kudu 3 replicações
Tempo total de execução	584,32	854,83	1463,41	1423,82	**106,92**	1883,07	1862,00
Tempo total de execução excluído as Q4	584,32	854,83	1463,41	1423,82	**60,02**	771,18	800,77

Tendo em conta os resultados da Tabela 11, é possível perceber, como era esperado, que a combinação com melhor tempo total de execução excluindo as *queries* do grupo 4 é armazenamento *Parquet* questionado pelo *Impala,* com um tempo de execução total de 60,02s, o que corresponde a uma redução de 92% relativamente ao melhor desempenho do *Kudu*.

De destacar que o armazenamento *Hive* tem o melhor desempenho entre os armazenamentos questionados pelo *Presto*, conseguindo inclusive melhor desempenho que o *Kudu* questionado pelo Impala, obtendo uma redução no tempo total de execução de 24%.

Em conformidade com o que se verifica nos outros fatores de escala, o *Kudu* apresenta uma grande dependência do *Impala*. Quando questionado pelo *Presto* o aumento no tempo total de execução excluído as *queries* do grupo 4, sofre um aumento de 90% no caso da não utilização de replicação e de 78% utilizando três replicações.

Focando a análise na escalabilidade das ferramentas, e à semelhança do que acontece no conjunto de dados TPH-C, as Tabela 12 Tabela 13 apresentam a comparação de resultados obtidos entre os SF 10 e 30 e os SF 10 e 100 pelas diversas ferramentas quando questionadas pelo *Presto* e *Impala* respetivamente. Mais uma vez os valores a negrito representam os casos em que se verificou um aumento de desempenho associado ao aumento do fator de escala.

Como referido anteriormente, o *Presto* não conseguiu executar o conjunto de *queries* 4 com sucesso no SF 100, pelo que os rácios entre o SF 10 e 100 do conjunto de *queries* 4 não se encontram disponíveis na Tabela 12, assim sendo o rácio total apresentado nestes casos exclui o tempo de execução do conjunto de *queries* 4 no SF 10.

Observando a Tabela 12, é possível perceber que a ferramenta com melhores resultados de escalabilidade é o *Parquet*, obtendo melhorias de desempenho em todas as *queries* na comparação com ambos os fatores de escala.

Tabela 12 - Escalabilidade SSB usando *Presto*

	Presto							
	Hive		Parquet		Kudu s/ replicação		Kudu 3 replicações	
	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100
Q1.1	**2,4**	**6,1**	**1,1**	**3,9**	3,0	13,3	**1,8**	**5,9**
Q1.2	**1,9**	**4,9**	**1,6**	**5,1**	**2,3**	**7,9**	**1,8**	**5,5**
Q1.3	**1,8**	**5,0**	**1,7**	**5,0**	**2,2**	**7,5**	**1,9**	**5,7**
Q2.1	**2,8**	**9,0**	**2,1**	**7,5**	3,1	11,7	**2,8**	**9,4**
Q2.2	**2,7**	**8,9**	**2,4**	**8,2**	3,2	13,3	**2,8**	10,6
Q2.3	**2,6**	**8,5**	**2,5**	**8,7**	3,0	10,8	**2,9**	**9,9**
Q3.1	**2,9**	**8,7**	**2,4**	**8,7**	3,0	**9,4**	3,1	10,1
Q3.2	3,0	**8,6**	**2,4**	**8,2**	**2,8**	**8,9**	**2,9**	10,1
Q3.3	3,0	**8,4**	**2,6**	**8,8**	**2,8**	**8,3**	3,2	11,2
Q3.4	3,1	**8,8**	**2,4**	**8,7**	**2,9**	**9,0**	3,1	10,4
Q4.1	3,2	-	**2,8**	-	**2,8**	-	3,1	-
Q4.2	3,0	-	**2,6**	-	**2,9**	-	3,2	-
Q4.3	**2,9**	-	**2,8**	-	**2,8**	-	3,1	-
Total	**2,9**	**8,2**	**2,4**	**7,7**	**2,9**	10,2	3,0	**9,9**

Relativamente ao *Kudu* os resultados não são muito esclarecedores, na comparação entre os SF 10 e 30 o *Kudu* sem replicação obtém um melhor desempenho total, mas na comparação com o SF 100 a utilização de 3 replicações significa um melhor desempenho. De realçar que as diferenças entre os resultados obtidos são muito reduzidas e que em ambos os casos, utilizando ou não utilizando replicação, o desempenho é dos SF 30 e 100 é semelhante ao do SF 10.

Na Tabela 13, em que são apresentados os resultados de escalabilidade do *Impala*, é possível perceber que a ferramenta com melhores resultados é o *Parquet* com melhor desempenho em todas as *queries* nas duas comparações à exceção da Q1.3 na comparação entre o SF 10 e o SF 100 que obtém o mesmo desempenho.

Neste caso em que é utilizado o *Impala*, verifica-se que o *Kudu* obtém melhores resultados sem a utilização de replicação. Na comparação entre os SF 10 e 30 observa-se um melhor desempenho em todas as queries com a exceção da Q3.1 que tem o mesmo desempenho. Quando a comparação é feita entre os SF 10 e 100 verifica-se uma perda significativa de desempenho no *Kudu* (46% sem a utilização de replicação e 65% utilizando três replicações) o que indicia que um aumento acentuado do volume de dados reduz o desempenho do *Kudu*.

Tabela 13 - Escalabilidade SSB usando *Impala*

	Impala					
	Parquet		Kudu s/ replicação		Kudu 3 replicações	
	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100	SF=10 vs. SF=30	SF=10 vs. SF=100
Q1.1	**1,5**	**6,7**	**2,0**	**6,1**	**1,9**	**7,9**
Q1.2	**1,4**	**2,9**	**1,8**	**4,4**	**1,8**	**5,1**
Q1.3	**1,5**	3,0	**1,7**	**4,4**	**1,9**	**4,9**
Q2.1	**1,5**	**5,4**	**1,5**	**4,4**	**1,8**	**4,8**
Q2.2	**1,6**	**3,5**	**1,9**	**6,3**	**2,4**	**7,5**
Q2.3	**1,6**	**3,6**	**1,8**	**6,0**	**2,4**	**7,5**
Q3.1	**1,6**	**4,8**	3,0	13,1	**2,8**	12,6
Q3.2	**1,6**	**3,8**	**2,9**	12,9	3,1	14,7
Q3.3	**1,6**	**3,7**	**2,8**	12,8	3,4	16,3
Q3.4	**1,4**	**3,2**	**1,7**	**7,4**	**1,8**	**9,9**
Q4.1	**2,1**	**8,1**	**2,6**	17,9	3,3	21,3
Q4.2	**2,2**	**9,2**	**2,5**	19,6	3,3	23,3
Q4.3	**1,5**	**3,6**	**2,6**	19,8	3,2	22,9
Total	**1,7**	**5,0**	**2,6**	14,6	3,0	16,5

Comparando o *Presto* com o *Impala*, é possível observar que relativamente à escalabilidade os melhores resultados são obtidos pelo Impala a questionar o *Parquet*. No entanto relativamente ao *Kudu* o *Presto* consegue melhores resultados de escalabilidade.

5.3 Síntese de resultados

Ao longo da análise de ambos os cenários, é possível perceber a grande dependência que o *Kudu* tem relativamente ao *Impala*, quando utilizado em conjunto com o *Presto* observa-se um aumento muito significativo no tempo de execução de *queries* chegando esse tempo a ser três vezes superior.

Com os testes realizados, é possível concluir que o *Kudu* não é uma alternativa viável às ferramentas comummente utilizadas. Em sistemas que disponham do *Impala* o seu desempenho é inferior ao do *Parquet*, essa diferença de desempenho é muito significativa quando são utilizados modelos de dados normalizados. Da mesma forma, em sistemas que não disponham do *Impala* o desempenho do *Kudu* mostra-se consideravelmente pior, como já referido anteriormente, sendo preferível a utilização de ferramentas como o *Hive* com armazenamento ORC ou o *Parquet*.

Em modo geral é também possível concluir que a utilização do *Kudu* com replicação, não tem um impacto relevante no seu desempenho, apesar de algumas melhorias em quantidades de dados mais

reduzidas, essas melhorias não se verificam em conjuntos maiores. Apesar de a replicação multiplicar o espaço utilizado pelo *Kudu* e não se mostrar relevante na melhoria do seu desempenho, é importante salientar que a sua utilização é imprescindível para manter a tolerância à falha do sistema.

De destacar também, o completo domínio do armazenamento *Parquet* questionado pelo *Impala* quando são utilizadas tabelas normalizadas, obtendo o melhor tempo de processamento em todas as *queries* realizadas no cenário 2. Com o aumento do volume de dados é possível observar redução do tempo total de processamento das *queries* na ordem dos 90%.

Comparando as ferramentas *Impala* e *Presto*, é facilmente percetível que o *Impala* tem diversas vantagens relativamente ao *Presto*, para além do melhor desempenho demostrado foi capaz também de concluir com sucesso todas as queries executadas, ao contrario do *Presto* que não concluiu algumas das *queries* quando se verificou um aumento do volume de dados. A única vantagem observada pelo *Presto*, prende-se com a sua versatilidade, uma vez que consegue executar *queries* em todos os armazenamentos utilizados, ao contrário do *Impala* que não suporta o formato de armazenamento ORC.

Tendo em conta a Tabela 14, que contem o tamanho das tabelas dos diferentes formatos em ambos os cenários para o fator de escala 100, é possível perceber que o modelo TPC-H desnormalizado ocupa quase 8 vezes mais espaço que o modelo SSB em todos os armazenamentos, o que pode influenciar significativamente o tempo de leitura em disco das *queries*.

Tabela 14 - Tamanho das tabelas SF=100

Cenário 1 (TPC-H Desnormalizado)				**Cenário 2 (SSB)**			
Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações	Hive	Parquet	Kudu s/ replicação	Kudu 3 replicações
100GB	170GB	698GB	-	12,7GB	21,5GB	88,2GB	264,6GB

Relativamente à Tabela 14 e ao armazenamento *Kudu* em particular, é necessário destacar que apesar de não terem sido realizados os testes do cenário 1 com 3 replicações por falta de espaço de armazenamento o tamanho espectável da tabela seria 2094GB, cerca de 2TB, uma vez que corresponde ao triplo do tamanho da tabela sem replicação. De destacar também a diferença entre o tamanho das tabelas *Kudu* e o tamanho das tabelas dos restantes formatos (cerca de sete vezes maior que *Hive* e quatro vezes maior que *Parquet*), este facto deve-se à não utilização de nenhuma compressão de dados.

Para comparar os cenários 1 e 2, de modo a entender o impacto por um lado da desnormalização e por outro da utilização de um modelo em estrela que implica a utilização de *joins* durante a execução de *queries*, é necessário encontrar um conjunto de fatores comuns que possam ser comparados. Não sendo possível comparar o tempo de execução de *queries*, dado que as *queries* utilizadas são diferentes,

é possível identificar como fator comum o tamanho ocupado pelas tabelas, a escalabilidade na realização de *queries*, e a complexidade no carregamento dos dados. Em todos estes fatores o cenário 2 apresenta vantagens. O tamanho ocupado pelas tabelas é cerca de um oitavo do seu homólogo no cenário 1. Relativamente à escalabilidade é constatado um melhor desempenho de todas as ferramentas no cenário 2, apesar de em termos de processamento as *queries* do cenário 2 serem mais exigentes, uma vez que envolvem a utilização de *joins,* a leitura de maiores quantidades de dados a partir do disco revela-se penosa no cenário 1. Quando o termo de comparação é o carregamento dos dados, no cenário 2 é apenas necessário realizar algumas conversões de dados dependendo do formato de armazenamento, não sendo necessárias modificações complexas, no cenário 1 é necessário realizar todos os *joins* entre todas as tabelas, um processo bastante exige em termos de processamento. Assim sendo, é possível afirmar que a utilização de um modelo em estrela, e consequentemente a realização de queries envolvendo *joins*, é uma alternativa viável à utilização de um modelo desnormalizado.

Ao longo da realização dos testes foi possível identificar as seguintes vantagens na utilização do *Kudu*:

- O *Kudu* é uma ferramenta que pode ser utilizada de forma independente, não sendo necessária a instalação de outros componentes normalmente presentes num *cluster Big Data* como o HDFS ou o *Yarn*. Uma vez que dispõe do seu próprio sistema de ficheiros e gestor de recursos, é possível, depois de instalado e configurado, ser utilizado através da sua API ou de qualquer ferramenta compatível que disponibilize uma linguagem SQL-*on-Hadoop;*
- O fator de replicação é uma propriedade de cada tabela, sendo possível definir diferentes fatores para diferentes tabelas, possibilitando, por exemplo, a utilização de um fator maior em tabelas que contenham dados mais sensíveis.

Como desvantagens foi possível identificar os seguintes aspetos:

- Como se trata de uma ferramenta relativamente recente e pouco explorada, a documentação disponível é reduzida, o que torna difícil a investigação de configurações e/ou erros mais específicos;
- O *Kudu* não disponibiliza nenhuma consola ou linguagem SQL-*on-Hadoop* sendo necessária a utilização de outras ferramentas que suportem estas operações como o *Impala* ou o *Presto*;

- Com o objetivo de reduzir o tempo de processamento, é recomendado que os dados armazenados no *Kudu* não sejam comprimidos, o que leva a um aumento muito considerável de espaço ocupado pelos dados comparativamente com outras ferramentas de armazenamento;
- O *Kudu* não disponibiliza nenhum tipo de dados para armazenamento de datas, tornando a sua manipulação mais complicada, as datas devem ser convertidas em *timestamp* e armazenadas como valores inteiros.
- O fator de replicação de uma tabela só pode ser definido no momento da sua criação, assim sendo para alterar o seu fator de replicação é necessário criar uma nova tabela com o fator desejado e posteriormente copiar todos os dados.

6. CONCLUSÃO

A presente dissertação começa por apresentar, de forma detalhada, a motivação que leva à sua realização, os seus objetivos e a abordagem metodológica utilizada.

De seguida são explorados os principais conceitos associados ao termo *Big Data*, as suas características, as suas vantagens competitivas e os desafios e problemas a si associados. São também abordados os principais paradigmas relacionadas com a temática, tais como o *NoSQL* e o *MapReduce*. Após o enquadramento conceptual são identificadas e descritas as principais tecnologias envolvidas no processamento de *Big Data*, com especial destaque para a ferramenta em estudo na dissertação, o *Kudu*. É também descrito de uma forma geral o ecossistema Hadoop e as suas principais ferramentas, assim como todas as ferramentas e formatos de armazenamento utilizadas nesta investigação, nomeadamente *Hive*, *Presto*, *Impala,* ORC e *Parquet.*

Posteriormente é detalhado o ambiente em que foram realizados os testes, com a descrição da infraestrutura utilizada, os conjuntos de dados utilizados (TPC-H desnormalizado e SSB) e todas as alterações efetuadas aos conjuntos originais, bem como os dois cenários em que os testes são realizados.

Por fim são apresentados os resultados dos testes realizados. Estes resultados encontram-se divididos em duas secções distintas que expõem os resultados e as considerações associadas aos mesmo. Nestas secções são avaliados os tempos de execução de *queries* dos diferentes formatos de armazenamento e diferentes ferramentas de consulta em três fatores de escala distintos, 10GB, 30GB e 100GB. É também avaliada a escalabilidade das ferramentas em estudo. A apresentação de resultados termina com as considerações gerais às ferramentas utilizadas, uma comparação dos dois cenários de testes que visa avaliar a viabilidade da utilização, num *Data Warehouse*, de um modelo em estrela em detrimento de um modelo desnormalizado e um conjunto de vantagens e de desvantagens identificadas na utilização do *Kudu*.

6.1 Trabalho realizado

Esta dissertação teve como objetivo estudar a ferramenta *Kudu*, nomeadamente compreender a tecnologia e os principais contextos de utilização, avaliar o seu desempenho utilizando *benchmarks* de referência e caracterizar as vantagens e desvantagens da sua utilização na análise de vastas quantidades de dados que sofrem diversas alterações ao longo do tempo. Os objetivos propostos foram concretizados

na sua totalidade, com a ressalva que foram apenas caracterizadas as vantagens e desvantagens na análise de vastas quantidades de dados estáticos, tendo sido:

- Realizada a revisão de literatura dos principais conceitos associados à temática;
- Implementado um *cluster* na *cloud* de raiz, com a sua configuração e instalação de todas as ferramentas necessárias ao seu funcionamento;
- Compreendido o *Kudu* e as suas principais funcionalidades;
- Compreendido todas as restantes ferramentas utilizadas na dissertação;
- Realizados os *benchmarks* às ferramentas utilizadas;
- Avaliados os resultados obtidos;
- Descrito um conjunto de vantagens e de desvantagens na utilização do *Kudu*.

6.2 Dificuldade e Limitações

Ao longo da dissertação foram sentidas diversas dificuldades, começando pela pouca documentação associada ao *Kudu*, o que dificultou bastante a compreensão da ferramenta.

Posteriormente a necessidade de utilização do *Impala* implicou a implementação de um *cluster* na *cloud*, e consequentemente a instalação de um *cluster* Cloudera. Este passo mostrou-se particularmente difícil e desafiante, uma vez que foi o primeiro contacto com um *cluster* em *cloud*, com a instalação e configuração de um *cluster* e com a utilização de uma *stack* Cloudera em simultâneo.

Por fim, a principal limitação, associada à infraestrutura do *cluster,* nomeadamente o pouco espaço de armazenamento disponível, que não permitiu a realização de todos os testes desejados, nem a utilização de conjuntos de dados maiores. Esta limitação obrigou a constantes manobras de gestão de armazenamento, uma vez que o espaço limitado tornou imperativo que as tabelas utilizadas fossem removidas após a sua utilização para dar lugar a outras. Em alguns casos esta limitação obrigou a novos carregamentos das mesmas tabelas para execução de tarefas esquecidas ou que se revelaram mal executadas.

6.3 Trabalho Futuro

Sendo o *Kudu* uma ferramenta recente é possível identificar diversos assuntos pertinentes para serem abordados futuramente, nomeadamente:

- Partições de tabelas: estudar a forma como a partição das tabelas afeta o desempenho do *Kudu*, definindo um conjunto de boas praticas de utilização em diferentes contextos;
- Compressão de dados: estudar o real impacto da compressão dos dados armazenados no *Kudu*, avaliando a diferença de armazenamento utilizado, assim como o tempo de processamento de *queries* e de leitura/escrita em disco;
- Comparação com arquitetura mista: comparar o desempenho do *Kudu* com uma arquitetura mista que utilize, por exemplo, *Hive* e *Hbase*, de modo a testar não só a execução de *queries*, mas também atualização de registos, inserção de registos e execução de *queries* em simultâneo com atualizações e inserções de registos;
- Escalabilidade: estudar de forma mais alargada a escalabilidade, em conjuntos de dados maiores e num *cluster* também maior, de modo a estudar não só a variação dos dados, mas também do tamanho do *cluster.*

Esta página foi deixada intencionalmente em branco.

REFERÊNCIAS BIBLIOGRÁFICAS

Apache. (2017). Apache Hadoop YARN. Retrieved from http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Apache Software Foundation. (2015). Apache™ Orc. Retrieved January 25, 2018, from https://orc.apache.org/docs/index.html

Apache Software Foundation. (2016). Apache Kudu - Fast Analytics on Fast Data. Retrieved from https://kudu.apache.org/

Barham, H. (2015). Achieving Competitive Advantage Through Big Data : A Literature Review.

Borthakur, D. (2013). HDFS Architecture Guide. Retrieved from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *Computer*, *45*(2), 23–29. https://doi.org/10.1109/MC.2012.37

Charles, V., & Gherman, T. (2013). Achieving Competitive Advantage Through Big Data. Strategic implications. *Middle East Journal of Scientific Research*, *16*(8), 1069–1074. https://doi.org/10.5829/idosi.mejsr.2013.16.08.11811

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, *19*(2), 171–209. https://doi.org/10.1007/s11036-013-0489-0

Cloudera. (2016). Apache Kudu ™. Retrieved from https://www.cloudera.com/content/dam/www/marketing/resources/datasheets/cloudera-kudu-datasheet.pdf

Costa, C., & Santos, M. Y. (2017a). Big Data: State-of-the-art concepts, techniques, technologies, modeling approaches and research challenges. *IAENG International Journal of Computer Science*, *44*(3), 285–301.

Costa, C., & Santos, M. Y. (2017b). The SusCity Big Data Warehousing Approach for Smart Cities. *Proceedings of the 21st International Database Engineering & Applications Symposium on - IDEAS 2017*, 264–273. https://doi.org/10.1145/3105831.3105841

Costa, C., & Santos, M. Y. (2018). Evaluating several design patterns and trends in big data warehousing systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10816 LNCS*, 459–473. https://doi.org/10.1007/978-3-319-91563-0_28

DeRoos, D., Zikopoulos, P. C., Melnyk, R. B., Brown, B., & Coss, R. (2014). *Hadoop for dummies*. https://doi.org/10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C

Floratou, A. (2014). SQL-on-Hadoop : Full Circle Back to Shared-Nothing Database Architectures, 1295–1306.

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007

Grolinger, K., Hayes, M., Higashino, W. A., L'Heureux, A., Allison, D. S., & Capretz, M. A. M. (2014). Challenges for MapReduce in Big Data. In *2014 IEEE World Congress on Services* (pp. 182–189). https://doi.org/10.1109/SERVICES.2014.41

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. In *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011* (pp. 363–366). https://doi.org/10.1109/ICPCA.2011.6106531

Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology

tutorial. *IEEE Access*, *2*, 652–687. https://doi.org/10.1109/ACCESS.2014.2332453

Inoubli, W., Aridhi, S., Mezni, H., & Jung, A. (2016). An Experimental Survey on Big Data Frameworks. Retrieved from http://arxiv.org/abs/1610.09962

Jin, X., Wah, B. W., Cheng, X., & Wang, Y. (2015). Significance and Challenges of Big Data Research. *Big Data Research*, *2*(2), 59–64. https://doi.org/10.1016/j.bdr.2015.01.006

Khan, M. A. U. D., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education - "Engineering Education: Industry Involvement and Interdisciplinary Trends", ASEE Zone 1 2014*. https://doi.org/10.1109/ASEEZone1.2014.6820689

Kornacker, M., Behm, A., Bittorf, V., Bobrovytsky, T., Ching, C., Choi, A., ... Yoder, M. (2015). Impala: A Modern, Open-Source SQL Engine for Hadoop. *Cidr*. Retrieved from http://impala.io/

Krishnan, K. (2013). *Data Warehousing in the Age of Big Data*. Elsevier Inc.

Lipcon, T., Alves, D., Burkert, D., Cryans, J., Dembo, A., Percy, M., ... Wang, A. (2015). Kudu : Storage for Fast Analytics on Fast Data ∗. *Draft*.

NOSQL Databases. (2018). Retrieved January 25, 2018, from http://nosql-database.org/

O'Neil, P., O'Neil, B., & Chen, X. (2009). Star Schema Benchmark.

Patgiri, R., & Ahmed, A. (2017). Big Data: The V's of the Game Changer Paradigm. In *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016* (pp. 17–24). https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0014

Patil, P. R., & Kshirsagar, V. (2018). Efficient time compression earthquake database using hadoop Hive ORC format. *Proceedings of the 2017 International Conference on Intelligent Computing and Control Systems, ICICCS 2017*, *2018–Janua*, 1361–1364. https://doi.org/10.1109/ICCONS.2017.8250691

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Plase, D., Niedrite, L., & Taranovs, R. (2017). Accelerating data queries on Hadoop framework by using compact data formats. *2016 IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering, AIEEE 2016 - Proceedings*. https://doi.org/10.1109/AIEEE.2016.7821807

Qin, X., Chen, Y., Chen, J., Li, S., Liu, J., & Zhang, H. (2017). The Performance of SQL-on-Hadoop Systems - An Experimental Study. *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, 464–471. https://doi.org/10.1109/BigDataCongress.2017.68

Raj, E. D., Nivash, J. P., Nirmala, M., & Dhinesh Babu, L. D. (2015). A scalable cloud computing deployment framework for efficient MapReduce operations using Apache YARN. In *2014 International Conference on Information Communication and Embedded Systems, ICICES 2014*. https://doi.org/10.1109/ICICES.2014.7033945

Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Martinho, B. A., Lima, F. V., & Costa, E. (2017). Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware. *Proceedings of the 21st International Database Engineering & Applications Symposium on - IDEAS 2017*, 242–252. https://doi.org/10.1145/3105831.3105842

Shafer, J., Rixner, S., & Cox, A. L. (2010). The Hadoop distributed filesystem: Balancing portability and performance. In *ISPASS 2010 - IEEE International Symposium on Performance Analysis of Systems and Software* (pp. 122–133). https://doi.org/10.1109/ISPASS.2010.5452045

Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V. (2016). Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research*, pp. 263–286. https://doi.org/10.1016/j.jbusres.2016.08.001

Sogodekar, M., Pendey, S., Tupkari, I., & Manekar, A. (2016). Big Data Analytics Hadoop and Tools.

TPC Benchmark H. (2017), 1–137. Retrieved from http://www.tpc.org/tpch/default.asp

Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. In *Proceedings - RoEduNet IEEE International Conference*. https://doi.org/10.1109/RoEduNet.2011.5993686

Zicari, R. V. (2014). Big Data: Challenges and Opportunities. *Big Data Computing*, 103–128. https://doi.org/10.1007/978-3-319-13206-8_1

Zikopoulos, P., Eaton, C., DeRoos, D., Deutsch, T., & Lapis, G. (2011). *Understanding Big Data*. The McGraw-Hill Companies.

Esta página foi deixada intencionalmente em branco.

APÊNDICES

Apêndice 1 – *Queries* TPC-H adaptadas à versão desnormalizada

Query 1

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order
from lineitem_1
where l_shipdate <= date '1998-12-01' - interval '90' day
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

Query 2

```
select s_acctbal, s_name, s_n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from lineitem_1
where p_size = 15 and p_type like '%BRASS' and s_r_name = 'EUROPE' and ps_supplycost = (select min(ps_supplycost)
from lineitem_1 where p_size = 15 and p_type like '%BRASS' and s_r_name = 'EUROPE')
group by s_acctbal, s_name, s_n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
order by s_acctbal desc, s_n_name, s_name, p_partkey;
```

Query 3

```
select o_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority
from lineitem_1
where c_mktsegment = 'BUILDING' and o_orderdate < date '1995-03-15' and l_shipdate > date '1995-03-15'
group by o_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate;
```

Query 4

```
select o_orderpriority, count (distinct o_orderkey) as order_count
from lineitem_1
where o_orderdate >= date '1993-07-01' and o_orderdate < date '1993-07-01' + interval '3' month and l_commitdate < l_receiptdate
group by o_orderpriority
order by o_orderpriority;
```

Query 5

```
select c_n_name, sum (l_extendedprice * (1 - l_discount)) as revenue
from lineitem_1
where c_nationkey = s_nationkey and s_nationkey = c_nationkey and c_r_name = 'ASIA' and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year
group by c_n_name order by revenue desc;
```

Query 6

```
select sum (l_extendedprice * l_discount) as revenue
from lineitem_1
where l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and l_quantity < 24;
```

Query 7

select sum (l_extendedprice * (1 - l_discount)) as revenue, s_n_name, c_n_name, extract (year from l_shipdate) as l_year
from lineitem_1
where ((s_n_name = 'FRANCE' and c_n_name = 'GERMANY') or (s_n_name = 'GERMANY' and c_n_name = 'FRANCE')) and l_shipdate between date '1995-01-01' and date '1996-12-31'
group by s_n_name, c_n_name, extract (year from l_shipdate)
order by s_n_name, c_n_name, extract (year from l_shipdate);

Query 8

select o_year, sum (case when nation = 'BRAZIL' then volume else 0 end) / sum(volume) as mkt_share
from (**select** extract (year from o_orderdate) as o_year, l_extendedprice * (1-l_discount) as volume, s_n_name as nation **from** lineitem_1 **where** c_r_name = 'AMERICA' and o_orderdate between date '1995-01-01' and date '1996-12-31' and p_type = 'ECONOMY ANODIZED STEEL') as all_nations
group by o_year
order by o_year;''

Query 9

select nation, o_year, sum(amount) as sum_profit
from (**select** s_n_name as nation, extract (year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount **from** lineitem_1 **where** p_name like '%green%') as profit
group by nation, o_year
order by nation, o_year desc;

Query 10

select c_custkey, c_name, sum (l_extendedprice * (1 - l_discount)) as revenue, c_acctbal, c_n_name, c_address, c_phone, c_comment
from lineitem_1
where o_orderdate >= date '1993-10-01' and o_orderdate < date '1993-10-01' + interval '3' month and l_returnflag = 'R'
group by c_custkey, c_name, c_acctbal, c_phone, c_n_name, c_address, c_comment
order by revenue desc limit 20;''

Query 11

select ps_partkey, sum (ps_supplycost * ps_availqty) as value
from lineitem_1
where s_n_name = 'GERMANY'
group by ps_partkey
having sum (ps_supplycost * ps_availqty) > (**select** sum (ps_supplycost * ps_availqty) * 0.00001 **from** lineitem_1 **where** s_n_name = 'GERMANY')
order by value desc;

Query 12

select l_shipmode, sum (case when o_orderpriority = '1-URGENT'or o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count, sum (case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count
from (**select** l_shipmode, o_orderpriority, o_orderkey **from** lineitem_1 **where** l_shipmode in ('MAIL', 'SHIP') and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >= date '1994-01-01' and l_receiptdate < date '1994-01-01' + interval '1' year **group by** l_shipmode, o_orderpriority, o_orderkey)
group by l_shipmode order by l_shipmode;

Query 13

select c_count, count (*) as custdist
from (**select** c_custkey, count(o_orderkey) **from** lineitem_1 **where** o_comment not like '%:special%requests%' **group by** c_custkey) as c_orders (c_custkey, c_count)
group by c_count

order by custdist desc, c_count desc;

Query 14

select 100.00 * sum (case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount) else 0 end) / sum (l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem_1
where l_shipdate >= date '1995-09-01' and l_shipdate < date '1995-09-01' + interval '1' month;

Query 15

select s_suppkey, s_name, s_address, s_phone, total_revenue
from (**select** s_suppkey, s_name, s_address, s_phone, sum (l_extendedprice * (1 - l_discount)) as total_revenue **from** lineitem_1 **where** l_shipdate >= date '1996-01-01' and l_shipdate < date '1996-01-01' + interval '3' month **group by** s_suppkey, s_name, s_address, s_phone **order by** s_suppkey)
where total_revenue= (**select** max(total_revenue) **from** (select s_suppkey, sum (l_extendedprice * (1 - l_discount)) as total_revenue **from** lineitem_1 **where** l_shipdate >= date '1996-01-01' and l_shipdate < date '1996-01-01' + interval '3' month **group by** s_suppkey));

Query 16

select p_brand, p_type, p_size, count (distinct s_suppkey) as supplier_cnt
from lineitem_1
where p_brand <> 'Brand#45' and p_type not like 'MEDIUM POLISHED%' and p_size in (49, 14, 23, 45, 19, 3, 36, 9) and s_comment not like '%Customer%Complaints%'
group by p_brand, p_type, p_size
order by supplier_cnt desc, p_brand, p_type, p_size;

Query 17

select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem_1
where p_brand = 'Brand#23' and p_container = 'MED BOX' and l_quantity < (select 0.2 * avg(l_quantity) from lineitem_1);

Query 18

select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
from lineitem_1
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
having sum(l_quantity) > 300 order by o_totalprice desc, o_orderdate;

Query 19

select sum (l_extendedprice * (1 - l_discount)) as revenue
from lineitem_1
where (p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') and l_quantity >= 1 and l_quantity <= 1 + 10 and p_size between 1 and 5 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
or (p_brand = 'Brand#23' and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK') and l_quantity >= 10 and l_quantity <= 10 + 10 and p_size between 1 and 10 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
or (p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') and l_quantity >= 20 and l_quantity <= 20 + 10 and p_size between 1 and 15 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON');

Query 20

select s_name, s_address
from lineitem_1
where s_suppkey in (**select** s_suppkey **from** (**select** s_suppkey, s_name, s_address, ps_availqty, sum(l_quantity) **from** lineitem_1 **where** l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year and p_name like

'forest%' **group by** s_suppkey, s_name, s_address, ps_availqty **having** ps_availqty > 0.5 * sum(l_quantity))) and s_n_name = 'CANADA' group by s_name, s_address order by s_name;

Query 21

select s_name, count (*) as numwait
from lineitem_1 l1
where l1 o_orderstatus = 'F' and l1 l_receiptdate > l1 l_commitdate and l1 s_n_name = 'SAUDI ARABIA' and
exists (**select** * **from** lineitem_1 l2 **where** l2 o_orderkey = l1 o_orderkey and l2 s_suppkey <> l1 s_suppkey) and
not exists (**select** * **from** lineitem_1 l3 **where** l1 o_orderkey = l3 o_orderkey and l1 s_suppkey <> l3 s_suppkey and l3 l_receiptdate > l3 l_commitdate)
group by s_name
order by numwait desc, s_name;

Apêndice 2 – *Queries* SSB

Query 1.1

select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, datet
where lo_orderdate = d_datekey and d_year = 1993 and lo_discount between 1 and 3 and lo_quantity < 25;

Query 1.2

select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, datet
where lo_orderdate = d_datekey and d_yearmonthnum = 199401 and lo_discount between 4 and 6 and lo_quantity between 26 and 35;

Query 1.3

select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, datet
where lo_orderdate = d_datekey and d_weeknuminyear = 6 and d_year = 1994 and lo_discount between 5 and 7 and lo_quantity between 26 and 35;

Query 2.1

select sum(lo_revenue), d_year, p_brand
from lineorder, datet, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_category = 'MFGR#12' and s_region = 'AMERICA'
group by d_year, p_brand
order by d_year, p_brand;

Query 2.2

select sum(lo_revenue), d_year, p_brand
from lineorder, datet, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand between 'MFGR#2221' and 'MFGR#2228' and s_region = 'ASIA'
group by d_year, p_brand
order by d_year, p_brand;

Query 2.3

select sum(lo_revenue), d_year, p_brand
from lineorder, datet, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'EUROPE'
group by d_year, p_brand
order by d_year, p_brand;

Query 3.1

select c_nation, s_nation, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, datet
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;

Query 3.2

select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, datet where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Query 3.3

select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, datet where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and (c_city='UNITED KI1' or c_city='UNITED KI5') and (s_city='UNITED KI1' or s_city='UNITED KI5') and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Query 3.4

select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, datet
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and (c_city='UNITED KI1' or c_city='UNITED KI5') and (s_city='UNITED KI1' or s_city='UNITED KI5') and d_yearmonth = 'Dec1997'
group by c_city, s_city, d_year
order by d_year asc, revenue desc;

Query 4.1

select d_year, c_nation, sum (lo_revenue - lo_supplycost) as profit
from datet, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation;

Query 4.2

select d_year, s_nation, p_category, sum (lo_revenue - lo_supplycost) as profit
from datet, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category;

Query 4.3

select d_year, s_city, p_brand, sum (lo_revenue - lo_supplycost) as profit
from datet, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_nation = 'UNITED STATES' and (d_year = 1997 or d_year = 1998) and p_category = 'MFGR#14'
group by d_year, s_city, p_brand
order by d_year, s_city, p_brand;