



Universidade do Minho
Escola de Engenharia

Eduarda Alexandra Pinto da Costa

**Organização e Processamento de Dados em
Big Data Warehouses baseados em Hive**

**Organização e Processamento de Dados em
Big Data Warehouses baseados em Hive**

Eduarda Alexandra Pinto da Costa

UMinho | 2017

outubro de 2017



Universidade do Minho
Escola de Engenharia

Eduarda Alexandra Pinto da Costa

**Organização e Processamento de Dados em
Big Data Warehouses baseados em Hive**

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 2017

DECLARAÇÃO

Nome: Eduarda Alexandra Pinto da Costa

Endereço eletrónico: a69189@alunos.uminho.pt

Telefone: 912718445

Número do Bilhete de Identidade: 256822476

Título da dissertação: Organização e Processamento de Dados em *Big Data Warehouses* baseados em Hive

Orientador: Professora Doutora Maribel Yasmina Santos

Ano de conclusão: 2017

Designação do Mestrado: Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA DISSERTAÇÃO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE.

Universidade do Minho, 21 / 12 / 2017

Assinatura: Eduarda Alexandra Pinto da Costa

*“The pessimist complains about the wind.
The optimist expects it to change.
The realist adjusts the sails.”*

William A. Ward

This work was supported by: European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 002814; Funding Reference: POCI-01-0247-FEDER-002814].

AGRADECIMENTOS

A minha passagem pela Universidade do Minho como estudante do curso MIEGSI termina agora. É assustador como o tempo passa! Nada disto seria possível sozinha pelo que me resta, agora, agradecer a todos os que, direta ou indiretamente, fizeram parte desta grande jornada.

Um agradecimento muito especial à Professora Doutora Maribel Santos, um exemplo de profissional e pessoa que levarei para a vida. Obrigada por aceitar ser minha orientadora, pela paciência e por todo o tempo dispensado a ler tudo e mais alguma coisa ao mínimo pormenor. Muito obrigada por toda a confiança depositada em mim, pelas palavras de coragem e por acreditar nas minhas capacidades e no meu trabalho, mesmo quando eu duvidei, tendo sempre as palavras certas no momento certo.

Ao Carlos, obrigada pelo “muito” conhecimento transmitido, pelas várias sugestões e ideias, pelo tempo dispensando para me ajudar em todas as dúvidas que iam surgindo, pelas inúmeras questões sempre prontamente respondidas e por me socorrer em todos os momentos de pânico.

Ao grupo mais espetacular de trabalho, o do LID4, obrigada por tudo. Obrigada pelo carinho que senti logo que integrei o grupo, e por terem sempre os melhores conselhos e estarem sempre prontos para ajudar. Obrigada pelas conversas, pelos almoços e por todas as risadas que foram uma lufada de ar fresco nos dias em que o trabalho parecia demais.

Ao meu grupo de amigos, agradeço por todos os momentos, por todas as horas de trabalho, por todas as gargalhadas, pela amizade constante. Não consigo pensar em ninguém mais feliz e sortudo pelo grupo de amigos que reuniu. Estou muito orgulhosa de todos vocês e de tudo que vos vejo, agora, alcançar. É bom perceber que estamos todos encaminhados na construção do nosso próprio caminho e só agradeço poder continuar a fazer parte dele. Um agradecimento especial à Francisca, à Daniela e à Susana, as companheiras de tese, que têm sido um grande apoio nesta fase, com as suas palavras de motivação e com a troca de ideias constante.

Um agradecimento do tamanho do mundo à minha Família, obrigada pelo amor e apoio incondicional e por acreditarem sempre mais em mim do que eu própria acredito! Sei que nem sempre fui fácil, principalmente nos dias em que o cansaço e a pressão se apoderaram de mim, mas saibam que sem o vosso apoio, sem as vossas brincadeiras e sem os jantares de mesa cheia, não seria a mesma coisa. Um agradecimento ainda mais especial aos meus pais a quem devo tudo e a quem jamais irei conseguir transpor em palavras o quanto me sinto sortuda por tudo o que são para mim. Eu sei o esforço que fizeram e continuam a fazer para que possamos todos seguir os nossos sonhos e objetivos, e só me resta continuar a agradecer-vos, a deixar-vos orgulhosos e a tentar ser metade daquilo que vocês me ensinaram a ser.

RESUMO

A quantidade de dados que é produzida nos dias de hoje tem aumentado exponencialmente, como consequência da disponibilidade de novas fontes de dados e, também, devido aos avanços que vão surgindo na área de recolha e armazenamento de dados. Esta explosão de dados foi acompanhada pela popularização do conceito de *Big Data*, que pode ser definido como grandes volumes de dados, com diferentes graus de complexidade, muitas vezes sem estrutura e organização, que não podem ser processados ou analisados com processos ou ferramentas tradicionais. Os *Data Warehouses* surgem como uma peça central no armazenamento adequado dos dados, facilitando a análise dos dados sob várias perspectivas e permitindo a extração de informação que pode utilizada nos processos de tomada de decisão. No entanto, estes repositórios tradicionais, que se baseiam em bases de dados relacionais, já não conseguem responder às exigências desta nova realidade. Surge então a necessidade de seguir para o contexto de *Big Data Warehouses*, que trazem novos problemas e que implicam a adoção de novos modelos lógicos, usados nas bases de dados NoSQL ou nas tecnologias disponíveis no Hadoop, para obter maior flexibilidade na gestão de dados não estruturados, e a adoção de novas tecnologias que suportem grandes quantidades de dados. O Hive é uma ferramenta que permite a concretização de *Data Warehouse* para contextos de *Big Data*, que organiza os dados em tabelas, partições e *buckets*. Vários estudos têm sido conduzidos para compreender formas de otimizar o desempenho no armazenamento e no processamento de dados em *Big Data Warehouses*. No entanto, poucos destes estudos exploram se a forma como os dados são estruturados tem alguma influência na forma como o Hive responde a consultas. Assim, esta dissertação procura investigar o papel da modelação e organização de dados nos tempos de processamento de *Big Data Warehouses*, especificamente a definição de partições e *buckets* no Hive, de forma a definir um conjunto de boas práticas que auxiliem no processo de modelação dos dados e de definição da estrutura de dados a armazenar nestes repositórios. Os resultados obtidos com a aplicação de diversas estratégias de modelação e organização de dados no Hive, reforçam as vantagens associadas à implementação de *Big Data Warehouses* baseados em tabelas desnormalizadas e, ainda, o potencial benefício da utilização de técnicas de particionamento que, uma vez alinhadas com os filtros aplicados frequentemente nos dados, podem diminuir significativamente o tempo de processamento. As técnicas de *bucketing* não demonstraram grandes benefícios para o armazenamento e processamento de dados pelo que, na generalidade dos casos, é desaconselhada a sua utilização.

Palavras-Chave: *Big Data Warehouse*, Hive, Partições, *Buckets*, Desempenho.

ABSTRACT

The amount of data produced today has increased exponentially as a consequence of the availability of new data sources, such as social networks and sensors and, also, due to advances emerging in the area of collection and storage of data. This data explosion was accompanied by the popularization of the term Big Data that can be defined as large volumes of data, with varying degrees of complexity, often without structure and organization, that cannot be processed or analyzed using traditional processes or tools. Data Warehouses emerged as central pieces for adequate data storage, facilitating the analysis of data using different perspectives and allowing the extraction of valuable information that can be used in decision-making processes. Nevertheless, these traditional repositories, which are based on relational databases, can no longer answer to the demands of this new reality. There is a need to move to a Big Data Warehouses context, which brings new problems and imply the adoption of new logical models, used in the NoSQL databases or in the technologies available in Hadoop, in order to gain flexibility and to manage unstructured data, and to adopt new technologies that support large amounts of data. Hive is a tool that allows the implementation of Data Warehouses for Big Data contexts which organizes the data into tables, partitions and buckets. Several studies have been conducted to understand ways to optimize the performance in data storage and processing in Big Data Warehouses. However, few of these studies explore whether the way data is structured has any influence on how Hive responds to queries. Thus, this dissertation investigates the role of data organization and modelling in the processing times of Big Data Warehouses, specifically the definition of partitions and buckets on Hive, in order to identify a set of best practices that help in the process of data modelling and the definition of the data structures to be used to store data in these repositories. The results obtained with the application of several strategies of data modeling and organization in Hive reinforce the advantages associated to the implementation of Big Data Warehouses based on denormalized models and, also, the potential benefit of using partitioning techniques that, once aligned with the filters frequently applied on data, can significantly decrease the processing times. Bucketing techniques have not presented significant benefits for data storage and processing, therefore, in general, the use of such techniques is discouraged.

Keywords: Big Data Warehouse, Hive, Partitions, Buckets, Performance.

ÍNDICE

Agradecimentos.....	v
Resumo.....	vi
Abstract.....	vii
Índice.....	viii
Índice de Figuras.....	x
Índice de Tabelas.....	xii
Lista de Siglas e Acrónimos.....	xiv
1. Introdução.....	1
1.1 Enquadramento e Motivação.....	1
1.2 Objetivos.....	2
1.3 Abordagem Metodológica.....	3
1.3.1 Fases da Metodologia.....	4
1.3.2 Processo de Revisão de Literatura.....	5
1.4 Contributos do Trabalho.....	5
1.5 Organização do Documento.....	6
2. <i>Big Data Warehouses</i>	8
2.1 <i>Big Data</i>	8
2.1.1 Oportunidades, Problemas e Desafios.....	11
2.1.2 Processamento de Dados.....	13
2.2 Armazenamento de Dados.....	15
2.2.1 Bases de Dados Relacionais <i>versus</i> NoSQL.....	15
2.2.2 <i>Data Warehouses</i>	20
2.2.3 <i>Big Data Warehouses</i>	23
2.2.4 Modelos de Dados.....	25
2.3 Otimização do Armazenamento e Processamento.....	28
3. Hive para Armazenamento e Processamento em <i>Big Data</i>	33
3.1 Ecossistema do Hadoop.....	33
3.1.1 HDFS.....	36
3.1.2 MapReduce.....	37
3.2 Características do Hive.....	37
3.2.1 Arquitetura do Hive.....	38

3.2.2 Linguagem HiveQL	40
3.3 Hive como Repositório de Dados.....	41
3.4 Partições e <i>Buckets</i> de Dados.....	43
3.4.1 Partições.....	43
3.4.2 <i>Buckets</i>	45
4. Ambiente de Testes e Dados Utilizados	48
4.1 Infraestrutura Física	48
4.2 Conjunto de Dados	49
4.3 Análise dos Atributos	50
4.3.1 Cardinalidade.....	51
4.3.2 Distribuição.....	52
4.4 Protocolo de Testes	55
5. Organização e Processamento de Dados em Hive	58
5.1 Cenário A – Modelação dos Dados.....	58
5.2 Cenário B – Organização dos Dados	62
5.2.1 Cenário B1 - Particionamento.....	63
5.2.2 Cenário B2 - <i>Bucketing</i>	78
5.2.3 Cenário B3 - Particionamento e <i>Bucketing</i>	93
5.3 Síntese de Resultados.....	103
6. Conclusões.....	113
6.1 Trabalho Realizado	114
6.2 Dificuldades e Limitações	116
6.3 Investigação Futura.....	117
Referências Bibliográficas	118
Apêndices	124
Apêndice 1 - <i>Queries</i>	124
Apêndice 2 – Resultados do cenário B com <i>Distributed Join</i>	127

ÍNDICE DE FIGURAS

Figura 1 <i>Design Science Research Methodology for Information Systems</i>	3
Figura 2 Evolução de <i>Big Data</i>	8
Figura 3 Principais características de <i>Big Data</i>	9
Figura 4 O Valor como resultado do <i>Big Data</i>	11
Figura 5 Fases do processamento de <i>Big Data</i>	14
Figura 6 Teorema de CAP de Eric Brewer.....	17
Figura 7 Tipos de bases de dados NoSQL.....	19
Figura 8 Elementos principais da arquitetura de <i>Data Warehouse</i> para <i>Business Intelligence</i> de Kimball.	22
Figura 9 Processo de implementação dos modelos em <i>Data Warehouse</i>	26
Figura 10 Arquitetura de alto nível do Hadoop.....	34
Figura 11 Arquitetura do Hive.....	39
Figura 12 Exemplos da aplicação da linguagem HiveQL.....	41
Figura 13 Exemplo de criação das partições " <i>country</i> " e " <i>state</i> ".....	44
Figura 14 Diretorias criadas pelo Hive de acordo com as partições.....	44
Figura 15 Definição de <i>buckets</i> por " <i>code</i> ".....	45
Figura 16 Modelo de dados do SSB.....	50
Figura 17 Distribuição dos Atributos da Dimensão " <i>Part</i> ".....	53
Figura 18 Distribuição dos Atributos da Dimensão " <i>Supplier</i> ".....	53
Figura 19 Distribuição dos Atributos das Dimensão " <i>Customer</i> ".....	54
Figura 20 Distribuição dos Atributos de " <i>LineOrder</i> ".....	54
Figura 21 Cenários de Teste.....	56
Figura 22 Variação dos Tempos de Processamento para SF=300.....	60
Figura 23 Variação na Utilização de CPU pelo Esquema em Estrela comparada com a Tabela Desnormalizada.....	62
Figura 24 Variação dos Tempos de Processamento com Particionameto (Hive, FE=300, Esquema em Estrela).....	66
Figura 25 Variação nos Tempos de Processamento com Particionamento por <i>S_Region</i> (EE, FE=300).....	68
Figura 26 Query 3.4 (para o modelo desnormalizado).....	69
Figura 27 Variação na Utilização de CPU com Particionamento Simples (FE=100).....	71

Figura 28 Erro na criação da tabela desnormalizada com particionamento triplo	76
Figura 29 Variação na utilização de CPU com Particionamento Múltiplo (Presto; FE=100)	78
Figura 30 Variação de utilização de CPU com <i>Bucketing</i> Simples por <i>Orderkey</i>	88
Figura 31 Variação de utilização de CPU com <i>Bucketing</i> Simples por <i>Suppkey</i> (modelo normalizado).	89
Figura 32 Variação na utilização de CPU com <i>Bucketing</i> Simples por <i>Suppkey</i> (FE=300, Hive).....	89
Figura 33 Comparação da variação de utilização de CPU entre <i>bucketing</i> simples (<i>sorted by</i>) e particionamento simples.....	90
Figura 34 Variação na utilização de CPU com <i>Bucketing</i> Múltiplo	93
Figura 35 Variação nos tempos de Processamento com Particionamento por <i>S_Region</i> e <i>Bucketing</i> por <i>Suppkey</i> (Hive, EF=300)	98
Figura 36 Variação nos Tempos de Processamento com Particionamento por <i>S_Region</i> e <i>Bucketing</i> por <i>Suppkey</i> (Presto, EF=300)	98
Figura 37 Variação na utilização de CPU com combinação de Particionamento com <i>Bucketing</i> Simples	102
Figura 38 Variação na utilização de CPU com combinação de Particionamento com <i>Bucketing</i> Simples (Hive, FE=300)	103

ÍNDICE DE TABELAS

Tabela 1 Desafios/Problemas de <i>Big Data</i>	12
Tabela 2 Classificação das bases de dados NoSQL segundo o Teorema de CAP	18
Tabela 3 Grupos de modelos de dados e conceitos utilizados.	25
Tabela 4 Esquemas de dados para criação de um <i>Data Warehouse</i>	26
Tabela 5 Principais módulos do Hadoop.	35
Tabela 6 Exemplos de tecnologias do ecossistema Hadoop.	35
Tabela 7 Exemplos de operações sobre os dados recorrendo à HiveQL	41
Tabela 8 Elementos do Hive.	42
Tabela 9 Cardinalidade dos Atributos	51
Tabela 10 Informação sobre as tabelas do Cenário A	58
Tabela 11 Tempos de Processamento (em segundos) - Esquema em Estrela (EE) vs Tabela Desnormalizada (TD)	59
Tabela 12 Informações sobre as tabelas criadas com particionamento	63
Tabela 13 Tempos de Processamento com Particionamento por <i>Od_Year</i> (em segundos).....	65
Tabela 14 Particionamento por <i>S_Region</i> (em segundos)	67
Tabela 15 Tempos de Processamento com Particionamento por <i>P_MFGR</i> (em segundos).....	70
Tabela 16 Tempos de Processamento com Particionamento por <i>Od_Year</i> e <i>Od_MonthNumInYear</i> (em segundos)	72
Tabela 17 Tempos de Processamento com Particionamento por <i>Od_Year</i> e <i>S_Region</i> (em segundos) 73	
Tabela 18 Tempos de Processamento com Particionamento por <i>S_Region</i> , <i>S_Nation</i> e <i>S_City</i> (em segundos)	75
Tabela 19 Tempos de Processamento com Particionamento por <i>Od_Year</i> , <i>S_Region</i> e <i>P_MFGR</i> (em segundos)	77
Tabela 20 Informações sobre tabelas criadas com <i>bucketing</i>	79
Tabela 21 Tempos de Processamento para <i>Bucketing</i> por <i>Custkey</i> (em segundos)	81
Tabela 22 Tempos de Processamento com <i>Bucketing</i> por <i>Orderkey</i> (em segundos)	82
Tabela 23 Tempos de Processamento com <i>Bucketing</i> por <i>Od_Year</i> , <i>Sorted by P_Brand</i> (em segundos)	84
Tabela 24 Tempos de Processamento para <i>Bucketing</i> por <i>P_Brand</i> , <i>Sorted By Od_Year</i> (em segundos)	85

Tabela 25 Tempos de Processamento para <i>Bucketing</i> por <i>Orderkey, Sorted By Orderdate</i> (em segundos)	86
Tabela 26 Tempos de Processamento para <i>Bucketing</i> por <i>Suppkey</i> (em segundos)	87
Tabela 27 Tempo de Processamento com <i>Bucketing</i> por <i>Orderdate, Custkey, Suppkey</i> e <i>Partkey</i> (em segundos)	92
Tabela 28 Informações sobre tabelas criadas com particionamento e <i>bucketing</i>	93
Tabela 29 Tempos de Processamento com Particionamento por <i>Od_Year</i> e <i>Bucketing</i> por <i>Orderkey</i> (em segundos)	95
Tabela 30 Resultados gerais do cenário de Particionamento Simples por <i>Od_Year</i>	96
Tabela 31 Tempos de Processamento com Particionamento por <i>S_Region</i> e <i>Bucketing</i> por <i>Suppkey</i> (em segundos)	97
Tabela 32 Tempos de Processamento com Particionamento por <i>Od_Year+S_Region</i> e <i>Bucketing</i> por <i>Suppkey</i> (em segundos)	99
Tabela 33 Tempos de Processamento com Particionamento por <i>Od_Year</i> e <i>Bucketing</i> por <i>P_Brand</i> (em segundos)	101
Tabela 34 Melhores resultados por FE - Cenário A	103
Tabela 35 Melhores resultados por configuração de Particionamento e por FE - Cenário B1	105
Tabela 36 Melhores resultados por configuração de <i>Bucketing</i> e por FE - Cenário B2	107
Tabela 37 Melhores resultados por combinação de Particionamento e <i>Bucketing</i> e por FE - Cenário B3	109
Tabela 38 Melhor resultado por FE - Cenário B	110
Tabela 39 Tempos de Processamento com Particionamento e DJ para FE=30 (em segundos)	127
Tabela 40 Tempos de Processamento com <i>Bucketing</i> e Particionamento+ <i>Bucketing</i> e com DJ para FE=30 (em segundos)	127
Tabela 41 Tempos de Processamento com Particionamento e DJ para FE=100 (em segundos)	128
Tabela 42 Tempos de Processamento com <i>Bucketing</i> e Particionamento+ <i>Bucketing</i> e com DJ para FE=100 (em segundos)	128
Tabela 43 Tempos de Processamento com Particionamento e/ou <i>Bucketing</i> e com DJ para FE=300 (em segundos)	129

LISTA DE SIGLAS E ACRÓNIMOS

Este documento utiliza um conjunto de siglas e acrónimos que são apresentados de seguida:

ACID – *Atomicity, Consistency, Isolation, Durability*

AQUA – *Automatic Query Analyzer*

BASE – *Basically Available, Soft state, Eventual Consistency*

BD – *Big Data*

BDW – *Big Data Warehouses*

BI – *Business Intelligence*

BJ – *Broadcast Join*

CAP – *Consistency, Availability and Partition Tolerance*

DDL – *Data Definition Language*

DJ – *Distributed join*

DW – *Data Warehouses*

EE – *Esquema em Estrela*

EE-P – *Esquema em Estrela com Partições*

EE-PB – *Esquema em Estrela com Partições e Buckets*

ELT – *Extraction, Loading and Transformation*

ETL – *Extraction, Transformation and Loading*

FE – *Fator de Escala*

GFS – *Google File System*

HDFS – *Hadoop Distributed File System*

HiveQL – *Hive Query Language*

NDFS – *Nutch Distributed File System*

NoSQL – *Not Only SQL*

OLAP – *Online Analytical Processing*

OLTP – *Online Transaction Processing*

ORC – *Optimized Row Columnar*

SGBDR – *Sistema de Gestão de Bases de Dados Relacionais*

SQL – *Structured Query Language*

SSB – *Star Schema Benchmark*

TD – Tabela Desnormalizada

TD-P – Tabela Desnormalizada com Partições

TD-PB – Tabela Desnormalizada com Partições e *Buckets*

1. INTRODUÇÃO

Neste capítulo é apresentado um enquadramento ao tema da dissertação assim como a motivação da mesma. De seguida são apresentados os principais objetivos, a abordagem metodológica adotada e os contributos do trabalho, finalizando com a estrutura do documento.

1.1 Enquadramento e Motivação

Com os avanços na recolha e armazenamento de dados, e tendo em conta a disponibilidade de novas fontes de dados como as redes sociais e os sensores, o volume de dados recolhido pelas organizações está a crescer rapidamente (Cassavia, Dicosta, Masciari, & Saccà, 2014). Este aumento explosivo de dados provocado pela constante inovação e pela transformação, globalização e personalização dos serviços associados aos novos modelos de negócio, fez-se acompanhar pela popularização do termo *Big Data* que pode ser definido como “vastos 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” (Krishnan, 2013).

O processamento e a análise de *Big Data* estão, no entanto, associados a grandes dificuldades relacionadas não só com a quantidade de dados que é produzida, como também com a variedade dos formatos e a velocidade a que estes são gerados. Assim, tal como em *Business Intelligence (BI)*, os *Data Warehouses* surgem como peças centrais para o armazenamento apropriado dos dados, suportados por modelos de dados bem definidos, e que facilitam as pesquisas e as análises sob vários pontos de vista (Santos & Costa, 2016), permitindo a extração e produção de informação com valor e que possa ser utilizada nos processos de tomada de decisão (Di Tria, Lefons, & Tangorra, 2014).

No entanto, estes repositórios tradicionais, baseados em bases de dados relacionais e modelos de dados bem estruturados, já não conseguem dar resposta às exigências desta nova realidade. Torna-se evidente que os *Big Data Warehouses* trazem novos problemas e implicam a adoção de novos modelos lógicos, como os modelos utilizados nas bases de dados NoSQL, de forma a ganhar maior flexibilidade e assim conseguir gerir dados não estruturados e desnormalizados (Di Tria et al., 2014).

Surgiu também a necessidade de desenvolvimento de aplicações com uso intensivo de dados, altamente disponíveis e escaláveis e baseadas em sistemas de armazenamento de confiança (Bhardwaj, Vanraj, Kumar, Narayan, & Kumar, 2015). É neste sentido que surge o Hadoop, um ecossistema tecnológico popular, baseado em MapReduce e HDFS, que tem sido bastante usado para armazenar,

processar e analisar grandes quantidades de dados (Thusoo et al., 2010). Associado ao ecossistema Hadoop, surge o Hive, um software de *Data Warehousing* construído sobre o Hadoop (Krishnan, 2013). Esta plataforma estrutura os dados em tabelas, colunas, linhas, partições e *buckets*, suporta a maior parte dos tipos primitivos e complexos, e tem como base uma linguagem própria de consulta, o HiveQL, que é uma extensão da SQL (*Structured Query Language*) (Thusoo et al., 2010).

Assumindo estes conceitos e estas tecnologias, o desafio centra-se no facto de os tempos de processamento desta quantidade de dados poderem ser melhorados dadas as necessidades cada vez mais emergentes nas organizações, de obter informação de valor que ajude nos processos de tomada de decisão. Assim, a motivação inerente a este projeto de investigação é perceber até que ponto a forma como se modela e organiza os dados pode influenciar os tempos de processamento de *Big Data Warehouses*. Para além disso, sendo a existência de partições e *buckets* uma característica diferenciadora do Hive, já que estes permitem a separação dos dados em partes mais fáceis de gerir, pode ser importante para otimizar os tempos de resposta do Hive a uma consulta. Com isto, este trabalho tem, ainda, como principal motivação perceber a relação entre a definição do número de partições e *buckets* no Hive e a eficiência do processamento de dados.

1.2 Objetivos

Esta dissertação tem como finalidade o estudo do papel da organização dos dados nos tempos de processamento de *Big Data Warehouses*, nomeadamente na definição de partições e *buckets* do Hive, de forma a identificar boas práticas que ajudem no processo de modelação dos dados uma vez que este condiciona a forma como o armazenamento é realizado.

Com a realização deste projeto pretende-se a concretização dos seguintes objetivos:

- Realizar uma revisão de literatura com o estado da arte nesta área, discutindo os principais conceitos e problemas relacionados com *Big Data Warehouses* e a sua implementação;
- Compreender a forma de organização dos dados num *Big Data Warehouse*, baseado em Hive;
- Compreender o impacto que a definição de partições e *buckets* em Hive tem na eficiência do processamento de dados;
- Identificar um conjunto de boas práticas para a modelação e organização dos dados em Hive, através da realização de diversos *benchmarks* que comparem o desempenho das diferentes formas de estruturar os dados.

Para além disso, esta dissertação pretende dar resposta a quatro questões que permanecem sem contributos por parte da comunidade científica de profissionais da área:

- Um *Data Warehouse* baseado em modelos multidimensionais (esquemas em estrela ou constelação) é um modelo de *design* adequado para *Data Warehousing* concretizado em Hive?
- Os esquemas multidimensionais são mais eficientes do que as tabelas totalmente desnormalizadas, em contextos de *Big Data*?
- Há vantagens na utilização de partições e/ou *buckets*? Estas formas de organização têm algum impacto nos tempos de processamento?

1.3 Abordagem Metodológica

Esta dissertação enquadra-se dentro de um processo de investigação mais completo baseado em *Design Science Research*, e que neste caso é utilizado como método de investigação uma experiência laboratorial (*benchmarking*). Assim, dada a área de enquadramento desta dissertação, a metodologia selecionada para orientar esta investigação foi a “Design Science Research Methodology (DSRM) for Information Systems” de Peffers, Tuunanen, Rothenberger, & Chatterjee (2007) uma metodologia apropriada à investigação na área de Sistemas de Informação e que admite a existência de seis fases que são adequadas aos objetivos e resultados esperados desta dissertação, representadas na Figura 1.

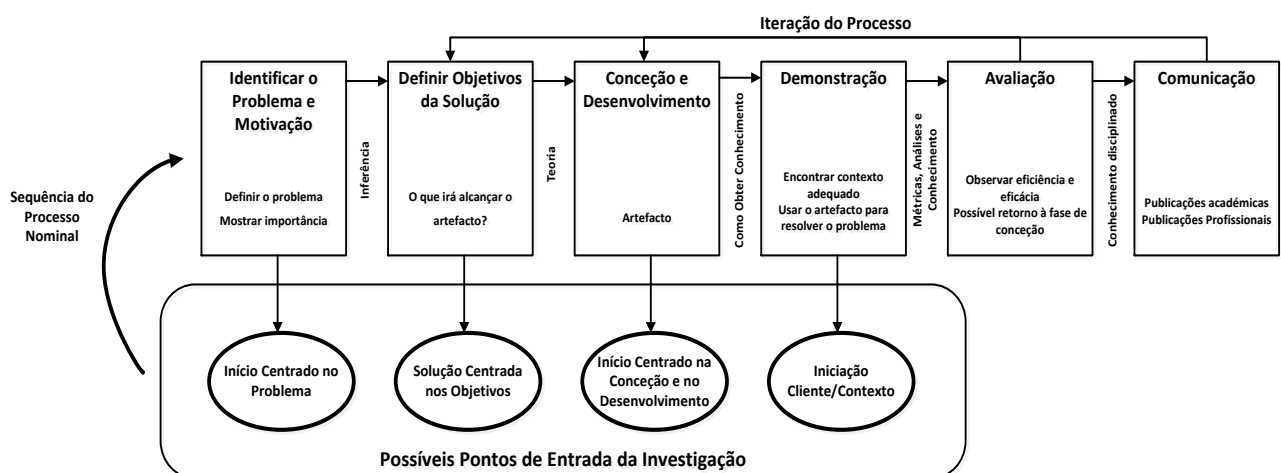


Figura 1 Design Science Research Methodology for Information Systems. Adaptado de (Peffers et al., 2007)

Os autores deste modelo, apesar de apresentarem um processo com ordem sequencial, admitem a flexibilidade do próprio modelo, podendo um projeto de investigação ser iniciado em qualquer um dos quatro pontos de partida acima representados. Neste caso, a investigação é iniciada no primeiro ponto

de partida, um início centrado no problema, começando por identificar o problema e as principais motivações.

1.3.1 Fases da Metodologia

Assumindo a abordagem metodológica de investigação selecionada e as datas de entregas intermédias exigidas pela instituição de ensino, surgem as seguintes tarefas a desenvolver:

1. Definição do Plano de Trabalho – criar um documento com o enquadramento da dissertação assim a como apresentação da motivação e principais objetivos, identificação da abordagem metodológica a utilizar e a descrição das tarefas e respetiva calendarização;
2. Identificação do problema e motivação – identificar o problema específico da investigação e justificar o valor que a solução encontrada poderá ter;
3. Definir os objetivos da solução – definir os objetivos a atingir tendo em conta o problema e o conhecimento daquilo que é possível e executável;
4. Revisão de literatura – identificar, analisar e compreender a literatura relevante expondo os conceitos apropriados e técnicas/experiências previamente estudadas e realizadas e que se relacionam com o tema da dissertação. Durante esta tarefa serão apresentados os conceitos e trabalhos de relevância na área de *Big Data Warehouses*, focando nos estudos que se preocuparam com a otimização do armazenamento e processamento de grandes quantidades de dados;
5. Enquadramento tecnológico – apresentação de tecnologias associadas ao tema da dissertação, sendo que, neste caso, será dada mais ênfase ao Hive;
6. Conceção e Desenvolvimento – criar o artefacto que pode conter modelos, métodos e instanciações desses modelos e métodos. Esta fase incluirá a definição de um protocolo de testes que terá em consideração as características e funcionalidades do Hive
7. Demonstração – demonstrar o uso do artefacto desenvolvido anteriormente. Nesta fase serão implementados os testes incluídos no protocolo de testes definido na fase anterior, que como caso de demonstração usa o modelo de dados do *Star Schema Benchmark*;
8. Avaliação – medir o resultado da demonstração executada. Nesta fase serão analisadas e validadas as boas práticas de modelação e organização de dados, a armazenar num *Big Data Warehouse* (concretizado em Hive), inferidas através de métricas de desempenho e de comparação de resultados, que pode ter como consequência a sua alteração e a realização de novos testes;

9. Comunicação – comunicar e apresentar os resultados obtidos a profissionais da área. Esta fase inclui a escrita e apresentação da dissertação assim como a publicação de artigos relacionados em jornais/revistas/conferências da área.

1.3.2 Processo de Revisão de Literatura

A criação de um processo para a recolha da literatura relevante para a fase de enquadramento conceptual é essencial para, não só facilitar o trabalho de pesquisa e leitura como para manter um método de trabalho consistente. Assim sendo, numa fase inicial foram definidas as palavras-chave a utilizar, as bases de dados de referência e o método de seleção dos artigos.

Dada a juventude da área em que este tema se enquadra não foram considerados filtros temporais uma vez que os artigos encontrados se concentravam, na sua maioria, entre 2001 e 2017. Para além disso, apesar da pesquisa de artigos se ter concentrado entre setembro de 2016 e fevereiro de 2017, foram criados alertas nas bases de dados de referência de forma a manter uma monitorização apertada de artigos que vão sendo publicados, que estejam relacionados com o tema e que possam vir a ser importantes para incluir e referenciar.

Assim foram selecionadas as seguintes palavras-chave: “big data”, “data warehouse”, “big data warehouse”, “hive”, “partitions”, “buckets”, “optimization” e “data models”, que foram utilizadas, individualmente ou combinadas entre si, nas bases de dados de referência. Estas bases de dados de referência incluíram o “Scopus”, o “Web of Science”, o “repositoriUM”, o “Google Scholar” e o “IEEE Xplorer”. Tendo em conta os resultados obtidos, numa primeira fase a seleção dos artigos foi feita através do título e *abstract* sendo posteriormente filtrados, com base numa leitura rápida do documento, em muito importantes ou potencialmente importantes. É importante realçar que foram também considerados artigos fornecidos pela orientadora da dissertação, alguns artigos que eram referenciados nos artigos que iam sendo extraídos e, ainda, sites oficiais das tecnologias.

Os artigos filtrados foram analisados com atenção, sendo extraídas as informações importantes de cada um e, posteriormente, utilizados e referenciados ao longo deste documento.

1.4 Contributos do Trabalho

O trabalho realizado numa fase inicial deste projeto, quer na análise de arquiteturas existentes, quer na compreensão e teste de diversas tecnologias de processamento de dados em *Big Data*, permitiu a publicação dos seguintes artigos científicos:

1. Santos, M. Y., Sá, J., Costa, C., Galvão, J., Andrade, C., Martinho, B., ... Costa, E. (2017). *A Big Data Analytics Architecture for Industry 4.0*. In *WorldCist'17 - 5th World Conference on Information Systems and Technologies*.
2. 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*. In *Proceedings of the 21st International Database Engineering & Applications Symposium* (pp. 242–252). New York, NY, USA: ACM. <https://doi.org/10.1145/3105831.3105842>
3. Santos, M. Y., e Sá, J. O., Andrade, C., Lima, F. V., Costa, E., Costa, C., ... Galvão, J. (2017). *A Big Data system supporting Bosch Braga Industry 4.0 strategy*. In *International Journal of Information Management*.

Estes artigos estão enquadrados no trabalho realizado no âmbito do projeto “P30 – *Business Intelligence Platform for Data Integration* ¹” entre a Universidade do Minho e a Bosch, no qual a autora deste documento é bolsista de investigação desde novembro de 2016.

Numa fase posterior, e já com resultados que emergem desta dissertação, foram publicados os seguintes artigos científicos:

1. Costa, E., Costa, C., & Santos, M. Y. (2017). *Efficient Big Data Modelling and Organization for Hadoop Hive-Based Data Warehouses*. In *14th European, Mediterranean, and Middle Eastern Conference (EMCIS)* (pp. 3–16).
2. Costa, E., Costa, C., & Santos, M. Y. (2018). *Partitions and Buckets on Hive-Based Data Warehouses*. In *WorldCist'18 - 6th World Conference on Information Systems and Technologies*.

1.5 Organização do Documento

Este documento está organizado em 6 capítulos aqui brevemente apresentados. O primeiro capítulo refere-se à introdução, apresentando o enquadramento, a motivação, os objetivos, a abordagem metodológica, os contributos e a organização do documento. O segundo capítulo inclui o enquadramento conceptual, apresentando os conceitos associados aos *Big Data Warehouses*, nomeadamente o termo

¹ O objetivo do projeto é desenvolver um sistema integrado de dados que permita, através de um processo interativo, o desenvolvimento do *Data Warehouse Organizacional*, contribuindo para o aumento da qualidade das operações de fábrica, em termos da eficiência do acesso e qualidade da informação crítica necessária para a tomada de decisão e envolvimento dos atores, a montante e jusante da cadeia de valor.

Big Data e a comparação entre as abordagens tradicionais e as abordagens emergentes no que toca ao armazenamento de dados, e terminando com a apresentação dos principais estudos relacionados com a otimização do armazenamento e processamento destas grandes quantidades de dados. O capítulo 3 apresenta o enquadramento tecnológico, sendo que se foca essencialmente no Hive, na sua arquitetura e modelo de dados, por ser esta a tecnologia base desta dissertação. O capítulo 4 apresenta o ambiente de testes, o conjunto de dados utilizados e a análise dos atributos que o compõem e, ainda, o protocolo de testes a seguir. O capítulo 5 apresenta os resultados obtidos para todos os cenários de teste, assim como uma avaliação e discussão desses resultados. O último capítulo apresenta as principais conclusões, sistematizando o trabalho realizado, identificando as dificuldades e limitações enfrentadas e apresentando trabalho futuro.

2. BIG DATA WAREHOUSES

De forma a enquadrar o propósito desta dissertação, neste capítulo serão apresentados os principais conceitos associados ao respetivo tema. Assim, será apresentado o conceito de *Big Data* e suas características, oportunidades e desafios inerentes. De seguida será realizada uma comparação entre as bases dados tradicionais e as bases de dados NoSQL assim como uma comparação entre *Data Warehouses* tradicionais e *Big Data Warehouses*. Por último, o capítulo dedica-se à questão da organização dos dados, sistematizando trabalho relacionado.

2.1 Big Data

A rápida evolução das tecnologias *Big Data* e sua rápida aceitação deixou pouco tempo para desenvolver ou amadurecer o conceito, existindo ainda pouco consenso sobre o porquê de *Big Data* ter sido qualificado como tal. O conceito de *Big Data* evoluiu de forma muito rápida o que originou alguma confusão (Gandomi & Haider, 2015). Admite-se, no entanto, que uma das razões fundamentais para o fenómeno *Big Data* existir é a extensão atual com que a informação pode ser gerada e disponibilizada (De Mauro, Greco, & Grimaldi, 2015). Na Figura 2 é possível perceber a forma como os dados evoluíram ao longo do tempo até atingir um contexto de *Big Data* que surge como consequência da complexidade e da heterogeneidade dos dados, para além do seu volume.

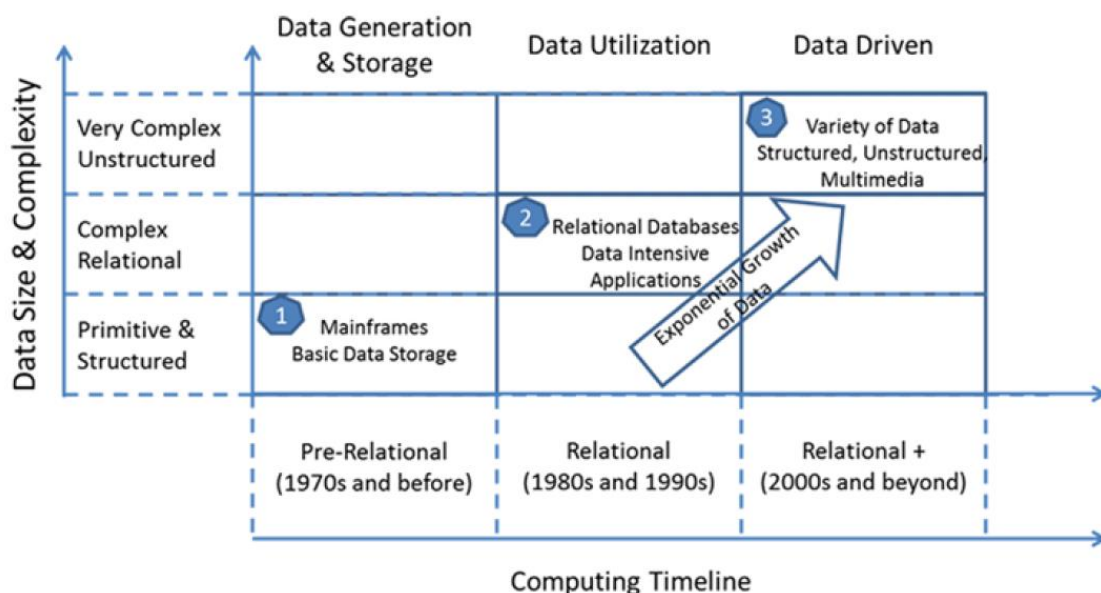


Figura 2 Evolução de Big Data. Retirado de (Mohanty et al., 2013)

Se esta grande quantidade de dados for efetivamente utilizada, os benefícios para as organizações são substanciais, nomeadamente a nível de inovação na produção ou eficiência operacional (Mohanty, Jagadeesh, & Srivatsa, 2013).

São vários os autores que definem *Big Data*, no entanto, é notável a existência de consenso de que *Big Data* pode ser definido como grandes volumes de dados, com vários graus de complexidade, sem estrutura e por vezes desorganizados, que não podem ser processados ou analisados utilizando processos ou ferramentas tradicionais (De Mauro et al., 2015; Krishnan, 2013; Zikopoulos & Eaton, 2011).

É notório, ainda, que a maior parte das definições se baseiam em três características principais: volume, variedade e velocidade (Zikopoulos & Eaton, 2011), representadas na Figura 3.

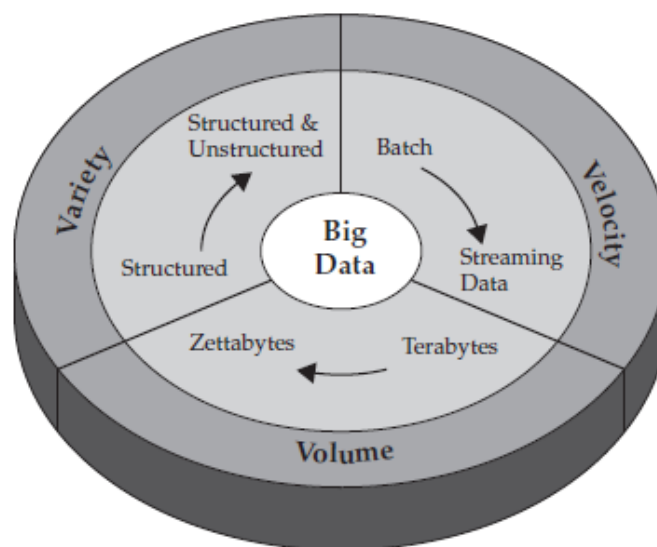


Figura 3 Principais características de Big Data. Retirado de (Zikopoulos & Eaton, 2011)

Claramente, o tamanho é a primeira característica que surge quando se fala em *Big Data*. Existem, nos dias de hoje, mais dados do que alguma vez existiu (Zikopoulos & Eaton, 2011). Assim, o **volume** refere-se à magnitude dos dados, sendo o seu tamanho descrito, geralmente, em múltiplos *terabytes* e *petabytes* (Gandomi & Haider, 2015). O volume de *Big Data* pode, então, ser caracterizado pela quantidade de dados que é gerado continuamente pelas mais variadas fontes de dados e que incluem texto, áudio, vídeo, redes sociais, pesquisas, dados médicos, relatórios de crimes, entre outros (M. A. Khan, Uddin, & Gupta, 2014).

Quando se introduz a característica **velocidade** é importante considerar não só a taxa de criação dos dados, mas também a velocidade a que estes dados devem ser analisados dependendo da necessidade da sua utilização (Gandomi & Haider, 2015). A alta velocidade a que os dados estão a ser

gerados é diretamente responsável pelo grande volume de dados descrito anteriormente (M. A. Khan et al., 2014). A velocidade é, assim, um aspeto muito importante na medida em que, tradicionalmente os dados eram analisados em *batches* (em lotes), que eram adquiridos ao longo do tempo. No entanto, com *Big Data*, os fluxos de dados contínuos só serão úteis se o tempo entre a recolha e o seu processamento for relativamente curto (Krishnan, 2013). É preciso analisar os dados quase em tempo real se as organizações esperam obter alguma compreensão dos dados e atuar com base nos mesmos (Zikopoulos & Eaton, 2011).

Não existe controlo sobre o formato ou estrutura dos dados que vão surgindo (Krishnan, 2013). A **variedade** refere-se, então, à heterogeneidade de estrutura num grande conjunto de dados. Existem dados estruturados, não estruturados e semiestruturados. Os primeiros referem-se, por exemplo, a dados tabulares encontrados em bases de dados relacionais, os não estruturados referem-se, por exemplo, a texto, imagens, vídeos, e os dados semiestruturados tem como exemplo típico a linguagem XML (Gandomi & Haider, 2015). Para Khan et al. (2014), a variedade dos dados afeta diretamente a integridade dos dados, pois mais variedade/heterogeneidade implicará mais erros.

Vários outros autores admitem a existência de mais propriedades que complementam as anteriores como complexidade, veracidade, validade, volatilidade e valor.

A complexidade surge associada à quantidade de fontes dos dados, o que impõe uma necessidade de conectar, associar, limpar e transformar os dados recebidos de todas essas fontes (Gandomi & Haider, 2015).

A veracidade está relacionada com o grau de certeza e confiança nos dados que são produzidos e, conseqüentemente, nas fontes de dados. Semelhante a esta característica, surge a validade que se refere à exatidão dos dados a ser usados. Os dados podem ser verídicos, mas ao mesmo tempo inválidos se não forem devidamente compreendidos. A volatilidade está relacionada com a política de retenção de dados de uma organização, referindo-se ao tempo em que os dados devem ser mantidos e a partir do qual devem/podem ser destruídos (N. Khan et al., 2014).

Para M. A. Khan et al. (2014), o valor, mais do que uma propriedade, é o resultado esperado do processamento de *Big Data*, tal como representado na Figura 4, e mede a utilidade dos dados nas tomadas de decisão (Kaisler, Armour, Espinosa, & Money, 2013). Só é possível retirar valor dos dados depois de analisados e visualizados ou então estes não passam de dados confusos e, possivelmente, com erros e em alteração constante (Gupta, 2015).

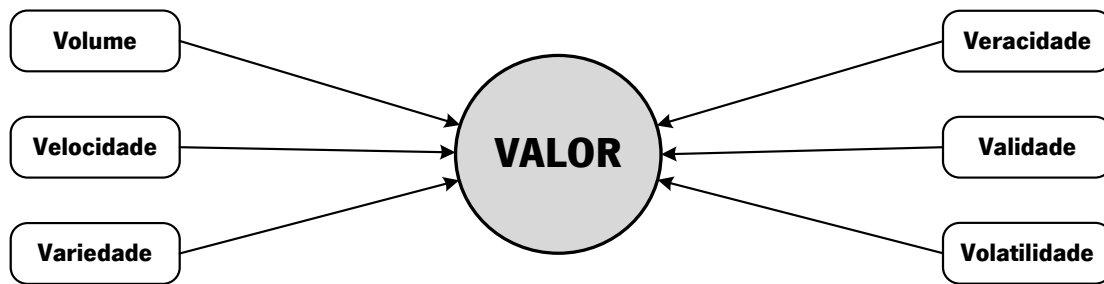


Figura 4 O Valor como resultado do Big Data. Adaptado de (M. A. Khan et al., 2014)

2.1.1 Oportunidades, Problemas e Desafios

O potencial de *Big Data* só é realmente aproveitado quando está associado a vantagens no processo de tomada de decisão (Gandomi & Haider, 2015). Recentemente, são várias as empresas e agências governamentais que têm demonstrado interesse nas potencialidades de *Big Data* e que têm anunciado grandes planos para acelerar a investigação e aplicação de *Big Data* (M. Chen, Mao, & Liu, 2014).

Big Data traz oportunidades únicas que não são tão exequíveis com os *datasets* tradicionais. De entre as muitas oportunidades surgem, por exemplo, a capacidade crescente de fornecer serviços personalizados (Fan, Han, & Liu, 2014). Tirar proveito de conhecimentos valiosos das grandes quantidades de dados virá a ser a competição básica para as empresas de hoje. Há muitas vantagens para o mundo empresarial incluindo o aumento da eficiência operacional, o desenvolvimento de um atendimento personalizado, identificação e desenvolvimento de novos produtos consoante as alterações no mercado, a identificação de novos clientes e mercados, entre outros (Philip Chen & Zhang, 2014).

Segundo Kaisler et al. (2013), o valor referido anteriormente advém da capacidade de analisar os dados e de criar informação relevante para as organizações. Para estes autores, as oportunidades oferecidas pelo *Big Data* na criação de valor incluem (Kaisler et al., 2013):

1. Criação de transparência, disponibilizando grandes quantidades de dados para análise;
2. Ajuda na segmentação de mercado, com base nas informações do cliente;
3. Ajuda no processo de inovação de produtos, com base nas necessidades dos clientes ou nas tecnologias que vão surgindo;

Contudo, esta explosão de dados traz consigo muitos problemas e desafios que devem ser ultrapassados para retirar o máximo valor possível da sua análise. Na Tabela 1 são especificados alguns desses problemas e desafios e respetivas considerações sobre cada um.

Tabela 1 Desafios/Problemas de Big Data

Desafio/ Problema	Caracterização
Heterogeneidade dos dados	Os algoritmos de análise esperam, geralmente, dados homogêneos (Jagadish et al., 2014). No entanto, os dados produzidos nos dias de hoje estão, na sua maioria, sem qualquer tipo de estrutura e em diferentes formatos (Katal, Wazid, & Goudar, 2013).
Integração de dados	<i>Big Data</i> requer, não só a capacidade de gestão e consulta de grandes volumes de dados, mas também a capacidade de consulta de dados de um grande número de fontes de dados (Salloum, Dong, Srivastava, & Tsotras, 2013). A análise de dados de múltiplas fontes pode estar associada a um aumento significativo do valor dos dados e da informação que deles se retira. Esta integração pode ser, no entanto, um problema, devido ao número de fontes de dados autónomas existentes (Rekatsinas, Dong, & Srivastava, 2014).
Problemas no armazenamento e comunicação	Os dados estão a ser criados por tudo e todos e a qualquer momento (Kaisler et al., 2013). Os meios de armazenamento tradicionais já não são suficientes para a quantidade de dados que é gerada todos os dias. Para além disso, esta quantidade de dados também excede as capacidades de transferência das redes de comunicação (Kaisler et al., 2013). Quer isto dizer que há uma necessidade crescente de revolucionar as tecnologias de armazenamento e de comunicação (Justin Samuel, Koundinya, Sashidhar, & Bharathi, 2006).
Problemas no Processamento	O processamento eficiente de <i>exabytes</i> de dados requer um processamento paralelo extenso e também novos algoritmos de análise de forma a conseguir fornecer informação adequada e a tempo da tomada de decisão (Kaisler et al., 2013). Isto porque, processar grandes quantidades de dados implica grandes quantidades de tempo, pelo que são necessários novos mecanismos que consigam reduzir consideravelmente o tempo de processamento (Katal et al., 2013)
Desafios Analíticos	<i>Big Data</i> pode trazer alguns problemas analíticos. O tipo de análise que deve ser realizado neste grande volume de dados requer competências avançadas e uma boa perceção dos resultados que se pretendem obter (Katal et al., 2013). Um desafio decorrente é a distinção entre quantidade e qualidade. Com tantos dados é preciso perceber quais são os mais relevantes e se são suficientes para as análises pretendidas, já que é preciso assegurar que os dados são confiáveis e precisos e que realmente é possível tirar valor desses dados de forma a que estes suportem o processo de tomada de decisão (Kaisler et al., 2013).

Desafio/ Problema	Caracterização
Privacidade e Segurança	<p>A privacidade tem-se mostrado uma das maiores preocupações na era de <i>Big Data</i> (Justin Samuel et al., 2006; Katal et al., 2013). Existe um grande receio em relação ao uso inadequado de dados pessoais obtidos através da ligação de várias fontes de dados (Jagadish et al., 2014). Em domínios como as redes sociais e a saúde, o excesso de dados armazenados criam preocupações de que as organizações irão saber demasiado sobre os indivíduos (Kaisler et al., 2013).</p> <p>A recolha de dados sobre interesses pessoais, hábitos ou informações pessoais pode acontecer com ou sem a permissão dos utilizadores, daí a necessidade da criação de mecanismos eficientes para a proteção da privacidade (Justin Samuel et al., 2006).</p>

Como se pode verificar, *Big Data* surge com muitos desafios que acabam por se relacionar com a capacidade de conceber sistemas apropriados que suportem eficientemente esta quantidade de dados e com a capacidade de analisar e extrair valor suficiente para tomar decisões adequadas. A privacidade permanece como uma das questões mais sensíveis e que inclui preocupações técnicas e legais (Katal et al., 2013), sendo uma das maiores preocupações na era de *Big Data*.

Para além disso, surgem alguns problemas relacionados com o armazenamento e transporte, com a sua gestão e, principalmente, relacionados com o processamento destas quantidades enormes de dados (Kaisler et al., 2013), tópico abordado de seguida.

2.1.2 Processamento de Dados

De acordo com Krishnan (2013), o processamento de dados é visto como a recolha, processamento e gestão dos dados e que tem como resultado a informação necessária aos consumidores finais. No entanto, em ambientes de *Big Data*, o autor afirma que o ciclo de processamento difere do ciclo de processamento dos dados transacionais. Num ambiente tradicional os dados são primeiro analisados, é criado um modelo de dados e posteriormente uma estrutura de base de dados para processar esses dados. Já em ambiente de *Big Data*, os dados são, numa primeira fase, recolhidos e armazenados numa plataforma, é criada uma estrutura de dados para o conteúdo e só depois se procede para a fase de transformação e análise.

Tal como representado na Figura 5, existem quatro fases principais para o processamento de *Big Data* (Krishnan, 2013):

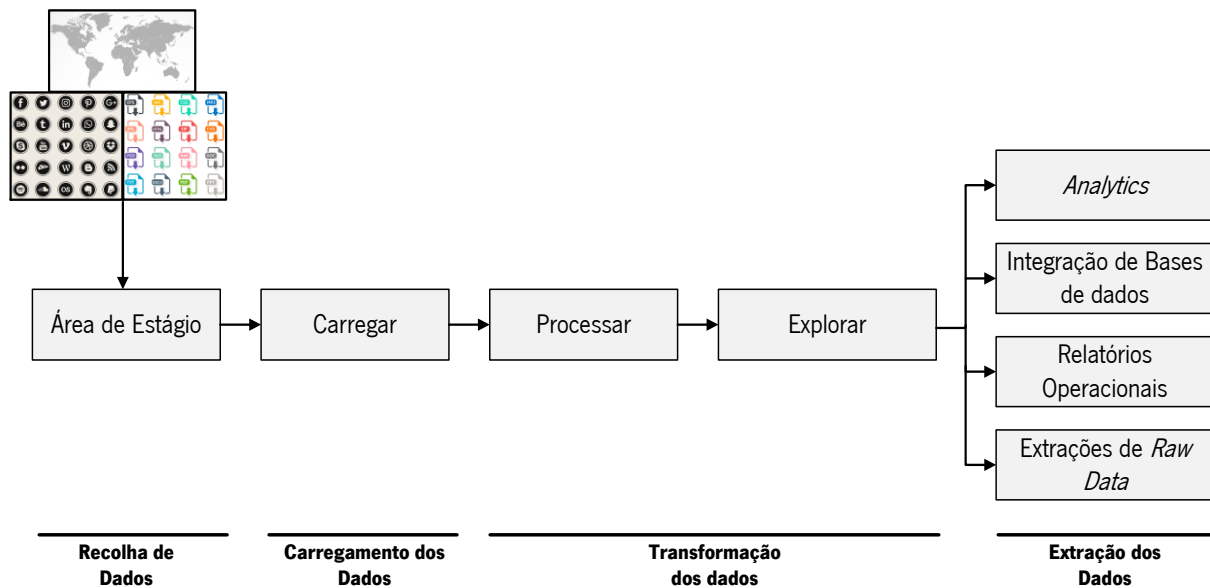


Figura 5 Fases do processamento de *Big Data*. Adaptado de (Krishnan, 2013)

1. Recolha: esta fase consiste na aquisição dos dados das várias fontes disponíveis e o seu armazenamento numa área de estágio.
2. Carregamento: carregamento dos dados com a aplicação dos metadados e com a aplicação de algum tipo de estrutura. Este processo de carregamento divide os dados recolhidos em arquivos mais pequenos, criando um catálogo de ficheiros;
3. Transformação: nesta fase os dados são tratados e transformados de acordo com as necessidades e regras de negócio;
4. Extração: nesta fase, os dados resultantes podem ser extraídos para fins de análise, criação de relatórios operacionais, integração num *Data Warehouse* e de visualização.

São estas as fases principais para a obtenção dos dados a serem armazenados num *Data Warehouse* sobre o qual se poderão ser executadas consultas sobre os dados, atuais ou históricos, facilitar o processo de previsão e desenvolvimentos de análises interpretativas a fim de extrair inteligência e conhecimento útil destes.

Para além disso, o processamento de dados pode ser de três tipos (Du, 2015):

1. Processamento em *batch*: usado para processar dados em lotes. Neste tipo de processamento, um conjunto de dados é lido da fonte de dados sendo posteriormente processado e escrito na fonte de destino.

2. Processamento em tempo real: consiste em processar os dados e obter os resultados quase imediatamente.
3. Processamento em *stream*: consiste em processar dados continuamente e atuar sobre os dados em *live stream* para obter os resultados.

2.2 Armazenamento de Dados

Sendo o armazenamento, tal como referido anteriormente, um dos grandes desafios na era do *Big Data*, nesta secção será feita uma contextualização dos sistemas de armazenamento de dados. Numa primeira fase será explicada a evolução das bases de dados relacionais para as bases de dados NoSQL, apresentando características de cada uma delas. De seguida é introduzido o conceito de *Data Warehousing*, as suas principais características e abordagens para, de seguida, introduzir, por fim, a evolução para *Big Data Warehouses* e as implicações que as grandes quantidades de dados têm nas características e na conceção destes repositórios e nas abordagens a seguir.

2.2.1 Bases de Dados Relacionais *versus* NoSQL

Ao falar de dados há três aspetos que têm que ser considerados: a sua estrutura (ou neste caso, a falta dela), a semântica (o perigo de se poderem referir a diferentes conceitos) e a integração (ligação das várias fontes de dados). Daqui resulta um dos maiores desafios: organizar e modelar os dados de forma a facilitar o processo de associação, transformação, processamento e análise dos dados recolhidos (Cassavia et al., 2014).

As bases de dados relacionais têm servido durante décadas e têm sido consistentemente utilizadas para estruturas de dados normalizadas (Alekseev et al., 2016). Os sistemas de gestão de bases de dados relacionais baseiam-se na linguagem SQL e em propriedades transacionais garantindo as propriedades ACID: atomicidade, consistência, isolamento e durabilidade.

Durante muitos anos estes sistemas de gestão de bases de dados relacionais alcançaram um grande nível de confiabilidade e estabilidade, funcionando como mecanismos poderosos de armazenamento e consulta de dados estruturados e garantindo consistência nestes processos (Gessert, Wingerath, Friedrich, & Ritter, 2016).

Mas a tecnologia tradicional já não suporta o processamento e as análises das quantidades enormes de dados que são produzidas atualmente (Alekseev et al., 2016).

Apesar de ser flexível e capaz de suportar processamento transacional e analítico (Cassavia et al., 2014), nos últimos anos a quantidade de dados úteis nas mais diversas áreas tornou-se tão grande que

as soluções tradicionais de bases de dados, baseadas em modelos relacionais, já não a conseguem guardar ou processar (Gessert et al., 2016).

Consequentemente, o desenvolvimento tecnológico levou ao aparecimento de novas arquiteturas e tecnologias de bases de dados. Uma das forças por trás destas mudanças é a necessidade de melhorar a análise a estas enormes quantidades de dados, chamada hoje de *Big Data Analytics*. Tendo em conta que o conceito de *Big Data Analytics* inclui conceção, desenvolvimento, monitorização e gestão de modelos analíticos, o volume, a variedade e a velocidade a que os dados têm que ser analisados limita a utilização de métodos analíticos tradicionais (Pokorný, 2015).

Surgiram então as bases de dados NoSQL (*Not Only SQL*), que na sua maioria oferecem escalabilidade horizontal e maior disponibilidade do que as bases de dados relacionais sacrificando, no entanto, características das bases de dados tradicionais como a consistência (Gessert et al., 2016)

As bases de dados NoSQL são a nova abordagem de conceção de bases de dados orientada para a gestão de grandes quantidades de dados, dados esses distribuídos (Han, Haihong, Le, & Du, 2011; Philip Chen & Zhang, 2014). O facto de se chamarem bases de dados NoSQL pode induzir em erro, podendo ser interpretado como “*NotSQL*”, no entanto, estas bases de dados não evitam o SQL. Existem alguns sistemas NoSQL que são totalmente não-relacionais, mas existem outros que apenas evitam algumas funcionalidades relacionais como as estruturas fixas ou as operações de *join* (Philip Chen & Zhang, 2014).

Com o desenvolvimento da Internet e da computação em *cloud*, são necessárias bases de dados que suportem o armazenamento e o processamento de *Big Data* de forma eficiente, e através das bases de dados NoSQL é possível resolver algumas das exigências desta nova realidade (Cassavia et al., 2014; Han et al., 2011; Huang & Luo, 2014):

1. Evitar complexidade desnecessária, de que são exemplo as propriedades ACID inerentes às bases de dados relacionais;
2. Armazenar grandes quantidades de dados sem necessidade de esquema/estrutura;
3. Conseguir uma taxa de transferência maior;
4. Replicar e particionar dados em muitos servidores;
5. Adicionar dinamicamente novos atributos aos registos;
6. Obter escalabilidade horizontal e não estar dependente dos recursos do hardware.

Ao contrário dos sistemas de gestão de bases de dados relacionais (SGBDR), os sistemas NoSQL são conhecidos pela escalabilidade horizontal, pela elasticidade, pela disponibilidade e pela flexibilidade

nos esquemas, que podem mesmo mudar durante o tempo de execução (Chavalier, El Malki, Kopliku, Teste, & Tournier, 2016; Vajk, Fehér, Fekete, & Charaf, 2013).

Segundo Pritchett (2008), algumas bases de dados são criadas para garantir consistência e serialização, através das propriedades ACID (atomicidade, consistência, isolamento e durabilidade), enquanto que outras procuram disponibilidade tendo em conta as propriedades BASE (disponibilidade básica, estado flexível, eventual consistência).

De acordo com N. Khan et al. (2014), um dos desafios iniciais de *Big Data* foi o desenvolvimento de um sistema distribuído em larga escala para armazenamento, processamento e análises eficientes. Segundo estes autores, e baseando-se no Teorema CAP de Eric Brewer, existem três fatores a considerar no uso dos sistemas distribuídos para armazenar grandes quantidades de dados, os quais são representados na Figura 6.

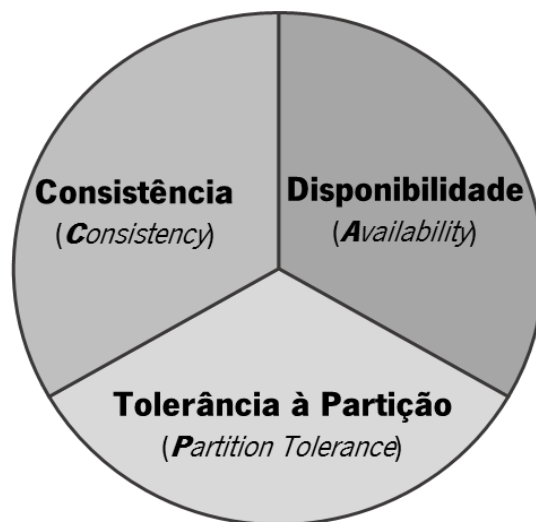


Figura 6 Teorema de CAP de Eric Brewer. Adaptado de (Han et al., 2011)

Assim, e segundo Han et al., 2011; Srivastava, Agarwal, Srivastava, & Pandey (2016), as bases de dados NoSQL podem ser classificadas de acordo com o teorema de CAP de Eric Brewer que assume que os sistemas distribuídos, apesar de ser o desejável, não conseguem responder às três propriedades simultaneamente, mas podem responder a pelo menos duas das três, que são:

1. **Consistência (C)**: esta característica, relacionada também com a atomicidade e isolamento, implica que todos os nós de um sistema tenham a mesma visão sobre os dados em qualquer momento (Gessert et al., 2016; Gilbert & Lynch, 2002).
2. **Disponibilidade (A)**: esta propriedade implica que qualquer pedido que seja recebido por um nó no sistema deve sempre resultar numa resposta, devendo o pedido terminar mesmo que existam falhas de rede (Gilbert & Lynch, 2002).

3. Tolerância à Partição (P): no caso de um sistema particionado não é possível manter a consistência e a disponibilidade simultaneamente (Gessert et al., 2016). Para modelar esta tolerância, a rede poderá perder arbitrariamente muitas mensagens enviadas de um nó para outro (Gilbert & Lynch, 2002). Assim, se o sistema continuar a trabalhar apesar da partição, há algum nó que perdeu contacto com os outros e, portanto, tem que se decidir se se continua com o processamento dos pedidos, e preservar assim a disponibilidade, ou se se rejeitam pedidos a fim de manter a consistência (Gessert et al., 2016).

Assim, tal como referido anteriormente, as bases de dados NoSQL podem ser classificadas segundo as duas propriedades a que conseguem responder, tal como se pode verificar na Tabela 2, que aproveita para dar exemplos de algumas bases de dados NoSQL.

Tabela 2 Classificação das bases de dados NoSQL segundo o Teorema de CAP

Combinação	Consistência	Disponibilidade	Tolerância à Partição	Exemplos
CA	X	X		Bases de dados relacionais; Greenplum ² ...
CP	X		X	Big Table ³ , Hypertable ⁴ , HBase ⁵ , MongoDB ⁶ ...
AP		X	X	CouchDB ⁷ , Voldemort ⁸ ; SimpleDB ⁹ ...

Pode então concluir-se que as bases de dados relacionais e as bases de dados NoSQL que incluem algumas das características das relacionais, conseguem assegurar consistência e disponibilidade, aliadas à atomicidade e isolamento. No entanto, as restantes bases de dados NoSQL, para garantir tolerância à partição, terão que abdicar da consistência ou da disponibilidade.

As bases de dados NoSQL são, normalmente classificadas em quatro tipos diferentes de acordo com o seu modelo de dados, representadas na Figura 7 (Cassavia et al., 2014).

² <http://greenplum.org/>

³ <https://cloud.google.com/bigtable/>

⁴ <http://www.hypertable.org/>

⁵ <https://hbase.apache.org/>

⁶ <https://www.mongodb.com/>

⁷ <http://couchdb.apache.org/>

⁸ <http://www.project-voldemort.com/voldemort/>

⁹ <https://aws.amazon.com/pt/simpledb/>

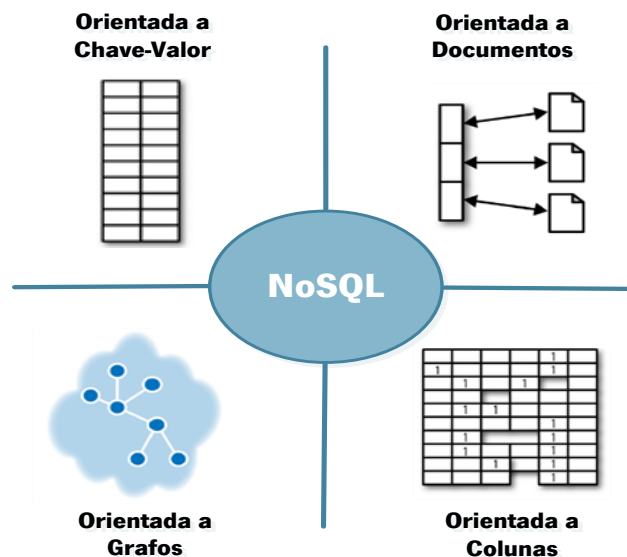


Figura 7 Tipos de bases de dados NoSQL. Adaptado de (Cassavia et al., 2014)

As bases de dados orientadas a documentos armazenam, como o nome indica, documentos, em formatos como XML, YAML ou JSON, sendo que cada documento contém determinado registo assim como os dados associados, reduzindo a necessidade de *joins* (Kaur & Rani, 2013). Estes sistemas tendem a incluir mapas e listas, permitindo hierarquias naturais. No nível mais simples deste tipo de modelos, os documentos podem ser armazenados e recuperados por um ID mas, no geral, dependem de índices criados de acordo com os seus atributos (Robinson, Webber, & Eifrem, 2015). Estes sistemas são livres de esquema e alguns exemplos de bases de dados deste tipo são o MongoDB e o CouchDB (Bhogal & Choksi, 2015).

As bases de dados orientadas a colunas armazenam grupos de dados em famílias de colunas e linhas. Cada linha pode estar associada a múltiplas colunas e ter uma linha. Numa base de dados de famílias de colunas, cada linha irá, então, representar uma determinada entidade abrangente (por exemplo, tudo sobre um cliente) (Robinson et al., 2015). Estas bases de dados são especialmente apropriadas em três situações: quando existe um grande número de colunas, se os dados forem de natureza muito dispersa e se existem mudanças frequentes no esquema de dados (Kaur & Rani, 2013). Exemplos deste tipo de base de dados são o HBase e o Cassandra¹⁰.

As bases de dados orientadas a grafos baseiam-se na criação de nós, nas suas relações e nas propriedades de cada nó. Nestes sistemas os dados são modelados como uma rede de relações entre pares chave-valor, daí serem apropriados quando se pretende perceber as relações entre grandes

¹⁰ <http://cassandra.apache.org/>

quantidades de dados diferentes. Exemplos de bases de dados deste tipo incluem o InfoGrid¹¹ e o Neo4J¹² (Kaur & Rani, 2013; Bhogal & Choksi, 2015).

Por último, as bases de dados chave-valor usam uma tabela de *hash* onde existe uma chave única e um apontador para um determinado conjunto de valores. Não existe esquema o que facilita a análise de dados sem estrutura (Bhogal & Choksi, 2015). A vantagem deste modelo de dados recai sobre a sua simplicidade, uma vez que este nível de abstração simples torna a partição e a consulta dos dados muito mais fácil, pelo que as bases de dados conseguem atingir baixa latência e altas taxas de transferência (Gessert et al., 2016). No entanto, quando são necessárias operações mais complexas sobre os dados então este modelo de dados já não é apropriado (Gessert et al., 2016).

Estas quatro classes de bases de dados lidam com diferentes tipos de conjuntos de dados, tendo cada um as suas vantagens ou desvantagens dependendo do contexto (Kaur & Rani, 2013).

Apesar das vantagens das bases de dados NoSQL, ainda existem alguns desafios inerentes, sendo a maturidade um dos principais problemas. Enquanto que as bases de dados relacionais são associadas à estabilidade e segurança, a verdade é que os sistemas NoSQL ainda são tecnologias muito novas e, na sua maioria, *open source* pelo que não fornecem suporte como acontece com a Oracle, Microsoft ou IBM (Huang & Luo, 2014).

Como o foco deste trabalho é o Hive, uma tecnologia particular baseada em tabelas e colunas, não se julga relevante estender as descrições associadas aos diferentes tipos de bases de dados NoSQL. Assim, será feita, posteriormente, uma descrição mais detalhada apenas para o Hive.

2.2.2 *Data Warehouses*

Até meados dos anos 80, as bases de dados armazenavam apenas dados operacionais, ou seja, dados criados pelas operações de negócio incluídas nos processos diários de gestão (compras, vendas, faturação...) (Golfarelli & Rizzi, 2009).

No entanto, no ambiente de negócios global e competitivo de hoje, perceber e gerir informação é crucial para as organizações de forma a conseguirem tomar decisões atempadas e responder às mudanças e exigências dos negócios (Helfert & Von Maur, 2001).

Data Warehousing é um fenómeno que surgiu como consequência das enormes quantidades de dados eletrónicos armazenados nos últimos anos e, também, da necessidade crescente de utilizar esses

¹¹ <http://infogrid.org/trac/>

¹² <https://neo4j.com/>

dados para alcançar aqueles objetivos que vão para além das tarefas rotineiras relacionadas com o processamento diário (Golfarelli & Rizzi, 2009). Para estes autores, *Data Warehousing* é um conjunto de métodos, técnicas e ferramentas usadas para suportar os “trabalhadores do conhecimento” (gestores, diretores, analistas) a conduzir análises de dados que os ajudem a melhorar o processo de tomada de decisão e também os recursos informacionais (Golfarelli & Rizzi, 2009).

Segundo Theodoratos & Sellis (1999), um *Data Warehouse* é uma base de dados que recolhe e armazena dados de múltiplas fontes de informação remotas e heterogéneas. Quando se processa uma consulta esta é avaliada localmente não sendo necessário aceder às fontes de informação originais.

Para Inmon (2005), um *Data Warehouse* é um conjunto de dados orientado a assuntos, integrado, não volátil, com variações no tempo, com dados corporativos granulares, que dá suporte às decisões da gestão. Quer isto dizer que um *Data Warehouse* está orientado para as principais áreas de uma empresa e que é alimentado por múltiplas fontes de dados o que implica que estas tenham que ser convertidas, formatadas, reorganizadas e integradas. Para além disso, é não volátil uma vez que quando armazenada a informação esta não se perde, sendo mantido um histórico de dados. É variante no tempo o que implica que apresente a evolução dos dados ao longo do tempo (Golfarelli & Rizzi, 2009; Inmon, 2005).

Um *Data Warehouse* é diferente de uma base de dados operacional. Enquanto que estas últimas são transacionais, os *Data Warehouses* têm as características diferenciadoras de estarem destinados a aplicações de apoio à decisão e de estarem otimizados para a recuperação de dados e não para o processamento de transações rotineiras (Elmasri & Navathe, 2003). Um *Data Warehouse* requer uma arquitetura que começa por um olhar pelo todo e a partir do qual vai descendo e trabalhando para os detalhes (Inmon, 2005).

Segundo Kimball & Ross (2013), num *Data Warehouse* existem quatro componentes que devem ser considerados e que estão representados na Figura 8.

1. Sistemas operacionais de origem: os sistemas que armazenam as transações do negócio, tratando-se, então, das fontes de dados que irão alimentar posteriormente o *Data Warehouse*.

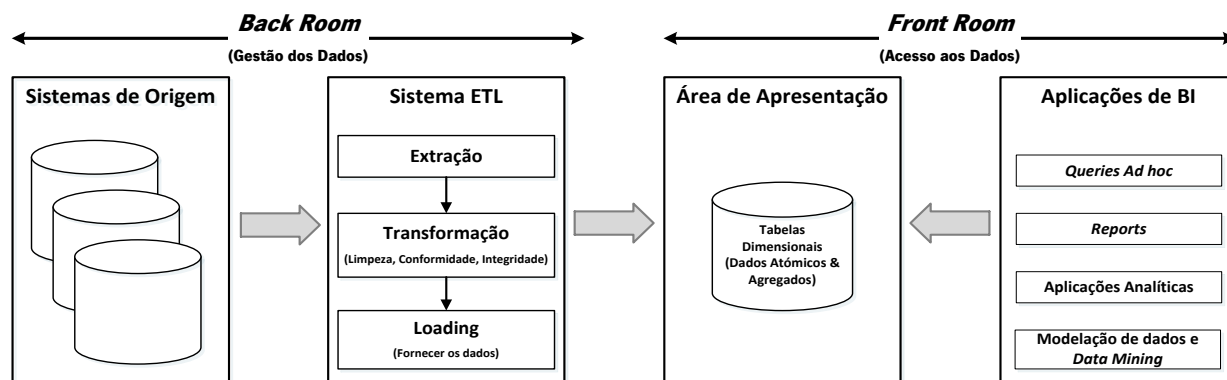


Figura 8 Elementos principais da arquitetura de Data Warehouse para Business Intelligence de Kimball. Adaptado de (Kimball & Ross, 2013)

2. Sistema ETL (extração, transformação e carregamento): engloba tudo o que acontece entre as fontes de dados operacionais e o *Data Warehouse*. A extração é o primeiro passo para a obtenção dos dados para o ambiente de *Data Warehousing*, e consiste na leitura e na compreensão das fontes de dados e na transferência dos dados necessários para futura manipulação. A transformação de dados inclui as tarefas de limpeza dos dados (correção de erros, resolução de conflitos, tratamento de *nulls...*) e de correspondência/ligação (combinar e integrar os dados) de forma a alterar e melhorar os dados a analisar. O carregamento dos dados e estruturação física dos dados consiste em carregar os dados para os modelos dimensionais que definem a estrutura do *Data Warehouse*, e que irá alimentar a camada de apresentação de dados (dimensões e factos).
3. Área de apresentação: é onde os dados são organizados, armazenados e tornados acessíveis para consultas por parte dos utilizadores. Esta deve conter detalhes, ou seja, dados atômicos, de forma a conseguir responder a consultas *ad hoc* por parte do utilizador.
4. Aplicações de *Business Intelligence*: referem-se ao conjunto de capacidades analíticas fornecidas aos utilizadores e que serão úteis para a tomada de decisão. Estas aplicações incluem ferramentas de consultas *ad hoc*, *dashboards* ou aplicações de *data mining*.

Genericamente, o esquema de um *Data Warehouse* consiste em dois tipos de elementos: factos e dimensões. Os factos são utilizados para armazenar indicadores relacionados com situações ou eventos, e as dimensões são utilizadas para analisar estas medidas através de operações de agregação (Schneider, 2008).

Segundo Di Tria, Lefons, & Tangorra (2014), os *Data Warehouses* tradicionais são desenhados de acordo com duas abordagens diferentes:

1. Orientada por dados: que consiste na reengenharia das fontes de dados. Esta metodologia dedica-se à definição de esquemas multidimensionais com base nas transformações pretendidas nas fontes de dados.
2. Orientada por requisitos: que tem como base os objetivos de negócio dos responsáveis pela tomada de decisão. Esta metodologia define esquemas multidimensionais através dos objetivos de negócio, sendo as fontes de dados consideradas mais tarde, no processo de ETL.

Estas duas abordagens podem ser combinadas, tirando partido dos dados e dos requisitos de negócio.

2.2.3 *Big Data Warehouses*

O volume de dados é, hoje, um grande desafio para o *Data Warehousing*. No entanto, também os tipos e os formatos dos dados hoje existentes surgem como um grande problema uma vez que colocam em causa as regras de processamento de um *Data Warehouse*. Isto porque, as regras inerentes a dados relacionais não podem ser aplicadas em textos, imagens ou vídeos gerados ou em alguns dados de sensores (Krishnan, 2013).

A influência da Web, dos dispositivos móveis e outras tecnologias que foram surgindo desafiaram as abordagens e os processos tradicionais. Os dados já não são centralizados e limitados aos sistemas das organizações estando altamente distribuídos e com diferentes estruturas e a crescer a uma taxa exponencial (Mohanty et al., 2013). Para estes autores, as organizações terão que desenvolver um novo ecossistema de plataformas que utilize não só os dados dos *Data Warehouses* tradicionais implementados, como as grandes quantidades de dados existentes e relevantes, através de arquiteturas de *Data Warehouse* híbridas cuidadosamente arquitetadas, um *Big Data Warehouse* (Mohanty et al., 2013).

Os *Big Data Warehouses* diferem substancialmente dos *Data Warehouses* tradicionais uma vez que o seu esquema deve ser baseado em modelos lógicos novos e mais flexíveis que os modelos relacionais (Di Tria et al., 2014). Como tal, nesta nova era de *Big Data*, os sistemas NoSQL têm atraído grande interesse como base para a investigação de novas oportunidades para *Data Warehousing*. Estas oportunidades incluem: explorar a escalabilidade, a flexibilidade e a possibilidade de armazenamento de dados heterogéneos (Chevalier, El Malki, Kopliku, Teste, & Tournier, 2016).

Segundo Di Tria et al. (2014), as metodologias para projetar um *Data Warehouse* em *Big Data* devem ser:

1. Híbridas, de forma a considerarem os benefícios inerentes às abordagens de *Data Warehouse* tradicionais, orientadas a dados e orientadas a requisitos;
2. Incrementais, com base nas abordagens ágeis;
3. Automáticas, a fim de tornar a sua conceção mais rápida, mesmo que se tenha que considerar várias fontes de dados.

Com a união destas duas últimas características às metodologias tradicionais espera-se ganhar rapidez na execução assim como rapidez na reação às mudanças nos requisitos do negócio (Di Tria et al., 2014).

Na mesma linha de trabalho, Mohanty et al. (2013) referem que a metodologia de conceção de um *Big Data Warehouse* deve ser uma abordagem altamente ágil e iterativa que, através da integração do maior número possível de fontes de dados (internas e externas), da definição ou não de modelos de dados, e da execução de algoritmos sofisticados, permita uma rápida compreensão e perceção dos dados.

Os *Big Data Warehouses* implicam novas características e mudanças na sua conceção (Goss & Veeramuthu, 2013; Mohanty et al., 2013), nomeadamente:

1. Capacidade de processamento de dados altamente distribuído;
2. Capacidade de suportar escalabilidade, a baixo custo;
3. Capacidade de analisar estes grandes volumes de dados sem recorrer a amostras;
4. Capacidade de processar e visualizar dados em tempo real, de forma a melhorar o processo de tomada de decisão;
5. Capacidade de integrar diversas estruturas de dados, seja de fontes de dados internas ou externas a uma organização;
6. Capacidade de gerir compensações entre consistência, disponibilidade e tolerância a partições;
7. Capacidade de suportar cargas de trabalho extremas, como consultas em profundidade ou consultas de amplitude (*ad hoc* e análises estratégicas) e ao carregar e processar dados em *batch* ou *streaming*.

Tal como referido anteriormente, no contexto de *Big Data* a modelação de dados muda de perspectiva uma vez que as bases de dados NoSQL, cada vez mais utilizadas neste contexto, são bases de dados livres de esquema podendo o esquema os dados mudar ao longo do tempo de acordo com as necessidades de armazenamento ou analíticas (Durham, Rosen, & Harrison, 2014; Santos & Costa, 2016a). No entanto, apesar de serem livres de esquema, estas bases de dados acabam por precisar de

modelos de dados para assegurar um armazenamento e uma análise adequados (Santos & Costa, 2016a). Este tópico será mais detalhado na secção seguinte.

2.2.4 Modelos de Dados

Uma característica fundamental das bases de dados é o facto de fornecerem um nível de abstracção dos dados, isto é, ocultam detalhes do armazenamento de dados que, na maior parte das vezes, não são necessários aos utilizadores. Os modelos de dados, um conjunto de conceitos que pode ser usado para descrever a estrutura de uma base de dados (tipos de dados, relações e restrições), fornecem os meios necessários para atingir esse nível de abstracção (Elmasri & Navathe, 2003).

Os modelos de dados são peças centrais em *Business Intelligence* e *Analytics*. São eles que garantem que as necessidades analíticas são devidamente consideradas, permitindo a sua análise através de diferentes perspetivas e de acordo com a abordagem utilizada para a definição das mesmas (Santos & Costa, 2016a, 2016b).

Elmasri & Navathe (2003) assumem a existência de vários modelos de dados que são classificados de acordo com os tipos de conceitos que usam para descrevem as estruturas de bases de dados e de acordo com o nível de detalhe que fornecem. Na Tabela 3 são apresentados os grupos de modelos de dados que são considerados por estes autores, dependendo do nível de abstracção utilizado.

Tabela 3 Grupos de modelos de dados e conceitos utilizados. Adaptado de (Elmasri & Navathe, 2003)

Modelo de Dados	Caracterização	Conceitos utilizados
Conceptual	Ou modelo de alto nível, fornece conceitos que são muito próximos da forma como os utilizadores compreendem os dados.	Entidades; Atributos; Relações.
Lógico	Ou modelo de implementação, fornece conceitos que podem ser compreendidos pelos utilizadores apesar de apresentarem detalhes sobre a forma como os dados são organizados no modelo de base de dados adaptado.	Estrutura dos registos.
Físico	Ou modelo de baixo nível, descreve e dá detalhes sobre a forma como os dados são armazenados nos computadores, sendo direccionados para especialistas na área.	Formatos dos registos; Organização dos registos; Caminhos de acesso...

Ao nível dos *Data Warehouses*, os autores Chevalier et al. (2016), Dehdouh (2016) e Jarke, Jeusfeld, Quix, & Vassiliadis (1999) distinguem, também, três níveis de abstracção, representados na

Figura 9: o modelo conceitual, que é independente de tecnologias; o modelo lógico, que corresponde a uma tecnologia específica, mas é independente do *software* utilizado; e o modelo físico que é o modelo de um *software* específico selecionado para a implementação. De acordo com estes autores, o esquema multidimensional é o modelo conceitual de referência para um *Data Warehouse*.

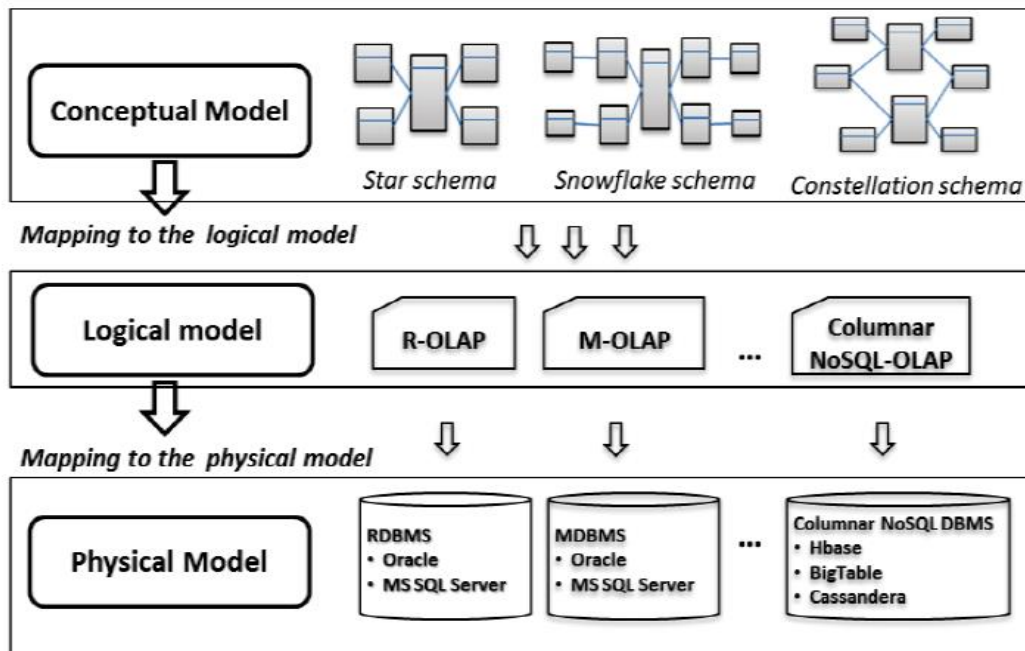


Figura 9 Processo de implementação dos modelos em Data Warehouse. Retirado de (Dehdouh, 2016)

Tal como referido anteriormente, a modelação multidimensional é amplamente aceite como a técnica de eleição para a estruturação de dados analíticos, uma vez que fornece dados compreensíveis aos utilizadores e proporciona, ao mesmo tempo, um rápido desempenho nas consultas, tornando a criação de bases de dados um processo mais simples (Kimball & Ross, 2013).

Considerando ainda a Figura 9, os modelos de dados mais populares e tradicionalmente utilizados para a criação de um *Data Warehouse* são o modelo em estrela, o modelo em floco de neve e o modelo em constelação (Dehdouh, Bentayeb, Boussaid, & Kabachi, 2015; Kimball & Ross, 2013). Sendo estes modelos os principais constituintes do modelo multidimensional, na Tabela 4 é apresentada uma breve descrição de cada um destes esquemas de dados.

Tabela 4 Esquemas de dados para criação de um Data Warehouse. Adaptado de (Kimball & Ross, 2013).

Modelo	Caracterização
Estrela	Estruturas dimensionais aplicadas num sistema de gestão de bases de dados relacionais. Estas consistem em tabelas de facto vinculadas a tabelas de dimensões através de relações de chave primária/secundária.

Modelo	Caracterização
Floco de Neve	Quando uma relação hierárquica numa tabela de dimensão é normalizada, os atributos de menor cardinalidade aparecem como tabelas secundárias dessa dimensão, criando um esquema em floco de neve. Este tipo de esquema dificulta a navegação pelos dados.
Constelação	União de um ou mais modelos em estrela, pela partilha de dimensões entre tabelas de factos.

Geralmente, o mapeamento entre o modelo conceptual e o modelo lógico é feito de acordo com três abordagens: ROLAP (*Relational OLAP*), MOLAP (*Multidimensional OLAP*) e HOLAP (*Hybrid OLAP*). Para além disso, os modelos de dados relacionais são os mais usados, sendo definidos segundo regras muito específicas relativamente aos requisitos que estes modelos devem considerar (Santos & Costa, 2016a). Contudo, os modelos lógicos tradicionais utilizados na construção de um *Data Warehouse* não são apropriados para os grandes volumes de dados inerentes a um contexto de *Big Data* pelo que se torna, mais uma vez, evidente que os *Big Data Warehouses* implicam a adoção de novos modelos lógicos, sendo as bases de dados NoSQL e algumas tecnologias disponíveis no Hadoop as mais recentes apostas para o armazenamento de dados (Dehdouh et al., 2015; Di Tria et al., 2014).

É de realçar que, ao contrário das bases de dados relacionais, onde a estrutura das tabelas é predefinida, nas soluções NoSQL são permitidas estruturas menos rígidas, não sendo necessárias grandes preocupações com a estrutura dos dados (Santos & Costa, 2016a; Vajk et al., 2013).

A total desnormalização dos dados pode ser uma forma de melhorar o desempenho das consultas (Vajk et al., 2013). Pode, no entanto, ser necessário adicionar estrutura aos dados quando as tarefas analíticas assim o exigem. Assim, dependendo do contexto, podem ser transformados num modelo de dados baseado em colunas a aplicar, por exemplo, no HBase quando se pensa num contexto de NoSQL, ou num modelo de dados tabular, como é o caso do Hive, sendo este último o modelo associado ao contexto de *Big Data Warehouses* (Santos & Costa, 2016a).

Assim, e focando no âmbito desta dissertação, quando o volume de dados justifica a implementação de um *Data Warehouse* em contexto de *Big Data*, um *Big Data Warehouse*, um modelo de dados tem de ser identificado de forma a definir a estrutura das tabelas que têm que ser criadas no Hive (Dere, 2017; Santos & Costa, 2016).

Desta forma, o estudo de Santos & Costa (2016b) é um bom exemplo da concretização de um *Big Data Warehouse*. Este propõe regras específicas para a estruturação de um modelo de dados no Hive, através da transformação de um modelo de dados multidimensional, normalmente utilizado para

modelar um *Data Warehouse* num contexto tradicional, num modelo de dados tabular que permita análises de dados com o Hive. Esta metodologia, dependendo do tempo disponível para a concretização da parte prática desta dissertação, poderá ser utilizada para um dos cenários de teste.

2.3 Otimização do Armazenamento e Processamento

De acordo com Fan et al. (2014), conceber sistemas computacionais de grande escala, altamente adaptáveis e tolerantes a falhas é um grande desafio e motiva a criação de uma infraestrutura de computação nova e confiável que suporte o armazenamento e processamento paralelos de dados em massa. A otimização é, assim, normalmente uma ferramenta, não um objetivo, na análise de *Big Data* (Fan et al., 2014).

As necessidades de processamento e análise de dados de baixa latência e quase em tempo real, surgem do facto de que os dados têm uma dimensão tempo associada e que se não forem processados e analisados naquele momento, o valor dos dados vai-se deteriorando significativamente (Mohanty et al., 2013). O processamento apropriado e atempado dos dados pode revelar novas informações sobre mercados, sobre a sociedade ou sobre o ambiente, permitindo uma melhor e mais rápida reação às oportunidades e mudanças que vão surgindo (J. Chen et al., 2013).

Dada a juventude desta área de investigação, não existem muitos trabalhos que se dediquem a analisar a problemática da organização dos dados, num *Big Data Warehouse*, para otimizar o seu processamento. De seguida são descritos alguns dos trabalhos disponíveis, a partir dos quais será destacado como o trabalho proposto nesta dissertação irá evoluir.

Recentemente, as plataformas de processamento de grande escala baseadas no *framework* de MapReduce (que será melhor explicado no capítulo seguinte) têm sido amplamente adotadas para o processamento escalável de dados estruturados ou dados sem estrutura (Kim, Ravindra, & Anyanwu, 2013). Com este *framework* é possível atingir um alto desempenho, ao explorar o paralelismo entre nós de processamento, ao mesmo tempo que fornece uma interface simples para as camadas superiores (Wu, Li, Mehrotra, & Ooi, 2011).

Nos estudos de Wu et al. (2011), apesar de se ter verificado uma melhoria nos sistemas de *Data Warehouse* com a integração do MapReduce, estes autores afirmaram que os sistemas de consultas baseados no MapReduce, de que é exemplo o Hive, ficam aquém das expectativas na otimização de consultas e das capacidades dos sistemas de bases de dados convencionais. Isto porque, por exemplo o Hive, para a execução de determinada consulta, traduz essa *query* num conjunto de trabalhos

MapReduce, assumindo que o utilizador otimizou a *query* antes de a introduzir no sistema, quando nem sempre é verdade.

Para melhorar este desempenho, neste estudo os autores apresentaram a conceção e implementação de um otimizador de consultas, o AQUA (*Automatic Query Analyzer*), para sistemas de *Data Warehouse* baseados em MapReduce. Assim, incorporaram, no Hive, um otimizador de consultas projetado para gerar um plano de *queries* eficiente baseado num modelo de custos proposto.

Para efeitos de comparação, estes autores avaliaram o desempenho de três planos: Hive *Manually Optimized*, onde todas as *queries* são otimizadas manualmente; o AQUA, que representa o melhor plano gerado pelo otimizador criado, e o Hive *Unoptimized*, que representa o pior plano para o modelo de custos. Para todos os testes executados, de otimização de *queries* e o efeito da escalabilidade na execução das mesmas, o AQUA apresenta, geralmente, os melhores resultados o que demonstra a eficiência deste otimizador de consultas.

Estes autores não consideraram, no entanto, a otimização do armazenamento dos dados, considerando apenas a otimização do processamento dos dados com base nas *queries* que são executadas e nas tarefas de MapReduce ou nos *joins* que serão, à partida, necessários.

Num outro trabalho, Chavalier et al. (2016) estudaram a implementação de um *Data Warehouse* baseada numa base de dados NoSQL orientada a documentos. Para isso, numa primeira fase analisaram as questões de modelação, consulta, carregamento de dados e de *cuboids* OLAP. Posteriormente, compararam os modelos orientados a documentos (com e sem normalização) com modelos de bases de dados relacionais, de forma a ilustrar as diferenças de desempenho. Os dados foram carregados a diferentes fatores de escala, isto é, com diferentes volumes de dados.

Ao comparar os dois modelos de dados utilizados, e testando o carregamento de dados a diferentes fatores de escala, estes autores concluíram que:

1. O modelo em estrela, normalizado, requer menos espaço e os dados são carregados de forma mais rápida;
2. Os modelos de dados desnormalizados mostram melhor desempenho nas consultas e as *queries* são mais fáceis de escrever.

Ao comparar os sistemas orientados a documentos com os sistemas de gestão de bases de dados relacionais, concluíram também que:

1. Os sistemas de gestão de bases de dados relacionais são mais rápidos a executar *queries* sobre o *raw data*, mas o desempenho diminui quando o volume de dados é superior ao suportado pela memória principal;
2. O sistema orientado a documentos analisado mostrou-se mais robusto, ou seja, não apresentou quedas significativas de desempenho com o aumento da escala, e mostrou, também, que beneficia da distribuição;
3. Os tempos de consulta diminuíram significativamente quando as *queries* são feitas diretamente sobre os *cuboids* OLAP.

Relativamente aos tipos de *cuboids* analisados (com e sem dados agregados), os autores concluíram ainda que:

1. Apesar de terem um custo de memória adicional, ambas tendem a executar bem em determinadas cargas de trabalho;
2. Os *cuboids nesting* (com agregados) suportam cargas de trabalho com consultas *drill-down*;
3. Os *cuboids detail* (com *raw data*) suportam consultas mais detalhadas;
4. Estes *cuboids* não são armazenados naturalmente em bases de dados relacionais, existindo a necessidade de transformar e dividir em tabelas separadas;
5. Os *cuboids* podem ser armazenados em sistemas orientados a documentos, sendo possível armazenar os *cuboids* devido à existência de *arrays* e de *nesting*.

Com este estudo os autores finalizam, concluindo que os sistemas orientados a documentos são um campo promissor para os *Data Warehouses* e propõem a investigação de otimizações com suporte ao MapReduce de forma a tornar possível uma análise mais rápida e avançada de dados.

Um outro esforço no sentido de otimizar o armazenamento e o processamento em contextos de *Big Data* surge no estudo de Yangu, Nabli, & Gargouri (2016), através da apresentação de uma abordagem para a implementação de um DW com base nos modelos NoSQL. Para isso, criaram um conjunto de regras que transformaram um modelo multidimensional em dois modelos NoSQL, um modelo orientado a colunas e um modelo orientado a documentos, definindo dois tipos de transformação para cada um: um simples e outro hierarquizado. Quer isto dizer que, no primeiro caso os factos e as dimensões foram armazenados numa família de colunas/coleções, enquanto que na segunda transformação são usadas diferentes famílias de colunas/coleções para armazenar os factos e dimensões, organizadas em hierarquias.

Para validar as transformações propostas, os autores implementaram quatro *Data Warehouses*: dois orientados a colunas, utilizando o Cassandra, e dois orientados a documentos, utilizando o MongoDB, e avaliaram-nos em termos de “latência dos pedidos de escrita” e de “latência dos pedidos de leitura” com o recurso ao *benchmark* TPC-DS.

Com os testes executados, puderam concluir que:

1. Em termos de latência nos pedidos de escrita, o *Data Warehouse* construído sobre o Cassandra é mais rápido que o *Data Warehouse* construído no MongoDB;
2. Em termos de latência nos pedidos de leitura, o *Data Warehouse* orientado a documentos é mais eficiente a lidar com os dois tipos de *queries* (simples, considerando apenas *joins* entre dimensões e atributos, e mais complexas, utilizando alguns operadores como *group by* ou *order by*) que foram executadas.
3. Os tempos de carregamento são mais rápidos no *Data Warehouse* orientado a colunas;
4. A maior diferença entre os dois sistemas está relacionada com as interrogações: para *queries* que exijam vários atributos, as abordagens orientadas a colunas levam mais tempo a responder.

Uma outra abordagem, proposta por Jukic, Jukic, Sharma, Nestorov, & Arnold (2017), pretende acelerar o processo de ETL para *Data Warehousing*, através da combinação de duas metodologias para o armazenamento de dados que poderiam aumentar a capacidade de processamento de grandes volumes de dados, e assim obter um suporte à tomada de decisão atempado e abrangente. Assim, a primeira metodologia envolvia a redução da complexidade do modelo de dados conceptual através de um processo de desnormalização do esquema em estrela, combinando com uma segunda metodologia que envolvia o uso de bases de dados baseadas em colunas (*columnar databases*), criando assim uma tabela totalmente desnormalizada (*pre-joined*) armazenadas em forma de coluna.

Os resultados obtidos por estes autores indicaram as vantagens desta combinação em termos de melhorias no desempenho do processo de ETL (um dos processos que, geralmente, consome mais tempo na implementação de *Data Warehousing*) e, ainda, melhorias no desempenho de determinadas *queries* analíticas.

Estes são apenas alguns exemplos de trabalhos que estudaram as transformações necessárias para o armazenamento e o processamento dos dados num contexto de *Big Data*, que incluem a tentativa de otimização das tarefas de MapReduce associadas ao Hive, assim como a otimização dos *Data*

Warehouses, através da sua implementação com base numa base de dados NoSQL orientada a documentos, num dos casos, ou através da sua implementação recorrendo à transformação de um modelo multidimensional em dois modelos lógicos NoSQL, orientado a colunas e orientado a documentos. Uma última tentativa aqui descrita procura a otimização do processo de ETL assim como do desempenho das consultas, através da implementação de tabelas desnormalizadas armazenadas em forma de coluna.

Como tal, e tendo em conta o foco deste trabalho, denota-se a ausência, na literatura, de trabalhos que se preocupem com a forma como os dados são modelados no Hive, e ainda como se pode otimizar a definição de partições e *buckets*, sendo esta uma característica relevante num contexto de *Big Data Warehouses* devido às melhorias no desempenho que podem ser conseguidas. Esta dissertação tem, então, como propósito implementar um estudo alargado sobre este assunto, a detalhar nos próximos capítulos.

3. HIVE PARA ARMAZENAMENTO E PROCESSAMENTO EM *BIG DATA*

Neste capítulo será, então, apresentado o Hive, uma solução *open source* para *Data Warehousing*, criada a partir do Hadoop¹³. No entanto, de forma a contextualizar esta ferramenta será, numa primeira fase, apresentado o ecossistema Hadoop, descrevendo as suas principais componentes e tecnologias. De seguida são apresentadas as principais características do Hive, como a linguagem de consultas utilizada e a sua arquitetura, sendo posteriormente feita a apresentação do Hive como repositório de dados. Por último, este capítulo dedica-se à apresentação de informação detalhada sobre partições e *buckets*, dada a sua importância para a presente dissertação.

3.1 Ecossistema do Hadoop

Com tem sido referido, o volume de dados produzido nos dias de hoje tem superado as capacidades de armazenamento e processamento. *Big Data* traz assim dois grandes desafios: como armazenar e processar com dados tão volumosos e como compreender os dados de forma a transformá-los em vantagem competitiva (Holmes, 2012).

O Hadoop surge como uma solução para resolver o processamento de *Big Data* numa plataforma de baixo custo e utilizando *commodity hardware* com alta escalabilidade e com processamento paralelo mais rápido, fornecendo, assim, os recursos computacionais para lidar com estas grandes quantidades de dados (Holmes, 2012; Krishnan, 2013).

O Hadoop é um *framework* de *software* baseado em Java para gestão e processamento distribuído de dados. Este utiliza o modelo de programação MapReduce e o seu próprio sistema de arquivos distribuído chamado HDFS (Hadoop Distributed File System) (Fan et al., 2014).

Inspirado no sistema de ficheiros da Google (GFS) e no seu paradigma de programação MapReduce, o Hadoop pode ser considerado um ambiente de computação construído sob um sistema de ficheiros e que oferece uma forma de paralelizar e executar programas num *cluster* de máquinas distribuído, concebido para operações com dados em larga escala (Holmes, 2012).

Segundo Gupta (2015) e Lam (2010), as características principais do Hadoop incluem:

1. **Acessibilidade:** O Hadoop corre em grandes *clusters* de máquinas comuns ou em serviços de computação em *cloud*, fornecendo um acesso fácil a todos os sistemas ultrapassando as barreiras da distância;

¹³ <http://hadoop.apache.org/>

2. Robustez: esta solução consegue ultrapassar facilmente falhas de funcionamento nas máquinas por ter sido arquitetado com o pressuposto de falhas de *hardware* frequentes;
3. Escalabilidade: o Hadoop escala linearmente de forma a suportar grandes quantidades de dados adicionando mais nós ao *cluster*;
4. Simplicidade: a sua simplicidade está presente no facto de permitir a escrita rápida e eficiente de programas paralelos de suporte, ao dar a possibilidade de os programadores desenvolverem programas em qualquer linguagem (Java, Python...);
5. Rentabilidade: prova ser rentável ao utilizar *hardware standard* e servidores de baixo custo.

Com esta ferramenta desaparecem as preocupações com o particionamento dos dados, com a determinação das tarefas que cada nó desempenha ou com a comunicação entre os nós, isto porque o Hadoop assegura essas tarefas (Lam, 2010).

Considerando agora a arquitetura de alto nível do Hadoop representada na Figura 10, este é caracterizado por ter uma arquitetura distribuída de *master-slave* que consiste num sistema de ficheiros distribuído (HDFS) para o armazenamento e o MapReduce para capacidades computacionais (Holmes, 2012).

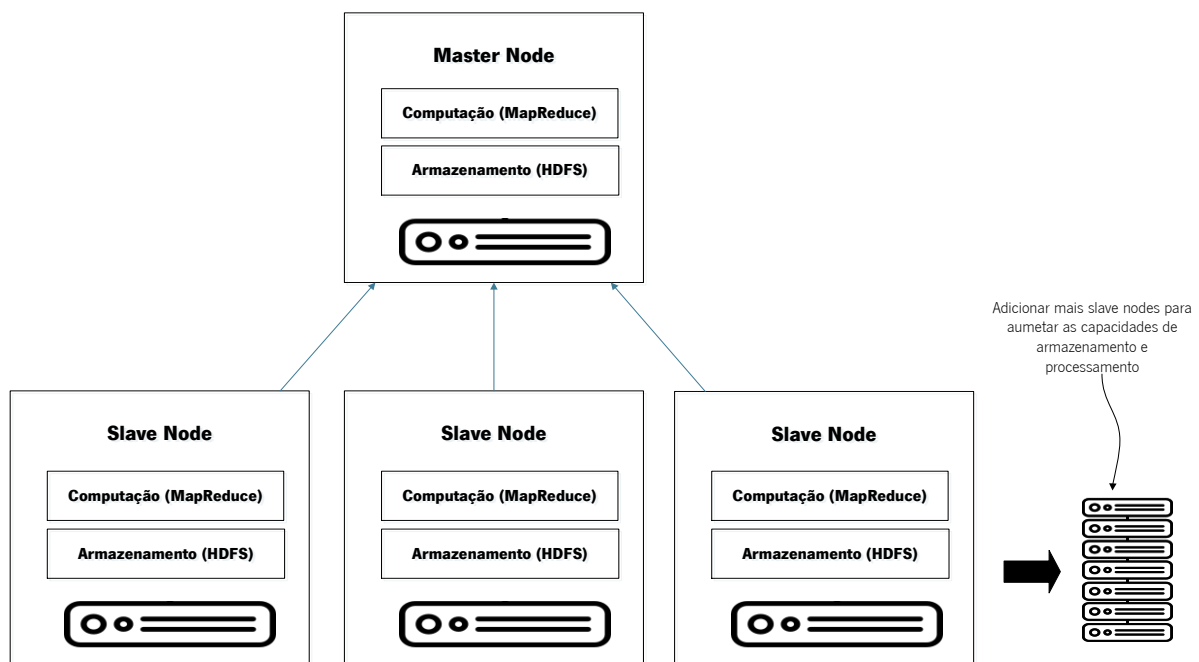


Figura 10 Arquitetura de alto nível do Hadoop. Adaptado de (Holmes, 2012)

Para lidar com os grandes volumes de dados, este fornece características de particionamento de dados e computação paralela, através de *slave nodes*. Assim, o MapReduce do *master* é responsável por organizar e agendar o trabalho computacional pelos vários *slave nodes* e o HDFS do *master* é responsável por particionar o armazenamento ao longo dos *slave nodes* e manter um registo de localização dos

ficheiros. O seu armazenamento e capacidades computacionais escalam com a adição de novos nós ao cluster, podendo atingir volumes de *petabytes* em *clusters* de milhares de nós.

Os principais módulos que constituem o Hadoop são apresentados na Tabela 5 (Apache, 2014; Sandoval, 2016).

Tabela 5 Principais módulos do Hadoop. Adaptado de (Apache, 2014; Sandoval, 2016)

Módulo	Funcionalidade
Common	Serviços comuns a todos as outras tecnologias, sendo o suporte para os mesmos.
HDFS	Hadoop Distributed File System (HDFS), um sistema de ficheiros distribuído responsável pelo armazenamento de grandes quantidades de dados.
Yarn	Framework de gestão de tarefas e de recursos do cluster.
MapReduce	Sistema utilizado para o processamento paralelo de grandes conjuntos de dados.

O Hadoop fornece várias ferramentas de análise, de *data warehousing* e de consulta, e ainda ferramentas de *data mining*, incluindo algoritmos de *machine learning* (Gupta, 2015). Estas tecnologias surgem como consequência da necessidade de aumentar a eficiência e o desempenho do Hadoop. O Hadoop, em conjunto com estas tecnologias, construídas com base neste, formam o ecossistema Hadoop. A Tabela 6 apresenta algumas destas tecnologias.

Tabela 6 Exemplos de tecnologias do ecossistema Hadoop. Adaptado de (Apache, 2014; Bhardwaj et al., 2015; White, 2012)

Tecnologia	Caracterização
Pig	Linguagem de scripts para a criação de programas para o Hadoop. Utiliza uma linguagem processual, o Pig Latin.
Hive	Utilizado para fins analíticos: agregação de dados, consultas <i>ad hoc</i> e análise de grandes quantidades de dados. <i>Data Warehouse</i> distribuído, gere os dados armazenados no HDFS e fornece uma linguagem de consultas baseada no SQL, o HiveQL.
Sqoop	Utilizado para transferir os dados entre o Hadoop e os servidores de bases de dados relacionais (importar dados das bases de dados relacionais para o HDFS e exportar do HDFS para sistemas relacionas).
HBase	São bases de dados usadas para armazenar as grandes quantidades de dados permitindo um acesso rápido e não sequencial às grandes quantidades de dados estruturados.

Tecnologia	Caracterização
Zookeeper	Permite a sincronização ao longo do cluster Hadoop, fornecendo controlo centralizado.
Flume	Utilizada para mover as grandes quantidades de dados de <i>datasets</i> de <i>streaming</i> para bases de dados centralizadas.
Oozie	Serviço para executar e agendar os fluxos de trabalho das tarefas do Hadoop.
Cassandra	Base de dados <i>multi-master</i> e escalável, sem pontos únicos de falha.
Chukwa	Sistema de recolha de dados para gestão dos sistemas distribuídos.
Spark	Fornecer um modelo de programação simples que suporta, por exemplo, o ETL, <i>machine learning</i> e processamento em <i>streaming</i> .
Tez	Substituição do MapReduce como mecanismo de execução subjacente. É um <i>framework</i> de programação de fluxo de dados geral, construído no Hadoop Yarn, que fornece um mecanismo poderoso e flexível para processar dados em batch ou interativos.
Impala	Permite consultas diretas aos dados armazenados no HDFS, melhorando o desempenho.
Presto	

De seguida são descritos os dois componentes de alto nível do Hadoop, o HDFS e o MapReduce.

3.1.1 HDFS

O HDFS é o sistema de ficheiros distribuído que surgiu com base no sistema de ficheiros da Google, sendo otimizado para altas taxas de transferência e para a leitura e escrita de grandes volumes de dados. A escalabilidade e a disponibilidade são, também, características chave do HDFS, devido em parte à replicação de dados e tolerância a falhas (tanto de *software* como de *hardware*) (Holmes, 2012).

O HDFS gere os dados do *cluster*, sendo cada bloco de dados replicado várias vezes (normalmente três cópias) de forma a que uma falha num disco ou no servidor não implique a perda de dados ou a capacidade de os processar (Capriolo, Wampler, & Rutherglen, 2012). Este re-replica automaticamente os blocos de dados nos nós que falham (Holmes, 2012).

A arquitetura do HDFS evoluiu a partir da arquitetura NDFS (*Nutch Distributed File System*, desenvolvida pela equipa Nutch), que por sua vez é baseada na arquitetura GFS (sistema de ficheiros desenvolvido pela Google) (Krishnan, 2013).

Os principais blocos constituintes do HDFS são: o *NameNode*, que mantém em memória os metadados sobre o sistema de ficheiros; e os *DataNodes*, que gere os blocos de dados para cada ficheiro (Krishnan, 2013).

3.1.2 MapReduce

O MapReduce é um modelo de computação que decompõe as tarefas de manipulação de grandes quantidades de dados em tarefas individuais que podem ser executadas em paralelo através do *cluster* de servidores (Capriolo et al., 2012).

Assim sendo, o MapReduce é um modelo de programação desenvolvido pela Google, para o processamento paralelo de dados (Capriolo et al., 2012; White, 2012). Este modelo simplifica o processamento paralelo ao abstrair a complexidade associada aos sistemas distribuídos, como a paralelização computacional, a distribuição das tarefas e lidar com *hardware* e *software* instável/não confiável. Com esta abstração, os programadores podem focar-se na resolução das necessidades do negócio em vez de se preocuparem com as complicações inerentes aos sistemas distribuídos (Holmes, 2012).

O MapReduce divide o processamento em duas fases principais: a fase de mapear (*Map*) e a fase de reduzir (*Reduce*) (Capriolo et al., 2012; White, 2012)). Cada fase tem pares chave-valor como *input* e *output* (White, 2012). De uma forma muito geral, na fase de mapear processa-se um par chave-valor, a partir do qual se gera um conjunto de pares chave-valor intermédios, e na fase de reduzir agregam-se todos os valores intermédios com a mesma chave intermédia (Cassavia et al., 2014).

3.2 Características do Hive

O Hive foi criado pelo Facebook em janeiro de 2007 como forma de melhorar as capacidades de consulta do Hadoop até então muito limitadoras e pouco produtivas (Thusoo et al., 2010). Assim, o Hive é uma infraestrutura de *Data Warehouse open source*, construída sobre o Hadoop, que facilita as consultas e a gestão de grandes volumes de dados armazenados em ambiente distribuído (Cassavia et al., 2014; Sandoval, 2016; Santos & Costa, 2016).

Esta ferramenta mantém os conceitos de tabelas, colunas, partições e *buckets* num ambiente não estruturado, mantendo a extensibilidade e a flexibilidade inerentes ao Hadoop (Thusoo et al., 2010).

Para cada base de dados, são criadas tabelas e cada tabela é armazenada numa diretoria do HDFS (Cassavia et al., 2014).

Segundo a documentação oficial, algumas funcionalidades/recursos fornecidos pelo Hive incluem (Dere, 2017):

1. Ferramentas para acesso fácil aos dados através de SQL, permitindo tarefas de *Data Warehousing* como extração/transformação/carregamento (ETL), *reporting* e análise de dados.
2. Mecanismos para impor uma estrutura na variedade de formatos de dados existentes;
3. Acesso a ficheiros armazenados tanto no HDFS como em outros sistemas de armazenamento de dados como o HBase;
4. Execução de queries através do Tez¹⁴, Spark¹⁵ ou através do MapReduce;
5. Linguagem processual com o HiveQL.

Não existe um formato único no qual os dados devem ser armazenados, sendo que este suporta ficheiros de texto com separação de atributos através de vírgulas ou Tabs (CSV ou TSV), ou outros formatos como Parquet, ORC (*Optimized Row Columnar*) ou outros (Dere, 2017). Uma vez que não é um tema central para esta dissertação, não serão dados pormenores acerca dos formatos existentes.

3.2.1 Arquitetura do Hive

Os principais componentes do Hive, maioritariamente representados na Figura 11, são (Dere, 2017; Krishnan, 2013; Thusoo et al., 2009, 2010):

1. *Metastore*: o catálogo do sistema que armazena os metadados sobre tabelas, colunas e partições.
2. *Driver*: o componente que recebe as queries. Este gere os detalhes das sessões, a forma como é assegurada, as estatísticas e gere o ciclo de vida de uma instrução HiveQL à medida que esta é processada no Hive. O *driver* compila os *inputs*, otimiza a computação e executa os pedidos através de três componentes:
 - a. *Compiler*: componente que decompõe a *query*, realizando a análise semântica da mesma e que gera um plano de execução de acordo com os dados do *metastore*.
 - b. *Optimizer*: componente que otimiza o plano criado pelo *compiler*,
 - c. *Execution Engine*: componente que executa o plano criado pelo *compiler*. Este gere as dependências entre as diferentes fases do plano e executa estas fases nos componentes adequados do sistema.
3. *Thrift server*: dá acesso ao Hive através de uma única porta, permitindo conexão programática ao Hive. É um *framework* para serviços de *cross-language* (cruzamento de

¹⁴ <https://tez.apache.org/>

¹⁵ <http://spark.apache.org/>

linguagens), em que um servidor escrito numa determinada linguagem (por exemplo, em Java), consegue suportar clientes de outras linguagens.

4. JDBC/ODBC: APIs que fornecem acesso ao Hive através de diferentes aplicações. Por exemplo, o JDBC permite o acesso a aplicações Java e o ODBC a aplicações C++.
5. CLI (*Comand Line Interface*) e HWI (*Hive Web Interface*): duas interfaces do cliente que permitem a comunicação com o HDFS através do *driver*. A primeira permite a execução em linhas de comando e a segunda é uma *interface web* (por exemplo, o Hue).

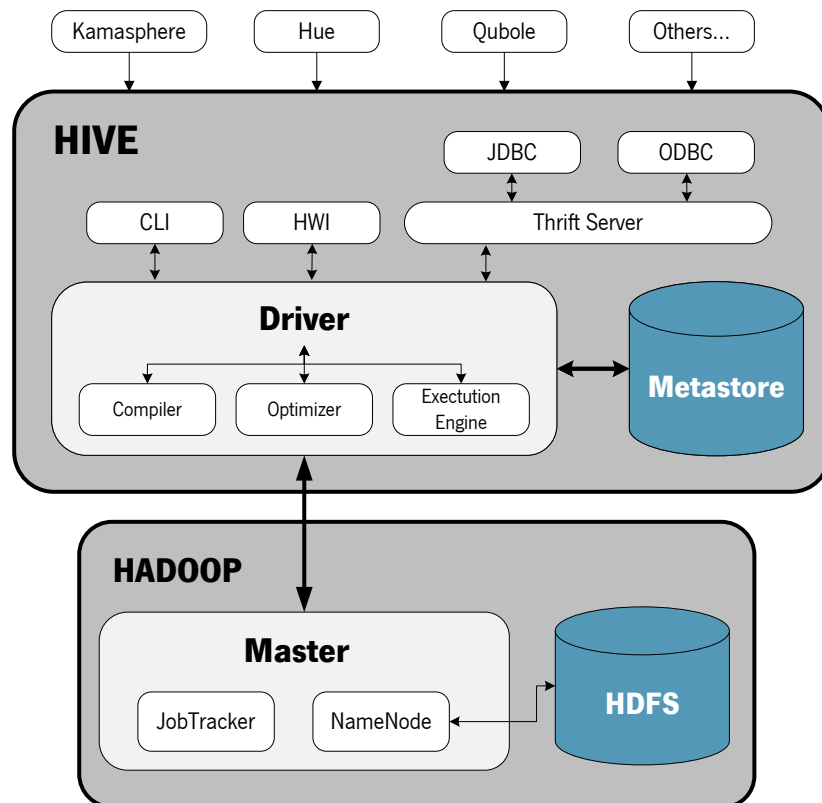


Figura 11 Arquitetura do Hive. Adaptado de (Capriolo et al., 2012; Dere, 2017; Krishnan, 2013)

Considerando, ainda, a arquitetura representada na Figura 11, esta representa também um fluxo típico do sistema, em que um utilizador, através de uma das *interfaces*, executa um comando HiveQL. Todos os comandos e *queries* vão para o *driver* que compila os *inputs*, otimiza a computação necessária e executa os passos necessários para processar os pedidos, normalmente com tarefas MapReduce. Isto é, o *driver* inicia uma sessão e envia a *query* para o *compiler* que extrai os metadados do *Metastore* e gera um plano de execução. Este plano lógico é, então, otimizado pela componente de otimização de *queries* do Hive e enviado para o *execution engine* que traduz esse plano em múltiplas fases, executando-as de acordo com as suas dependências e nos componentes apropriados (no *JobTracker*, se o plano inclui tarefas de MapReduce; no *Metastore* se este inclui uma operação aos metadados; ou no

NameNode, se este inclui operações diretas ao HDFS). Por exemplo, para executar um conjunto de tarefas de *map* e de *reduce*, o Hive comunica, então, com o *JobTracker*, normalmente de forma remota, de forma a iniciar a tarefa MapReduce (Capriolo et al., 2012; Dere, 2017; Krishnan, 2013; Thusoo et al., 2010).

Geralmente os ficheiros de dados a processar estão no HDFS que é gerido pelo *NameNode*. O Hive armazena as tabelas criadas pelo utilizador ou acede aos dados através de *interfaces* com sistemas externos, e o *Metastore* armazena as informações sobre as tabelas, como número de partições ou o tamanho da tabela, que podem ser usadas para otimização de *queries* (Krishnan, 2013; Thusoo et al., 2010).

Depois de terminadas as tarefas de MapReduce, o *driver* recolhe os resultados da *query* enviando-os para o utilizador que a submeteu (Huai et al., 2014).

Esta arquitetura foca a execução de *queries* através da utilização do MapReduce. No entanto, com a evolução das tecnologias associadas ao ecossistema Hadoop, o Hive tem agora utilizado o Apache Tez como o seu mecanismo de execução. O Tez é uma nova aplicação construída sobre o Hadoop Yarn que pode executar tarefas complexas de processamento, sendo então considerado um sucessor do MapReduce, mais flexível e eficiente no planeamento e execução de consultas (Dere, 2017).

3.2.2 Linguagem HiveQL

O HiveQL é a linguagem de consultas do Hive. Esta linguagem organiza e faz consultas aos dados armazenados no Hive (Gruenheid, Omiecinski, & Mark, 2011), imitando a sintaxe da linguagem SQL para a criação, carregamento e consulta de tabelas.

Existem várias formas de manipular os dados no Hive, sendo possível criar, alterar e eliminar bases de dados, tabelas, *views*, funções e índices (Capriolo et al., 2012). Uma das grandes limitações do Hive estava relacionada com o facto de que até há algumas versões atrás, este não suportava operações de remoção ou atualização de registos. No entanto, a partir da versão 0.14 este passou a disponibilizar a execução destas operações desde que as tabelas suportem as propriedades ACID (Dere, 2017).

A Tabela 7 apresenta exemplos de algumas das operações que se podem executar sobre os dados recorrendo à linguagem HiveQL, sendo estas operações relacionadas com a definição, a manipulação e a consulta de dados.

Tabela 7 Exemplos de operações sobre os dados recorrendo à HiveQL

Tipo de Operação	Exemplos
Definição de dados	Create; Drop; Alter; Truncate; Show; Describe.
Manipulação de dados	Load; Insert; Update; Delete; Import/Export; Explain Plan.
Consulta de dados	Select.

A Figura 12 representa um dos exemplos mais simples associados à criação e à eliminação de uma base de dados no Hive.

```
CREATE DATABASE IF NOT EXISTS employees (
    code      string,
    name      string,
    country   string,
    city      string,
    salary    float
);

SELECT name, salary FROM employees WHERE country = 'PT';

DROP DATABASE IF EXISTS employees;
```

Figura 12 Exemplos da aplicação da linguagem HiveQL

Como se pode verifica, tanto com a Tabela 7 como com a Figura 12, a linguagem acaba por ser intuitiva para experientes na linguagem SQL, dada a sua semelhança (Capriolo et al., 2012).

Uma característica exclusiva do HiveQL é a possibilidade de inserção simultânea de várias tabelas. Como consequência, os utilizadores podem executar múltiplas consultas nos mesmos dados de entrada usando uma única *query* Hive QL, aumentando assim o desempenho dessas consultas (Dere, 2017).

A linguagem de consultas do Hive pode, ainda, correr em diferentes *frameworks* computacionais, como MapReduce, o Tez ou o Spark para melhorar o desempenho (Capriolo et al., 2012; Dere, 2017).

3.3 Hive como Repositório de Dados

Num contexto de *Big Data*, o Hive é usado como um mecanismo de armazenamento distribuído, fornecendo capacidades de *Data Warehousing* (Santos & Costa, 2016).

Este estrutura os dados com base em três elementos principais - tabelas, partições e *buckets* - e suporta a maior parte dos tipos primitivos - inteiros, *floats*, *doubles*, *strings* - assim como tipos complexos como mapas, listas e estruturas (Thusoo et al., 2010).

O Hive não é uma base de dados completa, já que as limitações de conceção e as limitações do próprio Hadoop e do HDFS acabam por limitar também as funcionalidades do Hive. Uma das grandes limitações, apesar de já se começar a considerar em versões mais recentes, é o facto de não fornecer, totalmente, operações de *update*, *insert* ou *delete* ao nível do registo. Para além disso, o Hive não trabalha ao nível transaccional, não fornecendo, por isso, os recursos necessários ao OLTP (*Online Transaction Processing*) (Capriolo et al., 2012).

Assim, o Hive é mais adequado para aplicações de *Data Warehouse* onde os dados a analisar são relativamente estáticos, sem grandes alterações ao longo do tempo ou alterações constantes e quando não são necessários tempos de resposta rápidos (Capriolo et al., 2012).

O modelo de dados do Hive fornece uma estrutura em tabelas de alto nível sobre o HDFS, suportando três tipos de estruturas de dados: tabelas, partições e *buckets*, caracterizadas na Tabela 8 (Dere, 2017; Thusoo et al., 2009).

Tabela 8 Elementos do Hive. Baseada em (Dere, 2017; Thusoo et al., 2009; Du, 2015; Santos & Costa, 2016b)

Elemento	Caracterização
Tabelas	Semelhantes às tabelas das bases de dados relacionais (estruturas comuns, com colunas e linhas). A cada tabela corresponde uma diretoria do HDFS, e podem ser filtradas, projetadas e unidas.
Partições	Cada tabela pode ter uma ou mais partições que determinam a distribuição dos dados dentro de subdiretorias da diretoria da tabela. Estas permitem dividir horizontalmente os dados e acelerar o processamento de <i>queries</i> .
<i>Buckets</i>	Os dados de cada partição podem ser divididos em <i>buckets</i> com base no <i>hash</i> de uma coluna na tabela. É uma técnica para agrupar dados verticalmente, agrupando-os por determinado atributo. Cada <i>bucket</i> é armazenado como um ficheiro dentro da diretoria da partição.

Como tal, e segundo Santos & Costa (2016a), em contextos de *Big Data*, a identificação de um modelo de dados tabular que suporte as necessidades analíticas é a base para o *Data Warehousing* no Hive.

O Hive também suporta o conceito de tabelas externas, em que uma tabela pode ser criada em arquivos ou diretorias do HDFS pré-existent, fornecendo o local apropriado para a DDL (*Data Definition Language*) da criação da tabela. Este suporta ainda a criação de tabelas temporárias que são automaticamente eliminadas quando terminada a sessão (Dere, 2017).

3.4 Partições e *Buckets* de Dados

Dada a importância do estudo das partições e dos *buckets* no âmbito desta dissertação, esta secção apresenta de forma mais detalhada os dois conceitos.

Tal como referido anteriormente, o Hive conta com 3 elementos para organizar os dados: as tabelas e as partições, e os *buckets* que podem ser definidos sobre essas mesmas tabelas e partições. Esta divisão da tabela em partições e, posteriormente, por *buckets* pode melhorar o desempenho das consultas executadas sobre o Hive, já que agrupam os dados em partes mais pequenas e mais fáceis de aceder (Santos & Costa, 2016).

3.4.1 Partições

Um dos grandes problemas num *Data Warehouse*, seja tradicional seja em ambientes de *Big Data*, é como fazer o particionamento de dados. Particionamento de dados refere-se à divisão de dados em unidades físicas separadas que podem ser tratadas independentemente. Um particionamento apropriado podem trazer benefícios para um *Data Warehouse* a nível de carregamento, acesso, armazenamento e monitorização dos dados (Inmon, 2005).

Segundo Shaw, Vermeulen, Gupta, & Kjerrumgaard (2016), o particionamento do Hive pode melhorar o desempenho de um conjunto muito específico de *queries* uma vez que elimina da pesquisa as partições que não são necessárias para responder a essas *queries*.

Os dados podem ser divididos segundo vários critérios: por data, por linha de negócio, por geografia, por unidade organizacional ou por todas as anteriores, sendo que, particionar por data é, normalmente, quase obrigatório (Inmon, 2005).

O atributo ou atributos pelos quais será aplicada a partição deverá ter baixa cardinalidade de forma a evitar a criação de um número muito elevado de subdiretorias, não sobrecarregando o HDFS. Assim, algumas das dimensões que podem ser usadas para particionar os dados são (Dere, 2017):

1. Partições como data e tempo: quando os dados estão associados por tempo, usar, por exemplo, partições por ano, por mês e por dia ou mesmo horas;
2. Partições por localizações: quando os dados são relacionados com a localização, usar, por exemplo, partições por país, estado ou cidade já que o número de valores possível é limitado.
3. Partições por lógica de negócio: quando os dados estão associados a alguma lógica de negócio particionar, por exemplo, por departamento, por região de vendas ou por clientes.

Considere-se o exemplo, representado na Figura 13. Neste caso, uma tabela de empregados é particionada por país e por estado. Mesmo que se tenha milhares de diretorias de países e de estados,

estando a tabela particionada e realizando consultas por determinado estado, todas as diretorias podem ser ignoradas com exceção da diretoria que contém esse estado.

```
CREATE DATABASE IF NOT EXISTS employees (  
    code      string,  
    name      string,  
    country   string,  
    state     string,  
    salary    float  
)  
PARTITIONED BY (country string, state string);
```

Figura 13 Exemplo de criação das partições "country" e "state"

Assim, particionar tabelas muda a forma como o Hive estrutura o armazenamento de dados (Capriolo et al., 2012). No seguimento do exemplo dado anteriormente, o Hive cria subdiretorias que refletem a estrutura de partições definida, tal como se pode verificar na Figura 14.

```
...  
.../employees/country=CA/state=AB  
.../employees/country=CA/state=BC  
...  
.../employees/country=US/state=AL  
.../employees/country=US/state=AK  
...
```

Figura 14 Diretorias criadas pelo Hive de acordo com as partições. Retirado de (Capriolo et al., 2012).

A característica de particionar os dados é muito útil no ambiente Hive, isto porque, normalmente, o Hive realiza análises completas sobre todos os dados para satisfazer uma consulta o que pode comprometer o seu desempenho. No entanto, é importante realçar que a criação de muitas partições pode otimizar algumas consultas, mas prejudicar outras consultas. Para conjuntos de dados muito grandes, o particionamento pode melhorar drasticamente o desempenho das consultas, mas isto só acontece se o esquema de partições refletir filtros comuns aos dados (Capriolo et al., 2012).

Portanto, se as partições são relativamente pequenas então, o custo associado à pesquisa por várias diretorias é superior à simples pesquisa pela totalidade dos dados, pelo que não se devem considerar. Para além disso, estas devem ter tamanhos semelhantes para evitar ter uma única execução muito longa (Capriolo et al., 2012).

Um esquema de partição ideal não deve resultar num grande conjunto de partições e diretorias, sendo que os ficheiros em cada diretoria devem ser grandes de forma a não só melhorar o desempenho na resposta às *queries* como para evitar comprometer o desempenho do HDFS, que foi concebido para muitos milhões de ficheiros grandes e não muitos milhões de ficheiros pequenos (Capriolo et al., 2012). Como tal, um esquema deve encontrar um balanço entre número de partições, de forma a manter um número equilibrado de diretorias, e o tamanho de cada partição, para que os ficheiros tenham um

tamanho adequado às características do HDFS e para que todas as diretorias tenham uma distribuição de valores aproximada (Capriolo et al., 2012; Santos & Costa, 2016).

3.4.2 Buckets

As partições oferecem uma forma conveniente de separar os dados e de, assim, otimizar as consultas. No entanto, nem sempre se consegue otimizar apenas com esta abordagem, dadas as dificuldades de obter um particionamento adequado dos dados (Capriolo et al., 2012). Como visto anteriormente, o particionamento requer um atributo específico que não implique a criação de um número elevado de pequenas partições. Em casos que os atributos têm um conjunto elevado de valores possíveis e um número baixo de registos por cada valor, o particionamento não será a melhor estratégia, já que um número elevado de ficheiros pequenos têm, normalmente, um processamento mais lento no Hadoop do que poucos ficheiros de tamanho maior. Assim sendo, o *bucketing* surge aqui como uma outra técnica para decompor grandes conjuntos de dados em partes mais fáceis de gerir (Capriolo et al., 2012; Shaw et al., 2016). Os *buckets* correspondem a segmentos de ficheiros no HDFS, podendo apenas ser aplicados a um único atributo. Estes ajudam a organizar os dados em cada partição através da sua divisão por vários ficheiros (Santos & Costa, 2016).

Considerando a Figura 15 tem-se, agora, como exemplo a tabela “employees” que usa o país como primeiro nível de partição e que cria *buckets* pelo código de empregado.

```
CREATE DATABASE IF NOT EXISTS employees (  
  code      string,  
  name      string,  
  country   string,  
  state     string,  
  salary    float  
)  
PARTITIONED BY (country string)  
CLUSTERED BY (code) INTO 32 BUCKETS;
```

Figura 15 Definição de buckets por “code”.

Se surgisse um pedido para considerar o código de empregado como o segundo nível de partição, isto levaria à criação de muitas partições pequenas e, conseqüentemente, a muitas diretorias. Ao criar *buckets* em vez de partições, o valor desta coluna sofre uma divisão segundo um número de *buckets* definido pelo utilizador (32), sendo os registos com o mesmo código armazenados no mesmo *bucket* (no mesmo segmento de ficheiro) (Capriolo et al., 2012; Du, 2015).

Assim, a definição de *buckets* será mais adequada para atributos com alta cardinalidade, ou seja, com um grande conjunto de valores possíveis. Para além disso, um mesmo conjunto de dados é sempre escrito no mesmo *bucket* o que ajuda na execução de *joins* entre tabelas (Santos & Costa, 2016).

Para além de possibilitar uma melhoria do desempenho das consultas, a criação de *buckets* apresenta outras vantagens (Capriolo et al., 2012):

1. O número de *buckets* é fixo, não mudando com oscilações nos dados;
2. É uma técnica ideal para amostragem, lidando com as amostras de forma mais eficiente;
3. Ajuda a fazer *mapside joins* eficientes (permite a união de *buckets* individuais de tabelas diferentes, na fase de mapeamento).

Tal como na definição de partições, para definir um número adequado de *buckets*, deve ter-se em conta a quantidade de dados que fica armazenada em cada *bucket*, evitando ter demasiados dados ou poucos dados (Du, 2015).

Assim, e considerando os estudos de Du, 2015; Prokopp (2014), Santos & Costa (2016), Shaw et al. (2016), assim como conselhos práticos dados por profissionais da área no site da comunidade Hortonworks¹⁶, surgem algumas considerações quanto à criação de *buckets*:

1. Deve-se selecionar um atributo com um número elevado de valores distintos para a criação de *buckets*;
2. As técnicas de *bucketing* são mais apropriadas para *join* de tabelas de factos (*join* de duas tabelas de grandes dimensões);
3. Tabelas com *buckets* pela mesma coluna potenciam o desempenho dos *map side joins*;
4. A estratégia de definição do número de *buckets* deve ter em consideração que um único núcleo do CPU escreve num único *bucket* o que, em casos de se ter um cluster de grande dimensão, este pode estar a ser subutilizado se o número de *buckets* criado for demasiado pequeno;
5. Os *buckets* ajudam a aumentar o desempenho do carregamento dos dados para a tabela, facilitando a definição do número de tarefas de redução a executar (normalmente o número de *buckets* é igual ao número de tarefas de redução executadas);
6. O número de *buckets* não pode ser alterado depois da tabela estar criada;
7. A criação de *buckets* por atributos do tipo *int* é mais estável do que a criação de *buckets* por atributos do tipo *string*, que podem levar a problema de assimetria nos dados (não conseguir prever de forma correta quais os valores relacionados e que devem ficar no mesmo *bucket*);

¹⁶ <https://community.hortonworks.com/index.html>

8. Há profissionais a referirem que um *bucket* deve ter pelo menos um tamanho mínimo de um bloco HDFS, ou múltiplos desse tamanho, no entanto outros acreditam que, para uma organização mais eficiente, o tamanho mínimo de um *bucket* deve ser 1GB;
9. Em todos os carregamentos dos dados para as tabelas deve-se reforçar/impor a utilização de *bucketing* através da execução do comando “set hive.enforce.bucketing=true”;
10. A utilização de técnicas de *bucketing* ajuda na organização dos dados pelos ficheiros pelo que pode trazer vantagens quando a ordem global dos dados é importante nas consultas executadas. Isto é, quando se consideram os atributos que estão presentes nas condições que agrupam/organizam os dados nas *queries* (*group by*, *order by*...), o processamento destas condições poderá ser potenciado.
11. A combinação de técnicas de bucketing com a utilização de técnicas de ordenação dos dados (*sorted by*) pode ter impacto positivo na rapidez de processamento.

Com esta secção é possível, então, concluir que, em geral, a definição de *buckets* funcionará bem quando o atributo tem elevada cardinalidade enquanto que a definição de partições funcionará melhor quando feita sobre atributos com baixa cardinalidade e quando os registos estão bem distribuídos pelos diferentes valores possíveis que o atributo pode assumir (Capriolo et al., 2012; Santos & Costa, 2016). Com a parte experimental desta dissertação serão utilizadas algumas destas considerações de forma a tentar verificar a sua validade, assim como perceber novos padrões na utilização deste tipo de estratégia de organização dos dados.

4. AMBIENTE DE TESTES E DADOS UTILIZADOS

Considerando o objetivo principal desta dissertação, que consiste na proposta de um conjunto de regras que guiem a modelação e organização dos dados a armazenar em *Big Data Warehouses*, numa primeira fase é necessário perceber o impacto que diferentes estratégias de organização dos dados terão nos tempos de processamento dos mesmos para, então, inferir algum tipo de padrão. Assim a fase experimental da dissertação vai integrar dois momentos principais:

1. Perceber variações nos tempos de processamento através de vários cenários de teste (diferentes modelos de dados, diferentes implementações de partições e *buckets*, diferentes volumes de dados);
2. Identificar as heurísticas para a modelação e organização dos dados, baseadas nos testes executados previamente.

Assim, numa primeira fase será apresentado o ambiente de testes em termos de infraestrutura física, o conjunto de dados utilizado, um estudo da cardinalidade e distribuição dos atributos e, ainda, o protocolo de testes que será seguido.

4.1 Infraestrutura Física

Para a realização dos testes necessários a infraestrutura utilizada integra um Cluster Hadoop com 5 nós (1 HDFS *NameNode* e 4 HDFS *Data Nodes*), contendo, cada um:

- Intel i5 quad core com velocidade entre 3.1-3.3 GHz;
- 32 GB de *Random Access Memory* (RAM), com 24GB disponível para processamento de *queries*;
- Samsung 850 EVO 500GB *Solid State Drive* (SSD), com capacidade de ler 540 MB/s e de escrever 520MB/s;
- 1 *gigabit Ethernet card*.

O sistema operativo utilizado em todos os nós é o CentOS 7 com um sistema de ficheiros XFS. A versão do Hadoop utilizada é a *Hortonworks Data Platform* (HDP) 2.6.0, instalada com as configurações de origem, com exceção do fator de replicação do HDFS que foi alterado para 2.

Para além do Hadoop, foi também instalado o Presto, estando o *master* alojado no *NameNode* e os 4 *workers* distribuídos pelos 4 *DataNodes* do cluster. Também o Presto manteve as configurações originais, exceto a de memória, que foi alterada para utilizar 24GB dos 32GB disponíveis.

4.2 Conjunto de Dados

Para implementar os testes necessários foi utilizado o *Star Schema Benchmark* (SSB), um *data mart* tradicional de vendas modelado de acordo com as estruturas multidimensionais (estrela). Este *benchmark* é baseado no TPC-H¹⁷, com algumas alterações relacionadas com a sua transformação num modelo em estrela e que são devidamente explicadas no trabalho de O’Neil, O’Neil, & Chen (2007). No entanto, é importante realçar, que algumas das alterações destes autores não foram implementadas nas tabelas que serão a base para os testes desta dissertação, sendo uma delas a diminuição do fator de escala das tabelas *customer* e *supplier*. Os autores justificaram esta diminuição com o facto de que, em contexto real, não existir um número tão grande de clientes e de fornecedores, tal como definido no TPC-H. No entanto, se se considerar empresas como Facebook e Amazon, uma grande quantidade de clientes, por exemplo, pode ser facilmente justificada, pelo que para esta dissertação decidiu-se manter os fatores de escala estabelecidos no TPC-H, não sendo consideradas nenhuma diminuições no processo de ETL criado para transformar o TPC-H no SSB. Para além disso, a dimensão temporal criada apresenta menos atributos do que os definido por O’Neil et al. (2007), uma vez que se decidiu utilizar apenas os necessários aos testes aqui implementados.

A Figura 16 apresenta o modelo de dados que serve como base para este *benchmark*, sendo que apresenta como tabela de factos a “Lineorder” e como dimensões as restantes 4 tabelas (*Customer*, *Supplier*, *Part* e *Date*).

Recorrendo a este conjunto de dados pré-definido é possível configurar o tamanho da base de dados desejada através da sua extração de acordo com o fator de escala (FE) pretendido. Assim é possível extrair, por exemplo, uma base de dados com fator de escala 10 que gera uma tabela de factos com cerca de 60.000.000 de linhas e as dimensões com os tamanhos correspondentes (calculadas de acordo com as fórmulas da Figura 16).

Para além disso, foram também utilizadas as 13 *queries* disponibilizadas pelos autores do SSB que permitem medir o desempenho do *Data Warehouse* modelado em fluxos de trabalho OLAP típicos. Estas *queries* podem ser consultadas no [Apêndice 1](#).

¹⁷ <http://www.tpc.org/tpch/>

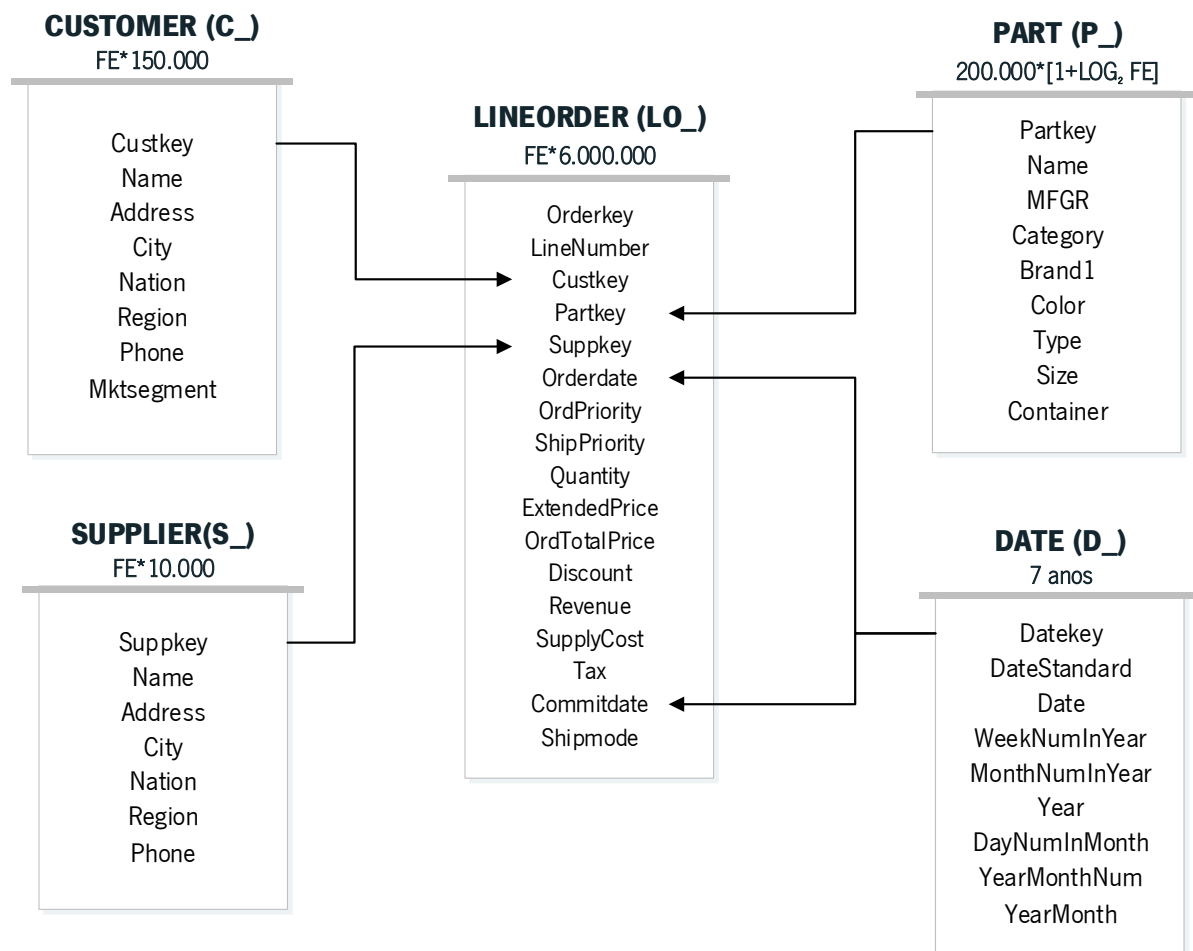


Figura 16 Modelo de dados do SSB. Adaptado de (O'Neil, O'Neil, & Chen, 2007)

4.3 Análise dos Atributos

Considerando o estudo de Santos & Costa (2016) referido anteriormente, uma das regras a considerar na criação de um modelo tabular passa pela definição de partições e *buckets*. Segundo estes autores, a identificação das partições e dos *buckets* deve considerar o balanço entre a cardinalidade dos atributos e a sua distribuição. Por considerar esta regra relevante, para qualquer contexto de teste, foi realizado este estudo sobre o conjunto de dados do SSB.

Importa realçar que, para esta fase de estudo dos atributos, apenas foi considerada uma amostra com FE 1 (que gerou cerca de 6.000.000 linhas da tabela “lineorder”) já que, sendo um *dataset* gerado automaticamente para efeitos de *benchmarks*, em que as amostras retiradas seguem a mesma distribuição, variando apenas o número de registos que é proporcional ao volume de dados, esta amostra foi considerada suficiente.

4.3.1 Cardinalidade

Uma vez que a amostra considerada é relativamente pequena, tendo em conta o contexto do projeto, a cardinalidade dos atributos é facilmente identificada no Tableau¹⁸ com a implementação de filtros por campos distintos. Assim, na Tabela 9 está representada a cardinalidade de cada atributo. É importante realçar que apenas são considerados atributos descritivos (nomes, tipos...), uma vez que os atributos analíticos/numéricos (preços, quantidades...) não permitem este tipo de análise nem são tipicamente usados para particionar ou segmentar os dados.

Tabela 9 Cardinalidade dos Atributos

Entidade	Atributo	Cardinalidade
Lineorder	<i>OrderPriority</i>	5
	<i>ShipPriority</i>	1
	<i>Tax</i>	9
	<i>ShipMode</i>	7
Supplier	<i>Suppkey</i>	10.000
	<i>Name</i>	10.000
	<i>Address</i>	10.000
	<i>City</i>	225
	<i>Nation</i>	25
	<i>Region</i>	5
	<i>Phone</i>	10.000
Customer	<i>Custkey</i>	150.000
	<i>Name</i>	150.000
	<i>Address</i>	150.000
	<i>City</i>	225
	<i>Nation</i>	25
	<i>Region</i>	5
	<i>Phone</i>	150.000
	<i>MKTSegment</i>	5
Part	<i>PartKey</i>	200.000
	<i>Name</i>	190.994
	<i>MFGR</i>	5
	<i>Category</i>	25
	<i>BrandI</i>	975
	<i>Color</i>	17.769
	<i>Type</i>	150
	<i>Size</i>	50
	<i>Container</i>	40

¹⁸ <https://www.tableau.com/>

Entidade	Atributo	Cardinalidade
Date_Dim	<i>DateKey</i>	2.466
	<i>Date</i>	2.466
	<i>Year</i>	7
	<i>YearMonthNum</i>	82
	<i>YearMonth</i>	82
	<i>DayNumInMonth</i>	31
	<i>MonthNumInYear</i>	12
	<i>WeekNumInYear</i>	53

Considerando os valores apresentados, pode-se concluir que atributos como *LO_OrderPriority*, *LO_ShipMode*, *S_Region*, *S_Nation*, *P_MFGR* ou *D_Year* são alguns exemplos de atributos que poderiam ser utilizados para a criação de partições uma vez que não implicariam a criação de um número elevado de pastas (subdiretorias). Já atributos como *Partkey*, *Suppkey*, *DateKey*, *Custkey*, *Orderkey* e *Name* são exemplos de atributos que não devem ser considerados para a criação de partições, mas que poderiam ser considerados para a criação de *buckets*.

4.3.2 Distribuição

Para a criação dos gráficos com a distribuição dos atributos foi, mais uma vez, utilizada a ferramenta Tableau, onde foram criados painéis com vários gráficos, divididos por assunto. De seguida são apresentadas as distribuições dos atributos divididos por entidade, ficando apenas a nota de que alguns deles não são possíveis de representar, nomeadamente as chaves de cada entidade, por apresentarem um valor por linha, e outros atributos analíticos, como o *LO_ExtendedPrice* e o *LO_Quantity*, que não permitem este tipo de representação nem serão considerados nos testes a executar.

Na Figura 17 estão representados os gráficos com a distribuição dos atributos relacionados com a entidade "Part".

Com a análise destes gráficos é possível perceber que os atributos *P_Category*, *P_MFGR*, *P_Type* e *P_Container* apresentam uma distribuição relativamente uniforme. O *P_Brand*, apesar de ter uma distribuição relativamente uniforme, tem um número de valores possível significativamente grande pelo que, caso se definisse como partição, criaria cerca de 900 pastas. Assim, estes atributos com cardinalidade elevada seriam mais apropriados para *bucketing* enquanto que os restantes poderiam ser apropriados para particionamento.

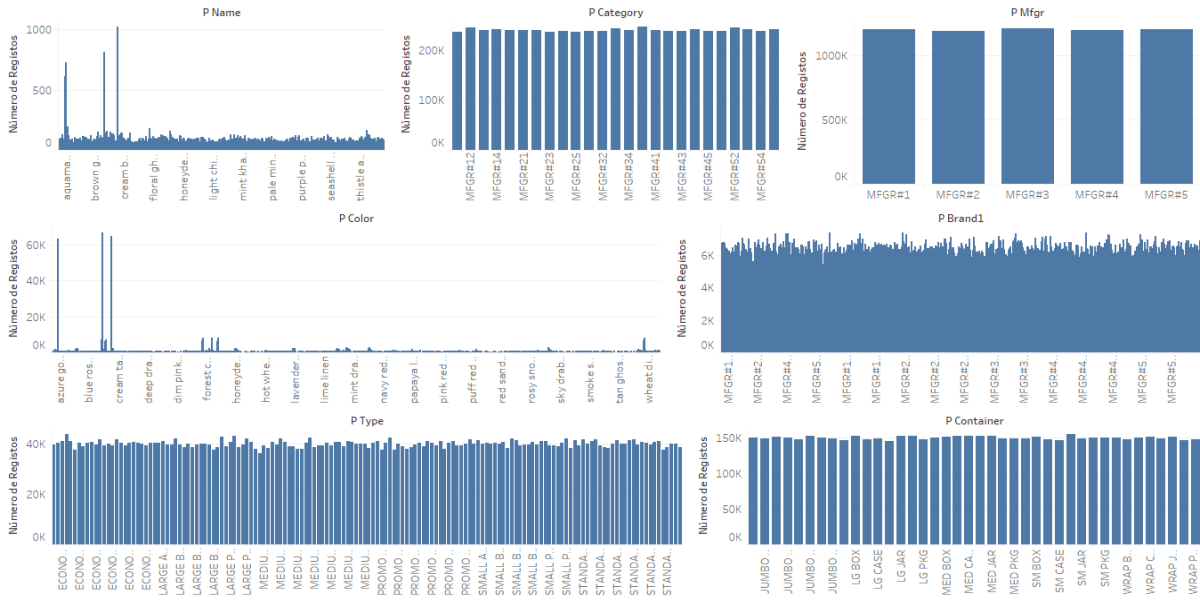


Figura 17 Distribuição dos Atributos da Dimensão "Part"

Na Figura 18 estão representados os gráficos com a distribuição dos atributos relacionados com a entidade "Supplier".

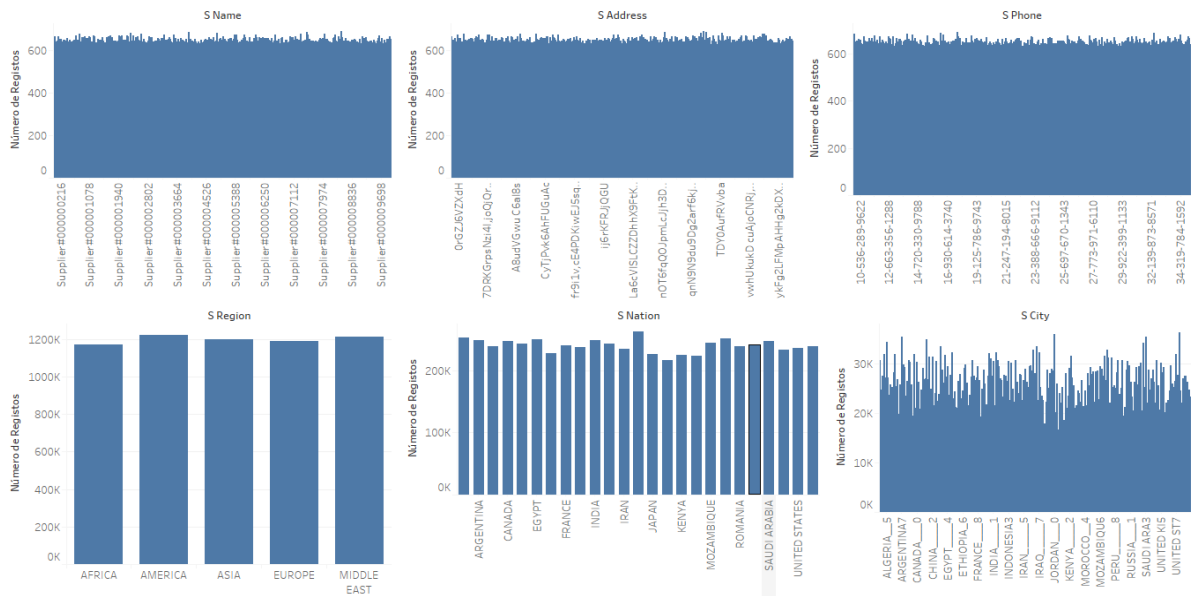


Figura 18 Distribuição dos Atributos da Dimensão "Supplier"

Com a análise destes gráficos é possível identificar uma distribuição relativamente uniforme nos atributos *S_Region* e *S_Nation*. Os restantes, para além de considerarem um número maior de valores possíveis, que em alguns casos dificulta esta análise, também não apresentam tanta consistência na sua distribuição.

Na Figura 19 estão representados os gráficos com a distribuição dos atributos relacionados com a entidade "Customer".

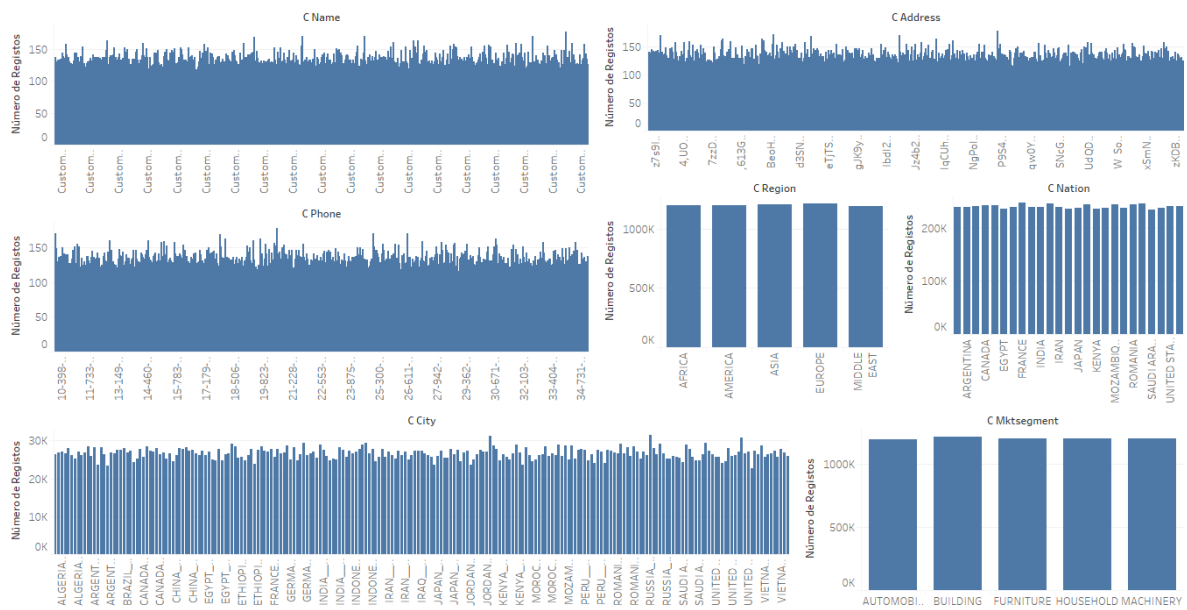


Figura 19 Distribuição dos Atributos das Dimensão "Customer"

Muito semelhante aos gráficos da entidade anterior, mais uma vez se verifica uma distribuição relativamente uniforme nos atributos S_Region e S_Nation , com a diferença que, aqui, também o $C_MKTsegment$ apresenta esta distribuição. Os restantes, mais uma vez, apenas poderiam ser considerados para *bucketing* dada a sua distribuição variável e o número elevado de valores possíveis.

Na Figura 20 estão representados os gráficos com a distribuição dos atributos relacionados com a entidade "Lineorder".

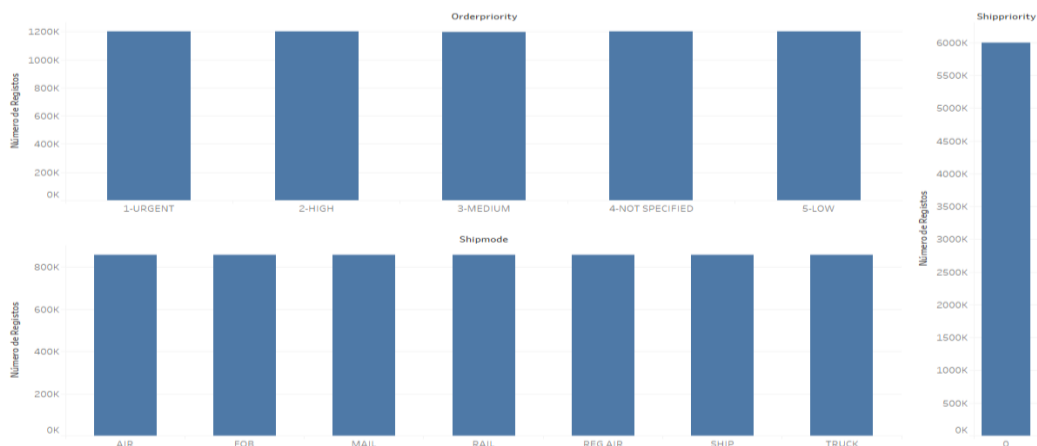


Figura 20 Distribuição dos Atributos de "LineOrder"

Neste caso, todos os atributos possíveis de representar apresentam distribuição claramente uniforme. O atributo $LO_ShipPriority$ apresenta apenas um valor possível pelo que pode ser considerado irrelevante para futuras investigações aqui realizadas.

4.4 Protocolo de Testes

Como mencionado anteriormente, esta dissertação propõe-se a analisar o impacto que diferentes modelos de dados e diferentes formas de organizar os dados têm num *Big Data Warehouse* e, portanto, os testes vão consistir em 2 cenários principais:

1. Cenário A, o cenário relacionado com a modelação dos dados, no qual se irá medir a variação dos tempos de processamento usando um modelo em estrela *versus* um modelo desnormalizado;
2. Cenário B, o cenário relacionado com a organização dos dados, no qual se irá medir a variação dos tempos de processamento com a criação de partições e/ou *buckets*, aplicando aos dois modelos de dados do cenário A, o que divide este cenário de testes em três grupos diferentes de testes:
 - a. Cenário B1, onde será estudado o particionamento simples e múltiplo, nos vários fatores de escala e aplicados aos dois modelos de dados;
 - b. Cenário B2, onde será estudado o *bucketing* simples e múltiplo, nos vários fatores de escala e aplicados aos dois modelos de dados;
 - c. Cenário B3, onde será estudada a combinação do particionamento com *bucketing*, nos vários fatores de escala e aplicado aos dois modelos de dados.

Com estes testes pretende-se perceber qual a estratégia de modelação de dados mais vantajosa a utilizar num *Big Data Warehouse* implementado em Hive, comparando os resultados obtidos nos vários cenários para as tabelas simples, perceber as implicações e o impacto que as partições e os *buckets* têm nos tempos de processamentos para os vários cenários de testes e perceber ainda o comportamento de cada um destes modelos de dados e diferentes organizações de dados com o aumento do volume de dados, aplicando todos os cenários em 3 fatores de escala (30, 100, 300).

O Hive é, então, utilizado como repositório de dados e para a execução de *queries* serão utilizados dois sistemas SQL-on-Hadoop (sistemas que recorrem ao SQL e aplicam-no no Hadoop): o Presto e o Hive (com base no Tez). Estas escolhas foram baseadas num *benchmark* realizado por Santos et al. (2017) que permitiu identificar as vantagens de utilizar o Presto em vez de Spark e do Drill e que demonstrou, ainda, as vantagens da utilização do Hive num contexto de aumento do volume de dados. Para além disso, considerando um dos objetivos desta dissertação de perceber o impacto das diferentes estratégias de organização dos dados, e sendo as partições e os *buckets* características inerentes ao Hive, achou-se relevante perceber como este se comporta quando se aplicam estas estratégias.

Como *interface* foi utilizado o Ambari¹⁹, um *software* que permite a gestão e monitorização de *clusters* Hadoop, uma vez que já tinha sido utilizado e explorado noutros projetos.

A Figura 21 apresenta, de forma sintetizada, os vários cenários de teste a executar e as suas variantes, nomeadamente os cenários A e B.

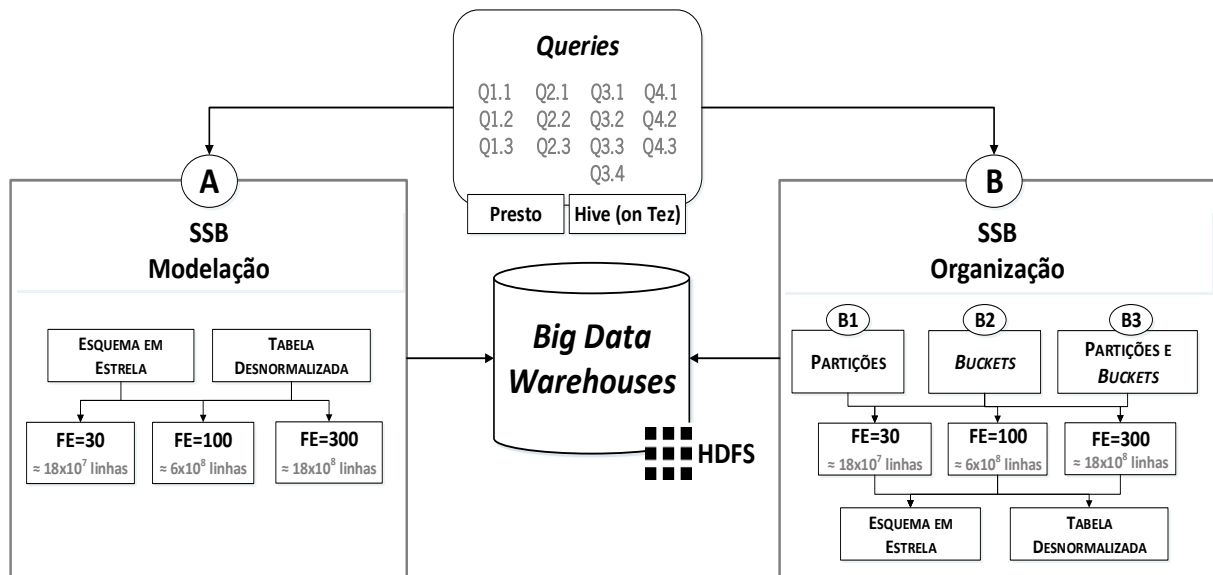


Figura 21 Cenários de Teste

Sistematizando, e tendo em conta o esquema anterior, serão testados dois modelos de dados diferentes (desnormalizado e em estrela) para três fatores de escala diferentes (30, 100 e 300) e aplicando a quatro estratégias de organização diferentes (sem partições ou *buckets*, apenas com partições, apenas com *buckets* e com partições e *buckets*). Para cada fator de escala, os dados do SSB serão armazenados no HDFS, onde o Hive irá aceder para criar as tabelas de acordo com o modelo de dados a implementar. Estando os dados armazenados, serão então executadas as *queries* através do Presto e do Hive (*on Tez*).

De forma a obter resultados mais rigorosos, foram codificadas várias *scripts* que permitem a execução de cada *query* quatro vezes, sendo depois o resultado final apresentado como a média dos quatro resultados obtidos. Estas *scripts* foram adaptadas de acordo com o sistema SQL-on-Hadoop utilizado (Presto ou Hive), com o modelo de dados aplicado (desnormalizado e em estrela), com a estratégia de organização utilizada (com ou sem partições e *buckets*), e podem ser consultadas em https://github.com/EduardaCosta/Scriptst_SSB.git.

¹⁹ <https://ambari.apache.org/>

Todas as tabelas foram criadas no formato ORC, um formato de ficheiros recente do Hive que fornece uma forma mais eficiente de armazenar os dados melhorando o desempenho do mesmo na leitura, escrita e processamento dos dados, e onde são utilizadas técnicas de compressão e de “*predicate push down*” (Dere, 2017).

Para além disso, importa destacar que para o Presto foram utilizadas duas estratégias de *join* diferentes: o *distributed join* (DJ, que o Presto executa por defeito) e o *broadcast join* (BJ). A diferença entre eles está no facto de o primeiro, apesar de conseguir lidar com operações de *join* complexas, é mais lento do que o BJ. Estes últimos são substancialmente mais rápidos, no entanto, apresentam restrições de memória uma vez que o lado direito do *join* (normalmente as dimensões) têm que se ajustar à fração de memória disponível para processamento de *queries* em cada nó. Esta distinção não é feita para o Hive uma vez que este assegura a utilização da melhor estratégia de *join* automaticamente e de acordo com a configuração do cluster.

5. ORGANIZAÇÃO E PROCESSAMENTO DE DADOS EM HIVE

Apresentado o ambiente de testes e os cenários a explorar, nesta secção serão apresentados os resultados obtidos que, depois de devidamente avaliados, serão a base para a sumarização de algumas recomendações para a modelação e organização de um *Big Data Warehouse* concretizado em Hive.




Importa realçar que os testes aqui apresentados deram origem a um trabalho paralelo já publicado, pelo que alguns dos cenários são aqui replicados e, conseqüentemente, algumas conclusões aqui apresentadas são baseadas nesse mesmo trabalho (Costa, Costa, & Santos, 2017).

5.1 Cenário A – Modelação dos Dados

Os *Data Warehouses* tradicionais têm sido um recurso importante da maioria das organizações para o processo de tomada de decisão (Kimball & Ross, 2013). No entanto, estes *Data Warehouses* tradicionais são baseados em modelos relacionais. Apesar dos mais recentes esforços para suportar a normalização nos sistemas Hadoop (Huai et al., 2014), a verdade é que ainda não é possível perceber se a normalização deveria ser implementada em ambientes de *Big Data* ou se a desnormalização é uma estratégia de modelação mais apropriada e com mais vantagens para estes contextos. E é neste sentido que surgem questões ainda não respondidas pela comunidade: será um *Data Warehouse* multidimensional um modelo de *design* adequado para *Data Warehousing* concretizado em Hive? Serão estes esquemas multidimensionais mais eficientes do que as tabelas totalmente desnormalizadas?

À semelhança do trabalho publicado, (Costa et al., 2017), foram realizados testes de comparação entre normalização e desnormalização em contextos de *Big Data Warehousing*. Para tal foram criadas as várias tabelas, para o esquema em estrela (EE) e para a tabela desnormalizada (TD) e considerando os vários fatores de escala, que têm as características apresentadas na Tabela 10.

Tabela 10 Informação sobre as tabelas do Cenário A

Modelo	FE			
EE	30	4 GB	19	280 MB
	100	16 GB	67	248 MB
	300	48 GB	100	496 MB
TD	30	14 GB	50	293 MB
	100	44 GB	200	228 MB
	300	157 GB	505	321 MB

 Espaço Ocupado;  Número de Ficheiros;  Tamanho médio de um ficheiro

Como se pode verificar as tabelas desnormalizadas ocupam um espaço efetivamente maior do que as estrelas (quase 3 vezes maior), o que, à partida poderá ser considerada uma desvantagem da desnormalização para contextos de *Big Data Warehousing*. No entanto, é necessário perceber se este aumento do volume dos dados é acompanhado, ou não, da diminuição do tempo necessário para processar os dados.

Avançando para a comparação dos tempos de processamento, a Tabela 11 apresenta esta comparação para os vários fatores de escala (FE) e para os dois modelos de dados (Esquema em Estrela representado por EE e Tabela Desnormalizada representado por TD), com o Hive e com o Presto (incluindo os dois tipos de *joins*, o DJ e o BJ), apresentando a negrito os melhores resultados por *query*.

Tabela 11 Tempos de Processamento (em segundos) - Esquema em Estrela (EE) vs Tabela Desnormalizada (TD)

	FE=30		FE=100			FE=300			
	HIVE								
	EE	TD	EE	TD	EE	TD			
Q1.1	25	24	31	29	44	51			
Q1.2	24	24	29	29	42	45			
Q1.3	24	23	29	30	43	45			
Q2.1	32	25	47	36	531	79			
Q2.2	31	36	46	73	531	161			
Q2.3	30	25	44	35	531	62			
Q3.1	35	28	59	40	651	98			
Q3.2	30	28	45	41	677	93			
Q3.3	33	25	219	38	665	59			
Q3.4	34	25	222	38	675	72			
Q4.1	38	27	86	41	226	103			
Q4.2	49	29	70	42	141	107			
Q4.3	34	29	54	42	116	114			
TOTAL	420	349	982	516	4874	1090			
DIF		-17%		-47%		-78%			
	PRESTO								
	EE			EE			EE		
	DJ	BJ	TD	DJ	BJ	TD	DJ	BJ	TD
Q1.1	6	5	5	17	13	13	44	36	37
Q1.2	5	5	5	14	13	14	37	34	38
Q1.3	5	4	5	14	13	14	36	35	39
Q2.1	27	8	4	88	19	10	257	59	36
Q2.2	27	7	4	88	18	10	253	51	32
Q2.3	27	7	4	88	17	10	255	49	29
Q3.1	22	8	5	70	29	12	205	81	33
Q3.2	19	6	5	60	17	12	173	51	29
Q3.3	18	5	4	58	15	9	167	43	27
Q3.4	18	6	5	57	15	12	168	43	33
Q4.1	41	13	6	135	43	14	393	119	42
Q4.2	27	9	6	88	26	14	256	69	49
Q4.3	27	8	5	86	23	12	252	63	49
TOTAL	270	92	63	864	262	155	2497	733	472
DIF			-32%			-41%			-36%

Através dos resultados obtidos é possível perceber que os tempos de processamento destas *queries* podem sofrer diminuições entre 4% e 78% com o Hive e, no caso do Presto, diminuições entre os 32% e os 41%. Isto significa que, o Hive, por exemplo, com uma tabela totalmente desnormalizada, com FE=300, consegue terminar o fluxo de trabalho 78% mais rápido do que com um esquema em estrela.

A Figura 22 apresenta, por exemplo, a percentagem de diminuição *query* a *query* no fator de escala 300 quando comparados os resultados da tabela desnormalizada com os do esquema em estrela.

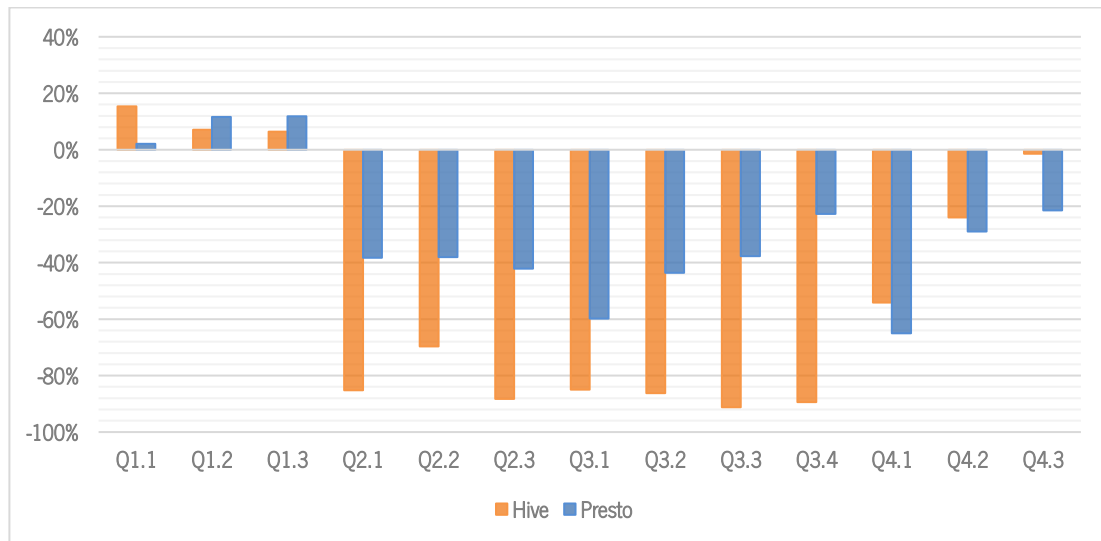


Figura 22 Variação dos Tempos de Processamento para SF=300

Com este gráfico é possível perceber que, com o fator de escala mais elevado, os resultados obtidos com a tabela desnormalizada são claramente melhores do que os obtidos com a estrela. Com a exceção das primeiras 3 *queries*, em que o nível de complexidade é tão baixo pelo que pode compensar fazer um *join* apenas do que correr toda a tabela desnormalizada, todas as outras *queries* sofrem diminuições com a aplicação da tabela desnormalizada. As *queries* 2.1, 2.3, 3.1, 3.2, 3.3 e 3.4 conseguem atingir diminuições superiores a 80% com o Hive.

Considerando novamente os resultados da Tabela 11, e analisando com pormenor os resultados obtidos com o Hive, o modelo em estrela apenas teve melhores resultados do que a desnormalizada em 10 execuções das 104 totais. No entanto, esta vantagem por parte da estrela é, apenas, notória no grupo de *queries* 1 e na query 2.2. Analisando as *queries* do primeiro grupo, verificou-se que estas só fazem o *join* da tabela de factos com uma dimensão (dimensão data) pelo que, em fatores de escala menores, parece compensar fazer apenas este *join* do que percorrer toda a tabela desnormalizada. Outra query que não é afetada positivamente pela desnormalização em algumas das execuções do Hive é a *query* 2.2. Analisando, também, esta *query*, a justificação para estes resultados poderá estar relacionada com

o predicado “*p_brand1 between ‘MFGR#2221’ and ‘MFGR#2228’*” que demonstra que, para responder a esta query, o sistema tem que percorrer um conjunto muito diversificado de *strings*.

Ainda assim, e apesar de não ser o teste que obtém o menor tempo de processamento, é no Hive que se encontra a maior percentagem de diminuição dos tempos de processamento por parte da tabela desnormalizada.

Analisando, agora, os resultados obtidos com o Presto, a vantagem da desnormalização torna-se ainda mais evidente. Com este sistema SQL-on-Hadoop, a estrela apenas superou a tabela desnormalizada em 6 execuções, apesar de que em todas elas se verificam variações pouco significativas. Na verdade, em 42% dos casos a estrela consegue ser 50% mais lenta do que a tabela desnormalizada, e este fenómeno fica ainda mais evidente, mais uma vez, com o aumento do tamanho do *dataset*.

Quando se verifica a diminuição obtida no FE=300, esta parece pequena quando comparada com as diminuições nos restantes fatores de escala, no entanto, e suportando mais uma vez as conclusões de Costa et al. (2017), esta é a primeira vez em que a tabela desnormalizada ultrapassa em tamanho, o espaço do cluster destinado ao processamento de *queries* (aproximadamente 96GB), já que ocupa cerca de 157GB de memória. A estrela, neste fator de escala, ocupa cerca de 50GB e, ainda assim, a tabela desnormalizada consegue ser 36% mais rápida, evidenciando ainda mais as vantagens desta estratégia de modelação para estes contextos.

Para além disso, fica claro que a normalização só pode ser comparada com a desnormalização quando são utilizadas técnicas de otimização, neste caso a utilização de *broadcast joins*, considerando os tempos maiores obtidos com o *distributed join* (assinalados a cinzento na tabela). No entanto, esta técnica tem a particularidade de exigir que o tamanho da tabela de *broadcast* (a tabela da direita do *join*) não ultrapasse a memória disponível para *queries* em cada nó. Apesar da maior parte das dimensões cumprir este requisito, se considerarmos o refrescamento das tabelas, nomeadamente a implementação do tipo 2 das *slowly changing dimensions*²⁰, estas podem aumentar o tamanho de tal forma que este tipo de *join* pode deixar de ser aplicável. Assim, este requisito de memória, que não se verifica em nenhum caso na desnormalização, pode ser considerado mais uma desvantagem da modelação multidimensional em contextos de *Big Data*.

Para além da análise do tempo de processamento necessário, foi ainda verificada a utilização do processador, por parte das diversas *queries*. A Figura 23 apresenta a variação na utilização de CPU por

²⁰ Alterações nas dimensões que implica a criação de um registo adicional na tabela para cada alteração.

parte da estrela em comparação com a tabela desnormalizada, resultados estes obtidos com o Presto para FE=100.

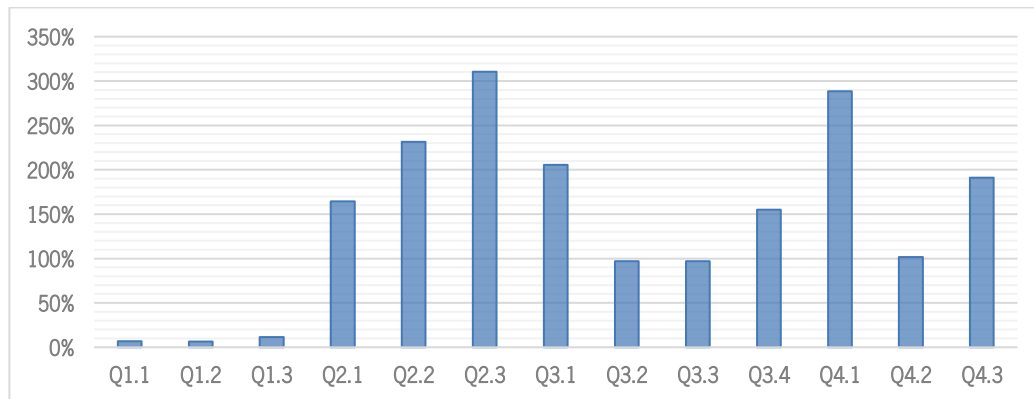


Figura 23 Variação na Utilização de CPU pelo Esquema em Estrela comparada com a Tabela Desnormalizada

Com estes resultados é possível perceber que a utilização adicional de CPU é mais uma desvantagem da normalização, uma vez que todas as *queries* utilizam mais tempo de CPU. Em média, a estrela tem uma utilização de CPU 143% superior e, no pior dos casos, chega mesmo a utilizar quatro vezes mais CPU do que a tabela desnormalizada.

5.2 Cenário B – Organização dos Dados

Apresentados os resultados para o primeiro cenário de teste proposto, segue-se a comparação de diferentes estratégias de organização dos dados em *Big Data Warehouses*. Considerando as funcionalidades de organização disponibilizadas pelo Hive, o sistema de *Data Warehousing* utilizado, esta dissertação propôs-se investigar o impacto do particionamento e do bucketing no processamento de *Big Data Warehouses* concretizados em Hive. Com isto, ao longo desta secção serão apresentadas diferentes estratégias que incluem particionamento, simples e múltiplo, *bucketing*, simples e múltiplo e ainda a combinação do particionamento com *bucketing*, aplicados nos dois tipos de modelação (normalizado e desnormalizado) e com diferentes fatores de escala.

Tal como referido anteriormente, os testes realizados com o modelo normalizado utilizando o Presto utilizaram duas estratégias de *joins*: o *distributed join*, que é o *join* aplicado por defeito, e o *broadcast join*, em que se força a sua utilização. Considerando que todos os resultados obtidos no cenário anterior, seja na forma de organização dos dados, seja no tipo de *join* utilizado, foram sempre melhores com o segundo tipo de *join* implementado, o *broadcast join*, todos os resultados que serão de seguida

apresentados apenas consideram este tipo de *join*. No entanto, também foram realizados testes com o *distributed join*, cujos resultados podem ser consultados no [Apêndice 2](#).

Para além disso, testes que em contextos de pequenas quantidades de dados se demonstrem pouco interessantes considerando os resultados que vão sendo obtidos, não serão replicados para fatores de escala superiores para não sobrecarregar, desnecessariamente, o *cluster* utilizado, com testes que à partida não terão qualquer influência nas conclusões a retirar.





5.2.1 Cenário B1 - Particionamento





Esta secção apresenta os resultados obtidos com a aplicação de partições, testando o particionamento simples (apenas um atributo como partição) e o particionamento múltiplo (mais do que um atributo como partição).

Tendo em conta o estudo da cardinalidade e distribuição dos valores dos atributos que constituem o *dataset* e, ainda, os atributos que mais vezes aparecem na condição “where” das *queries* a executar, foram selecionados 6 atributos que serão utilizados, isoladamente ou combinados, para a criação de partições.

A Tabela 12 apresenta os vários cenários de particionamento que serão testados assim como informações sobre as diversas tabelas criadas (espaço efetivamente ocupado, número de pastas criadas com o particionamento, número de ficheiros por pasta e tamanho médio de um ficheiro de uma pasta) em cada fator de escala e dependendo do modelo de dados.

Tabela 12 Informações sobre as tabelas criadas com particionamento

Partições	Modelo	FE				
Od_Year	EE	30	5 GB	7	4	248 MB
		100	16 GB		12	256 MB
		300	54 GB		34	255 MB
	TD	30	16 GB	7	10	260 MB
		100	57 GB		26	355 MB
		300	149 GB		88	263 MB
S_Region	EE	30	4 GB	5	5	176 MB
		100	16 GB		14	248 MB
		300	49 GB		34	304 MB
	TD	30	14 GB	5	11	260 MB
		100	47 GB		26	390 MB
		300	162 GB		121	
P_MFGR	EE	30	4 GB	5	5	194 MB
		100	-	-	-	-
		300	-	-	-	-
	TD	30	14 GB	5	11	268 MB
		100	-	-	-	-

Partições	Modelo	FE				
		300	-	-	-	-
Od_Year, Od_MonthNumInYear	EE	30	4 GB	7x12=84	5	176 MB
		100	-	-	-	-
		300	-	-	-	-
	TD	30	18 GB	7x12=84	3	80 MB
		100	-	-	-	-
		300	-	-	-	-
Od_Year, S_Region	EE	30	5 GB	7x5=35	3	50 MB
		100	18 GB		3	200 MB
		300	55 GB		7	249 MB
	TD	30	16 GB	7x5=35	4	130 MB
		100	57 GB		8	250 MB
		300	172 GB		22	247 MB
S_Region, S_Nation, S_City	EE	30	4 GB	7x5x5=175	3	7 MB
		100	16 GB		3	25 MB
		300	-		-	-
Od_Year, S_Region, P_MFGR	EE	30	5 GB	7x5=35	3	11 MB
		100	18 GB		3	38 MB
		300	-		-	-

 Espaço Ocupado;  Número de Pastas;  Número de Ficheiros por pasta;  Tamanho médio de um ficheiro por pasta

Tal como no cenário anterior, as tabelas normalizadas com aplicação de particionamento continuam a ser significativamente menores do que as tabelas desnormalizadas também particionadas. As diferenças de tamanho das tabelas do mesmo fator de escala parecem estar relacionadas com as técnicas de compactação (ZLIB) e do algoritmo de compressão usado pelos ficheiros ORC, uma vez que, como terão configuração de ficheiros diferentes, de acordo com o tamanho de cada ficheiro, estes comprimem de uma forma específica, gerando tabelas com tamanho diferentes.

Assim, é de realçar ainda que, a maioria das tabelas com partições ocupam mais memória do que as tabelas sem partições, sendo variações pouco significativas para os contextos em que se criam poucos níveis de particionamento.

Para além disso, algumas das estratégias aplicadas criam ficheiros de dados muito pequenos, como é o caso do particionamento triplo, estratégias estas que, à partida, deverão ser melhor estudadas de forma a não comprometer o desempenho do HDFS.

a) Particionamento Simples

Nesta subsecção serão apresentados os resultados obtidos para as tabelas com particionamento simples, ou seja, considerando apenas um atributo como partição.

Assim, a Tabela 13 apresenta os resultados obtidos para a primeira estratégia de particionamento, utilizando como partição o atributo *Od_Year* (ano de encomenda). Este é o cenário de particionamento utilizado no artigo já publicado (Costa et al., 2017), no entanto, as tabelas particionadas aqui criadas foram atualizadas com outro tipo de mecanismos de otimização, que já tinha sido aplicado nas tabelas simples, nomeadamente a computação das estatísticas da tabela de forma a manter os metadados do Hive atualizados e facilitar o processamento de *queries*. Esta tabela evidencia, a negrito, os melhores resultados e utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição), TD (Tabela Desnormalizada), TD-P (Tabela Desnormalizada com Partições)

Tabela 13 Tempos de Processamento com Particionamento por *Od_Year* (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
Q1.1	25	16	31	22	44	25	5	2	13	4	36	7
Q1.2	24	23	29	31	42	45	5	5	13	13	34	34
Q1.3	24	18	29	22	43	25	4	2	13	4	35	7
Q2.1	32	33	47	56	531	658	8	9	19	24	59	69
Q2.2	31	31	46	53	531	652	7	8	18	21	51	59
Q2.3	30	28	44	53	531	643	7	8	17	20	49	54
Q3.1	35	32	59	52	651	646	8	9	29	28	81	76
Q3.2	30	29	45	48	677	625	6	6	17	18	51	49
Q3.3	33	36	219	77	665	632	5	6	15	15	43	42
Q3.4	34	31	222	243	675	687	6	6	15	17	43	46
Q4.1	38	39	86	99	226	242	13	14	43	47	119	127
Q4.2	49	38	70	63	141	90	9	6	26	16	69	41
Q4.3	34	26	54	47	116	73	8	5	23	13	63	34
TOTAL	420	381	982	868	4874	5042	92	87	262	239	733	645
DIF		-9%		-12%		3%		-5%		-9%		-12%
	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P
Q1.1	24	20	29	20	51	24	5	2	13	3	37	6
Q1.2	24	20	29	22	45	26	5	2	14	4	38	10
Q1.3	23	19	30	21	45	25	5	2	14	3	39	6
Q2.1	25	25	36	40	79	79	4	5	10	10	36	31
Q2.2	36	37	73	67	161	171	4	4	10	9	32	26
Q2.3	25	23	35	32	62	68	4	4	10	7	29	23
Q3.1	28	26	40	39	98	86	5	4	12	10	33	27
Q3.2	28	26	41	39	93	90	5	4	12	10	29	31
Q3.3	25	24	38	31	59	63	4	4	9	8	27	27
Q3.4	25	24	38	38	72	79	5	4	12	9	33	31
Q4.1	27	26	41	43	103	108	6	6	14	13	42	39
Q4.2	29	22	42	27	107	40	6	3	14	5	49	12
Q4.3	29	22	42	28	114	40	5	3	12	5	49	12
TOTAL	349	313	516	449	1090	901	63	46	155	97	472	280
DIF		-10%		-13%		-17%		-26%		-38%		-41%

Com estes resultados é possível concluir que, de forma geral, os tempos de processamento melhoram num contexto de particionamento. Com o Hive, os tempos de processamento podem diminuir até 17% e com o Presto os tempos podem diminuir cerca de 41% quando num contexto de desnormalização e de particionamento por ano de encomenda.

Tal como referido anteriormente, o atributo usado para o particionamento da tabela aparece na condição “where” de várias *queries* executadas, nomeadamente: Q1.1, Q1.3, Q3.1, Q3.2, Q3.3, Q4.2 e Q.4.3 (destacadas a negrito na Tabela 13). Verificando os resultados destas *queries* é possível notar que, efetivamente, estas são as *queries* mais influenciadas pelo particionamento, existindo, na maioria das execuções, uma diminuição no tempo de resposta. Nestas *queries*, em que o atributo utilizado para particionamento mais vezes aparece na condição “where”, a diminuição no tempo de resposta é sempre verificada com exceção das *queries* do grupo 3. Ao analisar este grupo similar de *queries* percebeu-se que estas apresentam uma condição que exclui apenas o ano 1998, o que implica que todas as pastas devem ser percorridas exceto a deste ano. Em contextos de menor quantidade de dados, que são maioritariamente os casos em que estas *queries* nem sempre são influenciadas pelo particionamento, percorrer 6 das 7 tabelas criadas ou procurar em toda a tabela desnormalizada acaba por não fazer grande diferença, executando praticamente no mesmo tempo nos dois contextos.

Contrariamente ao esperado, o padrão de diminuição que se verifica em todos os outros casos, não se confirma com o Hive, no fator de escala 300 num contexto de esquema em estrela. Os tempos de processamento obtidos, em segundos, de cada *query* assim como a percentagem da variação dos tempos de processamento estão apresentados na Figura 24.

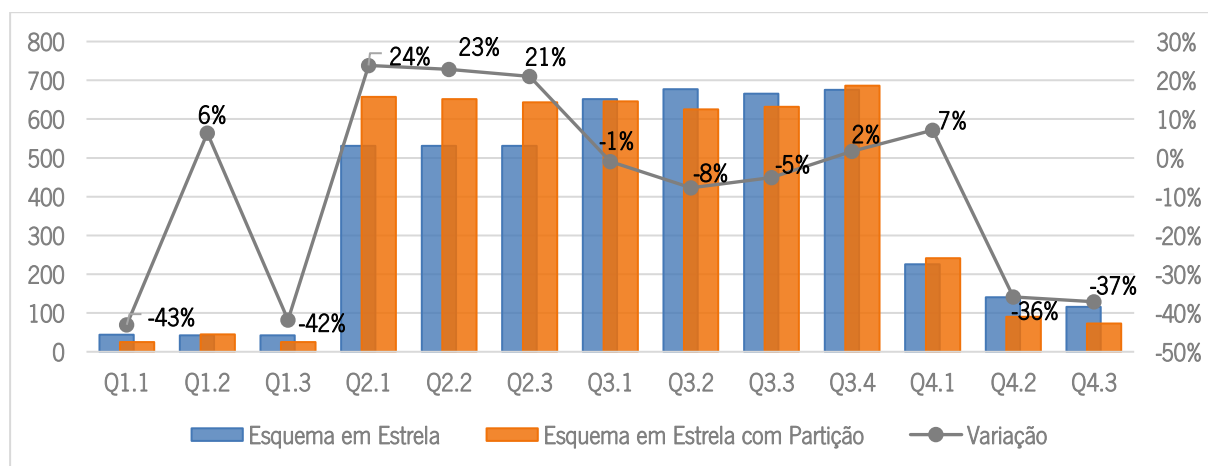


Figura 24 Variação dos Tempos de Processamento com Particionamento (Hive, FE=300, Esquema em Estrela)

Analisando *query* a *query*, é possível verificar que todas as que têm o filtro por ano são positivamente influenciadas pelo particionamento, uma vez que todas elas sofrem uma diminuição. No

entanto, as *queries* do grupo 2, por exemplo, dada a sua complexidade e a ausência de qualquer reconhecimento do filtro, são afetadas negativamente, sofrendo aumentos até 24% nos tempos de processamento. Assim, apesar das vantagens do particionamento serem evidentes para estas *queries*, as vantagens conseguidas não superam os aumentos sofridos pelas *queries* que não apresentam o filtro por ano, existindo um ligeiro aumento de 3% no valor total do cenário.

Ainda assim, já se começa a perceber as vantagens associadas ao particionamento uma vez que este torna mais rápida a execução de algumas *queries*. Como tal, apesar de depender sempre dos dados a serem trabalhados e dos filtros de pesquisa que se vão aplicando, o particionamento temporal pode ser uma boa estratégia de organização dos dados.

Considerando agora, o segundo atributo aqui referido para particionamento simples, o atributo *S_Region*, os resultados obtidos estão representados na Tabela 14, que evidencia, a negrito, os melhores tempos de processamento e que utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição), TD (Tabela Desnormalizada), TD-P (Tabela Desnormalizada com Partições).

Este cenário é um dos que apresenta melhores resultados. Como é possível verificar pelos resultados obtidos, a vantagem do particionamento está bastante evidenciada. Com o Hive é possível obter tempos de processamento até 32% mais rápidos e com o Presto é possível diminuir quase 50% do tempo de processamento. Para além disso, quanto maior o volume de dados mais evidenciada fica a vantagem do particionamento.

Tabela 14 Particionamento por *S_Region* (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
Q1.1	25	24	31	31	44	55	5	5	13	13	36	33
Q1.2	24	25	29	29	42	52	5	5	13	13	34	32
Q1.3	24	23	29	30	43	52	4	5	13	12	35	32
Q2.1	32	24	47	40	531	136	8	3	19	7	59	18
Q2.2	31	23	46	39	531	135	7	3	18	6	51	14
Q2.3	30	24	44	35	531	130	7	3	17	6	49	13
Q3.1	35	26	59	35	651	-	8	4	29	10	81	27
Q3.2	30	29	45	49	677	633	6	6	17	17	51	51
Q3.3	33	34	219	245	665	634	5	6	15	16	43	42
Q3.4	34	34	222	248	675	639	6	6	15	16	43	42
Q4.1	38	31	86	63	226	205	13	6	43	15	119	36
Q4.2	49	33	70	57	141	85	9	5	26	12	69	26
Q4.3	34	35	54	55	116	124	8	8	23	23	63	61
TOTAL	420	364	982	955	4874	2880	92	66	262	166	733	428
DIF		-13%		-3%		-32%		-28%		-37%		-42%

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P
Q1.1	24	24	29	38	51	83	5	5	13	11	37	38
Q1.2	24	25	29	37	45	82	5	5	14	12	38	39
Q1.3	23	25	30	36	45	83	5	5	14	12	39	40
Q2.1	25	21	36	23	79	30	4	2	10	3	36	8
Q2.2	36	24	73	31	161	48	4	2	10	2	32	7
Q2.3	25	21	35	22	62	26	4	2	10	2	29	6
Q3.1	28	22	40	24	98	32	5	2	12	4	33	9
Q3.2	28	24	41	35	93	73	5	3	12	6	29	15
Q3.3	25	24	38	34	59	77	4	3	9	6	27	18
Q3.4	25	25	38	35	72	80	5	4	12	7	33	24
Q4.1	27	21	41	24	103	32	6	2	14	4	42	11
Q4.2	29	21	42	24	107	35	6	2	14	4	49	11
Q4.3	29	24	42	35	114	81	5	3	12	7	49	22
TOTAL	349	300	516	397	1090	762	63	39	155	79	472	247
DIF		-14%		-23%		-30%		-38%		-49%		-48%

Importa realçar que, por erros de memória e como consequência das configurações do cluster utilizado, o Hive não conseguiu executar a *query* 3.1, no FE=300 e esquema em estrela, pelo que a diferença no tempo de processamento calculada não inclui esta *query* (tanto da tabela sem partições como da tabela particionada). Ainda assim, a diminuição conseguida com esta configuração neste fator de escala atinge os 32%.

Este atributo foi selecionado por aparecer nas condições “where” de várias *queries* executadas, nomeadamente as *queries* 2.1, 2.2, 2.3, 3.1, 4.1 e 4.2 (destacadas a negrito na Tabela 14). Analisando os resultados obtidos para estas *queries* é possível concluir que, na grande maioria das execuções, estas são, efetivamente, as *queries* mais influenciadas pelo particionamento e as grandes responsáveis pelas diminuições nos valores gerais dos cenários estudados. A Figura 25 apresenta um exemplo da variação nos tempos de processamento, *query* a *query*, obtida com o Presto para o esquema em estrela no FE=300.

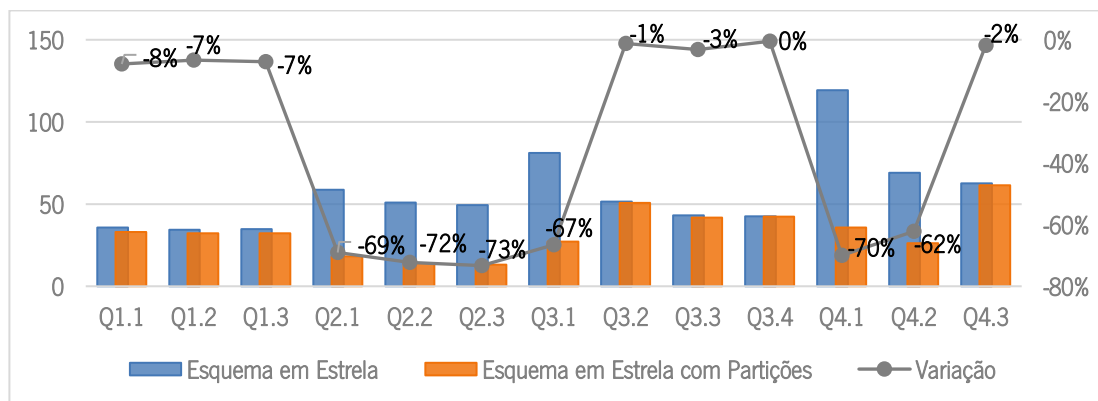


Figura 25 Variação nos Tempos de Processamento com Particionamento por S_Regio (EE, FE=300)

Como é possível verificar nesta figura, e em todos os resultados obtidos pelo Presto, há vantagens claras na distribuição dos dados em pastas em todos os contextos, confirmando que este é um atributo adequado para o particionamento deste conjunto de dados já que se mostra alinhado com as *queries* que são executadas. Assim, não só as *queries* que apresentam o filtro sofrem diminuição, como todas as outras são influenciadas positivamente por esta estratégia, apesar de sofrerem diminuições menos significativas.

Ainda assim, há um outro fenômeno que é importante explicar: a diminuição nos tempos de resposta em *queries* que não apresentam o atributo *S_Region* como filtro, que neste caso são todas as restantes *queries* que vão, em algumas execuções, obtendo melhores resultados com o particionamento mesmo não tendo o filtro. Analisando, por exemplo, a *query* 3.4, a Figura 26 apresenta destacados os atributos incluídos na condição “where”.

```
select c_city, s_city, od_year, sum(revenue) as revenue
from analytical_obj$2 part2
where (c_city='UNITED KI1' or c_city='UNITED KI5')
      and (s_city='UNITED KI1' or s_city='UNITED KI5')
      and od_yearmonth = 'Dec1997'
group by c_city, s_city, od_year
order by od_year asc, revenue desc;
```

Figura 26 Query 3.4 (para o modelo desnormalizado)

Apesar de não ter o filtro por *S_Region* apresenta na condição “where” o atributo *S_City*, um atributo relacionado hierarquicamente com o atributo da partição, sendo benéfico para as ferramentas uma vez que para responder à *query* os dados, para aquele determinado filtro, vão estar concentrados em determinada partição. Este fenômeno acontece tanto no Hive como no Presto e é conhecido como *predicate pushdown* ao nível da faixa ORC, que não é mais do que uma técnica filtragem de dados com base na leitura dos cabeçalhos e estatísticas dos blocos ORC criados para a tabela, em que se verifica primeiro se o ficheiro ORC poderá conter alguma linha que corresponda ao predicado da *query* e se o terá que ler (todo) ou se o ignora e avança para outro bloco (Hortonworks, 2017).

Assim, estando a tabela filtrada por *S_Region*, os resultados para esta *query* vão estar concentrados numa única partição que contém aquelas duas cidades (“United KI1” e “United KI5”), pelo que o número de blocos que terá que ler será menor, irá ler mais linhas por segundo e, conseqüentemente, o tempo de resposta irá diminuir. Já com a tabela simples (sem partições), a distribuição pelos blocos ORC vai ser muito variável, pelo que poderá ter que ler mais blocos para encontrar todas as linhas necessárias. Lendo mais blocos irá, conseqüentemente, ler mais linhas, uma

vez que, mesmo que só tenha uma linha naquele bloco que coincida com o predicado, este considera e percorre todas as linhas desse bloco.

Com tudo isto, este cenário vem evidenciar ainda mais as vantagens do particionamento uma vez que, para as *queries* aqui executadas, as vantagens não se verificam apenas para *queries* com o filtro por *S_Region* mas, também, para outras *queries* que apresentem condições com atributos facilmente reconhecidos, pelas ferramentas, como estando associados às partições criadas. Como tal, apesar de, mais uma vez, depender sempre dos dados a ser trabalhados e dos filtros de pesquisa que se vão aplicando, o particionamento espacial dos dados pode ser uma boa estratégia de organização.

Um terceiro teste do particionamento simples implica a adoção de um atributo pouco utilizado nas condições “where” das *queries* executadas, o *P_MFGR*. A Tabela 15 apresenta os resultados obtidos com este particionamento, apenas para o fator de escala 30, utilizando a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição), TD (Tabela Desnormalizada), TD-P (Tabela Desnormalizada com Partições).

Tabela 15 Tempos de Processamento com Particionamento por *P_MFGR* (em segundos)

	FE=30					FE=30			
	HIVE		PRESTO			HIVE		PRESTO	
	EE	EE-P	EE	EE-P		TD	TD-P	TD	TD-P
Q1.1	25	23	5	6	Q1.1	24	24	5	5
Q1.2	24	23	5	5	Q1.2	24	25	5	5
Q1.3	24	24	4	5	Q1.3	23	25	5	5
Q2.1	32	30	8	8	Q2.1	25	24	4	2
Q2.2	31	31	7	7	Q2.2	36	28	4	2
Q2.3	30	29	7	7	Q2.3	25	23	4	2
Q3.1	35	34	8	9	Q3.1	28	27	5	5
Q3.2	30	29	6	7	Q3.2	28	27	5	5
Q3.3	33	33	5	6	Q3.3	25	25	4	5
Q3.4	34	32	6	6	Q3.4	25	26	5	5
Q4.1	38	32	13	8	Q4.1	27	22	6	3
Q4.2	49	40	9	7	Q4.2	29	22	6	3
Q4.3	34	32	8	9	Q4.3	29	24	5	2
TOTAL	420	392	92	90	TOTAL	349	322	63	49
DIF		-7%		-2%	DIF		-8%		-22%

Contrariamente às expectativas, considerando a ausência do atributo nos filtros da maior parte das *queries*, este cenário mostrou-se vantajoso. No entanto, considerando o fenómeno do *predicate push down*, esta parece ser, mais uma vez, a razão para esta diminuição dos tempos de processamento. As únicas *queries* que apresentam diretamente este filtro são as *queries* 4.1 e 4.2 (destacadas a negrito na Tabela 15), e, tal como o esperado, são as *queries* que apresentam menores tempos de processamento

em todos os cenários, num contexto de particionamento. No entanto, as *queries* 2.1, 2.2, 2.3 e 4.3 apresentam predicados relacionados com o produto e, como tal, apresentam, também, diminuições com o particionamento em algumas das execuções, consequência da técnica de leitura dos cabeçalhos dos blocos ORC.

Apesar dos resultados obtidos demonstrarem vantagens neste cenário, este não foi replicado para fatores de escala maiores uma vez que acaba por não acrescentar nenhuma conclusão às que já foram retiradas nos dois cenários anteriores.

Antes de passar para a apresentação de resultados associados ao particionamento múltiplo, importa referir, de forma breve, qual é o impacto do particionamento simples na utilização do processador central.

Tal como aconteceu no cenário A, a par da análise da variação dos tempos de processamento das 13 *queries*, foi ainda estudada a utilização de CPU no processamento em ambos os modelos de dados, agora com a aplicação do particionamento simples. A Figura 27 apresenta a variação na utilização de CPU com a tabela particionada por *Od_Year*, em relação à utilização de CPU num contexto em que não existe qualquer partição (cenário A), resultados estes obtidos pelo Presto para FE=100.

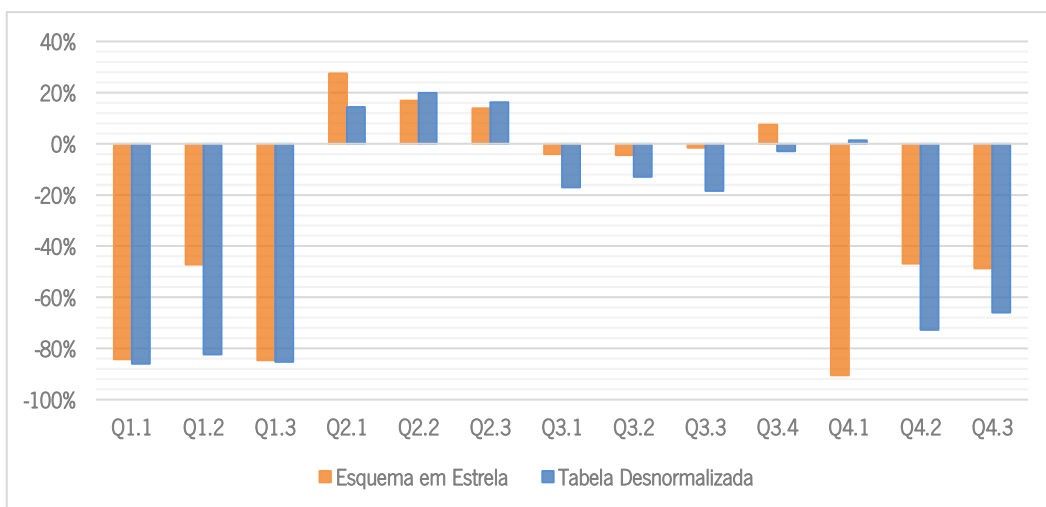


Figura 27 Variação na Utilização de CPU com Particionamento Simples (FE=100)

Relativamente aos resultados apresentados é possível perceber que as *queries* que apresentam o atributo como filtro (Q1.1, Q1.3, Q3.1, Q3.2, Q3.3, Q4.2 e Q4.3) são aquelas que sofrem as diminuições mais significativas na utilização de CPU. Destaca-se ainda a diminuição na Q1.2 e Q4.1, que apesar de não apresentarem o filtro por ano contêm outros filtros relacionados com este, e tal como aconteceu com os tempos de processamento, que diminuíram neste fator de escala, também aqui se verifica uma vantagem nesta query. Com isto, em média, há uma diminuição de cerca 30% em ambos os modelos

com a aplicação da estratégia de particionamento simples. Assim, para além da diminuição dos tempos de processamento conseguidos, com este tipo de estratégia, também se consegue melhorar o tempo de utilização de CPU.

b) Particionamento Múltiplo

Nesta subsecção serão apresentados os resultados obtidos para as tabelas com particionamento múltiplo, considerando combinações de dois e três atributos.

A Tabela 16 apresenta os tempos de processamento obtidos para o FE de 30 com o particionamento pelos atributos *Od_Year* e *Od_MonthNumInYear*, organizados em hierarquia do mais agregado para o mais detalhado. Esta tabela evidencia a negrito os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição), TD (Tabela Desnormalizada), TD-P (Tabela Desnormalizada com Partições).

Tabela 16 Tempos de Processamento com Particionamento por *Od_Year* e *Od_MonthNumInYear* (em segundos)

	FE=30					FE=30			
	HIVE		PRESTO			HIVE		PRESTO	
	EE	EE-P	EE	EE-P		TD	TD-P	TD	TD-P
Q1.1	25	20	5	3	Q1.1	24	22	5	8
Q1.2	24	26	5	13	Q1.2	24	25	5	3
Q1.3	24	21	4	3	Q1.3	23	19	5	3
Q2.1	32	35	8	15	Q2.1	25	29	4	42
Q2.2	31	36	7	14	Q2.2	36	42	4	40
Q2.3	30	34	7	14	Q2.3	25	27	4	38
Q3.1	35	34	8	13	Q3.1	28	29	5	37
Q3.2	30	31	6	13	Q3.2	28	29	5	39
Q3.3	33	39	5	12	Q3.3	25	28	4	36
Q3.4	34	36	6	13	Q3.4	25	27	5	22
Q4.1	38	40	13	17	Q4.1	27	32	6	40
Q4.2	49	40	9	7	Q4.2	29	23	6	11
Q4.3	34	33	8	6	Q4.3	29	23	5	11
TOTAL	420	428	92	143	TOTAL	349	354	63	330
DIF		2%		55%	DIF		2%		428%

Neste cenário foi criada uma hierarquia ano-mês, no entanto, como o segundo atributo não é utilizado por nenhuma das *queries* executadas e como implica a criação de 12 subpastas para cada pasta do ano, os resultados demonstram claramente o esforço extra no seu processamento.

Ainda que algumas *queries* sofram diminuição, importa realçar que estas são consequência do reconhecimento da primeira partição, uma vez que são as *queries* que utilizam este atributo como filtro (destacadas a negrito na Tabela 16) que são, de alguma forma, influenciadas.

As *queries* do grupo 3, apesar de apresentarem o filtro por ano, como implicam a procura em 6 das 7 pastas de ano criadas, adicionando as 12 subpastas do mês denota-se um esforço extra de processamento e, por isso, são *queries* que na maior parte das vezes não são afetadas de forma positiva por este tipo de particionamento (muito pelo contrário em alguns casos).

Assim, este é, para já, um primeiro exemplo de uma má estratégia de organização, que, considerando os resultados obtidos para o FE=30, não será replicada para fatores de escala maiores.

Voltando ao contexto das diminuições dos tempos de processamento ocorrerem em *queries* que utilizem os atributos utilizados para particionar como filtros, criaram-se tabelas particionadas por *Od_Year* e *S_Region*, os dois atributos que mais vezes aparecem nas condições “*where*” organizados por ordem crescente da sua utilização nos filtros das *queries*, estando os resultados obtidos apresentados na Tabela 17. Esta tabela evidencia a negrito os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição), TD (Tabela Desnormalizada), TD-P (Tabela Desnormalizada com Partições)

Tabela 17 Tempos de Processamento com Particionamento por *Od_Year* e *S_Region* (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
Q1.1	25	21	31	22	44	25	5	2	13	4	36	8
Q1.2	24	27	29	33	42	54	5	7	13	18	34	48
Q1.3	24	21	29	22	43	26	4	2	13	4	35	8
Q2.1	32	30	47	45	531	153	8	4	19	8	59	23
Q2.2	31	28	46	39	531	152	7	4	18	6	51	17
Q2.3	30	27	44	41	531	147	7	3	17	6	49	15
Q3.1	35	26	59	34	651	162	8	4	29	9	81	27
Q3.2	30	30	45	52	677	570	6	7	17	19	51	52
Q3.3	33	37	219	75	665	578	5	7	15	16	43	48
Q3.4	34	36	222	223	675	618	6	8	15	20	43	56
Q4.1	38	33	86	70	226	205	13	6	43	15	119	40
Q4.2	49	30	70	58	141	91	9	4	26	9	69	20
Q4.3	34	29	54	44	116	70	8	5	23	14	63	36
TOTAL	420	375	982	760	4874	2849	92	63	262	149	733	399
DIF		-11%		-23%		-42%		-32%		-43%		-46%

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P
Q1.1	24	20	29	21	51	29	5	2	13	3	37	8
Q1.2	24	26	29	36	45	80	5	2	14	5	38	16
Q1.3	23	21	30	21	45	30	5	2	14	3	39	8
Q2.1	25	21	36	23	79	30	4	3	10	5	36	16
Q2.2	36	24	73	32	161	50	4	3	10	6	32	16
Q2.3	25	21	35	22	62	29	4	3	10	5	29	17
Q3.1	28	21	40	23	98	31	5	2	12	3	33	11
Q3.2	28	25	41	29	93	60	5	4	12	5	29	32
Q3.3	25	25	38	29	59	62	4	5	9	6	27	44
Q3.4	25	28	38	40	72	108	5	7	12	13	33	81
Q4.1	27	22	41	24	103	34	6	3	14	6	42	20
Q4.2	29	17	42	21	107	25	6	2	14	4	49	12
Q4.3	29	21	42	24	114	34	5	4	12	7	49	19
TOTAL	349	292	516	346	1090	602	63	43	155	71	472	299
DIF		-16%		-33%		-45%		-32%		-54%		-37%

Comparando com os cenários apresentados anteriormente, este é aquele em que estão mais evidenciadas as vantagens do particionamento. Esta estratégia de organização inclui a utilização de dois atributos para particionamento, atributos esses que aparecem mais vezes nos filtros das *queries*, isolados ou combinados. As únicas *queries* que não reconhecem, diretamente, um destes filtros são as *queries* 1.2 e 3.4 mas, ainda assim, apresentam filtros relacionados como *YearMonth* e *S_City*, pelo que em algumas execuções podem sofrer diminuições relacionadas com a técnica de leitura dos cabeçalhos ORC. As *queries* 3.1 e 4.2 apresentam os dois atributos nas suas condições “where” e são, efetivamente, as *queries* em que se verificam maiores diminuições, por vezes, superiores a 50%.

Assim, quando num contexto de esquema em estrela, as diminuições nos tempos de processamento atingem os 42% no Hive e os 46% no Presto. Num contexto desnormalização, as diminuições no Hive variam entre os 16% e os 45% e com o Presto, atingem diminuições superiores a 50%, atingindo no melhor cenário os 54%.

Mais uma vez se verifica que o grupo de *queries* 3 nem sempre realça as vantagens do particionamento, estando mais uma vez relacionado com o filtro que exclui apenas um ano e que implica uma pesquisa em 6 das 7 pastas de *Od_Year* criadas, sendo que, em cada uma dessas pastas, têm que percorrer mais 5 pastas de *S_Region*, quando não são filtradas por este atributo (*queries* 3.2 e 3.3).

Ainda assim, no geral, como há *queries* que sofrem diminuição nos tempos de resposta iguais ou superiores a 50%, em 15% dos casos, estas compensam todas aquelas que são afetadas negativamente

por este tipo de particionamento. Ficam, assim, evidenciadas as suas vantagens em todos os fatores de escala e nos dois sistemas. Para além disso, as que sofrem aumentos nos tempos de processamento apresentam variações pouco significativas já que estes aumentos nunca ultrapassam os 14 segundos, pelo que, dependendo do contexto analítico poderá não ser relevante.

A combinação dos atributos *S_Region*, *S_Nation* e *S_City* surgiu com a ideia de, não só testar o particionamento triplo como criar uma hierarquia espacial de pastas, em que todos os atributos aparecem pelo menos uma vez na condição “*where*” de alguma *query*.

A criação desta combinação é complexa e, verificando a organização dos dados, é possível perceber que com a combinação de 5 regiões com 5 países e 9 cidades, os dados foram distribuídos por 225 pastas pelo que, por exemplo, com cerca de 30GB de dados, criaram-se ficheiros demasiado pequenos no HDFS (cada partição fica com três ficheiros pequenos, num total de, aproximadamente, 22MB por partição). Tal como referido em secções anteriores, este contexto não é aconselhável para um bom desempenho do HDFS.

Ainda assim, a Tabela 18 apresenta os resultados obtidos para os fatores de escala e modelos de dados em que foi possível criar esta configuração, consoante as capacidades do *cluster* que está a ser utilizado. Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição).

Tabela 18 Tempos de Processamento com Particionamento por *S_Region*, *S_Nation* e *S_City* (em segundos)

	FE=30		FE=100		FE=30		FE=100	
	HIVE				PRESTO			
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
Q1.1	25	32	31	41	5	11	13	36
Q1.2	24	31	29	44	5	11	13	37
Q1.3	24	30	29	43	4	11	13	35
Q2.1	32	30	47	46	8	4	19	10
Q2.2	31	29	46	43	7	4	18	10
Q2.3	30	30	44	45	7	4	17	9
Q3.1	35	28	59	38	8	4	29	11
Q3.2	30	24	45	28	6	2	17	4
Q3.3	33	27	219	35	5	2	15	3
Q3.4	34	26	222	33	6	2	15	3
Q4.1	38	37	86	79	13	6	43	16
Q4.2	49	42	70	63	9	5	26	14
Q4.3	34	30	54	45	8	3	23	5
TOTAL	420	396	982	582	92	70	262	193
DIF		-6%		-41%		-24%		-26%

Como é possível verificar, este cenário não foi replicado para as tabelas desnormalizadas devido às configurações de memória e de replicação do cluster em utilização, que eram ultrapassadas, tal como se pode verificar no erro da Figura 28. Para além disso, e pelos mesmos motivos, o modelo normalizado não foi replicado para o maior fator de escala de forma a não sobrecarregar o *cluster*, dada a complexidade deste cenário.

```

-----
      VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----
Map 1                RUNNING    100         0         0         100     40     0
-----
VERTICES: 00/01  [>>-----] 0%    ELAPSED TIME: 112.71 s
-----
ERROR : Status: Failed
ERROR : Vertex failed, vertexName=Map 1, vertexId=vertex_1506428884733_0105_3_00, diagnostics=[Task failed,
kAttempt 0 failed, info=[Error: Failure while running task:java.lang.RuntimeException: java.lang.RuntimeExce
e Runtime Error while processing row

```

Figura 28 Erro na criação da tabela desnormalizada com particionamento triplo

Apesar de implicar uma grande complexidade na sua criação, problemas de memória e a criação de poucos ficheiros, demasiado pequenos, como é um cenário em que os atributos que serviram como partições aparecem muitas vezes nos filtros, abrangendo desde a *query* 2.1 à *query* 4.3 (destacadas a negrito na Tabela 18), há uma diminuição evidente em todos os cenários executados.

A diminuição nos tempos de processamento é ainda mais evidente nas *queries* 3.3 e 3.4 uma vez que apresentam o filtro por cidade (o último nível de particionamento). Aliando a técnica de *predicate pushdown* nos dois níveis de particionamento anteriores com um filtro tão específico, no último nível, faz com que estas *queries* demorem menos 85% do tempo, no melhor dos casos.

Ainda que as vantagens sejam bastante claras, neste cenário fica a nota de que é preciso ter cuidado com este tipo de organização, para evitar particionamento excessivo acompanhado de perdas de desempenho por parte do HDFS. Assim, a criação de mais do que dois níveis de particionamento tem que ser cuidadosamente avaliada de forma a incluir um estudo do volume de dados total, da quantidade de dados armazenada em cada pasta final, assim como um estudo do tipo de refrescamento que será realizado, de forma a não dificultar este processo de atualização ou introdução de novos registos nas tabelas.

No seguimento do cenário apresentado anteriormente, decidiu-se criar uma outra tabela com particionamento triplo, desta vez combinando os três atributos que foram testados isoladamente e que são os atributos que aparecem mais vezes nas condições “where” das *queries*, abrangendo a maior parte delas: *Od_Year*, *S_Region* e *P_MFGR*. Os resultados obtidos para este cenário estão representados

na Tabela 19, que evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-P (Esquema em Estrela com Partição).

Tabela 19 Tempos de Processamento com Particionamento por *Od_Year*, *S_Region* e *P_MFGR* (em segundos)

	FE=30		FE=100		FE=30		FE=100	
	HIVE				PRESTO			
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
Q1.1	25	21	31	33	5	3	13	8
Q1.2	24	27	29	104	5	12	13	40
Q1.3	24	21	29	32	4	3	13	7
Q2.1	32	29	47	57	8	5	19	12
Q2.2	31	30	46	59	7	4	18	10
Q2.3	30	29	44	55	7	4	17	12
Q3.1	35	27	59	48	8	5	29	12
Q3.2	30	31	45	87	6	13	17	38
Q3.3	33	39	219	117	5	12	15	39
Q3.4	34	35	222	275	6	13	15	42
Q4.1	38	35	86	65	13	5	43	13
Q4.2	49	25	70	37	9	5	26	12
Q4.3	34	33	54	76	8	6	23	15
TOTAL	420	380	982	1045	92	91	262	259
DIF		-9%		6%		-1%		-1%

Tal como se verificou no contexto anterior, e tendo em conta também o erro da Figura 28, este cenário não foi replicado para as tabelas desnormalizadas nem para o fator de escala maior devido a problemas de memória.

Apesar de se verificarem diminuições nos tempos de processamento em 3 dos 4 contextos analisados, a verdade é que são diminuições, quando existem, demasiado pequenas e pouco significativas.

Ainda assim, *queries* que incluam na condição “where” combinações de dois dos 3 atributos utilizados, como é o caso das *query*3.1 que apresenta a combinação *Od_Year* + *S_Region* e a *query*4.1 que apresenta a combinação *S_Region* + *P_MFGR*, sofrem decréscimos consideráveis. Já a *query*4.1 apresenta todos os atributos das partições nos seus filtros, verificando-se, por isso, diminuições consideráveis em todos os casos.

No entanto, estes decréscimos não compensam os aumentos causados nas restantes *queries* pelo que acaba por não ter um impacto positivo significativo nos tempos gerais de processamento. Este cenário vem assim comprovar, mais uma vez, que o particionamento excessivo pode trazer complexidade acrescida tanto no carregamento dos dados como na criação das *queries* a executar (uma vez que tem que ter em conta a incorporação de combinações de atributos nos filtros com potencial para diminuir os tempos de processamento).

Para a utilização do processador central, foi estudada a utilização de CPU no processamento em ambos os modelos de dados, agora com a aplicação do particionamento múltiplo, utilizando como exemplo o particionamento por *Od_Year* + *S_Region*. A Figura 29 apresenta a variação na utilização de CPU por parte das tabelas particionadas por mais do que um atributo em relação às tabelas sem qualquer tipo de organização apresentadas no cenário A, quando se utiliza o Presto como ferramenta de *querying* e para FE=100.

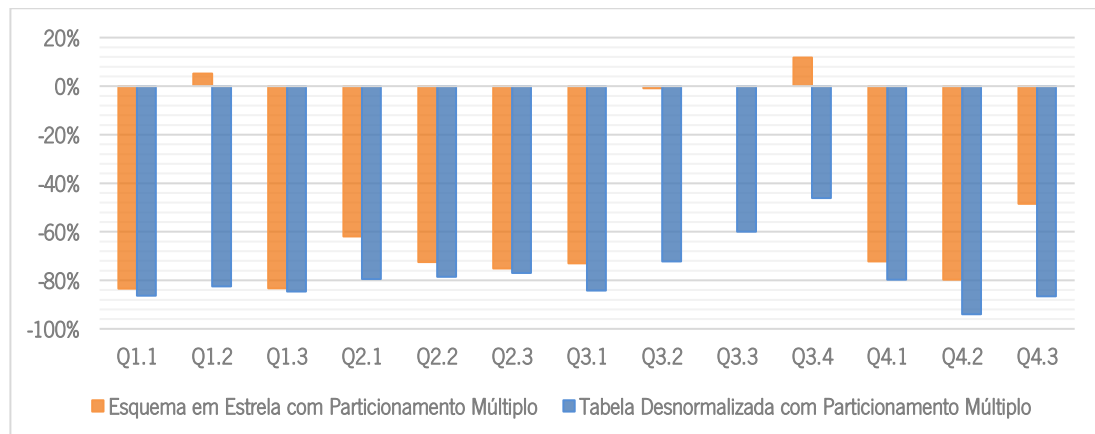


Figura 29 Variação na utilização de CPU com Particionamento Múltiplo (Presto; FE=100)

Relativamente aos resultados apresentados para o esquema em estrela é possível perceber que as *queries* (Q1.1, Q1.3, Q2.1, Q.2.2, Q2.3, Q3.1, Q3.2, Q3.3, Q4.1, Q4.2 e Q4.3) que apresentam os atributos como filtro (isolados ou combinados) são aquelas que sofrem diminuições na utilização de CPU. Relativamente à tabela desnormalizada, a vantagem é evidente em todas as *queries*, apesar de as diminuições serem menores nas *queries* que não apresentam estes atributos nos seus filtros. Assim, para além da diminuição dos tempos de processamento conseguidos, com este tipo de estratégia também se consegue menos tempo de utilização de CPU, melhorando o desempenho geral.

5.2.2 Cenário B2 - *Bucketing*

Segundo a literatura, apesar de poucas vezes referidos ou exemplificados, para a definição de *buckets* poderá ter-se em conta os atributos com elevada cardinalidade e ainda, a forma como se pretende agrupar/ordenar os dados de acordo com as *queries* executadas. Para além disso, o número de *buckets* deve ser estudado de forma a não criar ficheiros demasiado pequenos (Du, 2015; Hortonworks, 2017; Santos & Costa, 2016; Shaw et al., 2016).




Através de conhecimento prático obtido, admite-se que o número de *buckets* deve seguir a seguinte expressão:

$$\text{tamanho do ficheiro} \div \text{número de buckets} \geq \text{tamanho de um bloco HDFS}$$

Tendo em conta que no cluster utilizado nestas experiências o tamanho mínimo de um bloco HDFS é de 128MB, os cálculos seguintes irão ter, na maioria das vezes, esta expressão em consideração. Em alguns casos o tamanho do *dataset* será dividido por 1GB uma vez que, segundo (Shaw et al., 2016) os ficheiros devem ter pelo menos 1GB de dados para otimizar o armazenamento. Esta definição será mais adequada para fatores de escala maiores, uma vez que em grandes quantidades de dados a criação de ficheiros de cerca de 128MB de dados cria um número demasiado grande de pequenos ficheiros no HDFS podendo colocar em causa o seu desempenho.

A Tabela 20 apresenta os vários cenários de *bucketing* que serão testados assim como informações sobre cada tabela criada (espaço efetivamente ocupado, número de *buckets* definidos, tamanho médio de um *bucket*) em cada fator de escala e dependendo do modelo de dados.

Tabela 20 Informações sobre tabelas criadas com *bucketing*

<i>Buckets</i>	Modelo	FE			
Custkey	EE	30	5 GB	40	148 MB
		100	-	-	-
		300	-	-	-
	TD	30	18 GB	8	1,8 GB
		100	-	-	-
		300	-	-	-
Orderkey	EE	30	5 GB	5	1,1 GB
		100	19 GB	8	2,4 GB
		300	-	-	-
	TD	30	19 GB	15	1,3 GB
		100	69 GB	46	1,5 GB
		300	-	-	-
Suppkey	EE	30	4 GB	6	827 MB
		100	16 GB	10	1,7 GB
		300	49 GB	50	1020 MB
Od_Year (sorted by P_Brand)	TD	30	16 GB	7	2,5 GB
		100	58 GB	7	8,9 GB
		300	-	-	-
P_Brand (sorted by Year)	TD	30	17 GB	15	1,1 GB
		100	-	-	-
		300	-	-	-
Orderkey (sorted by Orderdate)	EE	30	4 GB	5	1011 MB
		100	-	-	-
		300	-	-	-
Orderdate, Suppkey, Custkey, Partkey	EE	30	5 GB	6	949 MB
		100	19 GB	16	1,2 GB
		300	60 GB	24	2,5 GB

 Espaço Ocupado;  Número de *Buckets*;  Tamanho médio de um *Bucket*

A definição do número de *buckets* foi um processo diferente de cenário para cenário, seguindo várias estratégias de acordo com o volume de dados e com o atributo pelo qual se definiam os segmentos. Assim, e de forma a simplificar a tabela anterior, a sua definição, assim como a seleção dos atributos utilizados, será explicada ao longo da apresentação dos vários cenários.

a) *Bucketing* Simples

Esta subsecção apresenta a maior parte dos cenários criados com a utilização dos *buckets* como estratégia de organização.

A primeira tabela criada foi o primeiro teste de *bucketing* realizado. Sendo um *dataset* relacionado com vendas, decidiu-se organizar os ficheiros por ID de cliente (*Custkey*), um dos filtros recorrentes em contextos reais. A Tabela 21 apresenta os resultados obtidos para esta primeira estratégia, para FE=30, evidenciando, a negrito, os melhores tempos de processamento e utilizando a seguinte notação: EE (Esquema em Estrela), EE-B (Esquema em Estrela com *Buckets*), TD (Tabela Desnormalizada), TD-B (Tabela Desnormalizada com *Buckets*).

Importa realçar que para a definição do número de *buckets* considerou-se o tamanho mínimo do bloco HDFS (128 MB), pelo qual se dividiu o tamanho total da tabela simples, tal como representado nas expressões seguintes.

$$\text{Número de } \textit{Buckets}_{FE=30, EE} \approx \frac{5088 \text{ MB}}{128 \text{ MB}} \approx 40 \textit{ buckets}$$

$$\text{Número de } \textit{Buckets}_{FE=30, TD} \approx \frac{14650 \text{ MB}}{128 \text{ MB}} \approx 114 \textit{ buckets} \rightarrow 10 \textit{ buckets}$$

Para a tabela desnormalizada não foi possível a criação dos 114 *buckets* uma vez que, através de conhecimento prático obtido com o Hive, para tabelas não particionadas em que se tenta criar um número excessivo de *buckets*, este não cria os ficheiros que ficariam vazios. Como na criação da tabela não foi possível criar 114 *buckets* uma vez que a ferramenta criava sempre um número menor, corrompendo a tabela (definiam-se 114 e o Hive não criava esse número de ficheiros), decidiu-se criar apenas 10 de acordo com o tipo do atributo. Como o atributo é do tipo *int*, considerando a função *hash* para a distribuição dos dados pelos *buckets*, cada ficheiro terá os dados correspondentes ao ID de cliente que termina com um algarismo específico. Assim, um dos *buckets* terá todos os valores de *C_Custkey* que terminem em 0, outro terá todos os valores que terminem em 1, e assim sucessivamente até 9.

Tabela 21 Tempos de Processamento para Bucketing por Custkey (em segundos)

	FE=30		FE=30			FE=30		FE=30	
	HIVE		PRESTO			HIVE		PRESTO	
	EE	EE-B	EE	EE-B		TD	TD-B	TD	TD-B
Q1.1	25	23	5	5	Q1.1	24	23	5	5
Q1.2	24	24	5	4	Q1.2	24	23	5	6
Q1.3	24	24	4	5	Q1.3	23	23	5	6
Q2.1	32	34	8	9	Q2.1	25	25	4	5
Q2.2	31	33	7	8	Q2.2	36	36	4	4
Q2.3	30	31	7	8	Q2.3	25	24	4	4
Q3.1	35	34	8	9	Q3.1	28	27	5	5
Q3.2	30	29	6	6	Q3.2	28	27	5	5
Q3.3	33	31	5	6	Q3.3	25	24	4	4
Q3.4	34	32	6	6	Q3.4	25	25	5	5
Q4.1	38	38	13	14	Q4.1	27	27	6	6
Q4.2	49	50	9	11	Q4.2	29	29	6	6
Q4.3	34	37	8	10	Q4.3	29	30	5	7
TOTAL	420	421	92	101	TOTAL	349	344	63	68
DIF		0%		9%	DIF		-1%		9%

Os resultados obtidos com esta estratégia de *bucketing* não demonstrou qualquer vantagem considerando os valores obtidos num contexto sem qualquer tipo de estratégia de organização. Mesmo com duas estratégias de definição do número de *buckets*, em quase todas as execuções os resultados são os mesmos e pequenas diminuições ou aumentos com *bucketing* não ultrapassam os 3 segundos, podendo ser consideradas pouco significativas para conclusões futuras.

Assim, este é, para já, um primeiro exemplo de uma má estratégia de organização por *buckets*, que, considerando os resultados obtidos para o FE=30, não foi replicada para fatores de escala maiores.

A Tabela 22 apresenta mais uma divisão por *buckets* por atributos com elevada cardinalidade, sendo neste caso o ID de encomenda (*Orderkey*) o selecionado e sendo a única diferença para o cenário anterior a fórmula para a definição do número de *buckets*. Assim, neste caso o tamanho total das tabelas simples foi dividido por 1GB de forma a criar *buckets* com tamanho maior, tal como representado nas expressões seguintes:

$$\text{Número de } \textit{Buckets}_{\text{FE=30, EE}} \approx \frac{5088 \text{ MB}}{1024 \text{ MB}} \approx 5 \text{ buckets}$$

$$\text{Número de } \textit{Buckets}_{\text{FE=30, TD}} \approx \frac{14650 \text{ MB}}{1024 \text{ MB}} \approx 15 \text{ buckets}$$

$$\text{Número de } \textit{Buckets}_{\text{FE=100, EE}} \approx \frac{16533 \text{ MB}}{1024 \text{ MB}} \approx 16 \text{ buckets} \rightarrow 8 \text{ buckets}$$

$$\text{Número de Buckets}_{FE=100, TD} \approx \frac{46800 \text{ MB}}{1024 \text{ MB}} \approx 46 \text{ buckets}$$

A alteração na terceira expressão está mais uma vez relacionada com o facto de o Hive não criar os 16 *buckets* solicitados, o que indica que apenas 8 seriam efetivamente preenchidos, razão pela qual se usa esta definição.

Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-B (Esquema em Estrela com *Buckets*), TD (Tabela Desnormalizada), TD-B (Tabela Desnormalizada com *Buckets*).

Tabela 22 Tempos de Processamento com Bucketing por Orderkey (em segundos)

	FE=30		FE=100		FE=30		FE=100	
	HIVE				PRESTO			
	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B
Q1.1	25	23	31	29	5	7	13	14
Q1.2	24	23	29	30	5	7	13	13
Q1.3	24	23	29	30	4	6	13	13
Q2.1	32	33	47	59	8	11	19	26
Q2.2	31	32	46	51	7	11	18	23
Q2.3	30	30	44	54	7	10	17	22
Q3.1	35	35	59	64	8	12	29	30
Q3.2	30	30	45	46	6	8	17	19
Q3.3	33	34	219	224	5	8	15	18
Q3.4	34	32	222	225	6	7	15	18
Q4.1	38	39	86	100	13	19	43	47
Q4.2	49	50	70	70	9	14	26	33
Q4.3	34	35	54	65	8	13	23	29
TOTAL	420	421	982	1047	92	133	262	305
DIF		0%		7%		44%		16%
	HIVE		HIVE		PRESTO		PRESTO	
	TD	TD-B	TD	TD-B	TD	TD-B	TD	TD-B
	TD	TD-B	TD	TD-B	TD	TD-B	TD	TD-B
Q1.1	24	23	29	31	5	5	13	15
Q1.2	24	23	29	30	5	6	14	15
Q1.3	23	23	30	30	5	5	14	15
Q2.1	25	26	36	42	4	6	10	14
Q2.2	36	35	73	69	4	5	10	12
Q2.3	25	23	35	34	4	4	10	10
Q3.1	28	27	40	45	5	5	12	13
Q3.2	28	27	41	44	5	5	12	12
Q3.3	25	24	38	35	4	4	9	10
Q3.4	25	25	38	42	5	5	12	12
Q4.1	27	27	41	44	6	7	14	16
Q4.2	29	29	42	46	6	6	14	17
Q4.3	29	29	42	47	5	6	12	17
TOTAL	349	342	516	539	63	71	155	178
DIF		-2%		5%		14%		15%

Mais uma vez, não existe vantagem no *bucketing* por ID mesmo com um número de ficheiros menor. Analisando os resultados gerais, aparentemente não há qualquer reconhecimento da estratégia de organização aplicada uma vez que os tempos de processamento aumentam sempre, com exceção do caso do Hive para FE=30, que apesar de não sofrer alteração, ou diminuir apenas 2%, são variações pouco significativas que podem alterar de execução para execução.

Assim, este cenário não parece representar uma prática adequada de organização dos dados para este *dataset* e para as *queries* executadas.

A documentação da Hortonworks e do Hive afirmam que a introdução da condição “Sorted By” na criação das tabelas com *buckets* pode ser vantajosa para a velocidade de processamento. Assim, foram testados alguns cenários com a sua aplicação.

A Tabela 23 apresenta os resultados obtidos para a tabela definida com *buckets* por *Od_Year* e com aplicação do “sorted by” por *P_Brand.*, evidenciando, a negrito, os melhores tempos de processamento e utilizando a seguinte notação: TD (Tabela Desnormalizada), TD-B(Tabela Desnormalizada com *Buckets*). Para além do estudo da aplicação desta técnica para ordenar os dados, este cenário pretende também estudar a definição de *buckets* recorrendo a atributos que aparecem várias vezes nas condições “group by” e “order by” das *queries*. Assim, apesar de ser um atributo com baixa cardinalidade, definiram-se *buckets* por ano de encomenda (*Od_Year*), tendo sido criados 7 *buckets* em ambos os fatores de escala, ficando os dados de cada ano armazenados num ficheiro diferente. Importa ainda realçar que apenas são aplicados à tabela desnormalizada uma vez que não é possível criar *buckets* por atributos apenas presentes nas dimensões, contrariamente ao que acontece com o particionamento. Para além disso, não foi possível replicar para o FE=300 devido a problemas de memória com o *cluster* utilizado.

De forma geral, os resultados demonstram diminuição dos tempos de processamento quando num contexto de *buckets* ordenados. Quase todas as *queries* que apresentam o atributo nas condições “group by” e “order by”, que abrangem desde a 2.1 à 4.3 (destacadas a negrito na Tabela 23), apresentam vantagens com esta estratégia de organização. De todas estas *queries*, as *queries* do grupo 3 são aquelas em que nem sempre se verifica esta diminuição nos tempos de processamento, no entanto, já tem ficado claro que estas são *queries* com grande complexidade e que apresentam filtros com grandes intervalos temporais, pelo que em contextos de maiores quantidades de dados pode ser necessário algum tempo extra de processamento.

Tabela 23 Tempos de Processamento com Bucketing por Od_Year, Sorted by P_Brand (em segundos)

	FE=30		FE=100		FE=30		FE=100	
	HIVE				PRESTO			
	TD	TD-B	TD	TD-B	TD	TD-B	TD	TD-B
Q1.1	24	18	29	21	5	3	13	8
Q1.2	24	19	29	21	5	3	14	9
Q1.3	23	18	30	22	5	3	14	8
Q2.1	25	18	36	20	4	2	10	4
Q2.2	36	18	73	20	4	2	10	3
Q2.3	25	18	35	16	4	2	10	4
Q3.1	28	26	40	39	5	5	12	14
Q3.2	28	25	41	40	5	5	12	13
Q3.3	25	23	38	32	4	4	9	11
Q3.4	25	26	38	39	5	5	12	13
Q4.1	27	22	41	30	6	3	14	9
Q4.2	29	20	42	23	6	2	14	5
Q4.3	29	14	42	15	5	2	12	4
TOTAL	349	265	516	337	63	41	155	103
DIF		-24%		-35%		-35%		-34%

Mesmo as *queries* que não apresentam esta condição, as *queries* do grupo 1, sofrem diminuição em todos os cenários. Analisando este grupo de *queries*, é possível perceber que são as *queries* de menor complexidade e são também aquelas que apenas lidam com dados temporais em quase todas as condições, pelo que uma organização dos ficheiros por ano acaba por facilitar o processamento das mesmas pelas técnicas de otimização utilizadas pelos sistemas de *querying*.

Para além disso, as *queries* 2.2, 2.3 e 4.3 apresentam, ainda, o atributo *P_Brand* no “select” assim como nos filtros e em todas as outras condições, e como os ficheiros estão ordenados por este atributo, facilita o processo tornando-as exemplos de *queries* em que fica mais evidente a diminuição nos tempos de processamento, diminuição essa que chega a ser superior a 50% na maior parte das execuções.

Este é, para já, o primeiro cenário que recorre a *buckets* e que apresenta alguma vantagem com este tipo de organização dos dados.

A Tabela 24 apresenta mais uma divisão por *buckets* por atributos com baixa cardinalidade e com a utilização da técnica de ordenação, sendo a única diferença para o cenário anterior a permuta dos atributos (os *buckets* foram definidos com o atributo *P_Brand* e a tabela foi ordenada por ano) e a fórmula para a definição do número de *buckets*. Assim, neste caso, o tamanho total da tabela simples foi dividido

por 1GB de forma a criar ficheiros relativamente grandes, estando o número de *buckets* representado na seguinte expressão:

$$\text{Número de } \mathit{Buckets}_{FE=30, EE} \approx \frac{5088 \text{ MB}}{1024 \text{ MB}} \approx 5 \text{ buckets}$$

$$\text{Número de } \mathit{Buckets}_{FE=30, TD} \approx \frac{14650 \text{ MB}}{1024 \text{ MB}} \approx 15 \text{ buckets}$$

Para além disso, este cenário é, mais uma vez, apenas aplicado à tabela desnormalizada como consequência da impossibilidade de criação de *buckets* por *P_Brand* na tabela de factos.

Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: TD (Tabela Desnormalizada), TD-B(Tabela Desnormalizada com *Buckets*).

Tabela 24 Tempos de Processamento para Bucketing por "P_Brand", Sorted By Od_Year (em segundos)

	FE=30			
	HIVE		PRESTO	
	TD	TD-B	TD	TD-B
Q1.1	24	20	5	2
Q1.2	24	20	5	2
Q1.3	23	20	5	2
Q2.1	25	25	4	4
Q2.2	36	36	4	3
Q2.3	25	23	4	3
Q3.1	28	26	5	4
Q3.2	28	26	5	4
Q3.3	25	24	4	3
Q3.4	25	26	5	4
Q4.1	27	28	6	5
Q4.2	29	22	6	2
Q4.3	29	22	5	2
TOTAL	349	317	63	42
DIF		-9%		-34%

Apesar dos poucos cenários testados, é possível perceber que, da mesma forma que o cenário apresentado anteriormente, há vantagens na criação deste tipo de organização que recorre a atributos que aparecem várias vezes na condição “*group by*” e “*order by*” e com a utilização de mecanismos de ordenação.

Uma vez que era um cenário com conclusões similares às do cenário anterior, este não foi replicado para fatores de escala superiores.

No seguimento destes dois últimos cenários, e dada a impossibilidade de criar *buckets* na tabela de factos com atributos das dimensões, para testar a implementação de *buckets* combinada com a

técnica do *Sorted By* neste modelo, criou-se a tabela com *buckets* por *Orderkey* e ordenada por *Orderdate*.

A definição do número de *buckets* teve em consideração a seguinte expressão:

$$\text{Número de } \textit{Buckets}_{\text{FE=30, EE}} \approx \frac{5088 \text{ MB}}{1024 \text{ MB}} \approx 5 \textit{ buckets}$$

Os resultados desta configuração estão apresentados na Tabela 25 que evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-B (Esquema em Estrela com *Buckets*).

Tabela 25 Tempos de Processamento para Bucketing por Orderkey, Sorted By Orderdate (em segundos)

	FE=30 HIVE		FE=30 PRESTO	
	EE	EE-B	EE	EE-B
Q1.1	25	22	5	5
Q1.2	24	22	5	5
Q1.3	24	23	4	5
Q2.1	32	33	8	11
Q2.2	31	31	7	10
Q2.3	30	31	7	9
Q3.1	35	31	8	10
Q3.2	30	28	6	7
Q3.3	33	32	5	6
Q3.4	34	30	6	7
Q4.1	38	39	13	16
Q4.2	49	48	9	12
Q4.3	34	35	8	11
TOTAL	420	405	92	115
DIF		-4%		25%

Como são dois atributos que não são reconhecidos em nenhuma das condições de “*Group By*” das *queries*, os resultados para este cenário não apresentam benefícios em comparação com os resultados obtidos para as tabelas sem qualquer tipo de estratégia de organização.

Apesar de existir uma diminuição nos resultados do Hive, a verdade é que esta pode ser considerada insignificante. Para além disso, o Presto aparentemente não reconhece este tipo de estratégia de organização considerando um aumento de 25% do tempo de processamento quando comparado com a estrela sem *buckets*.

Como tal, estes testes não serão replicados para fatores de escala superiores.

Tal como referido em secções anteriores, Du (2015, Santos & Costa (2016 e Shaw et al. (2016) defendem que a definição de *buckets* traz vantagens no *join* de duas ou mais tabelas, desde que ambas façam uso do *bucketing* pela mesma coluna. Assim, e só fazendo sentido para o modelo normalizado,

foram definidos dois cenários distintos na forma de criação de *buckets* mas que pretendem estudar a vantagem do *bucketing* nos *joins*.

A Tabela 26 apresenta o primeiro cenário, explicado anteriormente, que inclui a criação da tabela de factos com a definição de *buckets* por uma chave com a qual faz *join* com uma das dimensões. Decidiu-se criar para o *join* da tabela de factos com a dimensão *supplier*, pelo que ambas foram criadas utilizando como *bucket* o atributo *Suppkey* (atributo utilizado no *join*). O segundo cenário será apresentado na subsecção seguinte (*Bucketing* múltiplo). A definição do número de *buckets* para a tabela de factos teve em consideração as seguintes expressões, organizadas por fator de escala.

$$\text{Número de Buckets}_{FE=30, EE} \approx \frac{5088 \text{ MB}}{1024 \text{ MB}} \approx 5 \text{ buckets} \rightarrow 6 \text{ buckets}$$

Esta alteração de 5 para 6 *buckets* na definição do número de buckets na tabela de factos foi realizada de forma a este número ser par e múltiplo do número de *buckets* definido para a dimensão. Como a dimensão era de pequena dimensão foram definidos apenas 2 *buckets* na dimensão.

Esta tabela evidencia, a negrito, os melhores tempos de processamento e utilizando a seguinte notação: EE (Esquema em Estrela), EE-B (Esquema em Estrela com *Buckets*).

Tabela 26 Tempos de Processamento para Bucketing por Suppkey (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B
Q1.1	25	22	31	29	44	46	5	6	13	16	36	36
Q1.2	24	23	29	29	42	44	5	7	13	16	34	36
Q1.3	24	23	29	30	43	45	4	7	13	14	35	34
Q2.1	32	31	47	53	531	110	8	11	19	25	59	62
Q2.2	31	29	46	66	531	611	7	9	18	22	51	56
Q2.3	30	29	44	49	531	101	7	9	17	22	49	53
Q3.1	35	33	59	68	651	137	8	11	29	35	81	83
Q3.2	30	28	45	52	677	92	6	8	17	23	51	53
Q3.3	33	33	219	45	665	80	5	7	15	17	43	44
Q3.4	34	30	222	44	675	78	6	6	15	19	43	43
Q4.1	38	39	86	88	226	237	13	17	43	51	119	127
Q4.2	49	49	70	65	141	119	9	11	26	32	69	75
Q4.3	34	35	54	57	116	103	8	10	23	28	63	67
TOTAL	420	404	982	676	4874	1803	92	120	262	321	733	768
DIF		-4%		-31%		-63%		30%		22%		5%

Os resultados aqui apresentados demonstram o potencial do Hive neste tipo de estratégias de organização. Ao contrário do que se tem verificado, este tipo de organização que potencia os benefícios

dos *buckets* para o *join* de duas mais tabelas, assegura um melhor desempenho do Hive quando comparado com o Presto, apesar de este último continuar a obter melhores tempos de processamento.

O Hive, apesar de não aparecer nas explicações das execuções das queries, ativa, à partida e principalmente em fatores de escala superiores, o *bucket map join*, uma estratégia de junção apropriada para situações de tabelas de grandes dimensões, com *bucketing* pelo atributo do *join* e desde que o número de *buckets* de uma das tabelas seja múltiplo do número de *buckets* de outra (Du, 2015). Como estas condições se verificam neste cenário (apesar de em alguns fatores de escala os tamanhos das tabelas não serem assim tão grandes), há uma clara vantagem com a diminuição de 63% no fator de escala 300.

Como o Presto não apresenta este tipo de mecanismo, provavelmente não reconhece este tipo de estratégia de organização, e, por isso, as vantagens não são demonstradas com a utilização deste sistema.

Este seria, então, uma estratégia benéfica para situações em que se opta por um *Big Data Warehouse* baseado em modelos multidimensionais, em que o Hive fosse utilizado não só para armazenamento, mas também para o processamento.

Para além do estudo da velocidade de processamento, foi, ainda, estudada a utilização de CPU por cada *query* em ambos os modelos de dados, agora com a aplicação *bucketing* simples.

A Figura 30 apresenta a variação obtida no Presto na utilização de CPU, por parte das tabelas com *bucketing* simples em relação às tabelas sem qualquer tipo de organização apresentadas no cenário A. O exemplo de *bucketing* simples aqui utilizado foi o *bucketing* por *Orderkey*.

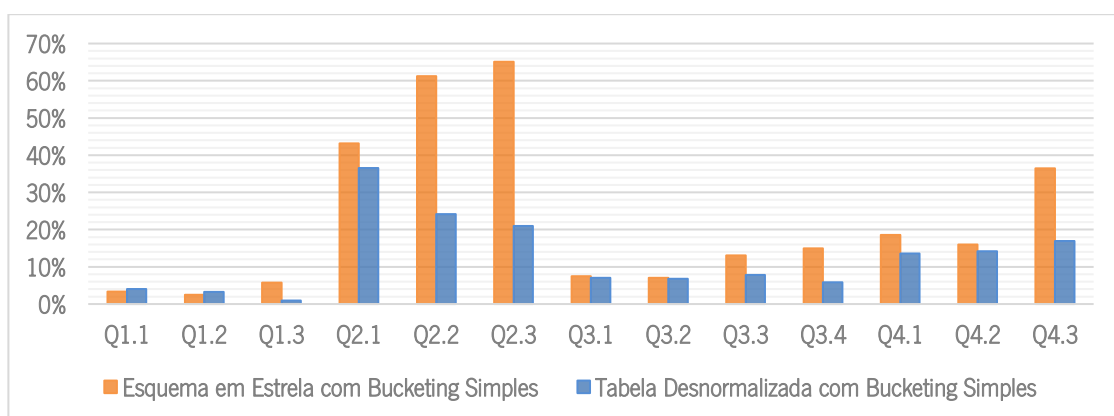


Figura 30 Variação de utilização de CPU com Bucketing Simples por Orderkey

Tal como no estudo dos tempos de processamento, este cenário demonstra desvantagem considerando a utilização do processador central. Todas as *queries*, em ambos os modelos, exigem uma maior utilização do processador, sendo que chega a precisar de mais de 60% de tempo de utilização do

que no cenário normalizado simples e mais de 35% do que no cenário desnormalizado sem qualquer estratégia de organização. Como tal, este é um exemplo de uma má estratégia de organização dos dados que pode trazer desvantagens ao nível do tempo de processamento assim como dos recursos necessários para tal.

Uma vez que foi com a aplicação de *bucketing* simples em duas tabelas pela mesma coluna, que se obtiveram os melhores resultados para o esquema em estrela, a Figura 31 apresenta as variações de utilização de CPU obtidas pelas duas ferramentas de *querying* para FE=100.

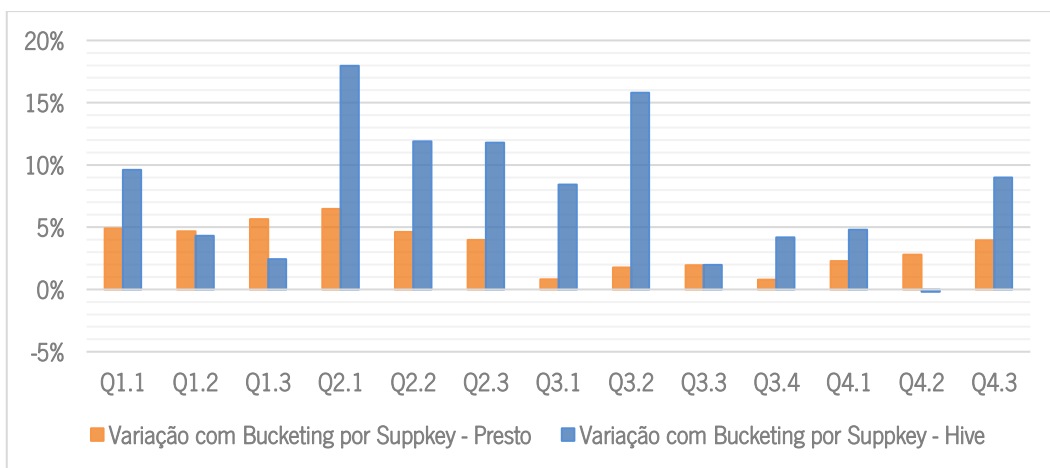


Figura 31 Variação de utilização de CPU com Bucketing Simples por Suppkey (modelo normalizado)

Tal como aconteceu com as variações no tempo de processamento, não é no contexto de FE = 100, e apesar de nesses testes o Hive ainda reconhecer melhor a estratégia de *bucketing* do que o Presto, este reconhecimento não implica a diminuição no tempo de utilização de CPU. Assim, não se verificam vantagens com esta estratégia relativamente à utilização dos recursos computacionais.

No entanto, se considerarmos esta estratégia para FE=300 e apenas com o Hive, já que o Presto não reconhece a estratégia em nenhum dos contextos, a Figura 32 apresenta esta variação.

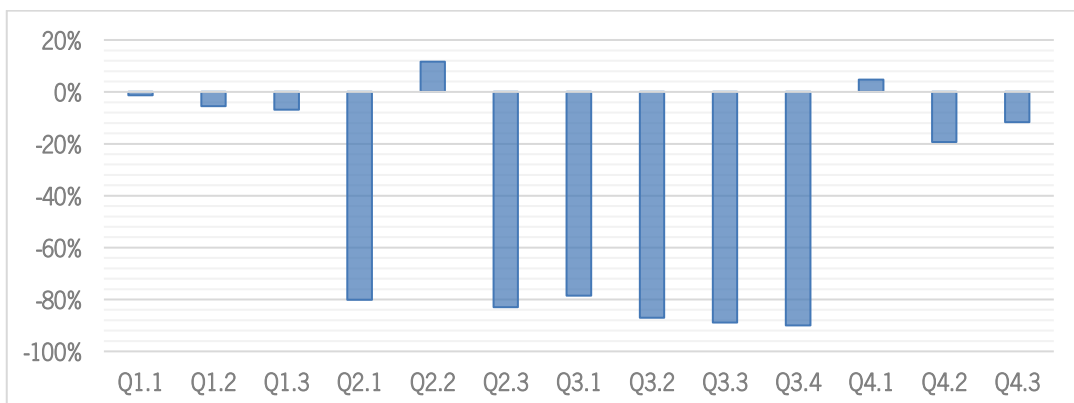


Figura 32 Variação na utilização de CPU com Bucketing Simples por Suppkey (FE=300, Hive)

Com estes valores aqui apresentados, é possível perceber que, com o Hive, não só se verificam diminuições significativas no tempo de processamento como também no tempo de utilização de CPU. Em média, há uma diminuição de cerca de 41% com a aplicação desta estratégia de organização no fator de escala maior (que é quando o Hive parece ativar mecanismos de otimização de joins).

Considerando a aplicação da técnica de *Sorted By*, o cenário de *bucketing* simples para a tabela desnormalizada que obteve melhores resultados, a Figura 33 apresenta a variação na utilização de CPU por parte das tabelas com *bucketing* simples com dados ordenados, em relação às tabelas sem qualquer tipo de organização apresentadas no cenário A e às tabelas com particionamento. O exemplo de *bucketing* simples aqui utilizado foi o *bucketing* por *Od_Year* (*Sorted by P_Brand1*) e a tabela de particionamento tem como partição o atributo *Od_Year*, e todos os resultados aqui apresentados são os resultados do Presto para FE=100 e para o modelo desnormalizado.

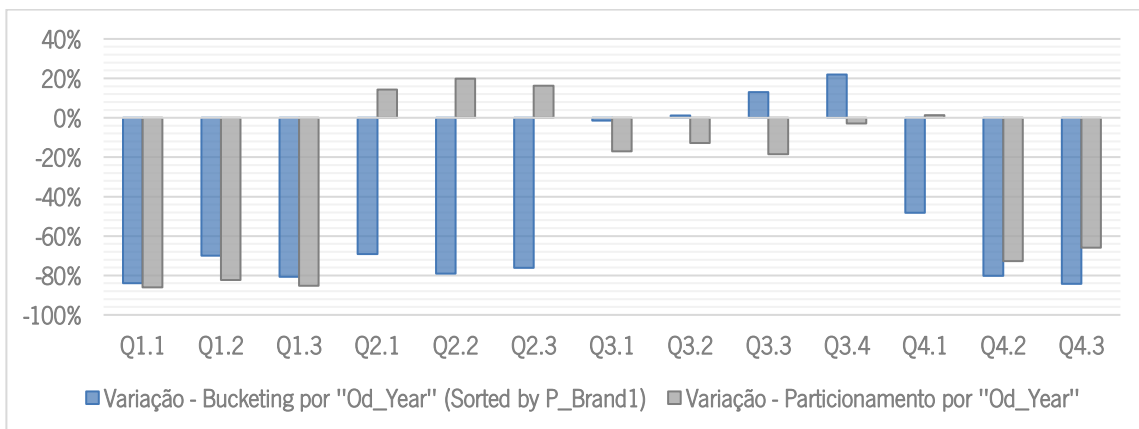


Figura 33 Comparação da variação de utilização de CPU entre *bucketing* simples (*sorted by*) e *particionamento* simples

Como é possível verificar, as queries que reconhecem o atributo *Od_Year* na condição "*Group By*" (que abrangem desde a Q2.1 à Q4.3) são as que sofrem uma maior diminuição em relação à utilização de CPU, com a exceção das *queries* do grupo 3 que, tal como aconteceu nos tempos de processamento, sofrem algum aumento por serem *queries* com grande complexidade e que apresentam filtros com grandes intervalos temporais, pelo que em contextos de maiores quantidades de dados pode ser necessário algum tempo extra na utilização de CPU. As *queries* do grupo 1 sofrem também diminuições uma vez que se tratam de *queries* que, como só têm filtros e condições temporais, beneficiam de uma organização dos ficheiros por ano.

Relativamente à comparação das variações na utilização de CPU por parte das tabelas com *Bucketing* simples (com dados ordenados) com as tabelas com *particionamento* simples, é possível concluir que como o atributo *Od_Year* aparece mais vezes na condição *Group By* do que na condição

Where, são mais as *queries* que beneficiam das técnicas de *bucketing* simples. No entanto, como no contexto de *bucketing* as *queries* do grupo 3 não apresentam diminuição, o número de *queries* que sofre aumento (Q3.1 à Q3.4) é o mesmo número de *queries* que sofre aumento no contexto de particionamento (Q2.1 à Q2.3 e Q4.1). O que se verifica é que as diminuições conseguidas com particionamento são inferiores às obtidas com o *bucketing*, daí se obter, em média, menos 19% do tempo de utilização de CPU em contextos de *bucketing* simples.

b) **Bucketing Múltiplo**

Tal como referido anteriormente, os *buckets* podem ter implicações nos *joins* de duas ou mais tabelas pelo que o segundo cenário para estudar este fenómeno é um cenário que implica a criação de múltiplos *buckets*. Apesar de vários autores afirmarem que só se cria *buckets* recorrendo a um atributo (Capriolo et al., 2012; Santos & Costa, 2016; Shaw et al., 2016; Thusoo et al., 2010), através de conhecimento prático obtido foi possível perceber que é possível a criação de *buckets* por múltiplos atributos (os atributos são considerados como uma única *string* – união de atributos – e o *join* irá beneficiar apenas quando todas as colunas são utilizados como critério do *join*), pelo que este cenário pretende testar se é mesmo possível, se as ferramentas reconhecem estas estratégias e se, efetivamente, existe algum tipo de vantagem com esta estrutura de dados. A tabela de factos foi, assim, definida com quatro *buckets* que correspondem às chaves com as quais faz o *join* com cada uma das dimensões (*Orderdate, Custkey, Suppkeye Partkey*), que foram, por sua vez, criadas com a definição do *buckets* pela chave correspondente.

$$\text{Número de } \mathit{Buckets}_{FE=30, EE} \approx \frac{5088 \text{ MB}}{1024 \text{ MB}} \approx 5 \text{ buckets} \rightarrow 6 \text{ buckets}$$

$$\text{Número de } \mathit{Buckets}_{FE=100, EE} \approx \frac{16533 \text{ MB}}{1024 \text{ MB}} \approx 16 \text{ buckets}$$

$$\text{Número de } \mathit{Buckets}_{FE=300, EE} \approx \frac{49700 \text{ MB}}{2048 \text{ MB}} \approx 24 \text{ buckets}$$

A transformação, na primeira expressão, está, mais uma vez, relacionada com a criação de um número múltiplo dos *buckets* criados nas dimensões (2). Para o FE=300 decidiu-se dividir por 2GB para não criar um número elevado de *buckets* e para testar outra configuração.

Os resultados obtidos com esta configuração estão apresentados na Tabela 27. Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-B (Esquema em Estrela com *Buckets*).

Tabela 27 Tempo de Processamento com Bucketing por Orderdate, Custkey, Suppkey e Partkey (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B	EE	EE-B
Q1.1	25	23	31	29	44	45	5	5	13	14	36	35
Q1.2	24	24	29	30	42	45	5	6	13	12	34	34
Q1.3	24	24	29	30	43	44	4	5	13	12	35	36
Q2.1	32	33	47	59	531	702	8	11	19	27	59	82
Q2.2	31	31	46	51	531	681	7	9	18	23	51	67
Q2.3	30	31	44	54	531	699	7	9	17	22	49	62
Q3.1	35	34	59	64	651	684	8	11	29	30	81	88
Q3.2	30	30	45	46	677	688	6	7	17	20	51	57
Q3.3	33	33	219	224	665	702	5	7	15	17	43	52
Q3.4	34	32	222	225	675	870	6	7	15	16	43	52
Q4.1	38	39	86	100	226	256	13	18	43	49	119	142
Q4.2	49	50	70	70	141	155	9	14	26	33	69	90
Q4.3	34	37	54	65	116	141	8	12	23	29	63	77
TOTAL	420	420	982	1047	4874	5712	92	121	262	305	733	876
DIF		0%		7%		17%		32%		16%		19%

Relativamente às possíveis otimizações que se conseguiriam com *bucketing* no *join* de duas ou mais tabelas, os resultados obtidos não comprovam esse contexto. Apesar das grandes expectativas para este cenário, os resultados obtidos demonstram uma clara desvantagem deste tipo de estratégia de organização, uma vez que em cerca de 92% dos casos continua a ser vantajoso manter tabelas simples do que ter uma organização em *buckets* recorrendo a esta estratégia. Mesmo as *queries* que incluem todos os atributos nos *joins* (Q4.1, Q4.2 e Q4.3) não demonstram qualquer benefício com esta configuração.

As desvantagens da aplicação do *bucketing* múltiplo estão, então, aqui evidenciadas, considerando o aumento nos tempos gerais de processamento, aumentos esses que atingem os 32%. Como tal, esta é uma estratégia de organização dos dados que não traz vantagens nem em termos de velocidade de processamento, nem em termos dos recursos necessários para tal.

Apesar de, por si só, os resultados do processamento das *queries* não demonstrarem qualquer tipo de benefício, estudou-se também a variação na utilização do processador num contexto de *bucketing* múltiplo. Esta variação está representada na Figura 34.

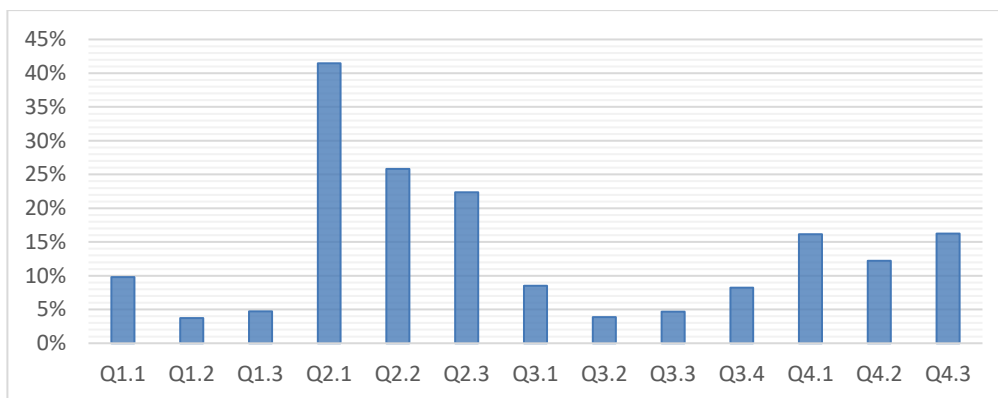


Figura 34 Variação na utilização de CPU com Bucketing Múltiplo

Tal como com a velocidade de processamento, também a desvantagem deste cenário relativamente à utilização de CPU se torna clara. Todas as *queries* exigem uma maior utilização do processador, sendo que chega mesmo a gastar mais 40% de tempo de utilização do que no cenário de um esquema em estrela sem *buckets*. Como tal, esta é mais uma estratégia de organização dos dados que não traz vantagens nem em termos de velocidade de processamento, nem em termos dos recursos necessários para tal.

5.2.3 Cenário B3 - Particionamento e *Bucketing*

Esta subsecção apresenta os resultados obtidos com a combinação de partições e *buckets*, tendo em consideração alguns dos resultados obtidos nos cenários apresentados anteriormente. A Tabela 28 apresenta os vários cenários com a combinação de particionamento e *bucketing* que serão testados, assim como informações sobre cada tabela criada (espaço efetivamente ocupado, número de pastas criadas com o particionamento, número de *buckets* definidos, tamanho médio de um *bucket*) em cada fator de escala e dependendo do modelo de dados.

Tabela 28 Informações sobre tabelas criadas com particionamento e *bucketing*

Partições (P) e <i>Buckets</i> (B)	Modelo	FE				
P = Od_Year B = Orderkey	EE	30	4 GB	7	6	135 MB
		100	19 GB		22	140 MB
		300	59 GB		9	1,1 GB
	TD	30	19 GB	7	10	308 MB
		100	66 GB	7	6	1,7 GB
		300	-	-	-	-

Partições (P) e <i>Buckets</i> (B)	Modelo	FE				
P = S_Region B = Suppkey	EE	30	4 GB	5	6	163 MB
		100	16 GB		2	1,6 GB
		300	48 GB		8	1,2 GB
P = Od_Year B = P_Brand	TD	30	16 GB	7	6	436 MB
		100	58 GB			1,5 GB
		300	-			-
P = Od_Year + S_Region B = Suppkey	EE	30	5 GB	7x5=35	2	83 MB
		100	18 GB			289 MB
		300	55 GB			868 MB
	TD	30	15 GB	7x5=35	2	242 MB
		100	57 GB			890 MB
		300	-			-

Espaço Ocupado; Número de Pastas; Número de Ficheiros por pasta; Tamanho médio de um ficheiro por pasta

A Tabela 29 apresenta os resultados para o primeiro cenário testado que considera como partição o atributo *Od_Year* (com baixa cardinalidade e encontrado várias vezes nos filtros das *queries*) e como *bucket* o atributo *Orderkey* (cardinalidade elevada). Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-PB (Esquema em Estrela com Partições e *Buckets*), TD (Tabela Desnormalizada), TD-PB (Tabela Desnormalizada com Partições e *Buckets*).

Importa realçar que não foi possível replicar este cenário para a tabela desnormalizada do FE=300 devido a problemas de memória do *cluster* utilizado. Para além disso, a definição do número de *buckets* teve em consideração as seguintes expressões:

$$\text{Número de } \textit{Buckets}_{FE=30, EE} \approx \frac{828 \text{ MB}}{128 \text{ MB}} \approx 6 \textit{ buckets}$$

$$\text{Número de } \textit{Buckets}_{FE=30, TD} \approx \frac{2561 \text{ MB}}{128 \text{ MB}} \approx 20 \textit{ buckets} \rightarrow 10 \textit{ buckets}$$

$$\text{Número de } \textit{Buckets}_{FE=100, EE} \approx \frac{2844 \text{ MB}}{128 \text{ MB}} \approx 22 \textit{ buckets}$$

$$\text{Número de } \textit{Buckets}_{FE=100, TD} \approx \frac{6247 \text{ MB}}{1024 \text{ MB}} \approx 6 \textit{ buckets}$$

$$\text{Número de } \textit{Buckets}_{FE=300, EE} \approx \frac{8670 \text{ MB}}{1024 \text{ MB}} \approx 9 \textit{ buckets}$$

Tabela 29 Tempos de Processamento com Particionamento por Od_Year e Bucketing por Orderkey (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB
Q1.1	25	16	31	21	44	26	5	2	13	4	36	7
Q1.2	24	23	29	32	42	42	5	6	13	13	34	44
Q1.3	24	18	29	21	43	25	4	2	13	4	35	8
Q2.1	32	33	47	60	531	682	8	11	19	26	59	98
Q2.2	31	32	46	53	531	677	7	10	18	23	51	76
Q2.3	30	30	44	52	531	670	7	9	17	22	49	74
Q3.1	35	31	59	56	651	667	8	10	29	29	81	100
Q3.2	30	28	45	50	677	634	6	7	17	19	51	63
Q3.3	33	33	219	78	665	648	5	6	15	16	43	53
Q3.4	34	31	222	228	675	674	6	7	15	19	43	59
Q4.1	38	39	86	102	226	253	13	17	43	50	119	164
Q4.2	49	35	70	63	141	91	9	7	26	18	69	48
Q4.3	34	28	54	50	116	77	8	6	23	14	63	38
TOTAL	420	378	982	865	4874	5166	92	100	262	256	733	835
DIF		-10%		-12%		6%		8%		-2%		14%
	HIVE		HIVE		HIVE		PRESTO		PRESTO		PRESTO	
	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB
	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB
Q1.1	24	20	29	23			5	2	13	4		
Q1.2	24	20	29	22			5	2	14	5		
Q1.3	23	19	30	21			5	2	14	4		
Q2.1	25	25	36	40			4	5	10	14		
Q2.2	36	37	73	69			4	4	10	13		
Q2.3	25	23	35	33			4	4	10	11		
Q3.1	28	26	40	41			5	4	12	13		
Q3.2	28	26	41	40			5	4	12	13		
Q3.3	25	24	38	32			4	4	9	11		
Q3.4	25	24	38	39			5	4	12	11		
Q4.1	27	26	41	45			6	6	14	18		
Q4.2	29	22	42	26			6	3	14	6		
Q4.3	29	22	42	26			5	3	12	6		
TOTAL	349	313	516	456			63	46	155	129		
DIF		-10%		-12%				-26%		-17%		

Com este tipo de cenários, e dada a dificuldade em perceber até que ponto a utilização do *bucketing* tem impacto nos tempos de processamento, surge sempre a dúvida se os resultados positivos com a configuração criada estão relacionados com a combinação das duas estratégias ou se estarão apenas relacionados com o reconhecimento do particionamento, que como foi possível verificar é, efetivamente, benéfico para o processamento de dados.

Analisando as *queries*, verifica-se que as *queries* que efetivamente sofrem algum tipo de decréscimo a nível de tempo de processamento são aquelas em que o atributo utilizado como partição aparece nos filtros de dados, nomeadamente as *queries* 1.1, 1.3, 3.1, 3.2, 3.3, 4.2 e 4.3.

Para além disso, relembrando os resultados gerais do cenário que incluía apenas o particionamento por ano, representados na Tabela 30, fica bastante claro que os resultados são muito similares.

Tabela 30 Resultados gerais do cenário de Particionamento Simples por Od_Year

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P	EE	EE-P
TOTAL	420	381	982	868	4874	5042	92	87	262	239	733	645
DIF		-9%		-12%		3%		-5%		-9%		-12%
	HIVE						PRESTO					
	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P	TD	TD-P
TOTAL	349	313	516	449	1090	901	63	46	155	97	472	280
DIF		-10%		-13%		-17%		-26%		-38%		-41%

Aliando estes resultados semelhantes com os resultados obtidos no cenário de *bucketing* simples por *Orderkey* que não foram nada benéficos, e com o potencial aumento dos tempos de processamento com o aumento do volume de dados (tendo em conta os resultados verificados para FE=300 no esquema em estrela, tanto para o Hive como para o Presto), pode concluir-se que este tipo de combinação não apresenta vantagens acrescidas, podendo manter-se como estratégia de organização o particionamento simples sem a definição de *buckets*.

A Tabela 31 apresenta os resultados obtidos para um outro cenário de combinação de particionamento simples com *bucketing* simples, sendo que neste caso se utilizou como partição o atributo *S_Region* (baixa cardinalidade e presente nos filtros de várias queries) e como *bucket* o atributo *Supkey* (elevada cardinalidade e atributo utilizado para realizar o *join* da tabela de factos com a dimensão "Supplier"). Como este teste pretende reintroduzir a questão das vantagens da definição de *buckets* nos *joins* de duas ou mais tabelas, este cenário é apenas aplicado ao esquema em estrela. Importa realçar que a dimensão "Supplier" foi também criada com *bucketing* por *Supkey* e o número de *buckets* foi definido consoante o tamanho médio de uma partição, dividindo-o por 1GB no caso das tabelas de factos dos fatores de escala maiores, e por 128MB no caso da tabela de factos de FE=30 e de todas as dimensões.

$$\text{Número de Buckets}_{FE=30, EE} \approx \frac{879 \text{ MB}}{128 \text{ MB}} \approx 7 \text{ buckets} \rightarrow 6 \text{ buckets}$$

$$\text{Número de Buckets}_{FE=100, EE} \approx \frac{3306 \text{ MB}}{1024 \text{ MB}} \approx 3 \text{ buckets} \rightarrow 2 \text{ buckets}$$

$$\text{Número de Buckets}_{FE=300, EE} \approx \frac{9830 \text{ MB}}{1024 \text{ MB}} \approx 9 \text{ buckets} \rightarrow 8 \text{ buckets}$$

A transformação em todas as expressões está, mais uma vez, relacionada com a criação de um número múltiplo dos *buckets* criados nas dimensões (2).

Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-PB (Esquema em Estrela com Partições e *Buckets*).

Tabela 31 Tempos de Processamento com Particionamento por *S_Region* e Bucketing por *Suppkey* (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB
Q1.1	25	22	31	32	44	48	5	6	13	24	36	36
Q1.2	24	23	29	29	42	47	5	6	13	23	34	37
Q1.3	24	23	29	30	43	46	4	6	13	22	35	37
Q2.1	32	25	47	39	531	44	8	4	19	10	59	19
Q2.2	31	21	46	39	531	143	7	4	18	9	51	14
Q2.3	30	21	44	38	531	42	7	4	17	9	49	13
Q3.1	35	23	59	37	651	67	8	4	29	15	81	28
Q3.2	30	29	45	46	677	96	6	7	17	33	51	52
Q3.3	33	33	219	219	665	77	5	7	15	29	43	46
Q3.4	34	32	222	220	675	75	6	7	15	29	43	45
Q4.1	38	30	86	61	226	118	13	7	43	22	119	36
Q4.2	49	34	70	58	141	67	9	5	26	17	69	26
Q4.3	34	34	54	60	116	110	8	11	23	44	63	63
TOTAL	420	349	982	908	4874	982	92	77	262	285	733	452
DIF		-17%		-8%		-80%		-16%		9%		-38%

As vantagens deste cenário são bastante claras no fator de escala maior, dada a diminuição no tempo de processamento de 80% com o Hive e de 38% com o Presto. Neste caso, apesar de o cenário definido apenas com particionamento simples com este atributo, apresentado em subseções anteriores, ter obtido resultados bastante positivos, não são comparáveis com o tipo de diminuição aqui atingido.

No entanto, esta grande vantagem apenas se verifica para o volume de dados maior, sugerindo que em cenários com menor volume de dados o Hive não chega a ativar o *bucket map join* (como explicado anteriormente) uma vez que as diminuições conseguidas nestes fatores de escala (30 e 100) são muito semelhantes aos obtidos no cenário de particionamento simples e verificam-se apenas nas *queries* que têm o atributo do particionamento como filtro. Em cenários de menores quantidades de dados, este tipo de estratégia não traz grandes benefícios pelo que se pode optar pelo particionamento simples como estratégia de organização.

Ainda assim, os resultados obtidos para fatores de escala superiores são, de facto, interessantes. A Figura 35 apresenta com mais detalhe a variação dos tempos de processamento, *query* a *query*, obtidos com o Hive, no fator de escala 300.

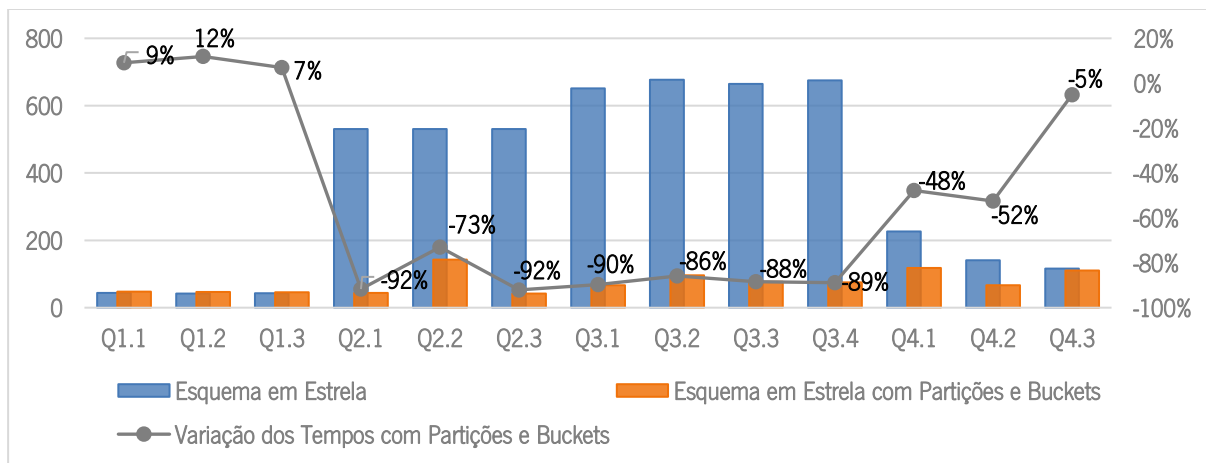


Figura 35 Variação nos tempos de Processamento com Particionamento por "S_Region" e Bucketing por "Supkey" (Hive, EF=300)

As únicas *queries* que não apresentam o *join* com a dimensão *supplier* são as *queries* 1.1, 1.2 e 1.3. Analisando o gráfico, estas são, efetivamente, as únicas *queries* que não sofreram diminuições nos tempos de processamento. Todas as outras apresentam decréscimos significativos, que variam entre o 5% e os 92%.

Com o Presto estas diminuições também ocorrem, apesar de em alguns casos não serem tão acentuadas, tal como representado na Figura 36.

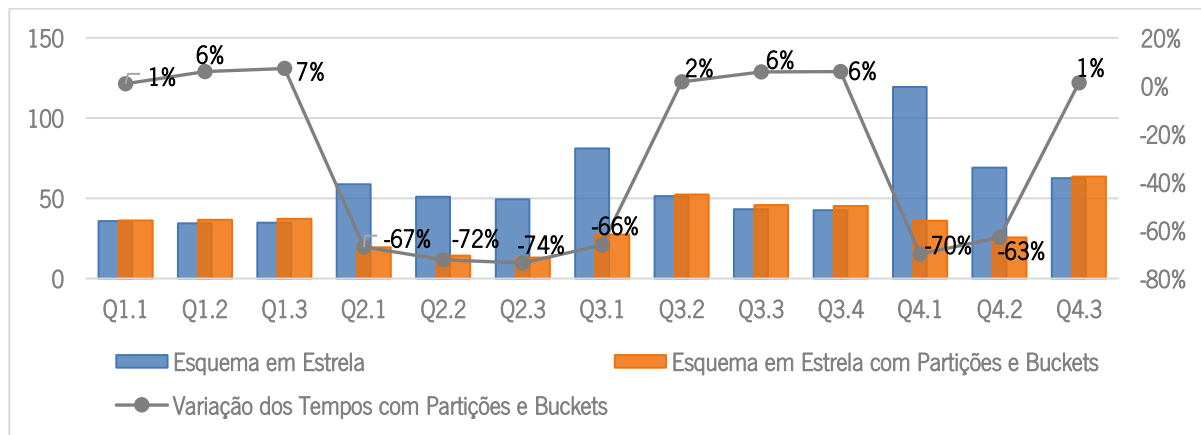


Figura 36 Variação nos Tempos de Processamento com Particionamento por "S_Region" e Bucketing por "Supkey" (Presto, EF=300)

Neste caso, as diminuições são apenas evidentes nas *queries* afetadas pelo particionamento, ou seja, as *queries* que apresentam o atributo *S_Region* nos filtros. Isto comprova o facto de o Presto não reconhecer a estratégia de *buckets* aqui implementada, tal como nos cenários anteriores em que se tentava verificar o impacto dos mesmos nos *joins* de duas tabelas.

Assim sendo, este tipo de estratégia de organização faria sentido em contextos de *Big Data Warehouses* baseados em modelos multidimensionais, utilizando o Hive como sistema de *querying*.

No seguimento destes dois cenários, e considerando que o particionamento múltiplo se mostrou, anteriormente, vantajoso para uma maior rapidez de processamento de dados, decidiu-se replicar o cenário anterior com a adição de um novo nível de particionamento. Assim, as tabelas foram particionadas por *Od_Year* e *S_Region* e criadas com *buckets* por *Suppkey*, estando os tempos de processamento representados na Tabela 32. Importa realçar que este cenário foi replicado para os dois modelos, apesar de só no esquema em estrela o atributo *Suppkey* surgir como forma de avaliar o *join* de duas tabelas. Os testes para a tabela desnormalizada surgem como forma de perceber a combinação do particionamento por atributos de baixa cardinalidade com o *bucketing* por atributos de elevada cardinalidade.

Esta tabela evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: EE (Esquema em Estrela), EE-PB (Esquema em Estrela com Partições e *Buckets*), TD (Tabela Desnormalizada), TD-PB(Tabela Desnormalizada com Partições e *Buckets*).

Importa, ainda, realçar que não foi possível replicar este cenário para a tabela desnormalizada do FE=300 devido a problemas de memória do *cluster* utilizado. Para além disso, como com dois níveis de particionamento se criam ficheiros pequenos por pasta, em todos os fatores de escala, decidiu-se fixar o número de *buckets* em 2 para todos os contextos.

Tabela 32 Tempos de Processamento com Particionamento por *Od_Year+S_Region* e *Bucketing* por *Suppkey* (em segundos)

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB	EE	EE-PB
Q1.1	25	19	31	22	44	27	5	3	13	5	36	12
Q1.2	24	25	29	32	42	51	5	8	13	21	34	65
Q1.3	24	19	29	22	43	25	4	2	13	5	35	12
Q2.1	32	28	47	43	531	50	8	5	19	12	59	37
Q2.2	31	26	46	41	531	160	7	5	18	10	51	27
Q2.3	30	26	44	41	531	45	7	4	17	9	49	26
Q3.1	35	25	59	36	651	66	8	5	29	14	81	44
Q3.2	30	30	45	50	677	92	6	9	17	31	51	100
Q3.3	33	36	219	78	665	78	5	9	15	25	43	86
Q3.4	34	33	222	226	675	81	6	10	15	30	43	91
Q4.1	38	33	86	70	226	127	13	9	43	24	119	65
Q4.2	49	30	70	57	141	60	9	4	26	13	69	25
Q4.3	34	31	54	47	116	72	8	7	23	21	63	60
TOTAL	420	362	982	765	4874	933	92	81	262	220	733	650
DIF		-14%		-22%		-81%		-12%		-16%		-11%

	FE=30		FE=100		FE=300		FE=30		FE=100		FE=300	
	HIVE						PRESTO					
	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB
Q1.1	24	19	29	23			5	2	13	5		
Q1.2	24	22	29	44			5	3	14	8		
Q1.3	23	18	30	25			5	3	14	6		
Q2.1	25	20	36	25			4	2	10	6		
Q2.2	36	22	73	35			4	2	10	5		
Q2.3	25	19	35	24			4	2	10	4		
Q3.1	28	19	40	27			5	2	12	6		
Q3.2	28	23	41	47			5	3	12	11		
Q3.3	25	23	38	45			4	3	9	12		
Q3.4	25	25	38	51			5	4	12	13		
Q4.1	27	20	41	28			6	3	14	7		
Q4.2	29	15	42	22			6	2	14	3		
Q4.3	29	21	42	29			5	2	12	5		
TOTAL	349	265	516	424			63	33	155	90		
DIF		-24%		-18%				-47%		-42%		

Através destes resultados é possível concluir que este é um cenário aparentemente benéfico para os tempos gerais de processamento. Contudo, pode-se repensar este benefício, mais uma vez, considerando os resultados obtidos nos testes realizados apenas com o particionamento com estes dois atributos, também, aqui combinados. No entanto, esta dúvida só surge para conjunto de dados menores, porque, mais uma vez, no fator de escala 300 há uma diminuição significativa no esquema em estrela, com o Hive (81%), o que indica que este sistema ativou mecanismos de otimização, que num caso de *joins* de duas tabelas, terá sido a ativação do *Bucket Map Join* tal como no cenário anterior.

Relativamente aos resultados do Presto, mais uma vez se verifica uma semelhança nos resultados obtidos com os do cenário de particionamento múltiplo sem *bucketing*, sendo na maior parte das vezes ainda menos benéfico do que este.

Como tal, se se decidir por um *Big Data Warehouse* baseado num esquema em estrela, utilizando o Hive como sistema de *querying*, conseguem-se resultados potencialmente benéficos, apesar de que, ainda assim, os tempos gerais de processamento são menores com o modelo desnormalizado e com o Presto como o sistema de *querying*.

Considerando, também, os resultados do *bucketing* por *Od_Year* e *P_Brand* na desnormalização, decidiu-se testar o particionamento por *Od_Year* e o *bucketing* por *P_Brand* para este modelo de dados, estando os resultados apresentados na Tabela 33, que evidencia, a negrito, os melhores tempos de processamento e utiliza a seguinte notação: TD (Tabela Desnormalizada), TD-PB(Tabela Desnormalizada

com Partições e *Buckets*). Importa, ainda, realçar que não foi possível replicar este cenário para a tabela desnormalizada do FE=300 devido a problemas de memória do *cluster* utilizado. Para além disso, definiram-se 6 *buckets* por pasta (partição) para os dois contextos aqui testados.

Tabela 33 Tempos de Processamento com Particionamento por *Od_Year* e Bucketing por *P_Brand* (em segundos)

	FE=30		FE=100		FE=30		FE=100	
	HIVE				PRESTO			
	TD	TD-PB	TD	TD-PB	TD	TD-PB	TD	TD-PB
Q1.1	24	19	29	21	5	2	13	3
Q1.2	24	21	29	22	5	2	14	5
Q1.3	23	20	30	21	5	2	14	4
Q2.1	25	26	36	40	4	5	10	14
Q2.2	36	36	73	68	4	4	10	11
Q2.3	25	23	35	32	4	4	10	9
Q3.1	28	25	40	40	5	5	12	13
Q3.2	28	26	41	39	5	5	12	13
Q3.3	25	22	38	31	4	4	9	10
Q3.4	25	25	38	39	5	4	12	10
Q4.1	27	27	41	43	6	6	14	17
Q4.2	29	22	42	26	6	3	14	5
Q4.3	29	21	42	27	5	3	12	5
TOTAL	349	312	516	449	63	47	155	119
DIF		-10%		-13%		-24%		-23%

Neste caso não se pode afirmar que as diminuições nos tempos de processamento se devem apenas ao particionamento uma vez que não são apenas as *queries* que apresentam este atributo como filtro que apresentam decréscimos.

As *queries* do grupo 1 apresentam diminuições em todas as execuções, no entanto, estas diminuições devem-se à baixa complexidade das próprias *queries* assim como ao facto de só lidarem com filtros temporais pelo que beneficiam apenas do particionamento por ano e do *predicate pushdown*.

Já as *queries* 2.2 e 2.3 sofrem diminuições em algumas das execuções, tendo valores iguais nas restantes. Analisando as *queries* percebe-se que estas apresentam o atributo *P_Brand* nas condições “group by” e “order by” pelo que pode ser a causa desta diminuição. No entanto, a *query* 2.1 também apresenta o atributo nesta condição e, ao contrário destas, não sofre qualquer tipo de diminuição. Analisando as outras condições das duas *queries* iniciais, conclui-se que estas apresentam este atributo não só nas condições “group by” e “order by” mas também na condição “where” o que indica que o facto de os ficheiros estarem segmentados com *buckets* por *P_Brand* facilita a pesquisa por linhas que coincidam com os predicados das *queries*.

Assim sendo, este é um cenário que, dependendo dos filtros que se aplicam mais vezes nos dados, pode ser estudado e aplicado em contextos de *Big Data Warehouses* com tabelas desnormalizadas.

Para concluir os diversos testes realizados aos cenários, apresenta-se a variação na utilização do processador central por *query*, em ambos os modelos de dados, agora com a combinação das duas estratégias de organização. A Figura 37 apresenta esta variação para o fator de escala 100, tendo como exemplo o cenário que combina o particionamento múltiplo (*Od_Year + S_Region*) com *bucketing* pelo atributo *Suppkey*, já que foi o cenário com melhores resultados a nível de tempos de processamento.

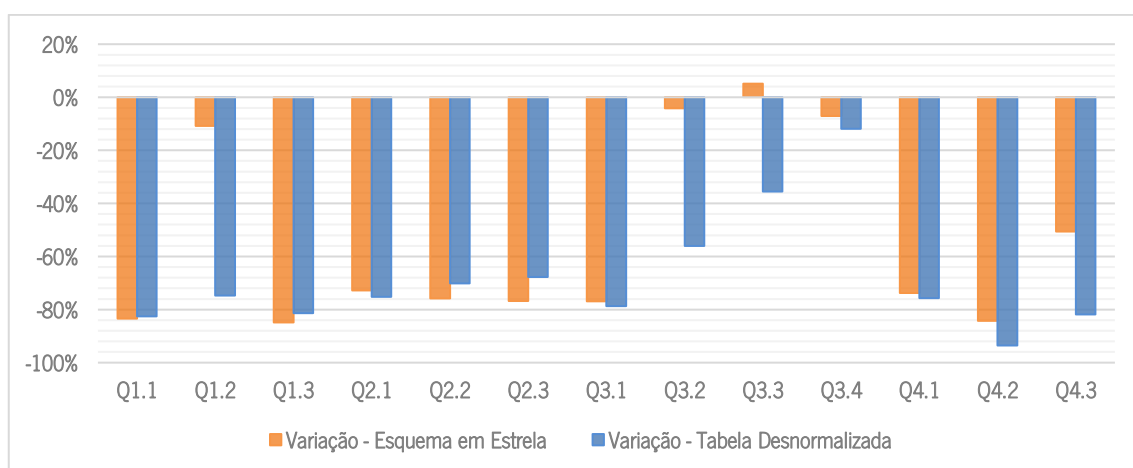


Figura 37 Variação na utilização de CPU com combinação de Particionamento com Bucketing Simples

Como é possível verificar, as diminuições conseguidas para este fator de escala em ambos os modelos de dados são bastante semelhantes às diminuições obtidas com o cenário de particionamento múltiplo por estes atributos sem estratégia de *bucketing* (apresentado anteriormente), em que as *queries* que apresentam os atributos na condição “where” (todas exceto a Q1.2 e a Q3.4) são as que sofrem diminuições mais significativas na utilização de CPU. Isto comprova que em contextos de volumes de dados de tamanho inferior, aplicar estratégias de *bucketing* não faz grande diferença para a estratégia de particionamento múltiplo.

No entanto, considerando que foi com o Hive, no modelo em estrela e com fator de escala 300, que se obteve a maior diminuição de tempo de processamento, por possível ativação de mecanismos de otimização dos *joins*, a Figura 38 apresenta a variação na utilização dos recursos por *query*, medidas com o Hive em vcores-segundos²¹.

²¹ Vcores-segundos = número de vcores (3 por nó do cluster) *os segundos que cada vcore trabalhou

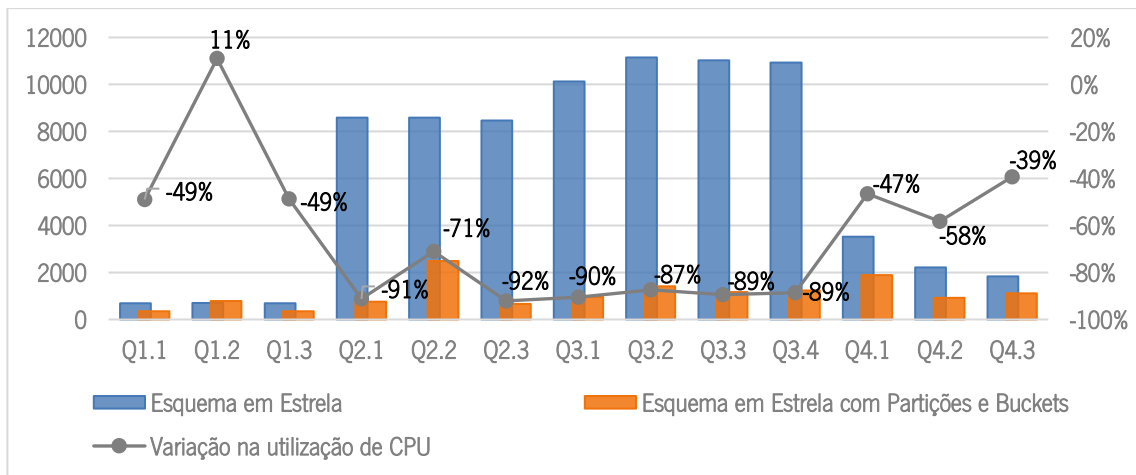


Figura 38 Variação na utilização de CPU com combinação de Particionamento com Bucketing Simples (Hive, FE=300)

Com estes valores aqui apresentados, é possível perceber que, com este contexto, não só se verificam diminuições significativas no tempo de processamento como também no tempo de utilização de CPU. Em média, há uma diminuição de cerca de 65% com a aplicação desta estratégia de organização no fator de escala maior (que é quando o Hive parece ativar mecanismos de otimização de *joins*).

5.3 Síntese de Resultados

Analisando os resultados na generalidade, os melhores resultados para cada fator de escala e para cada configuração serão apresentados ao longo desta secção. Em todos os cenários, o tempo referido é o tempo total necessário para processamento de todas as *queries* que fazem parte do conjunto testado sendo este o total obtido pelo Presto, uma vez que é com esta ferramenta que se obtém, em todos os contextos, o menor tempo de processamento. A Tabela 34 apresenta os menores tempos obtidos (em segundos) para cada fator de escala por modelo de dados, assim como a diminuição conseguida com a tabela desnormalizada em relação ao esquema em estrela.

Tabela 34 Melhores resultados por FE - Cenário A

CENÁRIO A: MODELAÇÃO			
FATOR DE ESCALA	EE	TD	DIFERENÇA
30	92	63	-32%
100	262	155	-41%
300	733	472	-36%

Considerando os resultados gerais obtidos para o cenário A, e considerando tudo o que foi apresentado na subsecção anterior referente à modelação dos dados, é possível concluir que a total desnормalização é mais vantajosa para contextos de *Big Data*.

Apesar do esquema em estrela apresentar benefícios em algumas consultas específicas assim como benefícios a nível de memória, a tabela desnормalizada implica, na generalidade dos casos, tempos de processamento inferiores assim como uma diminuição significativa na utilização do processador.

Considerando que o esquema em estrela implica a execução de operações de *join*, operações normalmente “pesadas” também em contextos tradicionais, e que, dependendo da sua complexidade, envolvem esforço extra de processamento, este tipo de esquema em contextos de *Big Data* implica, ainda, a criação de estratégias de otimização destas operações, tal como verificado nos resultados obtidos para os dois tipos de estratégia aplicados (*distributed join* e *broadcast joins*). Sendo o *broadcast join* o tipo de *join* com melhores resultados, a sua aplicação, por si só, apresenta necessidades de memória específicas que não se verificam para contextos de desnормalização, podendo estes requisitos ser considerados como mais uma desvantagem do esquema em estrela.

Por outro lado, a modelação em estrela implica menos utilização de memória, ocupando, na generalidade dos casos, três vezes menos espaço do que a tabela desnормalizada. No entanto, considerando que o espaço de armazenamento, na atualidade, é cada vez mais barato, tendo em conta, por exemplo, a evolução da *cloud*, esta é a única desvantagem apresentada pela tabela desnормalizada que pode ser ultrapassada recorrendo a estes serviços ou opções semelhantes, compensando as desvantagens apresentadas pela modelação em estrela.

Portanto, considerando as questões de investigação relacionadas com a modelação dos dados a que esta dissertação se propôs responder, é possível concluir que, apesar de ser possível a implementação de um *Big Data Warehouse* concretizado em Hive baseado num esquema em estrela, a verdade é que uma tabela totalmente desnормalizada parece ser a mais eficiente para estes contextos, tanto a nível de processamento como de utilização de recursos.

Relativamente à organização dos dados, particularmente às estratégias de particionamento, a Tabela 35 apresenta os melhores tempos gerais de processamento obtidos para cada configuração testada, por fator de escala. Estes resultados são comparados com os melhores resultados obtidos no cenário A, apresentando a diferença obtida com a aplicação de técnicas de particionamento. Importa

realçar que, mais uma vez, todos os resultados apresentados correspondem aos resultados obtidos com o Presto já que foi a ferramenta que obteve os melhores tempos de processamento em todos os contextos.

Tabela 35 Melhores resultados por configuração de Particionamento e por FE - Cenário B1

FE	MODELO	CENÁRIO A	CENÁRIO B1: PARTICIONAMENTO						
			<i>Od_Year</i>	<i>S_Region</i>	<i>P_MFGR</i>	<i>Od_Year, Monthnuminyear</i>	<i>Od_Year, S_Region</i>	<i>S_Region, S_Nation, S_City</i>	<i>Od_Year, S_Region, P_MFGR</i>
30	EE	92	87 -5%	66 -28%	90 -2%	143 55%	63 -32%	70 -24%	91 -1%
	TD	63	46 -26%	39 -38%	49 -22%	330 428%	43 -32%	-	-
100	EE	262	239 -9%	166 -37%	-	-	149 -43%	193 -26%	259 -1%
	TD	155	97 -38%	79 -49%	-	-	71 -54%	-	-
300	EE	733	645 -12%	428 -42%	-	-	399 -46%	-	-
	TD	472	280 -41%	247 -48%	-	-	299 -37%	-	-

Antes de elaborar uma reflexão dos resultados obtidos, importa realçar que uma boa estratégia de organização dos dados no Hive depende do contexto analítico dos dados, do tipo de filtros que são utilizados na maior parte das vezes assim como o tipo de refrescamento que se realiza sobre os dados. Com esta síntese pretende-se, apenas, realçar algumas boas práticas associadas ao armazenamento dos dados de forma a conseguir obter respostas atempadas que suportem o processo de tomada de decisão.

Analisando os melhores resultados apresentados na Tabela 35, é possível verificar que existem vantagens evidentes no particionamento de dados, uma vez que se conseguem diminuir os tempos de processamento em 6 das 7 configurações aqui testadas. Na melhor execução do fator de escala maior, com cerca de 1,8 biliões de registos, o tempo total de processamento das 13 queries não ultrapassa os 247 segundos, o que equivale a uma média de 19 segundos por *query*.

Considerando as configurações aplicadas, e tendo em conta os resultados obtidos, é possível concluir que o particionamento deve considerar atributos que são utilizados muitas vezes nos filtros aplicados nas consultas aos dados, isto é, nas condições “*where*” das *queries* que se pretendem executar sobre os dados.

Os particionamentos temporais e espaciais poderão, também, ser das melhores estratégias de particionamento a implementar, estando os resultados aqui obtidos alinhados com os estudos de vários autores referidos ao longo deste documento. Efetivamente, e considerando a maioria dos contextos organizacionais reais, facilmente se percebe que as consultas executadas sobre os dados implicam, na generalidade dos casos, filtros por ano ou por região/país/cidade, pelo que a tendência é o particionamento por atributos deste género.

Para além disso, considerando os requisitos de desempenho do HDFS, uma boa estratégia de organização deve ter em consideração o tamanho dos ficheiros pelos quais os dados são distribuídos. Quer isto dizer que é preciso ter cuidado com o particionamento excessivo, que para além da complexidade de processamento associada à existência dos múltiplos níveis de pastas, importa considerar, que quanto maior for o número de subdiretorias criadas, menos informação será armazenada por pasta pelo que a tendência é existirem poucos ficheiros demasiado pequenos, o que pode comprometer o desempenho dos sistemas utilizados.

Considerando novamente contextos reais que não foram aqui testados, mas que pode influenciar o desempenho do *Big Data Warehouse*, o refrescamento dos dados é outra operação importante e que pode implicar complexidade acrescida na organização dos dados. Assim, a estratégia de particionamento deve ponderar o carregamento de novos dados para o *Big Data Warehouse*, em que se forem, por exemplo, atualizados por mês e/ou por departamento, estes podem ser atributos com potencial para conseguir uma boa estratégia de organização, não envolvendo complexidade acrescida na criação de novas partições ou na atualização da partição correspondente.

Portanto, considerando os resultados obtidos e tudo o que foi sendo realçado, é possível perceber o benefício do particionamento em qualquer modelo de dados e em qualquer fator de escala, pelo que considerando os filtros aplicados, é possível obter na maioria dos casos otimizações no armazenamento e, conseqüentemente, no processamento dos dados.

Relativamente às estratégias de organização que recorrem ao *bucketing*, a Tabela 36 apresenta os melhores tempos gerais de processamento obtidos para cada configuração testada, por fator de escala e comparando com os melhores resultados obtidos no cenário A. Importa realçar que, mais uma vez e

pelas mesmas razões, todos os resultados apresentados correspondem aos resultados obtidos com o Presto.

Tabela 36 Melhores resultados por configuração de Bucketing e por FE - Cenário B2

FE	MODELO	CENÁRIO A	CENÁRIO B2: BUCKETING						
			<i>Custkey</i>	<i>Orderkey</i>	<i>Suppkey</i>	<i>Orderkey (Sorted by Orderdate)</i>	<i>Orderdate, Custkey, Suppkey, Partkey</i>	<i>Od_Year (sorted by P_Brand1)</i>	<i>P_Brand (sorted by Od_Year)</i>
30	EE	92	101 9%	133 44%	120 30%	115 25%	121 32%	- -	- -
	TD	63	68 9%	71 14%	- -	- -	- -	41 -35%	42 -34%
100	EE	262	- -	305 16%	321 22%	- -	305 16%	- -	- -
	TD	155	- -	178 15%	- -	- -	- -	103 -34%	- -
300	EE	733	- -	- -	768 5%	- -	876 19%	- -	- -
	TD	472							

Analisando o panorama geral dos melhores resultados, este não se mostra a favor da organização dos dados recorrendo a *buckets*. Analisando os tempos de processamento obtidos, é possível perceber que só em casos muito específicos poderá existir alguma vantagem neste tipo de organização, no entanto a sua definição e a criação de uma boa estratégia é um processo mais complexo.

Ao longo dos testes que foram realizados recorrendo a *bucketing*, comprovou-se que a sua definição depende do tipo de modelo aplicado sobre os dados. O único cenário testado para os dois modelos de dados que se baseava na definição de *buckets* por atributos com elevada cardinalidade, nomeadamente IDs, não se mostrou benéfico em nenhum dos contextos pelo que não se considera, para este conjunto de dados e para estas *queries*, uma boa estratégia de organização.

Relativamente à modelação em estrela recorrendo a *bucketing*, comprovou-se a ineficiência da aplicação de *bucketing* múltiplo, uma vez que se trata de uma estratégia que não é vantajosa para nenhuma das ferramentas de *querying* utilizadas. No entanto, no contexto do *bucketing* simples, existiram cenários em que se verificaram algumas vantagens, nomeadamente com o Hive e para o fator de escala maior. Assim, o único contexto em que se verificou algum tipo de vantagens na definição de

buckets trata-se do contexto do *bucketing* de duas tabelas pelo mesmo atributo, atributo esse utilizado na operação de *join* dessas tabelas. Este contexto obteve, no Hive, diminuições de cerca de 80% comparando com um cenário sem qualquer tipo de estratégia, o que indica que o Hive é a melhor ferramenta para lidar com este tipo de estratégia. Com o Presto, quando se verificaram diminuições, na maioria das vezes estavam relacionadas com outro tipo de estratégia, e noutros casos verificaram-se aumentos nos tempos de processamento. Ainda assim, o melhor resultado a nível de processamento continua a ser com o Presto, daí este contexto aparecer na Tabela 36 sem qualquer vantagem associada, apesar do tempo de processamento ser menor quando comparado com o tempo obtido com o Hive.

Relativamente às tabelas desnormalizadas, os únicos cenários em que se verificaram alguns benefícios são os cenários em que se definem *buckets* por atributos que aparecem várias vezes na condição “*group by*” combinando com a ordenação dos atributos por um atributo com relevância para as *queries* executadas, sendo que a maior parte aplica sempre uma condição “*order by*”. Assim, apesar de num dos cenários ter sido utilizado um atributo com baixa cardinalidade (*Od_Year*), como a maioria das *queries* estavam orientadas para condições temporais, em todos os cenários concluídos verificaram-se diminuições significativas pelo que poderá ser uma estratégia a considerar num contexto em que não se pretende particionar.

Ainda assim, surge também a questão de não serem permitidas alterações ao número de *buckets* depois da tabela ter sido criada, pelo que se cria aqui um problema relacionado com o refrescamento dos dados. Assim, sempre que surgir a necessidade de adicionar novos registos à tabela terá que se criar uma nova tabela com a estratégia de *bucketing* repensada, se necessário, e fazer o carregamento dos dados da tabela antiga e dos dados novos. Esta complexidade adicional mostra, assim, mais uma desvantagem deste tipo de estratégia de segmentação dos dados.

Relativamente à combinação destas duas estratégias de organização, a Tabela 37 apresenta os melhores tempos gerais de processamento obtidos para cada combinação de particionamento e *bucketing* testada, por fator de escala e comparando com o cenário sem particionamento ou *bucketing*. Mais uma vez, os resultados apresentados são os resultados obtidos com o Presto e algumas das configurações não foram testadas para alguns fatores de escala devido a problemas de memória e de configurações do *cluster* utilizado.

Analisando a generalidade dos resultados e diminuições/aumentos conseguidos e apresentados na tabela anterior é possível perceber que este tipo de estratégia traz vantagens para o processamento dos dados principalmente para as tabelas desnormalizadas.

Tabela 37 Melhores resultados por combinação de Particionamento e Bucketing e por FE - Cenário B3







FE	MODELO	CENÁRIO A	CENÁRIO B3: PARTICIONAMENTO + BUCKETING			
			<i>P = Od_Year</i> <i>B = Orderkey</i>	<i>P = S_Region</i> <i>B = Suppkey</i>	<i>P = Od_Year</i> <i>B = P_Brand</i>	<i>P = Od_Year,</i> <i>S_Region</i> <i>B = Suppkey</i>
30	EE	92	100 8%	77 -16%	- -	81 -12%
	TD	63	46 -26%	- -	47 -24%	33 -47%
100	EE	262	256 -2%	285 9%	- -	220 -16%
	TD	155	129 -17%	- -	119 -23%	90 -42%
300	EE	733	835 14%	452 -38%	- -	650 -11%
	TD	472	- -	- -	- -	- -



No entanto, na maioria dos casos os resultados positivos obtidos estão mais relacionados com o reconhecimento do particionamento e não tanto com o reconhecimento do *bucketing*, uma vez que se obtêm resultados similares, nas mesmas *queries* e nas mesmas proporções, que os resultados obtidos nos cenários em apenas se utilizava como estratégia o particionamento.

O único cenário em que se verificam vantagens da combinação, sem ser apenas por mérito do particionamento, é no modelo em estrela e com o Hive e, mais uma vez, com a estratégia de *bucketing* que potencia o *join* de duas tabelas com *buckets* pela mesma coluna. No entanto, continua a não ser o melhor tempo de processamento conseguido, uma vez que o Hive tem sempre tempos de processamento superiores aos obtidos com o Presto.

Realizada a síntese dos resultados obtidos para o cenário B, a Tabela 38 apresenta a melhor configuração (com o menor tempo de processamento total conseguido) por fator de escala.

Tabela 38 Melhor resultado por FE - Cenário B

CENÁRIO B: ORGANIZAÇÃO							
FATOR DE ESCALA	PARTIÇÕES		BUCKETS		PARTIÇÕES + BUCKETS		CONFIGURAÇÃO (melhor cenário)
	EE	TD	EE	TD	EE	TD	
30						 33s 	Particionamento Múltiplo por <i>Od_Year</i> + <i>S_Region</i>
100						 71s 	Particionamento Múltiplo por <i>Od_Year</i> + <i>S_Region</i>
300		 247s 					Particionamento Simples por <i>S_Region</i>

 Menor tempo de processamento  Melhor cenário

Tal como se foi verificando, o particionamento, por si só, consegue solucionar a maior parte dos problemas de organização, uma vez que uma tabela adequadamente particionada potencia melhores tempos de processamento e menos recursos consumidos por *query* executada sobre os dados. Assim, e considerando as questões de investigação relacionadas com a organização dos dados em *Big Data Warehouses* concretizados em Hive, a que esta dissertação se propôs responder (“Há vantagens na utilização de partições e/ou *buckets*? Estas técnicas têm algum impacto nos tempos de processamento?”), é possível concluir que as partições são, efetivamente, vantajosas e têm um impacto positivo significativo nos tempos de processamento e na utilização do processador, ao contrário dos *buckets* que, para além de não demonstrarem vantagens na maioria dos casos testados, acrescentam complexidade na criação da tabela (definição do atributo a utilizar como *buckets* e o número de *buckets* a criar) e no refrescamento da mesma. A considerar os *buckets* num *Big Data Warehouse*, seria num contexto de modelo em estrela ou então com atributos que sejam usados com frequência nas condições “*group by*” e “*order by*” das *queries* a executar e com a aplicação da técnica “*sorted by*” para potenciar ainda mais os resultados positivos.

Apesar de não ser o foco deste trabalho, importa ainda realçar o desempenho do Presto como sistema de *querying*, já que apresentou os melhores tempos de processamento em todos os contextos aqui estudados. No entanto, destaca-se ainda o potencial do Hive em contextos de *bucketing* de duas tabelas pelo atributo utilizado no *join*, onde o Presto demonstrou não reconhecer nem possuir qualquer

tipo de mecanismo de otimização para lidar com esta estratégia de organização dos dados num contexto de modelação em estrela. Ainda assim, os tempos gerais obtidos pelo Presto, mesmo nestes cenários, são consideravelmente inferiores pelo que, só em contextos de aumento significativo do volume de dados se poderia afirmar se este padrão se manteria e se o Presto continuaria a ter perdas de desempenho significativas e, até mesmo, ser ultrapassado pelo Hive.

De todas as conclusões aqui avaliadas, é possível inferir algumas boas práticas para a modelação e organização dos dados:

1. Utilização dos modelos de dados baseados em tabelas desnormalizadas;
2. Fazer um estudo da cardinalidade e distribuição dos atributos que compõem o conjunto de dados para identificar quais os atributos mais apropriados para particionamento (atributos de baixa cardinalidade e com distribuição uniforme) e/ou para *bucketing* (atributos de elevada cardinalidade);
3. Implementação de técnicas de particionamento:
 - a. Conhecendo as *queries* à partida, particionar pelos atributos que aparecem com frequência nos filtros aplicados sobre os dados;
 - b. Prestar atenção ao particionamento excessivo (evitar a criação de um grande número de subdiretórias);
 - c. Optar por particionamento temporal, geográfico ou departamental, dependendo dos filtros executados em contexto real e da forma como se atualizam os dados.
4. Evitar o uso de técnicas de *bucketing*; A usar técnicas de *bucketing*:
 - a. Definir um número de *buckets* adequado ao tamanho do *dataset*, de forma a evitar criar poucos ficheiros e de pequena dimensão;
 - b. Optar por atributos que apareçam com frequência nas condições relacionadas com a estrutura dos dados ("*group by*" ou "*order by*") das *queries* a executar;
5. Estudar a implementação da combinação das duas técnicas (particionamento e *bucketing*) dependendo do modelo de dados e das *queries* a executar sobre os dados;
6. Se a velocidade de processamento for crucial, utilizar o Presto, ou semelhante, como ferramenta de *querying*.

Através das experiências realizadas, ficou, também, claro que há um conjunto de otimizações que se podem aplicar:

1. Aplicar sobre todas as tabelas criadas (sem *buckets*) a função “*alter table concatenate*”, de forma a otimizar a distribuição dos dados pelos ficheiros (transformar muitos ficheiros de pequena dimensão em poucos ficheiros de maior dimensão) e conseqüentemente o desempenho do HDFS;
2. Aplicar sobre todas as tabelas criadas a função “*analyze table compute statistics*” e “*analyze table compute statistics for columns*”, de forma a manter os metadados e estatísticas do Hive atualizados e, otimizar, assim as consultas sobre os dados;
3. Num contexto de modelo em estrela, forçar a utilização do *broadcast join* no Presto, uma vez que os resultados obtidos são melhores do que os obtidos com o *distributed join* (o *join* definido por defeito por este sistema).

6. CONCLUSÕES

Tendo esta dissertação como tema a organização e o processamento de dados em *Big Data Warehouses* baseados em Hive, foram propostos um conjunto de objetivos que foram sendo explorados e trabalhados ao longo deste documento.

Este documento começa, então, por expor o enquadramento conceptual de *Big Data Warehouses* assim como a identificação dos principais trabalhos já realizados na área. Assim, numa primeira fase fez-se o levantamento dos principais conceitos associados a *Big Data*, apresentando o conceito, as principais características, potenciais oportunidades, problemas e desafios e a forma como são processados os dados, neste novo contexto. De seguida foi exposta uma comparação entre as abordagens de armazenamento tradicionais com as abordagens emergentes para os contextos de grandes volumes de dados, tendo sido realizada a comparação entre bases de dados relacionais com bases de dados NoSQL, a apresentação do contexto de *Data Warehouses* tradicionais e a sua evolução para *Big Data Warehouses*, e a forma como os modelos de dados vão evoluindo e sendo alinhados com estas novas estratégias de armazenamento. Por fim, apresentaram-se alguns exemplos de trabalhos que estudaram algumas das transformações necessárias para o armazenamento e processamento dos dados num contexto de *Big Data*. Com este capítulo foi possível perceber que *Big Data* traz novos desafios não só a nível de processamento de grandes volumes de dados, como também a nível de armazenamento, pelo que existe, ainda, um grande caminho a percorrer no sentido de perceber a forma como a organização dos dados pode influenciar a forma como os dados são processados.

Posteriormente, foi feito o enquadramento tecnológico desta dissertação, que se focou na compreensão da ferramenta base para todo o trabalho, o Hive. Assim, foi possível perceber o seu modo de funcionamento assim como as características de particionamento e de *bucketing* que este fornece e que demonstram ser uma potencial ajuda na diminuição dos tempos de processamento de dados, quando implementados de forma adequada.

Realizados os enquadramentos conceptuais e tecnológicos, começou-se a preparar o ambiente de testes e os dados a utilizar. Como tal, fez-se uma caracterização da infraestrutura física que suportou os testes executados e realizou-se uma análise do conjunto de dados utilizado, em particular um estudo sobre a cardinalidade e distribuição dos atributos que o compõem, estudo esse importante para a definição dos atributos a utilizar como suporte à definição das estratégias de organização aplicadas no capítulo seguinte. Todas estas análises serviram para a criação de um protocolo de testes a executar,

dividido em duas áreas principais voltadas para a modelação e para a organização dos dados, respetivamente.

Por fim, elaboraram-se um conjunto de *benchmarks* com base no protocolo de testes criado, tendo sido apresentados os principais resultados obtidos que contam com as variações nos tempos de processamento com a aplicação de diferentes modelos de dados e de diferentes estratégias de organização e com a variação na utilização dos recursos que utilizam no seu processamento. Por fim, foi elaborada uma discussão e síntese desses mesmos resultados. Assim, os resultados obtidos no primeiro cenário reforçam as vantagens associadas à implementação de *Big Data Warehouses* baseados em tabelas desnormalizadas, por apresentarem vantagens em todos os contextos testados. Os resultados obtidos no segundo cenário de teste vêm reforçar o potencial benefício da segmentação dos dados, uma vez que, dependendo da consulta executada, estas técnicas podem diminuir de forma significativa o tempo de processamento. No entanto, as grandes vantagens são verificadas mais para técnicas de particionamento do que propriamente nas técnicas de *bucketing*, um processo mais complexo que depende de muitas variáveis e que cria restrições em atualizações futuras das tabelas. Conclui-se assim que, o particionamento tem, efetivamente, um impacto positivo nos tempos de processamento de *Big Data Warehouses* sendo uma técnica, que definida adequadamente e alinhada de acordo com os filtros executados sobre os dados, traz grandes benefício tanto a nível de armazenamento (melhor organização e distribuição dos dados) como no processamento (tempos de resposta inferiores suportando de forma mais rápida o processo de tomada de decisão, assim como nos recursos utilizados (diminuição na utilização de CPU). Por outro lado, a implementação de técnicas de *bucketing* não demonstrou qualquer benefício para o armazenamento e processamento de dados.

6.1 Trabalho Realizado

Esta dissertação teve como finalidade estudar o papel da organização dos dados nos tempos de processamento de *Big Data Warehouses*, nomeadamente o impacto da modelação e da definição de partições e *buckets* do Hive, de forma a definir um conjunto de recomendações que ajudem no processo de modelação dos dados. Assim, os objetivos que este trabalho se propôs atingir, foram concretizados na sua totalidade, tendo sido:

- Realizada uma revisão de literatura que discutiu os principais conceitos, problemas e desafios relacionados com *Big Data Warehouses* e a sua implementação;
- Compreendida a forma de organização dos dados num *Big Data Warehouse* concretizado em Hive, tanto a nível de modelação como de estratégias de segmentação;

- Compreendido o impacto da definição de partições e *buckets* em Hive nos tempos de processamento de um conjunto de *queries* previamente definidas;
- Propostas um conjunto de recomendações para a definição do modelo de dados e organização dos dados em Hive, que podem ser utilizadas na modelação destes repositórios;
- Identificadas e testadas as boas práticas propostas num caso de demonstração (SSB) através da aplicação de vários cenários com diferentes estratégias de organização dos dados.

Assim, com todos os testes executados e após uma análise extensa dos resultados obtidos para cada cenário e para cada estratégia aplicada sobre o armazenamento dos dados foi, ainda, possível dar resposta às quatro questões a que esta dissertação se propôs explicar:

- Um *Data Warehouse* multidimensional é um modelo de *design* adequado para *Data Warehousing* concretizado em Hive? É possível afirmar que apesar de ser possível a implementação deste tipo de modelo de dados multidimensional em contextos de *Big Data Warehousing*, a verdade é que esta não se mostra a configuração mais vantajosa para o processo de tomada de decisão, já que obtém tempos de processamento consideravelmente superiores em todos os contextos estudados (mesmo com as diferentes estratégias de organização aplicadas);
- Os esquemas multidimensionais são mais eficientes do que as tabelas totalmente desnormalizadas em contextos de *Big Data*? No seguimento da questão anterior, e considerando as diminuições conseguidas com a tabela desnormalizada, tanto a nível de tempo de processamento como utilização de CPU, é possível concluir que o esquema em estrela mostra-se menos eficiente do que as tabelas desnormalizadas na generalidade dos casos. O esquema em estrela mostra-se mais eficiente do que a tabela desnormalizada a nível de memória ocupada, ocupando cerca de três vezes menos memória, no entanto essa é uma desvantagem da tabela desnormalizada que pode ser facilmente ultrapassada considerando a evolução dos serviços de armazenamento atualmente existentes.
- Há vantagens na utilização de partições e/ou *buckets*? Estas técnicas têm algum impacto nos tempos de processamento? Considerando todos os resultados previamente apresentados ficaram evidentes as vantagens associadas à utilização de técnicas de particionamento, uma vez que provocam diminuições consideráveis ao nível de processamento dos dados. Já a utilização de técnicas de *bucketing* fica aquém das expectativas uma vez que foram raros os cenários em que demonstraram algum tipo de

benefício para as variáveis aqui estudadas. Ainda assim, a sua utilização em contextos muito específicos, com um estudo aprofundado de como os definir, e até mesmo combinando com técnicas de particionamento, poderá assegurar algumas vantagens no armazenamento e processamento de dados.

Pode-se então afirmar que o resultado final desta dissertação oferece um conjunto de boas práticas possível de ser adaptado noutros contextos, pelo que se pensa ter dado um contributo importante para a área da modelação e organização dos dados em contextos de *Big Data Warehousing*.

Para além disso, importa, ainda, referir que uma parte deste trabalho deu, também, origem a uma publicação científica, já publicada e apresentada na “European, Mediterranean and Middle Eastern Conference on Information Systems” (EMCIS 2017), para além de outros contributos adicionais já referidos na secção 1.4.

6.2 Dificuldades e Limitações

Dada a juventude desta área de investigação e a velocidade a que as ferramentas e as técnicas vão evoluindo, surgiram algumas dificuldades na concretização dos objetivos a que este trabalho se propôs atingir.

Uma das principais dificuldades surge com a necessidade de um conhecimento muito aprofundado das várias ferramentas necessárias e integradas no *cluster* utilizado, as quais se devem conhecer de forma a otimizar a integração com as ferramentas principais e necessárias à dissertação. Assim, numa primeira fase certas técnicas de otimização não foram consideradas por desconhecer a sua importância pelo que, aquando do conhecimento delas, rapidamente foram introduzidas de forma a obter um conjunto de resultados o mais correto e confiável possível.

Uma outra dificuldade surge na implementação de técnicas de *bucketing*, na maioria das vezes desaconselhada pela comunidade científica e profissionais da área. Apesar de ser um dos grandes objetivos desta dissertação, a verdade é que para chegar a um conjunto de boas práticas para a sua definição em vários cenários não foi um caminho fácil devido, tanto à falta de informação sobre este tipo de técnica, como à falta de suporte científico, assim como o desaconselhamento da sua utilização por parte de vários autores ou profissionais mais práticos da área.

A principal limitação deve-se à infraestrutura utilizada, que apesar de já nos permitir chegar a um volume de dados considerável, a verdade é que dificultou, muitas vezes, a criação de alguns cenários por falta de memória ou por não ter capacidade para lidar com a segmentação que era solicitada.

Uma outra limitação está relacionada com o tempo necessário para correr todos os contextos aqui testados, uma vez que foram considerados muitos cenários por fator de escala, em que para cada cenário as 13 *queries* eram executadas quatro vezes (para se obter a média), para dois sistemas SQL-on-Hadoop (Hive e Presto) e, no caso do modelo em estrela, para dois tipos de estratégia de *joins* diferentes. Para além disso, e verificando-se mais para contextos de grandes volumes de dados, a criação e carregamento dos dados para a tabela tornava-se, em muitos dos casos, um processo complexo que necessitava, também, de muito tempo e de muitos recursos do *cluster*.

No entanto, todas as dificuldades e limitações foram enfrentadas e ultrapassadas, pelo que na generalidade, o trabalho correu dentro do esperado e todos os objetivos foram cumpridos.

6.3 Investigação Futura

Apesar de se terem atingido todos os objetivos, existem várias vertentes que não foram consideradas e que poderão complementar o trabalho aqui proposto.

Um possível trabalho futuro, interligado com os resultados desta dissertação, seria testar as boas práticas aqui propostas, noutros cenários inovadores, em diferentes contextos e com os mais diversos *datasets* de forma a inferir, com mais certezas, regras mais específicas para a organização e modelação dos dados.

Para além disso, seria importante testar os contextos que não foram implementados neste trabalho como consequência das capacidades e configurações do *cluster* utilizado ou pela falta de tempo para considerar todas as combinações possíveis. Assim, seria importante testar a implementação de técnicas de *bucketing* na tabela desnormalizada do fator de escala maior e, também, a implementação de técnicas de *bucketing* com atributos descritivos de elevada cardinalidade em vários contextos.

Considerando, ainda, a desvantagem aqui apresentada para as tabelas desnormalizadas, relacionada com a memória utilizada, seria interessante aprofundar a análise do real impacto desta em contextos de maior volume de dados e identificar técnicas de mitigação para esta problemática.

Para além disso, e tendo em conta a sua vantagem em contextos relacionais, seria interessante seguir uma linha de investigação relacionada com a criação de um conjunto de *guidelines* estruturadas para a criação automática de vistas materializadas em contextos de *Big Data Warehousing*, e avaliar o seu impacto a nível de processamento e de utilização de recursos.

REFERÊNCIAS BIBLIOGRÁFICAS

- Alekseev, A. A., Osipova, V. V., Ivanov, M. A., Klimentov, A., Grigorieva, N. V., & Nalamwar, H. S. (2016). Efficient Data Management Tools for the Heterogeneous Big Data Warehouse. *Physics of Particles and Nuclei Letters*, 13(5), 689–692. <https://doi.org/10.1134/S1547477116050022>
- Apache. (2014). Apache Hadoop. Retrieved from <http://hadoop.apache.org/>
- Bhardwaj, A., Vanraj, Kumar, A., Narayan, Y., & Kumar, P. (2015). Big Data Emerging Technologies: A Case Study with Analyzing Twitter Data using Apache Hive. In *Recent Advances in Engineering & Computational Sciences (RAECS), 2nd International Conference on*. IEEE. <https://doi.org/10.1109/RAECS.2015.7453400>
- Bhogal, J., & Choksi, I. (2015). Handling Big Data Using NoSQL. In *Advanced Information Networking and Applications Workshops (WAINA), 29th International Conference on* (pp. 393–398). IEEE. <https://doi.org/10.1109/WAINA.2015.19>
- Capriolo, E., Wampler, D., & Rutherglen, J. (2012). *Programming Hive*. O'Reilly Media, Inc.
- Cassavia, N., Dicosta, P., Masciari, E., & Saccà, D. (2014). Data Preparation for Tourist Data Big Data Warehousing. In *Proceedings of 3rd International Conference on Data Management Technologies and Applications (DATA)* (pp. 419–426). SciTePress. <https://doi.org/10.5220/0005144004190426>
- Chavalier, M., El Malki, M., Kopliku, A., Teste, O., & Tournier, R. (2016). Document-Oriented Data Warehouses: Models and Extended Cuboids. In *10th International Conference on Research Challenges in Information Science (RCIS)* (pp. 1–11). IEEE. <https://doi.org/10.1109/RCIS.2016.7549351>
- Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., & Zhou, X. (2013). Big data challenge: A data management perspective. *Frontiers of Computer Science*, 7(2), 157–164. <https://doi.org/10.1007/s11704-013-3903-7>
- 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>
- Chevalier, M., El Malki, M., Kopliku, A., Teste, O., & Tournier, R. (2016). Document-oriented Models for Data Warehouses - NoSQL Document-oriented for Data Warehouses. In *Proceedings of the 18th International Conference on Enterprise Information Systems* (Vol. 1, pp. 142–149). SCITEPRESS - Science and Technology Publications. <https://doi.org/10.5220/0005830801420149>
- Costa, E., Costa, C., & Santos, M. Y. (2017). Efficient Big Data Modelling and Organization for Hadoop

- Hive-Based Data Warehouses. In M. Themistocleous & V. Morabito (Eds.), *14th European, Mediterranean, and Middle Eastern Conference (EMCIS)* (pp. 3–16). Coimbra: Springer International Publishing. https://doi.org/10.1007/978-3-319-65930-5_1
- De Mauro, A., Greco, M., & Grimaldi, M. (2015). What is Big Data? A Consensual Definition and a Review of Key Research Topics. In *AIP Conference Proceedings* (Vol. 1644, pp. 97–104). AIP Publishing. <https://doi.org/10.1063/1.4907823>
- Dehdouh, K. (2016). Building OLAP Cubes from Columnar NoSQL Data Warehouses. In *Bellatreche L., Pastor Ó., Almendros Jiménez J., Ait-Ameur Y. (eds) Model and Data Engineering. MEDI 2016* (Vol. 9893 LNCS, pp. 166–179). Springer, Cham. https://doi.org/10.1007/978-3-319-45547-1_14
- Dehdouh, K., Bentayeb, F., Boussaid, O., & Kabachi, N. (2015). Using the column oriented NoSQL model for implementing big data warehouses. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (pp. 469–475).
- Dere, J. (2017). Apache Hive. Retrieved from <https://cwiki.apache.org/confluence/display/Hive/Home>
- Di Tria, F., Lefons, E., & Tangorra, F. (2014). Design Process for Big Data Warehouses. In *Data Science and Advanced Analytics (DSAA), 2014 International Conference on* (pp. 512–518). IEEE. <https://doi.org/10.1109/DSAA.2014.7058120>
- Du, D. (2015). *Apache Hive Essentials*. Packt Publishing Ltd.
- Durham, E.-E. A., Rosen, A., & Harrison, R. W. (2014). A Model Architecture for Big Data applications using Relational Databases. In *Big Data (Big Data), 2014 IEEE International Conference on* (pp. 9–16). [https://doi.org/978-1-4799-5666-1/14/\\$31.00](https://doi.org/978-1-4799-5666-1/14/$31.00)
- Elmasri, R., & Navathe, S. B. (2003). *Fundamentals of Database Systems, Fourth Edition*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Fan, J., Han, F., & Liu, H. (2014). Challenges of Big Data Analysis. *National Science Review*, *1*(2), 293–314. <https://doi.org/10.1093/nsr/nwt032>
- 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>
- Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2016). NoSQL database systems: a survey and decision guidance. *Computer Science - Research and Development*, 1–13. <https://doi.org/10.1007/s00450-016-0334-3>
- Gilbert, S., & Lynch, N. (2002). Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News*, *33*(2), 51–59.

- Golfarelli, M., & Rizzi, S. (2009). *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, Inc.
- Goss, R. G., & Veeramuthu, K. (2013). Heading Towards Big Data: Building A Better Data Warehouse For More Data, More Speed, And More Users. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2013 24th Annual SEMI* (pp. 220–225). IEEE.
- Gruenheid, A., Omiecinski, E., & Mark, L. (2011). Query optimization using column statistics in Hive. In *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/2076623.2076636>
- Gupta, A. (2015). Big Data Analysis Using Computational Intelligence and Hadoop: A Study. In *2nd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 1397–1401). IEEE.
- Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* (pp. 363–366). IEEE. <https://doi.org/10.1109/ICPCA.2011.6106531>
- Helfert, M., & Von Maur, E. (2001). A Strategy for Managing Data Quality in Data Warehouse Systems. In *Sixth International Conference on Information Quality* (pp. 62–76).
- Holmes, A. (2012). Hadoop in a heartbeat. In *Hadoop in practice* (pp. 3–23). Manning Publications Co.
- Hortonworks, I. (2017). Hortonworks. Retrieved October 22, 2017, from <https://hortonworks.com>
- Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E. N., O'Malley, O., ... Zhang, X. (2014). Major technical advancements in Apache Hive. In ACM (Ed.), *SIGMOD'14 - International Conference on Management of Data* (pp. 1235–1246). Association for Computing Machinery. <https://doi.org/10.1145/2588555.2595630>
- Huang, Y., & Luo, T.-J. (2014). *NoSQL database: A Scalable, Availability, High Performance Storage for Big Data. Joint International Conference on Pervasive Computing and the Networked World* (Vol. 8351 LNCS). Springer. https://doi.org/10.1007/978-3-319-09265-2_19
- Inmon, W. H. (2005). *Building the Data Warehouse, Third Edition*. John wiley & sons.
- Jagadish, H. V. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M. J. M., Ramakrishnan, R., & Shahabi, C. Big Data and Its Technical Challenges, 57 Communications of the ACM § (2014). Association for Computing Machinery. <https://doi.org/10.1145/2611567>
- Jarke, M., Jeusfeld, M. A., Quix, C., & Vassiliadis, P. (1999). Architecture and quality in data warehouses: An extended repository approach. *Information Systems*, 24(3), 229–253. [https://doi.org/10.1016/S0306-4379\(99\)00017-4](https://doi.org/10.1016/S0306-4379(99)00017-4)
- Jukic, N., Jukic, B., Sharma, A., Nestorov, S., & Arnold, B. K. (2017). Expediting analytical databases

- with columnar approach. *Decision Support Systems*, 95, 61–81.
- Justin Samuel, S., Koundinya, R. V. P. V. P., Sashidhar, K., & Bharathi, C. R. R. (2006). A Survey on Big Data and Its Research Challenges. *ARPJ Journal of Engineering and Applied Sciences*, 10(8), 3343–3347.
- Kaisler, S., Armour, F., Espinosa, J. A. A., & Money, W. (2013). Big Data : Issues and Challenges Moving Forward. In *System sciences (HICSS), 2013 46th Hawaii international conference on* (pp. 995–1004). IEEE. <https://doi.org/10.1109/HICSS.2013.645>
- Katal, A., Wazid, M., & Goudar, R. H. (2013). Big data: Issues, challenges, tools and Good practices. In *Contemporary Computing (IC3), 2013 Sixth International Conference on* (pp. 404–409). IEEE. <https://doi.org/10.1109/IC3.2013.6612229>
- Kaur, K., & Rani, R. (2013). Modeling and querying data in NoSQL databases. In *Big Data, 2013 IEEE International Conference on* (pp. 1–7). IEEE. <https://doi.org/10.1109/BigData.2013.6691765>
- Khan, M. A., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data: Understanding Big Data to extract Value. In *American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the* (pp. 1–5). IEEE. <https://doi.org/10.1109/ASEEZone1.2014.6820689>
- Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., ... Gani, A. (2014). Big Data: Survey, Technologies, Opportunities, and Challenges. *The Scientific World Journal*, 2014. <https://doi.org/10.1155/2014/712826>
- Kim, H., Ravindra, P., & Anyanwu, K. (2013). Optimizing RDF(S) queries on cloud platforms. In *22nd International Conference on World Wide Web* (pp. 261–264). ACM.
- Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition*. John Wiley & Sons.
- Krishnan, K. (2013). *Data Warehousing in the Age of Big Data*. Elsevier Inc.
- Lam, C. (2010). Introducing Hadoop. In *Hadoop in Action* (pp. 1–22). Manning Publications Co.
- Mohanty, S., Jagadeesh, M., & Srivatsa, H. (2013). *Big data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics*. Apress.
- O'Neil, P., O'Neil, B., & Chen, X. (2007). The star schema benchmark (SSB).
- Peffer, 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>
- Philip Chen, C. L., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275, 314–347.

<https://doi.org/10.1016/j.ins.2014.01.015>

- Pokorný, J. (2015). Database Technologies in the World of Big Data. In *CompSysTech'15 - International Conference on Computer Systems and Technologies* (Vol. 1008, pp. 1–12). ACM. <https://doi.org/10.1145/2812428.2812429>
- Pritchett, D. (2008). BASE: An Acid Alternative. *Queue*, 6(3), 48–55. <https://doi.org/10.1145/1394127.1394128>
- Prokopp, C. (2014). *The Free Hive Book*. GitHub. Retrieved from <https://github.com/Prokopp/the-free-hive-book>
- Rekatsinas, T., Dong, X. L., & Srivastava, D. (2014). Characterizing and selecting fresh data sources. In *2014 ACM SIGMOD International Conference on Management of Data* (pp. 919–930).
- Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Inc.
- Salloum, M., Dong, X. L., Srivastava, D., & Tsotras, V. J. (2013). Online ordering of overlapping data sources. *Proceedings of the VLDB Endowment*, 7(3), 133–144.
- Sandoval, L. J. (2016). Design of business intelligence applications using big data technology. In *Central American and Panama Convention (CONCAPAN XXXV), 2015 IEEE Thirty Fifth* (pp. 1–6). Institute of Electrical and Electronics Engineers Inc. <https://doi.org/10.1109/CONCAPAN.2015.7428454>
- Santos, M. Y., & Costa, C. (2016a). Data Models in NoSQL Databases for Big Data Contexts. In Y. Tan & Y. Shi (Eds.), *International Conference on Data Mining and Big Data* (pp. 475–485). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-40973-3_48
- Santos, M. Y., & Costa, C. (2016b). Data Warehousing in Big Data: From Multidimensional to Tabular Data Models. In *C3S2E'16 - Ninth International C* Conference on Computer Science & Software Engineering* (p. 10). <https://doi.org/10.1145/2948992.2949024>
- Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Martinho, B., Vale Lima, F., & Costa, E. (2017). Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware [Experiments and Analyses]. In *IDEAS '17 - 21st International Database Engineering & Applications Symposium*.
- Shaw, S., Vermeulen, A. F., Gupta, A., & Kjerrumgaard, D. (2016). *Practical Hive: A Guide to Hadoop's Data Warehouse System*. Apress.
- Srivastava, S., Agarwal, S., Srivastava, A., & Pandey, A. K. (2016). Big data - An emerging and innovative technology: Survey. In *Computational Intelligence & Communication Technology (CICT), 2016 Second International Conference on* (pp. 180–183). IEEE. <https://doi.org/10.1109/CICT.2016.43>
- Theodoratos, D., & Sellis, T. (1999). Designing data warehouses. *Data & Knowledge Engineering*, 31(3),

279–301. [https://doi.org/10.1016/S0169-023X\(99\)00029-4](https://doi.org/10.1016/S0169-023X(99)00029-4)

- Thusoo, A., Sarma, J. Sen, Jain, N., Shao, Z., Chakka, P., Anthony, S., ... Murthy, R. (2009). Hive - A Warehousing Solution Over a Map-Reduce Framework. In *Proceedings of the VLDB Endowment* (pp. 1626–1629). <https://doi.org/10.14778/1687553.1687609>
- Thusoo, A., Sen Sarma, J., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... Murthy, R. (2010). Hive - A Petabyte Scale Data Warehouse using Hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (pp. 996–1005). IEEE. <https://doi.org/10.1109/ICDE.2010.5447738>
- Vajk, T., Fehér, P., Fekete, K., & Charaf, H. (2013). Denormalizing data into schema-free databases. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on* (pp. 747–752). IEEE.
- White, T. (2012). *Hadoop: The definitive guide* (3rd Editio). O'Reilly Media, Inc.
- Wu, S., Li, F., Mehrotra, S., & Ooi, B. C. (2011). Query optimization for massively parallel data processing. In *2nd ACM Symposium on Cloud Computing* (p. 12). ACM. <https://doi.org/10.1145/2038916.2038928>
- Yangui, R., Nabli, A., & Gargouri, F. (2016). Automatic Transformation of Data Warehouse Schema to NoSQL Data Base: Comparative Study. *Procedia Computer Science, 96*, 255–264.
- Zikopoulos, P., & Eaton, C. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data* (1st ed.). McGraw-Hill Osborne Media.

APÊNDICES

Apêndice 1 - *Queries*

Q1.1

```
select sum(lo.extendedprice*lo.discount) as revenue
from lineorder lo join date_dim d on lo.orderdate = d.datekey
where d.year = 1993 and lo.discount between 1 and 3 and lo.quantity < 25;
```

Q1.2

```
select sum(lo.extendedprice*lo.discount) as revenue
from lineorder lo join date_dim d on lo.orderdate = d.datekey
where d.yearmonthnum = 199401 and lo.discount between 4 and 6 and lo.quantity between 26 and 35;
```

Q1.3

```
select sum(lo.extendedprice*lo.discount) as revenue
from lineorder lo join date_dim d on lo.orderdate = d.datekey
where d.weeknuminyear = 6 and d.year = 1994 and lo.discount between 5 and 7 and lo.quantity between
26 and 35;
```

Q2.1

```
select sum(lo.revenue), d.year, p.brand1
from lineorder lo join date_dim d on lo.orderdate = d.datekey
                join part p on lo.partkey = p.partkey
                join supplier s on lo.suppkey = s.suppkey
where p.category = 'MFGR#12' and s.region = 'AMERICA'
group by d.year, p.brand1
order by d.year, p.brand1;
```

Q2.2

```
select sum(lo.revenue), d.year, p.brand1
from lineorder lo join date_dim d on lo.orderdate = d.datekey
                join part p on lo.partkey = p.partkey
                join supplier s on lo.suppkey = s.suppkey
where p.brand1 between 'MFGR#2221' and 'MFGR#2228' and s.region = 'ASIA'
group by d.year, p.brand1
order by d.year, p.brand1;
```

Q2.3

```
select sum(lo.revenue), d.year, p.brand1
from lineorder lo join date_dim d on lo.orderdate = d.datekey
      join part p on lo.partkey = p.partkey
      join supplier s on lo.supplykey = s.supplykey
where p.brand1 = 'MFGR#2221' and s.region = 'EUROPE'
group by d.year, p.brand1
order by d.year, p.brand1;
```

Q3.1

```
select c.nation, s.nation, d.year, sum(lo.revenue) as revenue
from lineorder lo join customer c on lo.custkey = c.custkey
      join supplier s on lo.supplykey = s.supplykey
      join date_dim d on lo.orderdate = d.datekey
where 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;
```

Q3.2

```
select c.city, s.city, d.year, sum(lo.revenue) as revenue
from lineorder lo join customer c on lo.custkey = c.custkey
      join supplier s on lo.supplykey = s.supplykey
      join date_dim d on lo.orderdate = d.datekey
where 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;
```

Q3.3

```
select c.city, s.city, d.year, sum(lo.revenue) as revenue
from lineorder lo join customer c on lo.custkey = c.custkey
      join supplier s on lo.supplykey = s.supplykey
      join date_dim d on lo.orderdate = d.datekey
where (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;
```

Q3.4

```
select c.city, s.city, d.year, sum(lo.revenue) as revenue
from lineorder lo join customer c on lo.custkey = c.custkey
      join supplier s on lo.supkey = s.supkey
      join date_dim d on lo.orderdate = d.datekey
where (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;
```

Q4.1

```
select d.year, c.nation, sum(lo.revenue - lo.supplycost) as profit
from lineorder lo join date_dim d on lo.orderdate = d.datekey
      join customer c on lo.custkey = c.custkey
      join supplier s on lo.supkey = s.supkey
      join part p on lo.partkey = p.partkey
where 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;
```

Q4.2

```
select d.year, s.nation, p.category, sum(lo.revenue - lo.supplycost) as profit
from lineorder lo join date_dim d on lo.orderdate = d.datekey
      join customer c on lo.custkey = c.custkey
      join supplier s on lo.supkey = s.supkey
      join part p on lo.partkey = p.partkey
where 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;
```

Q4.3

```
select d.year, s.city, p.brand1, sum(lo.revenue - lo.supplycost) as profit
from lineorder lo join date_dim d on lo.orderdate = d.datekey
      join customer c on lo.custkey = c.custkey
      join supplier s on lo.supkey = s.supkey
      join part p on lo.partkey = p.partkey
where 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.brand1
order by d.year, s.city, p.brand1;
```

Apêndice 2 – Resultados do cenário B com *Distributed Join*

A Tabela 39 e a Tabela 40 apresentam os tempos de processamento obtidos com o Presto para FE=30, utilizando o *DJ* e comparando com os resultados obtidos com a utilização do *BJ*.

Tabela 39 Tempos de Processamento com Particionamento e DJ para FE=30 (em segundos)

	Partições						
	<i>Od_Year</i>	<i>S_Region</i>	<i>P_MFGR</i>	<i>Od_Year, Monthnum</i> <i>year</i>	<i>Od_Year, S_Region</i>	<i>S_region, S_Nation, S_City</i>	<i>Od_Year, S_Region, P_MFGR</i>
Q1.1	2	6	6	4	2	13	3
Q1.2	5	6	6	14	7	12	13
Q1.3	2	6	6	3	2	11	3
Q2.1	28	7	29	37	7	8	9
Q2.2	28	7	28	37	7	8	9
Q2.3	28	7	28	36	7	8	8
Q3.1	20	6	22	27	6	7	7
Q3.2	18	19	19	24	18	3	25
Q3.3	17	18	18	23	17	2	24
Q3.4	18	18	18	26	20	2	26
Q4.1	43	10	18	48	11	11	6
Q4.2	11	7	12	13	4	8	3
Q4.3	11	27	28	13	12	3	13
TOTAL	232	143	239	305	121	97	150
TOTAL (BJ)	87	66	90	143	63	70	91
DIF	166%	115%	164%	113%	93%	38%	64%

Tabela 40 Tempos de Processamento com Bucketing e Particionamento+Bucketing e com DJ para FE=30 (em segundos)

	Buckets					Partições + Buckets		
	<i>Custkey</i>	<i>Orderkey</i>	<i>Suppkey</i>	<i>Orderkey (Sorted by Orderdate)</i>	<i>Orderdate, Custkey, Suppkey, Partkey</i>	<i>P=Od_Year B=Orderkey</i>	<i>P=S_region B=Suppkey</i>	<i>P=Od_Year, S_Region B= Suppkey</i>
Q1.1	6	7	6	6	7	2	7	2
Q1.2	4	7	6	6	6	6	7	8
Q1.3	5	6	6	6	6	2	6	3
Q2.1	30	33	30	31	38	31	7	9
Q2.2	29	33	30	31	35	30	7	8
Q2.3	28	32	30	31	35	31	7	8
Q3.1	10	26	24	24	29	22	7	7
Q3.2	6	22	20	20	27	19	20	22
Q3.3	6	23	21	22	25	19	20	21
Q3.4	6	20	20	19	27	21	20	24
Q4.1	43	49	44	47	53	45	10	12
Q4.2	31	32	29	31	38	12	8	5
Q4.3	29	32	31	32	38	12	30	13
TOTAL	233	321	296	306	365	251	156	141
TOTAL (BJ)	101	133	120	115	121	100	77	81
DIF	132%	141%	147%	166%	201%	152%	102%	73%

A Tabela 41 e a Tabela 42 apresentam os tempos de processamento obtidos com o Presto para FE=100, utilizando o DJ e comparando com os resultados obtidos com a utilização do BJ.

Tabela 41 Tempos de Processamento com Particionamento e DJ para FE=100 (em segundos)

	Partições				
	<i>Od_Year</i>	<i>S_Region</i>	<i>Year, S_Region</i>	<i>S_Region, S_Nation, S_City</i>	<i>Od_Year, S_Region, P_MFGR</i>
Q1.1	4	17	4	38	9
Q1.2	14	14	20	38	43
Q1.3	4	14	4	35	9
Q2.1	94	19	20	24	43
Q2.2	92	19	20	25	8
Q2.3	91	19	20	24	26
Q3.1	65	18	16	22	25
Q3.2	56	59	56	5	26
Q3.3	55	57	54	3	20
Q3.4	59	58	62	3	76
Q4.1	143	30	31	35	76
Q4.2	35	20	10	24	83
Q4.3	34	87	35	6	16
TOTAL	745	431	352	281	459
TOTAL (BJ)	239	166	149	193	259
DIF	211%	160%	137%	46%	77%

Tabela 42 Tempos de Processamento com Bucketing e Particionamento+Bucketing e com DJ para FE=100 (em segundos)

	Buckets			Partições + Buckets		
	<i>Orderkey</i>	<i>Suppkey</i>	<i>Orderdate, Custkey, Suppkey, Partkey</i>	<i>P=Od_Year B=Orderkey</i>	<i>P=S_Region B=Suppkey</i>	<i>P= Od_Year, S_Region B=Suppkey</i>
Q1.1	17	18	18	4	25	6
Q1.2	14	16	14	14	24	25
Q1.3	14	15	13	3	23	5
Q2.1	95	93	118	95	24	24
Q2.2	92	91	112	95	24	24
Q2.3	94	93	108	94	23	23
Q3.1	72	77	90	64	21	21
Q3.2	62	64	80	57	77	74
Q3.3	60	63	77	54	80	70
Q3.4	58	63	77	61	79	74
Q4.1	143	143	177	196	36	36
Q4.2	95	94	119	36	26	10
Q4.3	95	92	118	35	142	40
TOTAL	913	921	1121	811	605	431
TOTAL (BJ)	305	321	305	256	285	220
DIF	199%	187%	268%	216%	113%	95%

Por sua vez, a Tabela 43 apresentam os tempos de processamento obtidos com o Presto para FE=300, utilizando o DJ e comparando com os resultados obtidos com a utilização do BJ.

Tabela 43 Tempos de Processamento com Particionamento e/ou Bucketing e com DJ para FE=300 (em segundos)

	Partições			Buckets		Partições + Buckets		
	<i>Od_Year</i>	<i>S_Region</i>	<i>Od_Year, S_Region</i>	<i>Suppkey</i>	<i>Orderkey, Orderdate, Custkey, Suppkey, Partkey</i>	<i>P=Od_Year B=Orderkey</i>	<i>P=S_Region B=Suppkey</i>	<i>P=Od_Year, S_Region B=Suppkey</i>
Q1.1	9	43	9	45	53	10	44	13
Q1.2	38	36	53	37	40	45	39	71
Q1.3	7	36	8	39	39	8	34	11
Q2.1	274	54	58	167	348	301	55	67
Q2.2	268	54	56	161	329	292	53	69
Q2.3	268	54	57	162	319	293	54	70
Q3.1	194	48	46	207	277	214	47	59
Q3.2	165	173	163	176	243	180	172	219
Q3.3	155	165	156	168	235	185	170	218
Q3.4	171	164	177	172	229	190	172	237
Q4.1	412	87	90	414	536	443	89	112
Q4.2	102	55	25	264	360	111	56	29
Q4.3	99	251	102	261	353	107	261	118
TOTAL	2163	1219	998	2274	3360	2377	1246	1292
TOTAL (BJ)	645	428	399	768	876	835	452	650
DIF	235%	184%	150%	196%	284%	185%	175%	99%

Importa realçar que todos os resultados, aqui apresentados, comprovam o benefício da utilização do *broadcast join* como estratégia de otimização dos *joins*.